Diving deeper into AngularJS

Objectives:

- Two Way Binding
- Dynamic content with Services

Binding

In this module, we want to go deeper on the binding that is in AngularJS. When you create a model, you can create properties that get or set a value on that model. Additionally, you can create functions that return the results of a model. Let's take a look at how that works.

First, let modify our **index.html** file and the "checknumber" HTML and add some additional markup we're going to need. Notice that we surrounded this content in another bootstrap "row". Since this gives us twelve columns, we'll break the content into four columns and eight columns. Next, add some labels and input boxes and a span to collect some name information.

```
<div class="row">
    <div class="col-sm-4">
        <input ng-model="smallnumber" type="text">
        <p>{{ theMessage }}</p>
        <button ng-click="checkNumber()">Submit</button>
    </div>
    <div class="col-sm-8">
        <label>FirstName</label><input type="text"/></br>
        <label>LastName</label><input type="text"/></br>
        <label>FullName</label><span></span>
    </div>
</div>
```

Save and load **index.html** in your browser. You should see the new fields on the page. They don't line up like we would want at this point, so let's add a style in the **app.css** file to clean up the layout.

```
label {
    padding: 0 10px;
    width: 100px;
}
```

Adding a User Object to our Scope

Next, we want to modify the AngularJS controller that we created in the last module to include a new property. Previously, the $scope properties were single value fields or a function like Date().

This time, let's modify the **app.js** file and the property is going to be a complex object that we will call "User". The User object is going to have it's own properties for "firstName", "lastName", and "fullName". Notice that each of these are functions that modify local variable values for "_first" and "_last".

```
app.controller('MainController', ['$scope', function ($scope) {
    $scope.timestamp = Date();
    $scope.smallnumber = 0;
    $scope.theMessage = "";

    ...

    var _first = "Your";
    var _last = "Name";
    $scope.User = {
        firstName: function (first) { return arguments.length ? (_first = first) :
        lastName: function (last) { return arguments.length ? (_last = last) : _las
        fullName: function () { return _first + ' ' + _last; }
    }
}]);
```

Two things that need to be pointed out here. First, notice the use of the "_" for the variables for "_first" and "_last". Programmers will often use this convention to help someone reading the code recognize that these are values that are "private" to the function. You can't access these two variables by themselves, you have to use the $scope.User object to manipulate these values.

Second, notice the function for the firstName or the lastName. We are using a special operator with the `?`. This is similar to an `if...then...else` statement. It works like this `if ? then : else`. Any non-zero value is considered true. So in the case of "firstName" the function is saying if the arguments.length (something was passed in to the function) is not zero the assign the value of "_first" to the argument value of "first" otherwise return the value of the "_first" variable.

In the case of the "fullName" function, no modifications to the private variables are made. The values are simply concatenated together and returned as the value of "fullName".

Wire Up the Model in HTML

Let's turn back now to the HTML and connect up the model that we just added to our controller. We're going to use the "ng-model" attribute to connect each input, but notice that it is we use "User.firstName". That dot in the is telling the code to use the firstName value of the User object. We will also add the "ng-

model-options" and set to `"{ getterSetter: true }"` which tells AngularJS that we are using functions for getting the value and setting the value. This is because we could use javascript somewhere else on the page to update the User object.

```
<div class="row">
    <div class="col-sm-4">
        <input ng-model="smallnumber" type="text">
        <p>{{ theMessage }}</p>
        <button ng-click="checkNumber()">Submit</button>
    </div>
    <div class="col-sm-8">
        <label>FirstName</label><input ng-model="User.firstName" type="text" ng-mod
        <label>LastName</label><input ng-model="User.lastName" type="text" ng-model
        <label>FullName</label><span ng-bind="User.fullName()"></span>
    </div>
</div>
```

Save and load the **index.html** page in your browser. Notice that as you enter values in the FirstName and LastName textbox, the span below for FullName is automatically updated.

This is just a simple of example of the binding that you can do, but imagine if you had a shopping cart where you could calculate the new price or the total as soon as the values were entered into the text box. Again, we are relying on the AngularJS framework to do the user interface manipulations for us so we don't have to write all the code to handle all of the updates.

Dynamic Page Content

Finally, let's turn our attention to dynamic page content. Most websites that are popular have content that is refreshed on a regular basis. The content that we have created so far is static to your page. If you wanted to update the content, you would have to edit and republish the page.

In this next section, we are going to look at how to make the content on the web page dynamic and pull from a service. You will see that the main content of your page is merely a template, and all of the content will be dynamic from a server.

We are going to break this down into steps so that you can see what is happening as we move forward.

First, we need to do is add an additional module to our main **angularApp** module. The module we are going to add is **ngSanitize**. This provides functionality to use HTML that is sent from another website and manipulate it without messing up the markup that is sent from that server.

