Day04 Deployment

part 1 – Folder Setup

1. Copy the folder called **Part12** from **Day03** paste it where all the other days are and rename it to **Day04-Deployment**.
2. Copy the folder called **AngularJS** from **Day02** paste it inside of the folder you just renamed to **Day04-Deployment**. Rename **AngularJS** inside of **Day04** to just **HTML**. It should contain all the HTML/JS/CSS from the second day of sessions.
3. Your **Day04-Deployment** folder should look like the image below, before running **npm install**.

|  |
| --- |
|  |

1. Run **npm install** to create the **node**\_**modules** folder and setup the environment.
2. First task is to serve the **HTML** pages we built on **Day02**. Now that we have a **controller** file, everything that goes to the client will pass through there and the root function will handle serving our HTML page when users land on our root which at the moment is <http://localhost/8000>. We would need to change line 3 of controller.js to serv the index.html file instead.

|  |
| --- |
| **const Weight = require('../models/employees);**  **exports.getdefault=function(req, res){**  **res.send(‘../HTML/index.html’);**  **};** |

1. Execute **node http\_server** and go to the address from #5. Of course this will not work because whatever is between the single quotes will simply print out on the browser screen. Node has a different function to handle html files, it is called **sendFile()** and it is attached to the response object.

|  |
| --- |
| **const Weight = require('../models/employees);**  **exports.getdefault=function(req, res){**  **res.sendFile(‘../HTML/index.html’);**  **};** |

Remember to stop and start the service

1. This did not work but it provided some clues, something about a path. In order to serve static pages, we need the **path** package, so in the controller.js file, declare a variable and point it to the **path** package.

|  |
| --- |
| **const Weight = require('../models/employees);**  **const path = require("path");**  **exports.getdefault=function(req, res){** |

1. The path object has a method called **join()** which we can use to obtain the current path of the application. If we then concatenate the root path with the path where our **HTML** files live, we can finally obtain a true absolute path to our files

|  |
| --- |
| **const path = require("path");**  **exports.getdefault=function(req, res){**  **res.sendFile(path.join(\_\_dirname + '/../HTML/index.html'));**  **};** |

1. Although the html file is served, it appears to not know that CSS and JS exists, we need to let Express know that these files exist and that it should use them. Open routes.js and include the following lines:

|  |
| --- |
| **module.exports = function(app){**  **const express= require('express');**  **app.use(express.static(\_\_dirname + '/../HTML'));**  **let controller = require('../controllers/controller');** |

Although we had to require **Express** in the http\_server.js file, we still have to do it again in this file. Also we just need the static method to know where the directory is that contains our html/css/js files.

Note: if the above code does not work in U18, use this instead:  
**app.use(express.static(\_\_dirname + './../HTML'));**

1. At this point, if you navigate to team weights in the navigation menu, you should see some records there already. If you do not see anything then you would need to complete the teamweights.html file.
2. Open the scripts.js file inside of the scripts folder. Make sure that the AngularJS controller is obtaining its data from the URL and not the JSON text file.

|  |
| --- |
| **let file = "json.txt";**  **let url = "http://localhost:8000/getallrecords";**  **//**  **let app = angular.module('SkillsApp', [] );**  **app.controller('Weights', function($scope, $http) {**  **$http.get(url).then(function(response){**  **$scope.allWeights = response.data;** |

1. There may be some left over problems from previous days and parts, so hit the f12 key to see if there are any errors and fix them. For example on the home page, there may be a message that angular is not defined. Just include the AngularCDN in the head of the document. Do this for any page that throws this error.

part 2 – Configuring Team Weights

1. We did not have to do much work with teamweights.html, but we can improve the display a bit. Open that html file and go to where the **h2** element is, should be around line 29. Add a new pair of **div** tags underneath in order to wrap the   
   **ng-repeat** block of code.

|  |
| --- |
| **<div id="showRecords">**  **<div id="records" ng-repeat="emp in allWeights">**  **{{emp.empName}} weighed in at {{emp.empWeight}} Kgs**  **</div>**  **</div>** |

Give it an id as well.

1. Now we can target that id in the css, so create a new style for our display of records:

|  |
| --- |
| **#showRecords {**  **margin-left:34px;**  **width:80%;**  **}** |

Refresh the teamweights.html file and adjust the CSS to your liking.

