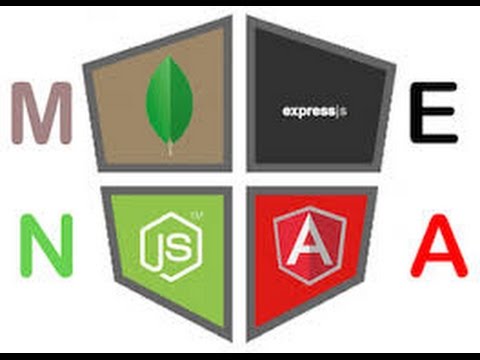
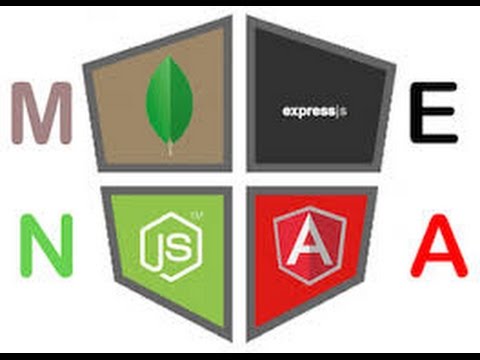
**Exercise06\_02\_01 – Step 1**

1. We will convert our database functionality over to Mongoose to take advantage of its advanced data modeling capabilities. The first step is to locally install it with npm, making a dependency:  
   ***npm install mongoose –save***Check ***package.json*** for a correct install.
2. Let’s begin to replace our MongoDB native code with Mongoose code. We will begin in ***server.js*** by replacing our mongoDB ***require()***: and change some of our variables:  
   **var mongoose = require('mongoose');  
   var dbName = 'test';  
   var url = 'mongodb://localhost:27017/' + dbName;**
3. Next we will make minor variations to our database connect code to do it Mongoose style:  
   ***mongoose.connect(url, (err, db) => {  
    if (err) {  
    return console.log('Error: ' + err);  
    }  
    console.log('Connected to database: ' + dbName);  
   });***  
   Start up the server and let’s do a connection test.
4. Now let’s create a Mongoose ***schema*** and the ***model***. We will store it in a variable, capitalized, to indicate that it is going to be an ***object***. Notice that we are going to name the fields in a ***JSON*** object, and also indicate to Mongoose what the ***data*** ***type*** is:  
   var database;  
   ***var Message = mongoose.model('Message', {  
    msg: String  
   });***
5. We need to modify the code in our app.post() route to use the Mongoose style object-oriented technique:  
   app.post('/api/message', (req, res) => {  
    console.log(req.body);  
    ***var message = new Message(req.body);  
    message.save();*** res.status(200);  
   });  
   Let’s run the server and try another front-end post. Check the results in the node console and in RoboMongo. Notice that Mongoose has added a ***new*** ***field*** to the document, ***\_v***. this is used to track revisions and changes.
6. Let’s build ourselves a ***function*** that will return all of our stored messages Mongoose style. We can do this directly below our ***app.post()*** route:  
   ***function GetMessages() {  
    Message.find({}).exec((err, result) => {  
    console.log(result);  
    });  
   }***
7. Let’s debug the function in our database ***connect***:  
    console.log('Connected to database: ' + dbName);  
    ***GetMessages();***Let’s run the server and give that a test, we should see all of the messages from our database in the node console.

**Exercise06\_02\_01 – Step 2**

1. Let’s create a new ***endpoint*** to get all of our messages. First, let’s remove our debug ***GetMessages()*** call from the database ***connect()*** method.
2. Now let’s turn ***GetMessages()*** into a function that will be our route ***callback*** by giving it ***req*** and ***res*** parameters, and removing the ***console.log()*** in favor of an HTTP ***response***:  
   function GetMessages(***req, res***) {  
    Message.find({}).exec((err, result) => {  
    ***res.send(result);*** });  
   }
3. We can create our new ***route*** above our ***app.post()*** route, and make use of the same URL, because we are using a different HTTP verb. We will also add the ***GetMessages()*** function to act as our route ***callback*** as follows:  
   ***app.get('/api/message', GetMessages);***Run the server, and we will test on the front-end. We could use Postman, but because it is a ***GET***, we can just open a clean browser window and type the following into the URL bar: ***localhost:8080/api/message***. This should show our data.

**Exercise06\_02\_01 – Step 3**

1. Time to do head over to the ***front-end*** and do some work in AngularJS to get ourselves a decent message display. Open up ***main.controller.js***, and we will create ourselves a new function called ***getMessages()*** above our postMessage() function. It will use a ***$http.get()***. The AngularJS $http service returns a ***Promise*** for a ***get()*** method, so we will make use of that:  
    ***getMessages() {  
    this.$http.get('http://localhost:8080/api/message')  
    .then((result) => {  
    console.log(result);  
    });  
    }***
2. We can call this function from the controller ***constructor***, because that will happen right away when it gets instantiated:  
    this.$http = $http;  
    ***this.getMessages();***Run the server, and we will test on the front-end. Make sure Developer Tools are open to the Console tab, regenerate the browser if necessary. The console should show us a ***data*** object, and if we ***expand*** down we can see an ***array*** holding our messages from the database.
3. To get AngularJS to handle the display work, we are going to need to get the data into our ***model***. First we need to save a reference to ***this*** in a variable inside the function. This is to protect against the Promise coming back and having ***this*** referencing something else instead of the controller object when we call the function from various places. We can then safely use it to set a ***property*** for the controller so that it will always be current in our model:  
    getMessages() {  
    ***var vm = this;*** this.$http.get('http://localhost:8080/api/message')  
    .then((result) => {  
    ***vm.messages = result.data;*** });  
    }
4. Now let’s get to work on our AngularJS by opening up ***main.html***. Start by making a copy of the entire ***<div>*** with class ***panel panel-default*** and paste it down below itself. Modify it by changing the content of the ***<div>*** with class ***panel-heading*** to ***Messages***, and removing the entire ***<div>*** with the class ***input-group***:  
    ***<div class="panel panel-default">  
    <div class="panel-heading">Messages</div>  
    <div class="panel-body">  
     
    </div>  
    </div>***
5. In the empty ***<div>*** with class ***panel-body***, let’s place a ***<ul>***. It will hold one ***<li>*** element with ***an ng-repeat*** directive. We will base its ***iterator*** on the ***messages*** array we have set in the controller with alias ***main***. We will use an AngularJS expression for the content of the ***<li>***  
    <div class="panel-body">  
    ***<ul class="list-group">  
    <li class="list-group-item"  
    ng-repeat="message in main.messages">  
    {{message.msg}}  
    </li>  
    </ul>*** </div>  
   Our server and browser should be running, if not start everything up. We should be showing a working message display.