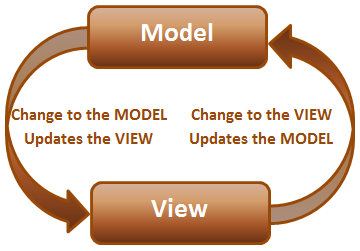
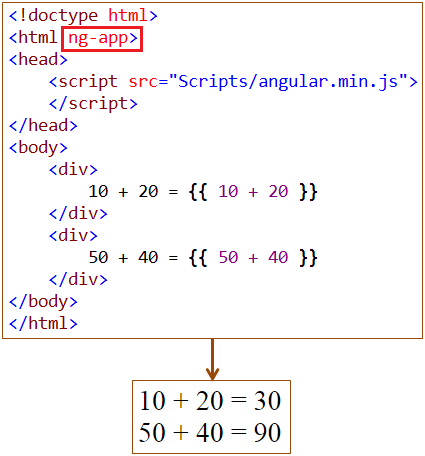
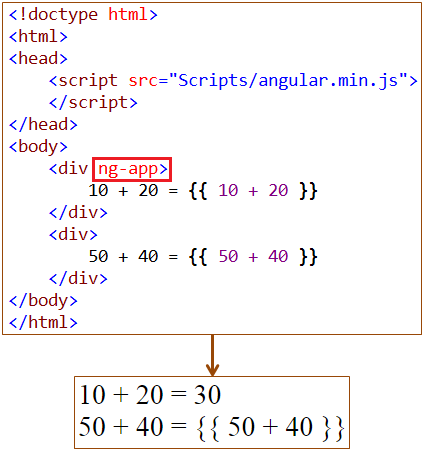
Angular JS

# Heading 1

### What is AngularJS

**Suggested Tutorials**  
[SQL Server Tutorial](https://www.youtube.com/playlist?list=PL08903FB7ACA1C2FB)  
[ASP.NET Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXQS_p1i-HLIftB9Y7Vnxlo)  
[jQuery Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhVDV697uvHpavA3K_eWGQap)

**What is AngularJS**  
AngularJS is a JavaScript framework that helps build applications that run in a web browser.    
  
**Who developed AngularJS**  
Google is the company that developed AngularJS. AngularJS is an open source project, which means it can be be freely used, changed, and shared by anyone.  
  
AngularJS is an excellent framework for building both Single Page Applications (SPA) and Line of Business Applications. Many companies are using Angular today, and there are many public facing web sites that are built with angular.  
  
There is a website, [https://www.madewithangular.com](https://www.madewithangular.com/), that has the list of web sites that are built using AngularJS. Within this list you can find many popular websites.  
  
**What are the benefits of using AngularJS**  
**1. Dependency Injection :** Dependency Injection is something AngularJS does quite well. If you are new to Dependency Injection, don't worry, we will discuss it in detail with examples in a later video.  
  
**2. Two Way Data-Binding :**One of the most useful feature in AngularJS is the Two Way Data-Binding. The Two Way Data-Binding, keeps the model and the view in sync at all times, that is a change in the model updates the view and a change in the view updates the model.   
   
  
**3. Testing :**Testing is an area where Angular really shines. Angular is designed with testing in mind right from the start. Angular makes it very easy to test any of it's components through both unit testing and end to end testing. So there's really no excuse for not testing any of your angular application code.  
  
**4. Model View Controller :**With angular it is very easy to develop applications in a clean MVC way. All you have to do is split your application code into MVC components. The rest, that is managing those components and connecting them together is done by angular.  
  
**5. Many more benefits**like controlling the behaviour of DOM elements using **directives**and the flexibility that **angular filters**provide.  
  
We will discuss directives, filters, Modules, Routes etc with examples in our upcoming videos in this series.  
  
To build angular applications you only need one script file and that is angular.js.  
  
To get the script file visit [https://angularjs.org](https://angularjs.org/). From here   
1. You can download the angular script file   
2. CDN link - We discussed the benefits of using CDN in [Part 3](https://www.youtube.com/watch?v=hcpskGpuKaE&list=PL6n9fhu94yhVDV697uvHpavA3K_eWGQap&index=3) of [jQuery tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhVDV697uvHpavA3K_eWGQap).  
3. Various resources to learn angular - Here you will find videos, Free courses, Tutorials and Case Studies. You will also find API reference which is extremeley useful.  
  
**To get started with angular**  
1. Add a reference to the angular script  
2. Include ng-app attribute   
  
**What is ng-app**  
In angular, ng-app is called a directive. There are many directives in angular. You can find the complete list of directives on https://angularjs.org. The ng prefix in the directive stands for angular. The ng-app directive is a starting point of AngularJS Application. Angular framework will first check for ng-app directive in an HTML page after the entire page is loaded. If ng-app directive is found, angular bootstraps itself and starts to manage the section of the page that has the ng-app directive.   
  
So the obvious next question is, **where to place the ng-app directive on the page**  
It should be placed at the root of the HTML document, that is at the <html> tag level or at the <body> tag level, so that angular can control the entire page.   
  
However, there is nothing stopping you from placing it on any other HTML element with in the page. When you do this only that element and it's children are managed by angular.   
  
Double curly braces are called binding expressions in angular.  
  
**Example :**In the example below, the **ng-app**directive is placed at the <html> tag level. So the binding expressions in both the div elements are evaluated and displayed as expected.   
   
  
**Example :**In the example below, the **ng-app** directive is placed on one of the <div> element. So the binding expressions in the <div> element that has the ng-app directive is evaluated but not the binding expression in the other <div> element.  
   
  
**All the following are valid expressions in angular**  
**{{** 1 == 1 **}}** - Evaluates to true  
**{{** { name: 'David', age : '30' }.name **}}** - Returns the name property value  
**{{** ['Mark', 'David', 'Sara'][2] **}}** - Returns the 2nd element from the array

### Angular modules and controllers

**Suggested Videos**  
[Part 1 - What is AngularJS](http://csharp-video-tutorials.blogspot.com/2015/10/what-is-angularjs.html)

**What is a module in AngularJS**  
A module is a container for different parts of your application i.e controllers, services, directives, filters, etc. In this video we will also discuss controllers. We will discuss services, filters and directives in a later video.   
  
**Why is a module required**  
You can think of a module as a Main() method in other types of applications. For example, a Dot Net console application has a Main() method which is the entry point into the application and it wires together the different parts of the application.  
  
Modules are the angular application's equivalent of the Main() method. Modules declaratively specify how the angular application should be bootstrapped.   
  
There are several benefits of the modular approach. It may be difficult to comprehend all those benefits right now, so we will defer the discussion of the benefits to a later video.  
  
**How to create a module**  
Creating a module in angular is staright forward. Use the angular object's module() method to create a module. The angular object is provided by angular script. The following example, creates a module.   
  
var myApp = angular.module("myModule", [])  
  
The first parameter specifies the name of the module.   
The second parameter specifies the dependencies for this module  
  
A module can depend on other modules. We will discuss an example of module dependencies in a later video. Right now, the module that we are creating is not dependent on any other external modules, so I am passing an empty array as the value for the second parameter.  
  
**What is a controller in angular**  
In angular a controller is a JavaScript function. The job of the controller is to build a model for the view to display. The model is the data. In a real world application, the controller may call into a service to retrieve data from the database.  
  
**How to create a controller in angular**  
Simple, create a JavaScript function and assign it to a variable  
  
var myController = function ($scope) {  
    $scope.message = "AngularJS Tutorial";  
}  
  
**What is $scope**  
$scope is a parameter that is passed to the controller function by angular framework. We attach the model to the $scope object, which will then be available in the view. With in the view, we can retrieve the data from the scope object and display.  
  
**How to register the controller with the module**  
Use module object's controller function to register the controller with the module  
  
myApp.controller("myController", myController);  
  
**Here is the complete code**

//Create the module

var myApp = angular.module("myModule", []);

//Create the controller

var myController = function ($scope) {

    $scope.message = "AngularJS Tutorial";

}

// Register the controller with the module

myApp.controller("myController", myController);

**The above code can also be written as shown below**

//Create the module

var myApp = angular.module("myModule", []);

// Creating the controller and registering with the module all done in one line.

myApp.controller("myController", function ($scope) {

    $scope.message = "AngularJS Tutorial";

});

**How to use the module that we created to bootstrap the angular application**  
Associate the module name with ng-app directive  
ng-app="myModule"  
  
Similarly associate the controller using ng-controller directive  
ng-controller="myController"  
  
**Here is the complete HTML**

<!doctype html>

<html ng-app="myModule">

<head>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body>

    <div ng-controller="myController">

**{{** message **}}**

    </div>

</body>

</html>

**Here is the complete JavaScript**

/// <reference path="angular.min.js" />

//Create module

var myApp = angular.module("myModule", []);

// Register controller with the module

myApp.controller("myController", function ($scope) {

    $scope.message = "AngularJS Tutorial";

});

### Controllers in AngularJS

**Suggested Videos**  
[Part 1 - What is AngularJS](http://csharp-video-tutorials.blogspot.com/2015/10/what-is-angularjs.html)   
[Part 2 - Angular modules and controllers](http://csharp-video-tutorials.blogspot.com/2015/10/angular-modules-and-controllers.html)

**We will discuss**  
1. Attaching complex objects to the scope  
2. What happens if a controller name is mis-spelled  
3. What happens if a property name in the binding expression is mis-spelled  
4. How to create module, controller and register the controller with the module, all in one line  
  
**The job of the controller is to build a model for the view**. The controller does this by attaching the model to the scope. The scope is not the model, it's the data that you attach to the scope is the model.   
  