1. We may also want to change the background color of alternating rows just for easier reading, again you can play with the background colors of this code:

|  |
| --- |
| **#showRecords div:nth-child(odd) {**  **background: lightgray;**  **display: block;**  **margin:5px 0px;**  **}** |

1. (optional)  
   you may want to include row numbers, and we can tap into Angular’s **$index** for that.

|  |
| --- |
| **<div id="records" ng-repeat="emp in allWeights">**  **{{$index + 1}} {{emp.empName}} weighed in at {{emp.empWeight}} Kgs**  **</div>** |

1. If you did this, you may notice that the start of each sentence does not really line up. One trick is to pad the leading character with a zero, in other words, mix regular JS and AngularJS together:

|  |
| --- |
| **<div id="records" ng-repeat="emp in allWeights">**  **{{ ("0"+($index+1)).slice(-2) }} {{emp.empName}} weighed in at {{emp.empWeight}} Kgs**  **</div>** |

You could also try this:  
{{ ("&nbsp;"+($index+1)).slice(-2) }} {{emp.empName}} weighed in at {{emp.empWeight}} Kgs

1. Another way if you don’t want the leading zero (or use replace):

|  |
| --- |
| **<div id="showRecords">**  **<div id="records" ng-repeat="emp in allWeights">**  **<span ng-If="($index + 1)<10">&nbsp;&nbsp;{{$index + 1}} {{emp.empName}} weighed in at {{emp.empWeight}} Kgs</span>**  **<span ng-If="($index + 1)>9">{{$index + 1}} {{emp.empName}} weighed in at {{emp.empWeight}} Kgs</span>**  **</div>**  **</div>** |

This can be improved using ng templates and the **ng if else** structure.

part 3 – Configuring MyWeights

For myweights.html, we already have the file created and almost ready to go. We would have to create an html element to display the individual’s record, once found. I am building this on the assumption that we are displaying one record, but in reality, we could be displaying several records, so in that case you can follow the code for teamweights.html.

1. Remove all the **p** tags except for one to show the database record and give it an id:

|  |
| --- |
| **</div>**  **</form>**  **<p id="displaySingle">On [date] you weighed [empWeight] Kgs.</p>**  **</div>**  **<div id="aside">** |

1. In the controller.js file, copy the **deletebyname()** function and rename it something like **getbyformname().**

|  |
| --- |
| **exports.getbyformname=function(req, res){**  **let empToDelete = req.params.employeeName;**  **Weight.deleteOne({empName:empToDelete}, function(err, result) {**  **if (err)**  **res.send(err);**  **res.end(`Deleted ${empToDelete}`);**  **});**  **};** |

1. Change empToDelete to empToFind and the values wont be coming via the url, so **params** wont work, we need to use **body.** Also in the form, our field is called **empName** not **employeeName**, so change this also.

|  |
| --- |
| **exports.getbyformname=function(req, res){**  **let empToFind = req.body.empName;**  **Weight.deleteOne({empName:empToDelete}, function(err, result) {**  **if (err)**  **res.send(err);**  **res.end(`Deleted ${empToDelete}`);**  **});**  **};** |

1. Also change Weight.deleteOne to Weight.find and the parameters inside to match the changes in part 3

|  |
| --- |
| **exports.getbyformname=function(req, res){**  **let empToFind = req.body.empName;**  **Weight.find({empName:empToFind}, function(err, result) {**  **if (err)**  **res.send(err);**  **res.end(`Deleted ${empToDelete}`);**  **});**  **};** |

1. Finally for this part, change the message we send back to the client and change result to result:

|  |
| --- |
| **exports.getbyformname=function(req, res){**  **let empToFind = req.body.empName;**  **Weight.find({empName:empToFind}, function(err, results) {**  **if (err)**  **res.send(err);**  **res.json(results);**  **});**  **};** |

1. Create a route to point to this function, make sure it’s a **post** request/method.

|  |
| --- |
| **app.route('/getbyformname').post(controller.getbyformname);** |

1. We would need to create a new function in the scrpts.js file so that we can post to the proper **api**, right now we have a **get** and a **frmSubmit().** Create a new function within the **controller** scope, just copy the **frmSubmit()** function and modify the name first.

|  |
| --- |
| **$scope.frmFindSingle = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **});**  **};** |

1. Also remove the e**mpWeight** part from the **data** line, this is what we are trying to find.

|  |
| --- |
| **$scope.frmFindSingle = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **});**  **};** |

1. The controller method in controler.js will return whatever the mongo **find()** method gathers, which is quite a lot, so lets reconfigure our new **frmFindSingle()** method to pick out the data we need:

|  |
| --- |
| **$scope.frmFindSingle = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **}).then();**  **};** |

Because angular’s **http()** method works on the basis of **promises** we can chain a **then()** method onto it, to handle any response from the **http()** method.

