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>

THE BASICS OF MEAN

* **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 scalabale. 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 for your application, the resulting environment is extraordinarily expressive, readable, and quick to develop. It also provides some awesome features like two-way data binding
* **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.
* **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 runa 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

**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