In the following example, $scope is not the model. The message property that we have attached to the scope is the model.  
  
myApp.controller("myController", function ($scope) {  
    $scope.message = "AngularJS Tutorial";  
});  
  
The view will then use the data-binding expression to retrieve the model from the scope. This means the controller is not manipulating the DOM directly, thus keeping that clean separation between the model, view and the controller. So when you are developing controllers, make sure, you are not breaking that clean separation between the model, view and the controllers. The controller should only be used for setting up the $scope object and adding behavior it. We will discuss, when and why should we add behvior to the scope object in a later video.   
  
In the example above, **message is a simple property**. You can also attach a complex object to the scope. In the example below, we have an employee object which is a complex object with 3 properties attached to the view.   
  
myApp.controller("myController", function ($scope) {  
  
    var employee = {  
        firstName: 'Mark',  
        lastName: 'Hastings',  
        gender: 'Male'  
    };  
  
    $scope.employee = employee;  
});  
  
With in the view, we can then retrieve the employee properties and display them in the view as shown below

<div ng-controller="myController">

    <div>First Name : **{{** employee.firstName **}}**</div>

    <div>Last Name : **{{** employee.lastName **}}**</div>

    <div>Gender : **{{** employee.gender**}}**</div>

</div>

**What happens if the controller name is misspelled**  
When the controller name is misspelled, 2 things happen  
1. An error is raised. To see the error, use browser developer tools  
2. The binding expressions in the view that are in the scope of the controller will not be evaluated  
  
If you are using the minified version of the AngularJS script, the error messages may not be readable. To get readable error message  
1. In the developer tools, click on the link next to the error. This will lead you to a page, where you can see a much clean error message without all the encoding symbols.  
  
2. Another option you have is, if you are in the development environment, you may use the non-minified version of the AngularJS script, which produces readable error message.  
  
**What happens if a property name in the binding expression is misspelled**  
Expression evaluation in angular is forgiving, meaning if you misspell a property name in the binding expression, angular will not report any error. It will simply return null or undefined.  
  
**How to create module, controller and register the controller with the module, all in one line**  
Use the method chaining mechanism as shown below  
  
var myApp = angular  
    .module("myModule", [])  
    .controller("myController", function ($scope) {  
        var employee = {  
            firstName: 'Mark',  
            lastName: 'Hastings',  
            gender: 'Male'  
        };  
        $scope.employee = employee;  
    }); 

### AngularJS ng-src directive

Let us understand this with an example : We want to display the name of the country, capital and it's flag.   
  
   
  
**AngularJS Code :** The controller builds the country model. The flag property of the country object has the path of the image.

var myApp = angular

                .module("myModule", [])

                .controller("myController", function ($scope) {

                    var country = {

                        name: "United States of America",

                        capital: "Washington, D.C.",

                        flag: "/Images/usa-flag.png"

                    };

                    $scope.country = country;

                });

**HTML Code :** In the view we are binding country.flag property with the src attribute of the image element.

<!doctype html>

<html ng-app="myModule">

<head>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body>

    <div ng-controller="myController">

        <div>

            Country : {{ country.name }}

        </div>

        <div>

            Capital : {{ country.capital }}

        </div>

        <div>

            <img src="{{country.flag}}"

                 alt="{{ country.name + ' Flag' }}"

                 style="height:100px; width:200px" />

        </div>

    </div>

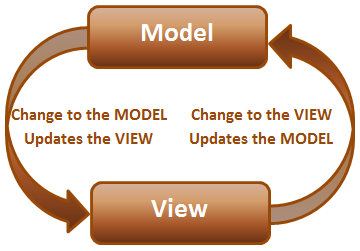
</body>

</html>

When you view the page in the browser, the country details and the flag are displayed as expected. The problem with the img src attribute is that we get a 404 error. To see the error, launch the developer tools.  
  
**Let's now understand the reason for the 404 error**  
As soon as the DOM is parsed, an attempt is made is to fetch the image from the server. At this point, AngularJS binding expression that is specified with the src attribute is not evaluated. Hence 404 (not found) error.  
  
**To fix the 404 error use the ng-src directive :** ng-src attribute ensures that a request is issued only after AngularJS has evaluated the binding expression

### Two way databinding in AngularJS

In this video we will discuss, **Two way data binding in AngularJS**. Along the way we also discuss one of the very useful directive in angular **ng-model**

   
When the model changes the view is automatically updated. This is achieved using the data binding expression in the view.   
  
**Script.js :**The code in the controller attaches message property to the scope which is the model.  

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                $scope.message = "Hello Angular"

            });

**HtmlPage1.html :**Whenever the message property value changes, the data binding expression in the view updates the view automatically. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        {{ message }}

    </div>

</body>

</html>

**How about the other way round**. How to keep the model up to date when the view changes. That's exactly is the purpose of ng-model directive.   
  
In the html below, notice the input element is decorated with **ng-model** directive. This ensures that whenever the value in the textbox is changed, angular will automatically update the message property of the $scope object. This means the ng-model directive automatically takes the form values and updates the model. The binding expression does the opposite, i.e whenever the model changes the view is automatically updated.    
  
Because of the two way data binding provided by angular, as you type in the textbox, the value is immediately displayed on the view just below the textbox. This two way binding feature provided by angular, eliminates the need to write any custom code to move data from the model to the view or from the view to the model. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="text" placeholder="Type your message here" ng-model="message" />

        <br /><br />

        {{ message }}

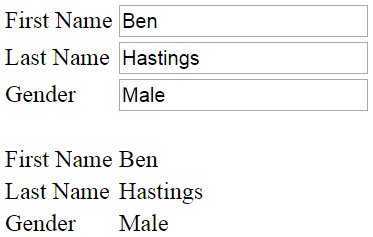
    </div>

</body>

</html>

**ng-model directive can be used with the following 3 html elements**

* input
* select
* textarea

**Two way binding example with complex object :**   
  
   
  
**Script.js code :** In the following example, the model is employee which is a complex object with properties like firstName, lastName and gender. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var employee = {

                    firstName: "Ben",

                    lastName: "Hastings",

                    gender: "Male"

                };

                $scope.employee = employee;

            });

**HtmlPage1.html :**When the view loads, the model data is display in both, the textbox and td elements on the page. As you start to type in any of the textboxes, the respective employee model object property is updated, and the change is immediately reflected in the respective td element. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <tr>

                <td>

                    First Name

                </td>

                <td>

                    <input type="text" placeholder="Firstname"

                           ng-model="employee.firstName" />

                </td>

            </tr>

            <tr>

                <td>

                    Last Name

                </td>

                <td>

                    <input type="text" placeholder="Lastname"

                           ng-model="employee.lastName" />

                </td>

            </tr>

            <tr>

                <td>

                    Gender

                </td>

                <td>

                    <input type="text" placeholder="Gender"

                           ng-model="employee.gender" />

                </td>

            </tr>

        </table>

        <br />

        <table>

            <tr>

                <td>

                    First Name

                </td>

                <td>

                    {{ employee.firstName }}

                </td>

            </tr>

            <tr>

                <td>

                    Last Name

                </td>

                <td>

                    {{ employee.lastName }}

                </td>

            </tr>

            <tr>

                <td>

                    Gender

                </td>

                <td>

                    {{ employee.gender }}

                </td>

            </tr>

        </table>

    </div>

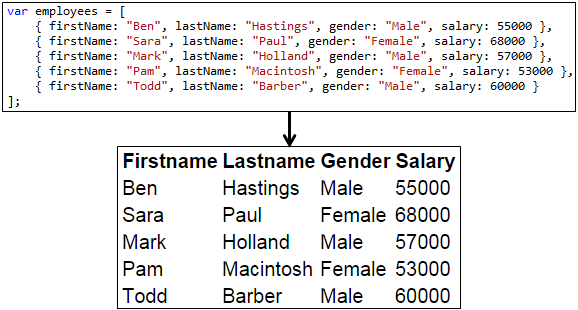
</body>

</html>

### AngularJS ng-repeat directive

**ng-repeat** is similar to foreach loop in C#.    
  
Let us understand this with an example. Here is what we want to do.  
1. For each employee we have in the employees array we want a table row. With in each cell of the table row we to display employee