1. Within the **then()** method we can place an anonymous function to handle the response:

|  |
| --- |
| **$scope.frmFindSingle = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **}).then(function(response){});**  **};** |

The **response** object will be passed into the function, once it is chained in this way

1. With the function in place, lets interrogate the **response** object to get the weight from the document we just found. Pass this to Angular’s **$scope** object.

|  |
| --- |
| **$scope.frmFindSingle = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **}).then(function(response){**  **$scope.empWeight = response.data[0].empWeight;**  **});**  **};** |

1. With our value stored, we just need a way to display it on the HTML (myweights.html) file itself, so we can use the sentence we already have:

|  |
| --- |
| **</form>**  **<p id="displaySingle">{{empName}}, on [date] you weighed {{empWeight}} Kgs.</p>**  **</div>** |

If the myweights.html file was not setup like the teamweights.html file, this may not work

1. We now have to make sure that angular is setup properly, so in the **maincontainer** div tag of the html, add the **ng-app** directive:

|  |
| --- |
| **</ul>**  **</nav>**  **<div id="container" ng-app="SkillsApp">**  **<div id="maincontent">** |

1. In the same way, add the directive **ng-controller="Weights"** to the **maincontent** div tag underneath

|  |
| --- |
| **</ul>**  **</nav>**  **<div id="maincontainer" ng-app="SkillsApp">**  **<div id="maincontent" ng-controller="Weights">**  **<h2>My Records</h2>**  **<form>** |

1. Make sure you call the correct function from the **form** tag

|  |
| --- |
| **<div id="maincontent" ng-controller="Weights">**  **<h2>My Records</h2>**  **<form name="frmCollectWeights" ng-submit="frmFindSingle()">**  **<div>**  **<label for="empName">Name</label>** |

1. Make sure that the form input boxes have the proper Angular directives

|  |
| --- |
| **<form name="frmCollectWeights" ng-submit="frmFindSingle()">**  **<div>**  **<label for="empName">Name</label>**  **<input id="empName" type="text" ng-model="empName"/>**  **</div>**  **<div>** |

1. Stop and start the server, then test the new route and method

part 4 – adding a new record

The HTML file enterweight.html is being used to add a new record to the database. We have to configure that file to work with the **putnewdoc** endpoint.

1. Open the scripts.js file and make sure that the **url** has the proper value, it should be pointing to [**http://localhost:8000/putnewdoc**](http://localhost:8000/putnewdoc). We wrote this **frmSubmit**() function on Day02.
2. Here is the entire function for **frmSubmit()**

|  |
| --- |
| **$scope.frmSubmit = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/putnewdoc',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **})**  **.then(function(response){**  **console.log(response.data);**  **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

1. In a case like this, it may be a good idea to send our user to a totally different page after the first entry, so that they don’t try to enter the same record again. We can use the **.then()** function for this.

|  |
| --- |
| **$scope.frmSubmit = function(){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/putnewdoc',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **})**  **.then(function(response){**  **if(response.data != null){**  **window.location = "creatednewdoc.html";**  **};**  **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

1. For creatednewdoc.html, just copy one of the AngularJS files like teamweights.html and remove the repeating code. Replace that code with something like below.

|  |
| --- |
| **</nav>**  **<div id="container">**  **<div id="main" ng-controller="Weights">**  **<h3>New document created</h3>**  **<p>Would you like to create another record?</p>**  **<p><button ng-click="gotoEdit()">Yes</button></p>**  **</div>**  **<div id="aside">**  **<div id="section">** |

So basically, remove everything between the **maincontent** div and replace those lines with the ones highlighted

1. In the scripts.js file, create a new function within the **Weights** scope called **gotoEdit()** and use that function to send the user back to enterweight.html

|  |
| --- |
| **$scope.empWeight = response.data[0].empWeight;**  **});**  **};**  **//**  **$scope.gotoEdit = function(){**  **window.location = "enterweight.html";**  **}**  **//**  **});**  **//** |

Test the application

part 6 – Deleting a Record

Deleting a document should be an admin function and should be in a separate folder. We also already have HTML files that do some of the work of sending off a name to be deleted, just like we can find a name.