First, in our **index.html** page, add a reference in the <head> of the page.

```
<head>
    <title>My First Web Page</title>
    <link href="app.css" rel="stylesheet"/>
    <script src="<http://code.angularjs.org/1.0.3/angular-sanitize.js>"></script>
    ...
</head>
```

Next, open the **app.js** file to update our app definition. We need to tell our **angularApp** to load the **ngSanitize** module. We do this by adding "ngSanitize" between the existing brackets in our module decleration

```
var app = angular.module("angularApp", ['ngSanitize']);
```

Move Existing Content into Controller

Next, let's move the main content to the scope of your controller. Open the **MainController.js** file and paste in this block of code.

```
  $scope.Articles = [
    {"Title":"Amazing Museums","Body":"<p>While most museums are a collection of ol
    {"Title":"SplashAnyone Waterpark","Body":"<p>While most museums are a collectio
    {"Title":"Best Zoo in Town","Body":"<p>Our waterpark is the biggest and best pa
    ];
```

Your file should look something like this (some of the code is removed for brevity):

```
app.controller('MainController', ['$scope', function ($scope) {
    $scope.timestamp = Date();
    $scope.smallnumber = 0;
    $scope.theMessage = "";

    $scope.checkNumber = function () {
        ...
    };

  $scope.Articles = [
    {"Title":"Amazing Museums","Body":"<p>While most ...</p>","Image":"http://lorem
    {"Title":"SplashAnyone Waterpark","Body":"<p>While most...</ p > ","Image":"htt
    {"Title":"Best Zoo in Town","Body":"<p>Our waterpark...</p>","Image":"http://lo
    ];

    var _first = "Your";
```

```
    var _last = "Name";
    $scope.User = {
        ...
    }
}]);
```

Now, lets remove all three of the <article> objects from your webpage. Replace them with the following content:

```
<div class="row">
    <article ng-repeat="article in Articles">
        <h2>{{ article.Title }}</h2>
        <div class="col-sm-3">
            <img ng-src="{{ article.Image }}" />
        </div>
        <div class="col-sm-9">
            <div ng-bind-html="article.Body"></div>
        </div>
    </article>
</div>
```

This HTML will use the AngularJS function "ng-repeat" to walk through each of the objects in the `$scope.Articles` variable. It will call each [object] an "article" and then we will use that in the header <h2> tags, image and body. Notice, we have three unique things we are doing with AngularJS items here.

1. When we are placing text between tags, we use the `{{ value }}` directly.
2. When we are setting the value of the "src" attribute in the image tag, we have to use the "ng-src" function to do this. This is because the browser will try to retreive the image for the `{{ value }}` before AngularJS has had a chance to replace it. After AngularJS has resolved the expression, the image will be loaded.
3. We have HTML in the content of our [object]. Because of this, things get a little messed up. This is where the ngSanitize helps us out. If we use the "ng-bind-html" function, AngularJS knows not to replace all of the HTML goodies in our data and simply render it all as HTML into the

Save and reload the **index.html** page, and you should have a page full of content that came from the **MainController.js**.

Notice how simple your HTML page got. Imagine now that you wanted to update the layout of your page. Before if you wanted to change all of the titles of your articles from an <h2> tag to an <h1> tag you would have had to change it on all three articles. What if you wanted to move the image to the right side? What if you had 15 or 20 articles. This could become very time consuming. MVC frameworks let you focus on the view and keep it separate from your data.

Moving Your Content Out of Your Code

Having all of that content in your code isn't a good idea. We've put it in a database, and have exposed it via an API for you to pull that content. Earlier we learned how to call an API with jQuery and Javascript. This time, we are going to use AngularJS to do the API calls for us.

In order to do this, we need to modify our controller definition to pass in an **$http** core service like we did with **$scope**. Modify your **MainController.js** to change your app.controller initialization.

```
app.controller('MainController', ['$scope', '$http', function ($scope, $http) {
```

Now that you have the **$http** service, we can take advantage of it in our controller to load data. Since the API call may take a minute, we don't want the "ng-repeat" in our page to crash because the "Articles" is undefined, so let's set our $scope.Articles to an empty object. Next, we'll call the POST method on the ChooseToCode API and get a few articles. When the method is complete, the result will be passed to the "then" function and we will store the data in the "$scope.Articles" value. When this change happens, the "ng-repeat" will rerun and bind all of the articles to the page. Be sure to modify **"YourClassName"** before you run the results.

```
    $scope.Articles = [];
    $http({method: 'POST', url: 'http://choosetocodeapi.azurewebsites.net/api/artic
        ClassName: 'YourClassName',  //modify this for your class
        Count: 3
    }}).then(function(result){
        $scope.Articles = result.data;
    });
```

Extra Challenge - Show What You Know

In this module we showed you how to get more out of AngularJS and how to bind data from an API. The content is coming from another service. Now it's time to show what you know. Modify the code for the $http POST so that it uses the number entered in the smallnumber box. Make sure that you are only using a valid number when you do this.

Good luck, and have fun!