* Firstname
* Lastname
* Gender
* Salary

  
  
This can be achieved very easily using **ng-repeat directive**   
  
**Script.js :** The controll function builds the model for the view. The model employees has the list of all employees. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var employees = [

                    { firstName: "Ben", lastName: "Hastings", gender: "Male", salary: 55000 },

                    { firstName: "Sara", lastName: "Paul", gender: "Female", salary: 68000 },

                    { firstName: "Mark", lastName: "Holland", gender: "Male", salary: 57000 },

                    { firstName: "Pam", lastName: "Macintosh", gender: "Female", salary: 53000 },

                    { firstName: "Todd", lastName: "Barber", gender: "Male", salary: 60000 }

                ];

                $scope.employees = employees;

            });

**HtmlPage1.html :** In the view, we are using ng-repeat directive which loops thru each employee in employees array. For each employee, we a get a table row, and in each table cell of the table row, the respective employee details (Firstname, Lastname, Gender, Salary) are retrieved using the angular binding expression. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Firstname</th>

                    <th>Lastname</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.firstName }} </td>

                    <td> {{ employee.lastName }} </td>

                    <td> {{ employee.gender }} </td>

                    <td> {{ employee.salary }} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Nested ng-repeat example :**The model contains an array of countries, and each country has an array of cities. The view must display cities nested under their respective country.

   
  
**Script.js :** The model is an array of countries. Each country contains an array of cities.

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var countries = [

                    {

                        name: "UK",

                        cities: [

                            { name: "London" },

                            { name: "Birmingham" },

                            { name: "Manchester" }

                        ]

                    },

                    {

                        name: "USA",

                        cities: [

                            { name: "Los Angeles" },

                            { name: "Chicago" },

                            { name: "Houston" }

                        ]

                    },

                    {

                        name: "India",

                        cities: [

                            { name: "Hyderabad" },

                            { name: "Chennai" },

                            { name: "Mumbai" }

                        ]

                    }

                ];

                $scope.countries = countries;

            });

**HtmlPage1.html :** Notice that we are using two ng-repeat directives in the view, one nested inside the other. The outer ng-repeat directive loops thru each country in the model. The inner ng-repeat directive loops thru each city of a given country.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <ul ng-repeat="country in countries">

            <li>

                {{country.name}}

                <ul>

                    <li ng-repeat="city in country.cities">

                        {{city.name}}

                    </li>

                </ul>

            </li>

        </ul>

    </div>

</body>

</html>

**Finding the index of an item in the collection :**

* To find the index of an item in the collection use $index property
* To get the index of the parent element
  + Use $parent.$index or
  + Use ng-init="parentIndex = $index"

The following example, shows how to retrive the index of the elements from a nested collection

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <ul ng-repeat="country in countries" ng-init="parentIndex = $index">

            <li>

                {{country.name}} - Index = {{ $index }}

                <ul>

                    <li ng-repeat="city in country.cities">

                        {{city.name}} - Parent Index = {{ parentIndex }}, Index = {{ $index }}

                    </li>

                </ul>

            </li>

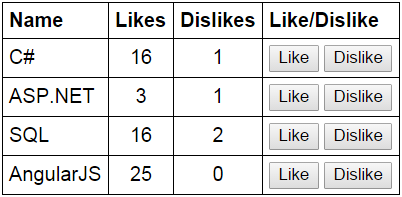
        </ul>

    </div>

</body>

</html>

### Handling events in AngularJS

Let us understand with an example. Here is what we want to do.  
  


**1.** Display the list of technologies in a table  
**2.** Provide the ability to like and dislike a technology  
**3.** Increment the likes and dislikes when the respective buttons are clicked

**Script.js :**In the controller function we have 2 methods to increment likes and dislikes. Both the functions have the technology object that we want to like or dislike as a parameter. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var technologies = [

                    { name: "C#", likes: 0, dislikes: 0 },

                    { name: "ASP.NET", likes: 0, dislikes: 0 },

                    { name: "SQL", likes: 0, dislikes: 0 },

                    { name: "AngularJS", likes: 0, dislikes: 0 }

                ];

                $scope.technologies = technologies;

                $scope.incrementLikes = function (technology) {

                    technology.likes++;

                };

                $scope.incrementDislikes = function (technology) {

                    technology.dislikes++;

                };

            });

**HtmlPage1.html :** Notice in the html below, we are associating **incrementLikes()** and **incrementDislikes()** functions with the respective button. When any of these buttons are clicked, the corresponsing technology object is automatically passed to the function, and the likes or dislikes property is incremented depending on which button is clicked.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Likes</th>

                    <th>Dislikes</th>

                    <th>Like/Dislike</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="technology in technologies">

                    <td> **{{** technology.name **}}** </td>

                    <td style="text-align:center"> **{{** technology.likes **}}** </td>

                    <td style="text-align:center"> **{{** technology.dislikes **}}** </td>

                    <td>

                        <input type="button" ng-click="incrementLikes(technology)" value="Like"/>

                        <input type="button" ng-click="incrementDislikes(technology)"value="Dislike" />

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css :**Styles for table, td and th elements

table {

    border-collapse: collapse;

    font-family:Arial;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

}

### AngularJS filters

**Filters in angular can do 3 different things**  
1. Format data  
2. Sort data  
3. Filter data   
  
Filters can be used with a binding expression or a directive  
  
To apply a filter use pipe (|) character  
  
**Syntax :** **{{** expression | filterName:parameter **}}**

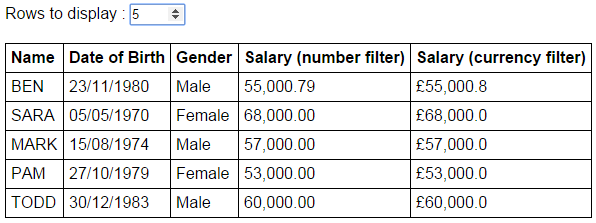
**Angular filters for formatting data**

|  |  |
| --- | --- |
| **Filter** | **Description** |
| lowercase | Formats all characters to lowercase |
| uppercase | Formats all characters to uppercase |
| number | Formats a number as text. Includes comma as thousands separator and the number of decimal places can be specified |
| currency | Formats a number as a currency. $ is default. Custom currency and decimal places can be specified |
| date | Formats date to a string based on the requested format |

**Angular Date formats**

|  |  |
| --- | --- |
| **Format** | **Result** |
| yyyy | 4 digit year. Exampe 1998 |
| yy | 2 digit year. Example 1998 => 98 |
| MMMM | January - December |
| MMM | Jan - Dec |
| MM | 01 - 12 |
| M | 1 - 12 (No leading ZEROS) |
| dd | 01 - 31 |
| d | 1 - 31 (No leading ZEROS) |

**Angular date format documentation**  
<https://docs.angularjs.org/api/ng/filter/date>  
  
**limitTo filter :** Can be used to limit the number of rows or characters in a string.  
  
**Syntax :** **{{** expression | limitTo : limit : begin**}}**

**The following example uses all the above filters**  
  
  
**Script.js**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                {

                    name: "Ben", dateOfBirth: new Date("November 23, 1980"),

                    gender: "Male", salary: 55000.788

                },

                {

                    name: "Sara", dateOfBirth: new Date("May 05, 1970"),

                    gender: "Female", salary: 68000

                },

                {

                    name: "Mark", dateOfBirth: new Date("August 15, 1974"),

                    gender: "Male", salary: 57000

                },

                {

                    name: "Pam", dateOfBirth: new Date("October 27, 1979"),

                    gender: "Female", salary: 53000

                },

                {

                    name: "Todd", dateOfBirth: new Date("December 30, 1983"),

                    gender: "Male", salary: 60000

                }

            ];

            $scope.employees = employees;

            $scope.rowCount = 3;

        });

**HtmlPage1.html** 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Rows to display : <input type="number" step="1"

                                 ng-model="rowCount" max="5" min="0" />

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Date of Birth</th>

                    <th>Gender</th>

                    <th>Salary (number filter)</th>

                    <th>Salary (currency filter)</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | limitTo:rowCount">

                    <td> **{{** employee.name | uppercase **}}** </td>

                    <td> **{{** employee.dateOfBirth | date:"dd/MM/yyyy" **}}** </td>

                    <td> **{{** employee.gender **}}** </td>

                    <td> **{{** employee.salary | number:2 **}}** </td>

                    <td> **{{** employee.salary | currency : "£" : 1 **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css** 

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

}

**Upcoming videos :** Sorting and filtering data using angular filters

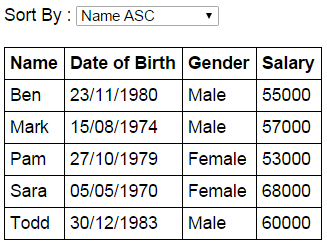
### Sorting data in AngularJS

In this video we will discuss **how to implement sorting in AngularJS**. This is continuation to [Part 8](http://csharp-video-tutorials.blogspot.com/2015/11/angularjs-filters.html). Please watch [Part 8](http://csharp-video-tutorials.blogspot.com/2015/11/angularjs-filters.html) from [AngularJS tutorial for beginners](https://www.youtube.com/playlist?list=PL6n9fhu94yhWKHkcL7RJmmXyxkuFB3KSl) before proceeding.

