**USEFUL LINKS**

* MEAN website - <http://mean.io/>
* MEAN stack tutorial for Beginners - <http://fullstacktutorials.net/mean-stack-tutorial-for-beginners/>
* Tutorial to build a full modern web app with MEAN - <https://thinkster.io/tutorials/mean-stack>
* Introduction to MEAN stack - <https://www.sitepoint.com/introduction-mean-stack/>
* Express Routing - <https://expressjs.com/en/guide/routing.html>
* Mongoose documentation - <http://mongoosejs.com/docs/index.html>
* Getting started with mongodb - <https://docs.mongodb.com/getting-started/shell/>
* MongoDB tutorial - <https://www.tutorialspoint.com/mongodb/>
* Installing MongoDB - <https://www.tutorialspoint.com/mongodb/mongodb_environment.htm>
* Angular tutorial, W3schools - <https://www.w3schools.com/angular/angular_intro.asp>
* Understanding mean stack, coding dojo blog - <http://www.codingdojo.com/blog/understanding-mean-stack/>
* setting up a mean stack single page application - <https://scotch.io/tutorials/setting-up-a-mean-stack-single-page-application>

**THE BASICS OF MEAN**

* The **MEAN** stack is a free and open-source, user-friendly javascript framework for building dynamic websites and web applications. Below are it’s 4 main components.
* **Node.Js**: It’s a server-side **javascript environment** based on V8. A platform built on Chrome’s Javascript runtime for easily building fast, scalable network applications.
* **Express.JS**: a minimal and flexible node.js web application **framework**, providing a robust set of features for building single, multi-page, and hybrid web applications. It has powerful middleware features and is inspired by the popular ruby framework Sinatra.
* **MongoDB**: the leading **NoSQL database**, empowering businesses to be more agile and scalable. It requires no schema and saves data in binary JSON format, which makes it easier to pass between client and server.
* **Angular.JS**: A frontend JavaScript **library/framework** created and maintained by google that is used to create single page applications. This lets you extend HTML vocabulary/attributes with “directives” and binds data to HTML with “expressions”
* **MEAN** is a big deal because it allows you to write JavaScript on the front and back end, where previously you needed to know another language to write the back end. A major benefit is that it’s extremely quick to prototype with, you can just use javascript, and quickly change your database without any migrations. JavaScript adds to its power, usefulness, flexibility, and overall popularity. It also is very useful for single page web applications.
* **Nodemon:** a utility that will monitor for any changes in your code and automatically restart your server. Perfect for development, it is installed via npm.
* **Bower**: a manager for client-side packages
* **Mongoose**: an ODM or Object Document Mapper that provides high level abstraction to make our lives much easier while working with MongoDB.
* **Grunt CLI**: a Javascript task runner that can be used to automate tasks such as deployment, unit testing, minification, concatenation, linting, etc. (**NOT YET FAMILIAR WITH**)

**THINGS TO KEEP IN MIND/REMEMBER**

* You can run a node server by typing **nodemon file\_name.js**, for example, to run most of my full mean projects, I would navigate to that project folder, then type **nodemon server.js**, since that is the key file that directs the project, but you could name it whatever you want, you specify to nodemon which file to run. You shut down the server by typing **ctrl + c**
* In order to use mongodb in any of your projects, you have to have a mongod server running, to do so, open a terminal, navigate to the root directory with “cd”, then type “**sudo mongod**”, this starts a mongod server, leave this tab open for as long as you need to access mongo. Be sure to exit the server with **ctrl + c** before you close the terminal or your computer or it will keep running in the background.

