Introduction to AngularJS

Objectives:

- Introduction to AngularJS
- Update HTML on the Fly with Javascript

What is AngularJS

In the next couple of modules, we are going to look at a JavaScript framework for dynamic web pages. Your HTML will move away from having so much content already filled in and become more like a template for displaying content we will pull from a service. Using a framework like AngularJS removes a lot of the code that you would otherwise have to write and simplifies your pages dramatically.

The AngularJS framework implements a common pattern for code design called "MVC". MVC stands for Model/View/Controller. Models will be created to hold data for display. Views are the way the data is presented. Controllers are what manage the interaction of the views and data. While we won't cover the design pattern much here, much of the code that we will create will follow the MVC naming approach.

Let's Get Started

As we have seen in earlier modules, we will add a reference to a content delivery version of the AngularJS libraries. Alternatively, these could be downloaded to your computer and stored in your script folder.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.5/angular.min.js
```

Next, we need to modify our existing app.js file to create a new AngularJS module. What is an AngularJS Module? You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.

We're to call this **angularApp**, but you could call it whatever you want. Let's put the following line of code at the top of your **app.js** file.

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

Now that we have an AngularJS module, we need to put a directive in out HTML that tells where the module can be used. This is also called the *application scope*.

```
<body onload="onLoad();" ng-app="angularApp">
```

Now, to keep the current time code functional, you will need to add some code to make that happen. Since this is the main page, let's add a new JavaScript file for the AngularJS code for this page. We'll use the standard naming and call it **MainController.js**. Let's also add the following code.

```
app.controller('MainController', ['$scope', function($scope) {
  $scope.timestamp = Date();
}]);
```

Back on the **index.html** page we need to make a few more modifications to the content. First, in the <head> tag, we need to add a reference to our **MainController.js** file

```
<head>
    <title>My First Web Page</title>
    ...
    <script src="scripts/app.js" type="application/javascript"></script>
    <script src="scripts/MainController.js" type="application/javascript"></script
</head>
```

Next, add a <div> around all of the content of the page and specify that it will be controlled by the MainController we just created. The "ng-controller" attribute will define the *controller scope* for our page. When the timestamp is set on the `$scope` it is on the scope of the controller.

```
<body ng-app="angularApp">
    <div ng-controller="MainController">
        <header class="container-fluid">
        ...
        </footer>
    </div>
</body>
```

Finally, change the footer to remove the id from the <p> tag and put in the AngularJS expression for the "timestamp" property of the controller scope:

```
<footer>
    <h1>Copyright 2015</h1>
    <p>{{ timestamp }}</p>
</footer>
```

Save and load the **index.html** page in your browser. Notice how our markup no longer has to define a "id" so that our code can replace the "timestamp". By simply using the scope property in the markup, it is more clear what the intent of this section of markup will do. This simplicity is part of what make frameworks like AngularJS so popular.

Before we move on, let's clean up the **app.js** code since the code to insert the timestamp isn't called any more. Good developers should always remove code that is no longer called to keep the needed code clutter free. This is our code right now...

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

function onLoad() {
    getAPIBadge();
}
```

A Word About Structure

All of the code files you create can be created right next to your HTML files and your image files. However, this can get messy very quickly, and it can be very difficult to find files when you need them.

Typical web projects structure files that are similar in their own folder like the following:

```
index.html
content (folder)
    app.css
scripts (folder)
    app.js
    MainController.js
images (folder)
    picture1.jpg
```

Let's do that now. Make an additional folder for "scripts" and move the **app.js** and **MainController.js** files to that folder. Additionally, make a folder called "content" and move the **app.css** file there. Make sure you update the content in the **index.html** file to reflect your new file locations.

```
<head>
    <title>My First Web Page</title>
    <link href="content/app.css" rel="stylesheet" />
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.c
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js
    <script src="https://code.jquery.com/jquery-1.10.2.js" type="application/javasc
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.5/angular.mi
    <script src="scripts/app.js" type="application/javascript"></script>
    <script src="scripts/MainController.js" type="application/javascript"></script>
</head>
```

Up until now, it wasn't a big deal because there was only one javascript file, one css file and one html page. Consider how many files we would have if we had downloaded the bootstrap, jquery and AngularJS files to our local machine. It gets messy very quickly.

Taking AngularJS farther

One of the other keys to a framework like AngularJS being so popular is that it takes little to no code to do what is called "data binding". Earlier, we put a textbox on the page that when the user clicked "submit" a message would warn them if the number wasn't valid. We had to have code that would find the control, get the number and then make a check, then find the element where we wanted to put the message and replace the text. With binding that goes away and we can assign the value as the data is entered, make the check and assign the message all using the controller *$scope*

Let's do that now.

First, let's add a few properties to our scope in the MainController. We'll add a **smallnumber** and **theMessage** values that we were tracking in our JavaScript code. While we're at it, let's add another property to the scope and assign to a function. We'll call it **checkNumber**.

Here's what your MainController should look like now:

```
app.controller('MainController', ['$scope', function($scope) {
    $scope.timestamp = Date();
    $scope.smallnumber = 0;
    $scope.theMessage = "";
    $scope.checkNumber = function(){
        if (isNaN($scope.smallnumber) || $scope.smallnumber < 1 || $scope.smallnumb
            $scope.theMessage = "Number was expected to be between 1 and 10";
        } else {
            $scope.theMessage = "Number is Good";
        }
    }
}]);
```

Notice that our **checkNumber** function is quite a bit simpler now. Go ahead and remove the **CheckNumber()** function from your **app.js** file now.

Back in the **index.html** file, we need to update the content to take advantage of the AngularJS MainController scope.

```
<input ng-model="smallnumber" type="text">
<p>{{ theMessage }} </p>
<button ng-click="checkNumber()">Submit</button>
```

Save your work, and reload the **index.html** page in a browser. Notice that when you enter a value in the textbox, the function on the controller scope executes and puts the correct message in the page.

In the next module, we will see how the scope controller can pull data from a web site and very easily layout the content on the page.