**To sort the data in Angular**  
**1.** Use orderBy filter  
    **{{** orderBy\_expression | orderBy : expression : reverse**}}**

**Example :** ng-repeat="employee in employees | orderBy:'salary':false"

**2.** To sort in ascending order, set reverse to false  
**3.** To sort in descending order, set reverse to true  
**4.** You can also use + and - to sort in ascending and descending order respectively  
    **Example :**ng-repeat="employee in employees | orderBy:'+salary'"

Let us understand sorting data with an example.  
  
  
The dropdownlist shows the columns and the direction we want to sort  
When a dropdownlist item is selected, the table data should be sorted by the selected column  
  
**Script.js :** The controller function builds the model. Also sortColumn property is added to the $scope object. Notice sortColumn property is initialized to name. This ensures that the data is sorted by name column in ascending order, when the form first loads.

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                {

                    name: "Ben", dateOfBirth: new Date("November 23, 1980"),

                    gender: "Male", salary: 55000

                },

                {

                    name: "Sara", dateOfBirth: new Date("May 05, 1970"),

                    gender: "Female", salary: 68000

                },

                {

                    name: "Mark", dateOfBirth: new Date("August 15, 1974"),

                    gender: "Male", salary: 57000

                },

                {

                    name: "Pam", dateOfBirth: new Date("October 27, 1979"),

                    gender: "Female", salary: 53000

                },

                {

                    name: "Todd", dateOfBirth: new Date("December 30, 1983"),

                    gender: "Male", salary: 60000

                }

            ];

            $scope.employees = employees;

            $scope.sortColumn = "name";

        });

**HtmlPage1.html :** The select element, has the list of columns by which the data should be sorted. + and - symbols control the sort direction. When the form initially loads notice that the data is sorted by name column in ascending order, and name option is automatically selected in the select element. Notice the orderBy filter is using the sortColumn property that is attached to the $scope object. When the selection in the select element changes, the sortColumn property of the $scope object will be updated automatically with the selected value, and in turn the updated value is used by the orderBy filter to sort the data.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Sort By :

        <select ng-model="sortColumn">

            <option value="name">Name ASC</option>

            <option value="+dateOfBirth">Date of Birth ASC</option>

            <option value="+gender">Gender ASC</option>

            <option value="-salary">Salary DESC</option>

        </select>

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Date of Birth</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | orderBy:sortColumn">

                    <td>

**{{** employee.name **}}**

                    </td>

                    <td>

**{{** employee.dateOfBirth | date:"dd/MM/yyyy" **}}**

                    </td>

                    <td>

**{{** employee.gender **}}**

                    </td>

                    <td>

**{{** employee.salary  **}}**

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>  
  
**Styles.css :** CSS styles to make the form look pretty.

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

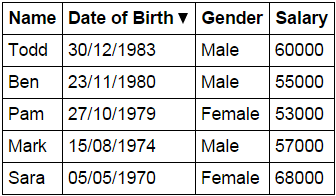
    border: 1px solid black;

    padding: 5px;

    text-align: left;

}

### AngularJS sort rows by table header

**Here is what we want to do**  
1. The data should be sorted when the table column header is clicked  
2. The user should be able to sort in both the directions - ascending and descending. Clicking on the column for the first time should sort the data in ascending order. Clicking on the same column again should sort in descending order.  
3. An icon should be displayed next to the column showing the sort column and direction   
  
   
  
**Script.js :** The controller function in the script does the following

* Sets up the model
* sortColumn and reverseSort properties are attached to the $scope object. These 2 properties are used to control the column by which the data should be sorted and the sort direction.
* sortColumn is set to name and reverseSort is set to false. This will ensure that when the form is initially loaded, the table data will be sorted by name column in ascending order.
* Depending on the column header the user has clicked, **sortData**() function sets the sortColumn and reverseSort property values.
* Based on the sort column and the sort direction, **getSortClass**() function returns the CSS class name to return. The CSS class controls the sort icon that will be displayed next to the sort column.

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                {

                    name: "Ben", dateOfBirth: new Date("November 23, 1980"),

                    gender: "Male", salary: 55000

                },

                {

                    name: "Sara", dateOfBirth: new Date("May 05, 1970"),

                    gender: "Female", salary: 68000

                },

                {

                    name: "Mark", dateOfBirth: new Date("August 15, 1974"),

                    gender: "Male", salary: 57000

                },

                {

                    name: "Pam", dateOfBirth: new Date("October 27, 1979"),

                    gender: "Female", salary: 53000

                },

                {

                    name: "Todd", dateOfBirth: new Date("December 30, 1983"),

                    gender: "Male", salary: 60000

                }

            ];

            $scope.employees = employees;

            $scope.sortColumn = "name";

            $scope.reverseSort = false;

            $scope.sortData = function (column) {

                $scope.reverseSort = ($scope.sortColumn == column) ?

                    !$scope.reverseSort : false;

                $scope.sortColumn = column;

            }

            $scope.getSortClass = function (column) {

                if ($scope.sortColumn == column) {

                    return $scope.reverseSort

                      ? 'arrow-down'

                      : 'arrow-up';

                }

                return '';

            }

        });

**HtmlPage1.html :** **sortData()** function is called when any table header is clicked, passing the name of the column by which the data should be sorted. The div element's, ng-class directive calls **getSortClass**() function, which returns the CSS class to be applied. The CSS displays the UP or DOWN arrow depending on the sort direction. Finally, with the orderBy filter sortColumn and reverseSort properties of the $scope object are used to control the column by which the data should be sorted and the sort direction. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th ng-click="sortData('name')">

                        Name <div ng-class="getSortClass('name')"></div>

                    </th>

                    <th ng-click="sortData('dateOfBirth')">

                        Date of Birth <div ng-class="getSortClass('dateOfBirth')"></div>

                    </th>

                    <th ng-click="sortData('gender')">

                        Gender <div ng-class="getSortClass('gender')"></div>

                    </th>

                    <th ng-click="sortData('salary')">

                        Salary <div ng-class="getSortClass('salary')"></div>

                    </th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | orderBy:sortColumn:reverseSort">

                    <td> **{{** employee.name **}}** </td>

                    <td> **{{** employee.dateOfBirth | date:"dd/MM/yyyy" **}}** </td>

                    <td> **{{** employee.gender **}}** </td>

                    <td> **{{** employee.salary  **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css :** CSS styles to make the form look pretty. 

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

    /\*cursor property displays hand symbol

        when hovered over the th element\*/

    cursor: pointer;

}

/\*This class displays the UP arrow\*/

.arrow-up {

     width: 0;

     height: 0;

     border-left: 5px solid transparent;

     border-right: 5px solid transparent;

     border-bottom: 10px solid black;

     display:inline-block;

}

/\*This class displays the DOWN arrow\*/

.arrow-down {

     width: 0;

     height: 0;

     border-left: 5px solid transparent;

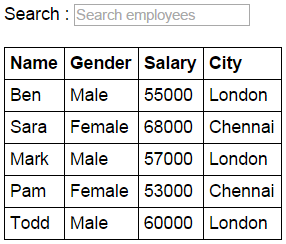
     border-right: 5px solid transparent;

     border-top: 10px solid black;

     display:inline-block;

}

### Search filter in AngularJS

   
  
As we type in the search textbox, all the columns in the table must be searched and only the matching rows should be displayed.    
  
**Script.js :**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000, city: "London" },

                { name: "Sara", gender: "Female", salary: 68000, city: "Chennai" },

                { name: "Mark", gender: "Male", salary: 57000, city: "London" },

                { name: "Pam", gender: "Female", salary: 53000, city: "Chennai" },

                { name: "Todd", gender: "Male", salary: 60000, city: "London" },

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :**

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Search : <input type="text" placeholder="Search employees"

                        ng-model="searchText" />

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                    <th>City</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | filter:searchText">

                    <td> **{{** employee.name **}}** </td>

                    <td> **{{** employee.gender **}}** </td>

                    <td> **{{** employee.salary  **}}** </td>

                    <td> **{{** employee.city  **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css :**

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

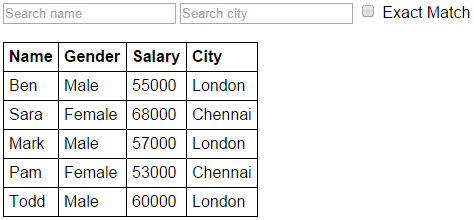
}

At the moment, the search is being done across all columns. If you want to search only one specific column, then change **ng-model** directive value on the search textbox as shown below. **With this change only city column is searched.**

<input type="text" ng-model="searchText.city" placeholder="Search employees" />

### Angularjs filter by multiple properties

In this video we will discuss **how to filter by multiple properties in AngularJS**.