1. Create a new folder called **admin** inside of HTML and copy the myweights.html file into that folder and rename it to deletedoc.html
2. Since deletedoc.html is within a folder one level deep, we may need to adjust the paths to reflect this. So wherever we are referring to files on the local file system, we need to put **../** in front of what we have already:

|  |
| --- |
| **<div class="wrapper">**  **<header>**  **<img id="logo" src="../images/chart.gif" />**  **<h1><a href="../index.html">Skillsoft Weight Tracker</a></h1>**  **<meta charset="utf-8"/>**  **<link rel="stylesheet" type="text/css" href="../styles/styles.css" />**  **</header>**  **.**  **.**  **.**  **</div>**  **</div><!--closes the container div-->**  **<script src="../scripts/scripts.js"></script>**  **</body>** |

Also the file is located in the admin folder so: **http://localhost:8000/admin/deletedoc.html**

1. Change the page title, button and message to reflect that this is a delete. We would also create a new method to handle interaction with the API method

|  |
| --- |
| **<div id="main" ng-controller="Weights">**  **<h2>Deleting A Record</h2>**  **<form name="frmCollectWeights" ng-submit="frmDeleteDoc()">**  **<div>**  **<label for="empName">Name</label>**  **<input id="empName" type="text" ng-model="empName"/>**  **</div>**  **<div>**  **<button>Delete Record</button>**  **</div>**  **</form>**  **<p id="deleteMessage">{{empName}}, was deleted.</p>**  **</div>**  **<div id="aside">** |

1. Turn attention now to the controller.js file and we already have a delete function that uses the **URL** params object, we can copy this function and configure it to use the **body** instead.

|  |
| --- |
| **exports.deletebyformname=function(req, res){**  **let empToDelete = req.body.empName;**  **Weight.deleteOne({empName:empToDelete}, function(err, result) {**  **if (err)**  **res.send(err);**  **res.end(`Deleted ${empToDelete}`);**  **});**  **};** |

Everything else should work as is

1. Create a corresponding route in routes.js

|  |
| --- |
| **app.route('/getbyname/:byname').get(controller.getbyname);**  **app.route('/deletebyname/:byname').delete(controller.deletebyname);**  **app.route('/deletebyformname').delete(controller.deletebyformname);**  **app.route('/putdoc').post(controller.putnewdoc);**  **app.route('/updatedoc').put(controller.updatedoc);** |

1. In scripts.js we need to create a function called **frmDeleteDoc()** as we eluded to in #3 above. Copy the **frmSubmit()** function and change it to interact with the corresponding **API**.

|  |
| --- |
| **$scope.frmDeleteDoc = function(){**  **$http({**  **method : 'DELETE',**  **url : 'http://localhost:8000/deletebyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **})**  **.then(function(response){**    **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

1. According to the API from #4 above, we are expected to get back some kind of message. We can either pass this message to our HTML or use Angular’s features to hide a prepopulated message. This has to be done in 2 places, in the deletedoc.html, add the **ng-hide** directive to the **deleteMessage** **div** tag

|  |
| --- |
| **<div>**  **<button>Delete Document</button>**  **</div>**  **<div ng-hide="deleteMessage">**  **<p id="deleteMessage">{{empName}}, was deleted.</p>**  **</div>**  **</form>** |

Then in scripts.js, complete the **then()** function to show the message on getting the proper response from the **API**

|  |
| --- |
| **data : 'empName='+$scope.empName**  **}).then(function(response){**  **if(response.data != null){**  **$scope.deleteMessage = false;**  **};**  **});** |

Also still in scripts.js, you would need to hide the **message div** prior to executing the call to the **API**. This happens at the very top of the file, but inside of the controller’s scope.

|  |
| --- |
| **let app = angular.module('SkillsApp', [] );**  **app.controller('Weights', function($scope, $http) {**  **$scope.deleteMessage = true;**  **$http.get(url).then(function(response){**  **$scope.allWeights = response.data;**  **});** |

1. Prior to testing the HTML form, it would be a good idea to test the API using the **REST** client.

Here is the complete **frmDeleteDoc()** function

|  |
| --- |
| **$scope.frmDeleteDoc = function(){**  **$http({**  **method : 'DELETE',**  **url : 'http://localhost:8000/deletebyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **})**  **.then(function(response){**  **if(response.data != null){**  **$scope.deleteMessage = false;**  **};**  **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

Here is the controller.js **API**

|  |
| --- |
| **exports.deletebyformname=function(req, res){**  **let empToDelete = req.body.employeeName;**  **Weight.deleteOne({empName:empToDelete}, function(err, result) {**  **if (err)**  **res.send(err);**  **res.end(`Deleted ${empToDelete}`);**  **});**  **};** |

part 7 – Editing a Record

Edits are not as easy as the other **CRUD** operations. We first have to find the record (myweights.html) then display the document’s data inside of two **input** tags (enterweight.html), then finally edit the record and send it back to the server.

