USEFUL LINKS

* Using ES6 Promises instead of callbacks for mongoose queries - <http://erikaybar.name/using-es6-promises-with-mongoosejs-queries/>
* Quora, difference between callback and a promise - <https://www.quora.com/Whats-the-difference-between-a-promise-and-a-callback-in-Javascript>
* software engineering, is there really a fundamental difference between callbacks and promises? - <https://softwareengineering.stackexchange.com/questions/302455/is-there-really-a-fundamental-difference-between-callbacks-and-promises>
* Promises (about) - <https://promisesaplus.com/#point-2>
* Switching out callbacks with promises in mongoose - <https://eddywashere.com/blog/switching-out-callbacks-with-promises-in-mongoose/>
* Avoiding callback hell while using mongoose - <http://www.summa.com/blog/avoiding-callback-hell-while-using-mongoose>
* Using Promises with mongoose ejs - <http://blog.revathskumar.com/2015/07/using-promises-with-mongoosejs.html>

ABOUT

* Callbacks and Promises are two ways to handle asynchronous programming (see bullet below for asynchronous), though both work, promises are generally considered to be better, due to avoiding ‘callback hell’. This concept and the differences between the two is pretty complex and complicated, so this word doc was made to help explain and understand everything.
* Asynchronous programming is when your code stops running synchronously, meaning your code stops running top to bottom. Synchronous is when your code runs from top to bottom, executing each line of code as you go. Asynchronous is when that stops for some reason, or when you have delayed code. An example of this is the http routes you hit in the factories of your full mean project. Ex: “factory.login = function(data, callback){ $http.post('/login', data).then(…..)”, in this example, you send a post to ‘/login’ and the code waits to get it back. When it does, the code after “.then” gets executed. But this is asynchronous code, it stops running normally and waits for the response. Though the code in this example is a callback, either way, if I had a console.log outside the .then() code, it would get run before the code inside of the .then(), hence the nature of asychronouse code
* What is a promise? A promise is a representation of a value that isn’t necessarily known at its creation time. When you use a promise, the function returns a promise rather than an asynchronous call accepting a callback. The code that needs the promise waits until the promise is fulfilled before executing the next step, in Javascript, this is shown as a .then() code.

WHY PROMISES OVER CALLBACKS

* Callback hell is when you have nested callbacks, it often gets complicated, hard to read, and very wide before high, meaning it looks like the code in blue below:

Department.find({name: 'Development'}, function(error, department){

Manager.find({departmentId: department.\_id}, function(error, manager){

Employee.find({manager: manager.name},function(error, employees){

res.render('employees', {

employees: employees

});

});

});

});

});

* The above code also runs into the issue of, if there is a block or error anywhere in that block, the entire thing gets blocked. Promises not only make your code easier to red, but instead of having an error handler (“if (err)”) for each level, we can have one error handler for multiple blocks. The blue code below is the same as the code above but with promises instead of callbacks:

Department.getDepartment("Development")

.then(function(department){

var Manager = mongoose.model('Manager');

return Manager.findManager(department);

})

.then(function(manager){

var Employee = mongoose.model('Employee');

return Employee.getEmployeesForManager(manager);

})

.then(function(employee){

res.send('/employees', {

employee: employee

});

})

.catch(error) {

console.log(error);

})

.done();

});

* The difference between the pyramid like callback one and the promise block above can already be seen, imagine code where there are even more nested callbacks/promises. Also, to provide an example of not only a mongoose/mongoDb case, but one where the callback needs an error handler for each level but the promise only needs one, see the callback part in red below:

User.findById('123', function(err, user) {

if (err) {

return console.log('error:', err);

}

user.name = 'Robert Paulson';

user.save(function(err) {

if (err) {

return console.log('error:', err);

}

console.log('updated user: ' + user.name);

});

});

* Now see the promise version with only one error handler in red below:

var promise = User.findById('123').exec();

promise.then(function(user) {

user.name = 'Robert Paulson';

return user.save();

})

.then(function(user) {

console.log('updated user: ' + user.name);

})

.catch(function(err){

console.log('error:', err);

});

* In that example, the .catch serves as the error handler for both .then()s

d