In the example below, we are using multiple search textboxes. As you type in the **"Search name"** textbox, only the name property is searched and matching elements are displayed. Similarly, as you type in the **"Search city"** textbox, only the city property is searched and the matching elements are displayed. When the **"exact match"**checkbox is checked, an exact match search is performed.   
  
   
  
**Script.js :**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000, city: "London" },

                { name: "Sara", gender: "Female", salary: 68000, city: "Chennai" },

                { name: "Mark", gender: "Male", salary: 57000, city: "London" },

                { name: "Pam", gender: "Female", salary: 53000, city: "Chennai" },

                { name: "Todd", gender: "Male", salary: 60000, city: "London" },

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :** 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="text" placeholder="Search name" ng-model="searchText.name" />

        <input type="text" placeholder="Search city" ng-model="searchText.city" />

        <input type="checkbox" ng-model="exactMatch" /> Exact Match

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                    <th>City</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | filter: searchText : exactMatch">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender }} </td>

                    <td> {{ employee.salary  }} </td>

                    <td> {{ employee.city  }} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css** 

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

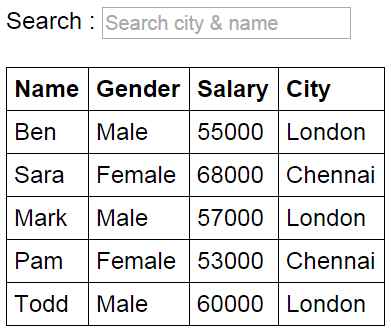
th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

}

The following example has a single search textbox, and is used to search multiple properties - name and city.   
  
   
  
**Script.js :**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000, city: "London" },

                { name: "Sara", gender: "Female", salary: 68000, city: "Chennai" },

                { name: "Mark", gender: "Male", salary: 57000, city: "London" },

                { name: "Pam", gender: "Female", salary: 53000, city: "Chennai" },

                { name: "Todd", gender: "Male", salary: 60000, city: "London" },

            ];

            $scope.employees = employees;

            $scope.search = function (item) {

                if ($scope.searchText == undefined) {

                    return true;

                }

                else {

                    if (item.city.toLowerCase()

                                 .indexOf($scope.searchText.toLowerCase()) != -1 ||

                        item.name.toLowerCase()

                                 .indexOf($scope.searchText.toLowerCase()) != -1) {

                        return true;

                    }

                }

                return false;

            };

        });

**HtmlPage1.html :** 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Search : <input type="text" placeholder="Search city & name"

                        ng-model="searchText" />

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                    <th>City</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | filter: search">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender }} </td>

                    <td> {{ employee.salary  }} </td>

                    <td> {{ employee.city  }} </td>

                </tr>

            </tbody>

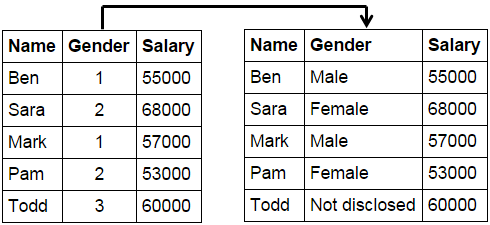
        </table>

    </div>

</body>

</html>

### Create a custom filter in AngularJS

**Custom filter in AngularJS**  
1. Is a function that returns a function  
2. Use the filter function to create a custom filter   
  
Let us understand creating custom filter with an example.   
   
  
**Script.js :** In the example below we are using the filter function to create a custom filter that converts **integer values 1, 2, 3** to **Male, Female and Not disclosed respectively**. gender is the name of the filter. With in the filter function we have an anonymous function that returns another anonymous function. The input parameter for the inner anonynous function is the gender integer value. The switch statement in the function returns the corresponding string value. 

var app = angular

        .module("myModule", [])

        .filter("gender", function () {

            return function (gender) {

                switch (gender) {

                    case 1:

                        return "Male";

                    case 2:

                        return "Female";

                    case 3:

                        return "Not disclosed";

                }

            }

        })

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: 1, salary: 55000 },

                { name: "Sara", gender: 2, salary: 68000 },

                { name: "Mark", gender: 1, salary: 57000 },

                { name: "Pam", gender: 2, salary: 53000 },

                { name: "Todd", gender: 3, salary: 60000 }

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :** In the view, we use the custom gender filter like any other angular built-in filter with a pipe character.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> **{{** employee.name **}}** </td>

                    <td> **{{** employee.gender | gender**}}** </td>

                    <td> **{{** employee.salary  **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

In the above example we have the custom filter in **Script.js** file. In a real world application you may the custom filters in a separate script file (**Filters.js** for example). To do this reference the module object and use the filter function.  
  
**Filter.js :** The custom filter is moved to a separate file

/// <reference path="Script.js" />

app.filter("gender", function () {

    return function (gender) {

        switch (gender) {

            case 1:

                return "Male";

            case 2:

                return "Female";

            case 3:

                return "Not disclosed";

        }

    }

});

**Script.js :**After moving the filter function to a separate **Filters.js** file, the **Script.js**file will now look as shown below.

/// <reference path="angular.min.js" />

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: 1, salary: 55000 },

                { name: "Sara", gender: 2, salary: 68000 },

                { name: "Mark", gender: 1, salary: 57000 },

                { name: "Pam", gender: 2, salary: 53000 },

                { name: "Todd", gender: 3, salary: 60000 }

            ];

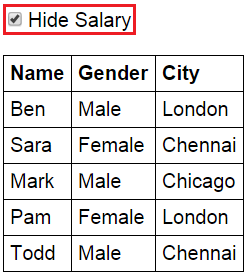
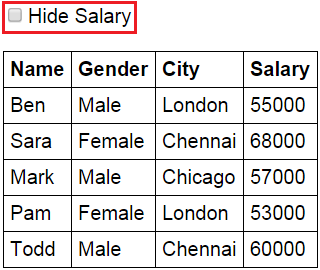
            $scope.employees = employees;

        });

**HtmlPage1.html :** The only change required in the view is to reference the Filters.js file

<script src="Scripts/Filters.js"></script>

### ng-hide and ng-show in AngularJS

ng-hide and ng-show directives are used to control the visibility of the HTML elements. Let us understand this with an example   
  
When Hide Salary checkbox is checked, the Salary column should be hidden.   
   
  
When it is unchecked the Salary column should be unhidden   
   
  
**Script.js :** The controller function builds the model 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", city: "London", salary: 55000 },

                { name: "Sara", gender: "Female", city: "Chennai", salary: 68000 },

                { name: "Mark", gender: "Male", city: "Chicago", salary: 57000 },

                { name: "Pam", gender: "Female", city: "London", salary: 53000 },

                { name: "Todd", gender: "Male", city: "Chennai", salary: 60000 }

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :** Notice **ng-model** directive on the checkbox is set to **hideSalary**. hideSalary variable is then used as the value for ng-hide directive on the th and td elements that displays Salary. When the page is first loaded, hideSalary variable will be undefined which evaluates to false, as a result Salary column will be visible. When the checkbox is checked, hideSalary variable will be attached to the $scope object and true value is stored in it. This value is then used by the ng-hide directive to hide the salary td and it's th element. When the checkbox is unchecked, false value is stored in the hideSalary variable, which is then used by the ng-hide directive to display the Salary column. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-hide="hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-hide="hideSalary"> {{ employee.salary  }} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

With the above example we can also use **ng-show** directive instead of **ng-hide**directive. For this example to behave the same as before, we will have to negate the value of hideSalary variable using ! operator. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-show="!hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-show="!hideSalary"> {{ employee.salary  }} </td>

                </tr>

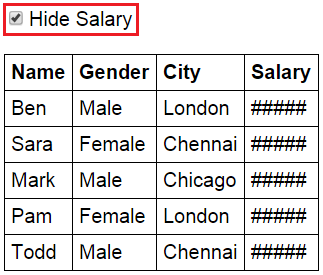
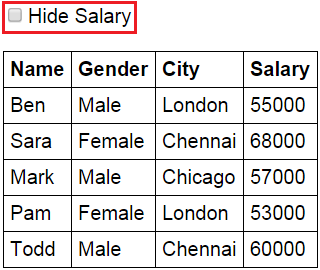
            </tbody>

        </table>

    </div>

</body>

</html>

The following example masks and unmasks the Salary column values using **ng-hide**and **ng-show**directives, depending on the checked status of the Hide Salary checkbox.  
  