Based on the pattern so far, we would need to use an existing **HTML** file and change the **HTML** code, create a new function in scripts.js and make sure we have a corresponding **API** to hook into.

1. Since we will be using both **input** tags, we can copy enterweight.html, paste and rename to editweight.html. This should go into the **admin** folder also.
2. Fix the paths as we did for deleting a record.
3. We should hide the weight input box initially and then show it if a record was found. As the **input** and **label** is already wrapped in a div tag (**formSeparator**) we can apply **ng-hide** to the entire **div** and hide both elements

|  |
| --- |
| **<input id="empName" type="text" ng-model="empName"/>**  **</div>**  **<div class="formSeparator" ng-hide="hideWeight">**  **<label for="empWeight">Your Weight Today</label>**  **<input id="empWeight" type="text" ng-model="empWeight"/>**  **</div>**  **<div>** |

1. Change the text on all elements appropriately, including the calling function to frmEditDocument()

|  |
| --- |
| **<div id="maincontent" ng-controller="Weights">**  **<h2>Editing weights</h2>**  **<form name="frmCollectWeights" ng-submit="frmEditDocument()">**  **<div>**  **<label for="empName">Name to Edit</label>**  **<input id="empName" type="text" ng-model="empName"/>**  **</div>**  **<div class="formSeparator" ng-hide="hideMessage">**  **<label for="empWeight">** **Current Weight</</label>**  **<input id="empWeight" type="text" ng-model="empWeight"/>**  **</div>**  **<div>**  **<button>{{** **editingButton}}</button>**  **</div>**  **</form>** |

Notice the button text will change based on *find* or *edit*.

Here is all the code so far in the form area:

|  |
| --- |
| **<form name="frmCollectWeights" ng-submit="frmEditDocument()">**  **<div>**  **<label for="empName">Name to Edit</label>**  **<input id="empName" type="text" ng-model="empName" />**  **</div>**  **<div class="formSeparator" ng-hide="hideWeight">**  **<label for="empWeight">Current Weight</label>**  **<input id="empWeight" type="text" ng-model="empWeight" />**  **</div>**  **<div>**  **<button>{{editingButton}}</button>**  **</div>**  **</form>** |

1. In scripts.js we need to declare the **hideWeight** and **editingButton** variables and assign the appropriate values. Do this at the top of the **controller** function.

|  |
| --- |
| **let app = angular.module('SkillsApp', [] );**  **//**  **app.controller('Weights', function($scope, $http) {**  **$scope.hideWeight=true;**  **$scope.editingButton="Find Document";**  **//**  **$http.get(url).then(function(response){** |

1. In scripts.js we need to create a new function, lets call it **frmEditDocument()**, so this is the function that our **HTML** form will call also. **Make sure that this is changed in editdocument.html.** I am using an arrow function here.

|  |
| --- |
| **$scope.frmEditDocument = () => {**  **$scope.frmFindSingle();**  **}** |

1. If **frmFindSingle()** does find the record, we will know because **$scope.empWeight** will contain a value greater than zero.

We can test this by displaying the weight found for the Employee in the html, so temporarily enter this code, just to test the value being returned. So just copy the div with the editingButton and replace editingButton with empWeight. Now test by searching for someone using the new editweight.html file.

|  |
| --- |
| **</div>**  **<div>**  **<button>{{ editingButton }}</button>**  **</div>**  **<div>**  **<button>{{ empWeight }}</button>**  **</div>**  **</form>** |

Once you see a result, remove the div tags.

1. Although this works here, from a different function we may not be able to access that variable. Enter this code in scripts.js within the frmEditDocument to see if we can get at the variable’s value

|  |
| --- |
| **$scope.frmEditDocument = () => {**  **$scope.frmFindSingle();**  **console.log($scope.empWeight);**  **}** |

It should show up as **undefined**.