**UNDERSTANDING HOW MEAN WORKS/ THE WORKFLOW**

* **NOTE**: I’m using the gamers\_for\_life\_message\_board app in my projects folder for reference of the workflow.
* **WHEN YOU START THE SERVER:** This is the default process that MEAN goes through whenever you boot up a server for a project. Whenever you type **nodemon server.js**, node goes to that file, server.js, and reads it. This file is the heart of your entire application, it tells it where to go, what to include, what port to run on, etc. From there, it goes down the page and loads what that file tells it too. From there, it all depends on what order you have the required files in, though most of the files require another so if they’re not in the right order things would error out. So **first**, the server file is read, between the start of that file and the next step, server.js tells it where the client side files and bower components are located, but that doesn’t really come into play until you visit the site, we talk about that below. **Second**, once it requires the mongoose file, it reads that. **Third**, at the end of the mongoose file, it makes it read all of the model files. **Fourth**, it then finishes the modesl and mongoose and goes back to server.js, where it is then redirected to the routes.js since it is required. **Lastly**, at the top of the routes.js file, it requires the controller files so those are the last files that it reads. Once it finishes with the controller files, it goes back to the routes.js file, finishes that, then goes back to and finishes the server.js file.
* **IMPORTANT NOTE FOR WHEN YOU VISIT THE WEBSITE/APPLICATION:** The important part to understand is what happens first. Say you go to <http://localhost:5000/>, ordinarily a routes.js file or its equivalent would tell you where to go for the ‘/’ or root route (ex: in a mean application “app.get(‘/’, function….)” would do it). And if you have that in your routes.js file and have that loaded before your client folder, then that will take precedence and tell it what to do. But normally, you don’t have that in your routes.js file (standard for my apps), and that file is loaded after your client folder. In this instance, you’re loading up the application but not telling node what to do for an empty/root route or the “/” route. So, like many webservers, express will automatically load “index.html” for the root route if it’s not told what to do. So, in a standard single page MEAN application, you hit the site and express loads the file called “index.html”, if you don’t have that, then nothing gets loaded and you have a blank screen.
* **WHEN YOU VISIT THE WEBSITE/APPLICATION - INITIAL LOADING:** **First**, as stated in the bullet point above, when you hit the app/website, express loads index.html. **Second**, once index is loaded, it starts reading the file and loads all of your stylesheets and bower components first (if you try to load angular things before the bower components, it will not work). The next big step is whenever it hits the angular files. These files should be loaded in the proper order since the factory requires the angular module, and the controller requires the module and usually a factory. So, the **third** big step is whenever it loads the file containing your angular module, usually app.js. This is where you load the major angular components/modules, and where you set up your angular routing. The **fourth** important thing happens is after it finishes with your app.js file, it loads first your factories then your controllers. Now, the **fifth** important part is **when angular routing kicks in**. I’m not certain on when exactly it kicks in, but it is most likely after all of the files in the header of index.html gets loaded and it hits the **ng-view** in the body. It goes to the app.js file to load a partial, and when you first hit the site, it either loads the ‘/’ route, or where the .otherwise tells it to go if the ‘/’ route isn’t listed. this tells it what partial to initially load within the ng-view.
* **THE REGULAR WORKFLOW OF MEAN WHEN USING THE APP:** So, after the startup, you are in your app with a partial loaded. Say you hit a button to load data from the database. **First**, is the action that makes it hit the controller, whether it be the loading of a partial or a button/form. **Second**, you go to the controller, which does its thing and can stop from there, but it usually hits the **third** step and sends you to the factory via a function usually. This is where the logic happens, once again, it can stop here, but if you need to hit the database, then the **fourth** step is when the factory makes an $http request (usually with some json data) and sends you to the server. The **fifth** step is after making the $http request, you are sent to the router.js file in /server/config, which tells you what to do for each route. The **sixth** step is what the router.js file tells you to do from there, which is ordinarily to go to a certain method within a certain controller. Once you are in the controller, usually the **seventh** step is to hit the model and mongoDB via a query and either callbacks or promises. The **eight** step is the response you get, once the query is done, the controller sends back a response in json format. The **ninth** and final step is sending the data back, first to the factory, then to the controller, and finally back to the view. Most likely the factory and controller would do something to it, but this is a simplified verions.

**PROJECTS TO LOOK AT FOR A REFRESHER**

\* you can look in the node and express folder for some basics/examples on them, but this isn’t necessary since this isn’t the format you’ll use for full MEAN

**EXPRESS, NODE, AND THE BASICS**

\*These projects aren’t full mean but help you to understand pieces of it, or things that will be used in full mean

See survey\_form (only server.js and result.ejs in /views) in the express folder for:

* An example of a server.js file, with some minor differences from a full MEAN version
* Explanations on what the 4 variables at the top of every server.js file is (ex: express, bodyparser)
* explanations on the app.use and app.set are for
* Examples and to understand app.get and app.post, basic express routing (used in full mean later on), on how data is sent over from the view
* how to send over data from the back to the front and how to access that data in the view

see quoting\_dojo\_redux in the express folder for:

* intro to mongo and connecting to the database
* creating a model in the database and basic validation
* Basic examples of find, update, and creating/saving queries
* Using a for statement to show query results or the data sent from the back end in the view

Next, see message\_board in the express folder for:

* Examples of using a promise instead of a callback and promise format
* Creating schema’s that have relationships (many to one, many to many, etc.) and how to create them with foreign keys
* Using the .populate() method in your queries to include the models that it is associated with.
* Creating new records when they are in a relationship (ex: find Message, create a Comment, then add that Comment to the Message.comments column, see file for more info)
* more examples of how I would use a promise instead of a callback for each query.

Next, see express\_modularization in the express folder for:

* a look at how you separate the back end into server.js, and the rest into the server folder with a config, controller, and models file.
* See examples of how the routes, mongoose, controller, and model files are supposed to look in full mean. Still slightly different from full mean, but I left comments to explain it.
* Also, explanations for what each file should do, ex: controllers, models, routes, mongoose, etc.

Next, see mongoose\_dashboard\_modularized in the express folder for:

* another example of full mean format when it comes to folders (config, controller, models) and separting the front from the server code (without angular anyway).
* More examples of promises with basic CRUD operations and using them in a semi-full-mean app (just missing angular)

**FOR SOCKET BASICS AND CODE**

\*Currently don’t have a full socket app, just the basics to look at

see socket\_practice in the express folder for:

* how to configure it on the back end so it can be used.
* How to send a response back to the front
* how to configure it in the view so it’s ready to use
* how to send something to the back end on an action and get a response back

Next, see survey\_form\_revisited in the express folder for:

* Sending data to the back through a form and socket
* Receiving the data, doing things with it, and sending it back
* Receiving the response on the front and putting it on the page without reloading it, using JSON.stringify() and appending the response to an existing div.

Next, see epic\_button\_game in the express folder for:

* Comments on my code
* And how to send data automatically every time someone connects
* Different examples, such as emitting to everyone
* Making something on the view equal a variable sent from the server and updating it

**ANGULARJS**

\*You should look at the word doc: angular\_1\_basics\_and\_refresher

Start with partials\_and\_custom\_users in angular for:

* The basics of angularjs, angular routing, factories, and controllers with full comments (all in the index.html file)
* Creating an angular module, basic routing, setting up your index.html file and partials
* Using partials
* communicating between the view, controllers, and factories (model)
* Using and not using $scope in your controllers
* Accessing/using data from your controller and factory in the view
* basic examples of angular directives
* sending form data back to the controller as an object, and calling controller functions from the view
* examples of the $location service
* Basic format and examples of your controller, factory, and app module/routes (app.js file)