   
  
 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-hide="hideSalary">Salary</th>

                    <th ng-show="hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-hide="hideSalary"> {{ employee.salary  }} </td>

                    <td ng-show="hideSalary"> ##### </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

### AngularJS ng-init directive

The **ng-init directive** allows you to evaluate an expression in the current scope.

In the following example, the **ng-init directive**initializes employees variable which is then used in the ng-repeat directive to loop thru each employee. In a real world application you should use a controller instead of **ng-init**to initialize values on a scope. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

</head>

<body ng-app>

    <div ng-init="employees = [

                    { name: 'Ben', gender: 'Male', city: 'London' },

                    { name: 'Sara', gender: 'Female', city: 'Chennai' },

                    { name: 'Mark', gender: 'Male', city: 'Chicago' },

                    { name: 'Pam', gender: 'Female', city: 'London' },

                    { name: 'Todd', gender: 'Male', city: 'Chennai' }

                ]">

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**ng-init** should only be used for aliasing special properties of ng-repeat directive. In the following example, ng-init is used to store the index of the parent element in parentIndex variable.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <ul>

            <li ng-repeat="country in countries" ng-init="parentIndex = $index">

                {{country.name}}

                <ul>

                    <li ng-repeat="city in country.cities">

                        {{city.name}} - Parent Index = {{ parentIndex }}, Index = {{ $index }}

                    </li>

                </ul>

            </li>

        </ul>

    </div>

</body>

</html>

**Script.js** 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var countries = [

                {

                    name: "India",

                    cities: [

                        { name: "Hyderabad" },

                        { name: "Chennai" }

                    ]

                },

                {

                    name: "USA",

                    cities: [

                        { name: "Los Angeles" },

                        { name: "Chicago" },

                    ]

                }

            ];

            $scope.countries = countries;

        });

### ng-include directive in AngularJS

ng-include directive is used to embed an HTML page into another HTML page. This technique is extremely useful when you want to reuse a specific view in multiple pages in your application.   
  
The value of ng-include directive can be the name of the HTML page that you want to reuse or a property on the $scope object that points to the reusable HTML page.  
  
**EmployeeList.html :** This is the HTML page that we intend to reuse on multiple HTML pages

<table>

    <thead>

        <tr>

            <th>Name</th>

            <th>Gender</th>

            <th>Salary</th>

        </tr>

    </thead>

    <tbody>

        <tr ng-repeat="employee in employees">

            <td> **{{** employee.name **}}** </td>

            <td> **{{** employee.gender**}}** </td>

            <td> **{{** employee.salary**}}** </td>

        </tr>

    </tbody>

</table>  
  
**Script.js :**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000 },

                { name: "Sara", gender: "Female", salary: 68000 },

                { name: "Mark", gender: "Male", salary: 57000 },

                { name: "Pam", gender: "Female", salary: 53000 },

                { name: "Todd", gender: "Male", salary: 60000 }

            ];

            $scope.employees = employees;

        });

**HTMLPage1.html :**This is the HTML page where we want to reuse EmployeeList.html. Notice that we are using ng-include directive and the value for it is the name of the HTML file that we want to reuse.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <div ng-include="'EmployeeList.html'">

        </div>

    </div>

</body>

</html>

In this example, we have specified the name of the HTML file in the view. You can also have a property attached to the $scope object that points to the HTML file that you want to reuse , and use that property with ng-include directive. Here are the changes required to use a model property with ng-include directive.  
  
**Script.js :**Notice, in the controller function we have employeeList property attached to the $scope object. This property points to the EmployeeList.html file that we want to reuse.

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000 },

                { name: "Sara", gender: "Female", salary: 68000 },

                { name: "Mark", gender: "Male", salary: 57000 },

                { name: "Pam", gender: "Female", salary: 53000 },

                { name: "Todd", gender: "Male", salary: 60000 }

            ];

            $scope.employees = employees;

            $scope.employeeList = "EmployeeList.html";

        });

**HTMLPage1.html :**Set the property employeeList that you have attached to the $scope object, as the value for ng-include directive

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

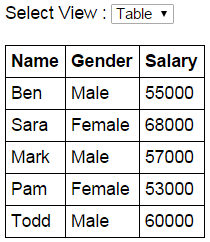
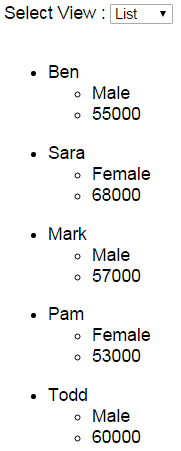
    <div ng-controller="myController">

        <div ng-include="employeeList"></div>

    </div>

</body>

</html>

**Example :** Create an HTML page with a dropdownlist that allows the user to select the view - Table or List. Depending on the selection we want to load the respective HTML page into the current HTML page i.e HTMLPage1.html  
  
If the user selects Table from the dropdownlist, the employee data should be presented using a Table  
  
  
If the user selects List from the dropdownlist, the employee data should be presented using an unordered list  
  
  
**EmployeeTable.html :** This HTML page presents the employee data using a table element

<table>

    <thead>

        <tr>

            <th>Name</th>

            <th>Gender</th>

            <th>Salary</th>

        </tr>

    </thead>

    <tbody>

        <tr ng-repeat="employee in employees">

            <td> **{{** employee.name **}}** </td>

            <td> **{{** employee.gender**}}** </td>

            <td> **{{** employee.salary**}}** </td>

        </tr>

    </tbody>

</table>

**EmployeeList.html :**This HTML page presents the employee data using 2 unordered list elements

<ul ng-repeat="employee in employees">

    <li>

**{{**employee.name**}}**

        <ul>

            <li>**{{**employee.gender**}}**</li>

            <li>**{{**employee.salary**}}**</li>

        </ul>

    </li>

</ul>

**Script.js :**The controller function attaches employeeView property to the $scope object and sets it to EmployeeTable.html. This means when the page is initially loaded the employee data will be presented using a table.

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000 },

                { name: "Sara", gender: "Female", salary: 68000 },

                { name: "Mark", gender: "Male", salary: 57000 },

                { name: "Pam", gender: "Female", salary: 53000 },

                { name: "Todd", gender: "Male", salary: 60000 }

            ];

            $scope.employees = employees;

            $scope.employeeView = "EmployeeTable.html";

        });

**HTMLPage1.html :** This HTML page loads either the EmployeeTable.html or EmployeeList.html page depending on the item the user has selected from the dropdownlist.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Select View :

        <select ng-model="employeeView">

            <option value="EmployeeTable.html">Table</option>

            <option value="EmployeeList.html">List</option>

        </select>

        <br /><br />

        <div ng-include="employeeView">

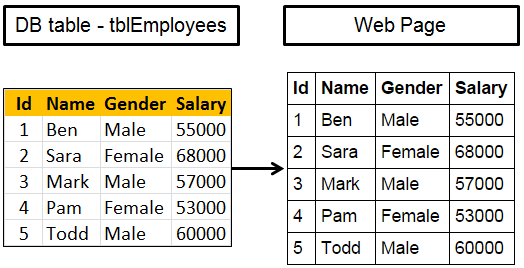
        </div>

    </div>

</body>

</html>

### Consuming ASP.NET Web Service in AngularJS using $http

   
  
**Here is what we want to do**  
1. Create an ASP.NET Web service. This web service retrieves the data from SQL Server database table, returns it in JSON formt.  
2. Call the web service using AngularJS and display employee data on the web page   
  
**Step 1 :**Create SQL Server table and insert employee data 

Create table tblEmployees

(

    Id int primary key identity,

    Name nvarchar(50),

    Gender nvarchar(10),

    Salary int

)

Go

Insert into tblEmployees values ('Ben', 'Male', 55000)

Insert into tblEmployees values ('Sara', 'Female', 68000)

Insert into tblEmployees values ('Mark', 'Male', 57000)

Insert into tblEmployees values ('Pam', 'Female', 53000)

Insert into tblEmployees values ('Todd', 'Male', 60000)

Go

**Step 2 :** Create new empty asp.net web application project. Name it Demo.   
  
**Step 3 :** Include the following settings in web.config file.

<?xml version="1.0" encoding="utf-8"?>

<configuration>

  <connectionStrings>

    <add name="DBCS"

         connectionString="server=.;database=SampleDB; integrated security=SSPI"/>

  </connectionStrings>

  <system.web>

    <webServices>

      <protocols>

        <add name="HttpGet"/>

      </protocols>

    </webServices>

  </system.web>

</configuration>

**Step 4 :** Add a class file to the project. Name it Employee.cs. Copy and paste the following code.

namespace Demo

{

    public class Employee

    {

        public int id { get; set; }

        public string name { get; set; }

        public string gender { get; set; }

        public int salary { get; set; }

    }

}

**Step 5 :** Add a new WebService (ASMX). Name it EmployeeService.asmx. Copy and paste the following code.

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Web.Script.Serialization;

using System.Web.Services;

namespace Demo

{

    [WebService(Namespace = "http://tempuri.org/")]

    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

    [System.ComponentModel.ToolboxItem(false)]

    [System.Web.Script.Services.ScriptService]

    public class EmployeeService : System.Web.Services.WebService

    {

        [WebMethod]

        public void GetAllEmployees()

        {

            List<Employee> listEmployees = new List<Employee>();

            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

            using (SqlConnection con = new SqlConnection(cs))

            {

                SqlCommand cmd = new SqlCommand("Select \* from tblEmployees", con);

                con.Open();

                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())

                {

                    Employee employee = new Employee();

                    employee.id = Convert.ToInt32(rdr["Id"]);

                    employee.name = rdr["Name"].ToString();

                    employee.gender = rdr["Gender"].ToString();

                    employee.salary = Convert.ToInt32(rdr["Salary"]);

                    listEmployees.Add(employee);

                }

            }

            JavaScriptSerializer js = new JavaScriptSerializer();

            Context.Response.Write(js.Serialize(listEmployees));

        }

    }

}

**Step 6 :** Add a new folder to the project. Name it Scripts. Download angular.js script file from [http://angularjs.org](http://angularjs.org/), and past it in Scripts folder.   
  