1. In order to have access to that value, we would need to call back into the **frmEditDocument()** function. It means we need to first change the function to accept a function parameter and postback using a chained **then()** method:

|  |
| --- |
| **$scope.frmFindSingle = function(callback){**  **$http({**  **method : 'POST',**  **url : 'http://localhost:8000/getbyformname',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName**  **})**  **.then(function(response){**  **$scope.empWeight = response.data[0].empWeight;**  **})**  **.then(callback)**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

1. When we call **frmFindSingle()** we need to wire up a call back function. Then we will have access to the value in **empWeight.** Lets use thatto first reveal the **empWeight** input box and insert into that box the value returned from **frmFindSingle().** Remember to remove the console.log() call.

|  |
| --- |
| **$scope.frmEditDocument = () => {**  **$scope.frmFindSingle(()=>{**  **if($scope.empWeight != ""){**  **$scope.hideWeight = false;**  **}**  **});**  **}** |

Because **empWeight** input box is wired up, it gets the weight value automatically

1. Once this works, we need to change the **button** to display something like “Update Document”, because we have already found the document and its corresponding weight. Do this in the same function call

|  |
| --- |
| **$scope.frmEditDocument = function (){**  **$scope.frmFindSingle(function(){**  **if($scope.empWeight != ""){**  **$scope.hideWeight = false;**  **$scope.editingButton = "Update Document";**  **}**  **});**  **}** |

As the form is loaded, “Find Document” will be displayed on the button, but this will have to change to “Update Document” once the record is found.

1. Initially the button will have the text “Find Document”, but if the document is found, then that button’s text changes to “Update Document”. We need to wrap up our code into an **if** statement to test for this text.

|  |
| --- |
| **$scope.frmEditDocument = function (){**  **if($scope.editingButton !== "Update Document"){**  **$scope.frmFindSingle(function(){**  **if($scope.empWeight != ""){**  **$scope.hideWeight = false;**  **$scope.editingButton = "Update Document";**  **}**  **});**  **} else {**  **$scope.frmDoEdit();**  **}**  **}**  **//** |

We now have to write the **frmDoEdit()** function. We can’t use the original **frmSubmit()** because that works with brand new records and edits are usually associated with the **HTTP PUT** verb

1. The **frmDoEDit()** function

|  |
| --- |
| **$scope.frmDoEdit = function(){**  **$http({**  **method : 'PUT',**  **url : 'http://localhost:8000/updatedoc',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **})**  **.then()**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

Notice the method value

1. If you look at the controller.js file, and the **updatedoc()** method, you will see that it is returning a message upon a successful edit, we can use that message in our front end. Create a new variable and a new **div** underneath the **button div**, we will reveal the message from the database there:

|  |
| --- |
| **<div>**  **<button>{{editingButton}}</button>**  **</div>**  **<div id="postMessage">**  **{{afterEdit}}**  **</div>**  **</form>** |

1. Back to the scripts.js file, chain the **then()** method to the frmDoEdit() method. Use the variable **afterEdit** so that the message appears in the HTML.

|  |
| --- |
| **$scope.frmDoEdit = function(){**  **$http({**  **method : 'PUT',**  **url : 'http://localhost:8000/updatedoc',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **})**  **.then((response) => {**  **if(response.data != "")**  **$scope.afterEdit = response.data;**  **else**  **$scope.afterEdit = "";**  **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

Here is the entire frmDoEdit() function:

|  |
| --- |
| **$scope.frmDoEdit = function(){**  **$http({**  **method : 'PUT',**  **url : 'http://localhost:8000/updatedoc',**  **headers : {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'},**  **data : 'empName='+$scope.empName + '&empWeight='+$scope.empWeight**  **})**  **.then((response) => {**  **if(response.data != "")**  **$scope.afterEdit = response.data;**  **else**  **$scope.afterEdit = "";**  **})**  **.catch(function(error){**  **console.log(error);**  **});**  **};** |

1. We could improve the security of frmEditDocumen() by first checking that the value in empWeight is in fact a number, if not we should display an error:

|  |
| --- |
| **$scope.frmEditDocument = () => {**  **if($scope.editingButton !== "Update Document"){**  **$scope.frmFindSingle(()=>{**  **if( weight = parseFloat($scope.empWeight )){**  **if($scope.empWeight> 0){**  **$scope.hideWeight = false;**  **$scope.editingButton = "Update Document";**  **}else {**  **$scope.afterEdit = "An error occured";**  **}**  **}**  **});**  **} else {**  **$scope.frmDoEdit();**  **}**  **}** |

Notes.

1. It may be better to send the user to a totally different page like we did after a new document was inserted. You could reconfigure the **gotoEdit()** function to do this.
2. After the document is updated, there is some extra data showing up, for example result 1. This is coming from the API, so either adjust it there or remove this last part, but if you are sending them to a new page, then there is no point.
3. If you get undefined, check the updatedoc() function in controller.js, it may be looking for fixname and newweight.