**Step 7 :** Add a new JavaScript file to the Scripts folder. Name it **Script.js**. Copy and paste the following code. 

/// <reference path="angular.min.js" />

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope, $http) {

            $http.get("EmployeeService.asmx/GetAllEmployees")

                 .then(function (response) {

                     $scope.employees = response.data;

                 });

        });

**Step 8 :** Add a new stylesheet to the project. Name it Styles.css. Copy and paste the following styles in it. 

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

}

**Step 9 :** Add an HTML page to the ASP.NET project. Copy and paste the following HTML and Angular code 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Id</th>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td>**{{**employee.id**}}**</td>

                    <td>**{{**employee.name**}}**</td>

                    <td>**{{**employee.gender**}}**</td>

                    <td>**{{**employee.salary**}}**</td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

### $http service in AngularJS

In Angular there are several built in services. $http service is one of them. In this video, we will discuss another built in service, $log. It is also possible to create our own custom services in Angular.   
  
**At this point several questions come to our mind**

* What are services in Angular
* When should we be creating services in Angular
* How to create our own custom Angular services
* Where do they fit, in an angular application architecture
* What are the benefits of using services

I will answer all these questions in a later video. The reason for postponing this discussion, is that, it is easier to understand the concept of Angular services and the benefits they provide, once we understand how to use use 1 or 2 built in angular services.   
  
So, let's start our discussion with $http service.   
  
$http service in Angular is used to make HTTP requests to remote server   
  
$http service is a function that has a single input parameter i.e a configuration object.   
  
**Example :** The following example issues a GET request to the specified URL 

$http({

    method: 'GET',

    url: 'EmployeeService.asmx/GetAllEmployees'

});

In the example above we are only using 2 properties of the configuration object. Check the link below for the complete list of properties supported by the configuration object  
<https://docs.angularjs.org/api/ng/service/$http#usage>  
  
Shortcut methods like get, post, put, delete etc are also available to be used with $http service  
  
**Example :** Using the short cut method get()  
$http.get('EmployeeService.asmx/GetAllEmployees')  
  
**$http service** returns a promise object. This means the functions are executed asynchronously and the data that these functions return may not be available immediately. Because of this reason you cannot use the return value of the $http service as shown below.  
  
~~$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployees');~~  
  
Instead you will use the **then**() method. The successCallback function that is passed as the parameter to the then function is called when the request completes. The successCallback function receives a single object that contains several properties. Use the data property of the object to retrieve the data received from the server.

$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployees')

            .then(function (response) {

                $scope.employees = response.data;

            });

You can use the **$log service** to log the response object to the console to inspect all of it's properties 

$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployees')

                        .then(function (response) {

                            $scope.employees = response.data;

                            $log.info(response);

                        });

If there is an error processing the request, the **errorCallback function**is called. The errorCallback function is passed as the second parameter to the then() function. The errorCallback function receives a single object that contains several properties. Use the data or statusText properties of the returned object to find the reasons for the failure.

$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployee')

                        .then(function (response) {

                            $scope.employees = response.data;

                        }, function (reason) {

                            $scope.error = reason.data;

                        });

You can use the $log service to log the response object to the console to inspect all of it's properties 

$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployee')

                        .then(function (response) {

                            $scope.employees = response.data;

                        }, function (reason) {

                            $scope.error = reason.data;

                            $log.info(reason);

                        });

You can also create separate functions and associate them as successCallback and errorCallback functions

var successCallBack = function (response) {

    $scope.employees = response.data;

};

var errorCallBack = function (reason) {

    $scope.error = reason.data;

}

$scope.employees = $http.get('EmployeeService.asmx/GetAllEmployees')

                        .then(successCallBack, errorCallBack);

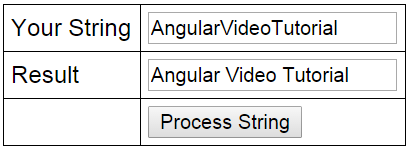
**Default Transformations provided by Angular's http service**

* If the data property of the request configuration object contains a JavaScript object, it is automatically converted into JSON object
* If JSON response is detected, it is automatically converted into a JavaScript object

### AngularJS Services

**What is a service in AngularJS**  
Before we talk about what a service is in Angular. Let's talk about a service in web development.    
  
**If you have any experience developing web applications**  
1. You might have heard about Web Services and WCF Services  
2. You might have also created objects that provide some services. For example, a Math object may provide services to add numbers.   
  
So, a service in Angular is simply an object that provide some sort of service that can be reused with in an angular application. The angular service object has properties and methods, just like any other JavaScript object.    
  
AngularJS has lot of built in services. We discussed two of the built in services - $http & $log, in our previous video. $http service is used to make AJAX calls. $log service is useful to log an object to the console, which is very useful when debugging applications. We can also create our own custom services, which we will discuss in a later video.   
  
For now let's understand the need for services.   
  
**Why do we need services in an angular application**  
The primary responsibility of the controller is to build the model for the view.  The controller should not be doing too many things. For example, if the controller also has the logic to compute Age from Date of Birth, it violates one of the SOLID principles i.e the Single Responsibility Principle. The Single Responsibility Principle states that an object should only have a Single Responsibility. So this kind a logic belongs in it's own service, which can then be injected into the object that needs that service.   
  
In our previous video session, we have injected 2 of the angular built in services i.e $http and $log service into the controller function that needs them.  
  
In general, if the logic with in your controller, is becoming too large or too complex, then it is time, to take a step back, and think if anything can be abstracted into it's own service.  
  
Services can be used by controllers, directives and filters.  
  
**What are the benefits of using services**  
Reusability : In a service you usually have a logic that you want to reuse with in your entire application. For example, any time you want to make AJAX calls, you can use one of the built in angular service - $http, simply by injecting it into the object that needs that service. The application is also easier to maintain when the reusable components are encapsulated into their own services.  
  
Dependency Injection : Another benefit of services, is that, they can simply be injected into controllers or other services that need them.  
  
Testability : Since services are injected into controllers or other services that need them, it becomes very easy to test them. Depending on which service you are testing, you can pass mock implementations or real implementations. If you are new to unit testing and mocking, don't worry, we will be covering these in detail in our upcoming videos.  
  
In our next video we will discuss, **how to create a custom service, register and use it**.

### Create custom service in AngularJS

Whenever the case changes from lower to upper, a single space character should be inserted. This means the string **"AngularVideoTutorial"** should be converted to **"Angular Video Tutorial"**.   
  
   
  
Let us first see, how to achieve this without using a custom service.   
  
**HtmlPage1.html :**  

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <tr>

                <td>Your String</td>

                <td><input type="text" ng-model="input" /> </td>

            </tr>

            <tr>

                <td>Result</td>

                <td><input type="text" ng-model="output" /></td>

            </tr>

            <tr>

                <td></td>

                <td>

                    <input type="button" ng-click="transformString(input)"

                           value="Process String" />

                </td>

            </tr>

        </table>

    </div>

</body>

</html>

**Script.js :**Notice, all the logic to insert a space when the case changes is in the controller. There are 2 problems with this  
1. The controller is getting complex  
2. This logic cannot be reused in another controller. If you have to use this logic in another controller, we will have to duplicate this same code with in that controller.   
  
When we use our own custom service to encapsulate this logic, both of these problems go away. The custom service can be injected into any controller where you need this logic. 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            $scope.transformString = function (input) {

                if (!input)

                    return input;

                var output = "";

                for (var i = 0; i < input.length; i++) {

                    if (i > 0 && input[i] == input[i].toUpperCase()) {

                        output = output + " ";

                    }

                    output = output + input[i];

                }

                $scope.output = output;

            };

        });

Now let's create a custom service. Here are the steps  
1. Add a JavaScript file to the Scripts folder in the project. Name it stringService.js.  
2. Copy and paste the following code. Notice we are using the factory method to create and register the service with Angular. 

app.factory('stringService', function () {

    return {

        processString: function (input) {

            if (!input)

                return input;

            var output = "";

            for (var i = 0; i < input.length; i++) {

                if (i > 0 && input[i] == input[i].toUpperCase()) {

                    output = output + " ";

                }

                output = output + input[i];

            }

            return output;

        }

    };

});

3. Copy and paste the following code in Script.js. Notice that we have injected stringService into the controller function. 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope, stringService) {

            $scope.transformString = function (input) {

                $scope.output = stringService.processString(input);

            };

        });

4. On HtmlPage1.html, only one change is required and that is to reference the stringService.js script file

<script src="Scripts/stringService.js"></script>

### AngularJS anchorscroll example

* $anchorscroll service is used to jump to a specified element on the page
* $location service hash method appends hash fragments to the URL
* $anchorscroll() method reads the hash fragment in the URL and jumps to that element on the page
* yOffset property specifies the vertical scroll-offset

**Example : HtmlPage1.html**

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="demoApp">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-controller="demoController">

    <button id="top" ng-click="scrollTo('bottom')">

        Go to bottom of the page

    </button>

    <br /><br />

    <div>

        <b>What is AngularJS</b>

        <br />

        AngularJS is a JavaScript framework that helps build applications that run in a web browser.

        <br /><br />

        <b>Who developed AngularJS</b>

        <br />

        Google is the company that developed AngularJS. AngularJS is an open source project, which means it can be be freely used, changed, and shared by anyone.

        <br /><br />

        AngularJS is an excellent framework for building both Single Page Applications (SPA) and Line of Business Applications. Many companies are using Angular today, and there are many public facing web sites that are built with angular.

        <br /><br />

        There is a website, https://www.madewithangular.com, that has the list of web sites that are built using AngularJS. Within this list you can find many popular websites.

        <br /><br />

        <b>What are the benefits of using AngularJS</b>

        <br />

        <b>1. Dependency Injection : </b>Dependency Injection is something AngularJS does quite well. If you are new to Dependency Injection, don't worry, we will discuss it in detail with examples in a later video.

        <br /><br />

        <b>2. Two Way Data-Binding : </b>One of the most useful feature in AngularJS is the Two Way Data-Binding. The Two Way Data-Binding, keeps the model and the view in sync at all times, that is a change in the model updates the view and a change in the view updates the model.

        <br /><br />

        <b>3. Testing : </b>Testing is an area where Angular really shines. Angular is designed with testing in mind right from the start. Angular makes it very easy to test any of it's components through both unit testing and end to end testing. So there's really no excuse for not testing any of your angular application code.

        <br /><br />

        <b>4. Model View Controller : </b>With angular it is very easy to develop applications in a clean MVC way. All you have to do is split your application code into MVC components. The rest, that is managing those components and connecting them together is done by angular.

        <br /><br />

        <b>5. Many more benefits like controlling the behaviour of DOM elements using directives and the flexibility that angular filters provide.</b>

        <br /><br />

        We will discuss directives, filters, Modules, Routes etc with examples in our upcoming videos in this series.

        <br /><br />

        To build angular applications you only need one script file and that is angular.js.

        <br /><br />

        <b>To get the script file visit https://angularjs.org. From here</b>

        <br />

        1. You can download the angular script file<br />

        2. CDN link - We discussed the benefits of using CDN in Part 3 of jQuery tutorial.<br />

        3. Various resources to learn angular - Here you will find videos, Free courses, Tutorials and Case Studies. You will also find API reference which is extremeley useful.<br />

        <br /><br />

        <b>To get started with angular</b>

        <br />

        1. Add a reference to the angular script

        <br />

        2. Include ng-app attribute

        <br /><br />

        <b>What is ng-app</b>

        <br />

        In angular, ng-app is called a directive. There are many directives in angular. You can find the complete list of directives on https://angularjs.org. The ng prefix in the directive stands for angular. The ng-app directive is a starting point of AngularJS Application. Angular framework will first check for ng-app directive in an HTML page after the entire page is loaded. If ng-app directive is found, angular bootstraps itself and starts to manage the section of the page that has the ng-app directive.

        <br /><br />

        <b>So the obvious next question is, where to place the ng-app directive on the page</b>

        <br /><br />

        It should be placed at the root of the HTML document, that is at the html tag level or at the body tag level, so that angular can control the entire page.

        <br /><br />

        However, there is nothing stopping you from placing it on any other HTML element with in the page. When you do this only that element and it's children are managed by angular.

        <br /><br />

        <span>Double curly braces are called binding expressions in angular.</span>

    </div>

    <br />

    <button id="bottom" ng-click="scrollTo('top')">

        Go to top of the page

    </button>

</body>

</html>

**Script.js**

/// <reference path="angular.js" />

var demoApp = angular.module("demoApp", [])

                     .controller("demoController", function

                         ($scope, $location, $anchorScroll) {

                         $scope.scrollTo = function (scrollLocation) {

                             $location.hash(scrollLocation);

                             $anchorScroll.yOffset = 20;

                             $anchorScroll();

                         }

                     });

**Styles.css**

div {

    width: 400px;

    border: 1px solid black;

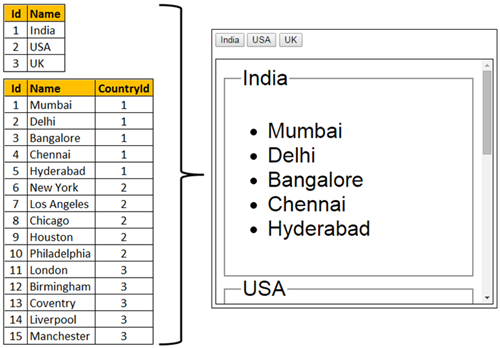
    font-family: Arial;

    font-size: large;

    padding: 5px;

}

### Angular anchorscroll with database data

So here is what we want to do. Retrieve the **countries**and **cities**data from respective tables in the SQL server database and display it on the web page. When we click on a country button, the page should automatically scroll to the respective country and it's cities.   
  
   
  
**Step 1 :** Create SQL Server tables and insert data 

Create Table tblCountry

(

    Id int primary key identity,

    Name nvarchar(50)

)

Go

Insert into tblCountry values ('India')

Insert into tblCountry values ('USA')

Insert into tblCountry values ('UK')

Go

Create Table tblCity

(

    Id int primary key identity,

    Name nvarchar(50),

    CountryId int foreign key references tblCountry(Id)

)

Go

Insert into tblCity values ('Mumbai', 1)

Insert into tblCity values ('Delhi', 1)

Insert into tblCity values ('Bangalore', 1)

Insert into tblCity values ('Chennai', 1)

Insert into tblCity values ('Hyderabad', 1)

Insert into tblCity values ('New York', 2)

Insert into tblCity values ('Los Angeles', 2)

Insert into tblCity values ('Chicago', 2)

Insert into tblCity values ('Houston', 2)

Insert into tblCity values ('Philadelphia', 2)

Insert into tblCity values ('London', 3)

Insert into tblCity values ('Birmingham', 3)

Insert into tblCity values ('Coventry', 3)

Insert into tblCity values ('Liverpool', 3)

Insert into tblCity values ('Manchester', 3)

Go

**Step 2 :** Create new empty asp.net web application project. Name it Demo.   
  
**Step 3 :** Include the following settings in web.config file.

<?xml version="1.0" encoding="utf-8"?>

<configuration>

  <connectionStrings>

    <add name="DBCS"

         connectionString="server=.;database=SampleDB; integrated security=SSPI"/>

  </connectionStrings>

  <system.web>

    <webServices>

      <protocols>

        <add name="HttpGet"/>

      </protocols>

    </webServices>

  </system.web>

</configuration>

**Step 4 :** Add a class file to the project. Name it **City.cs**. Copy and paste the following code.

namespace Demo

{

    public class City

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public int CountryId { get; set; }

    }

}

**Step 5 :** Add a class file to the project. Name it **Country.cs**. Copy and paste the following code.

using System.Collections.Generic;

namespace Demo

{

    public class Country

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public List<City> Cities { get; set; }

    }

}

**Step 6 :** Add a new WebService (ASMX). Name it CountryService.asmx. Copy and paste the following code.

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Web.Script.Serialization;

using System.Web.Services;

namespace Demo

{

    [WebService(Namespace = "http://tempuri.org/")]

    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

    [System.ComponentModel.ToolboxItem(false)]

    [System.Web.Script.Services.ScriptService]

    public class CountryService : System.Web.Services.WebService

    {

        [WebMethod]

        public void GetData()

        {

            List<Country> listCountries = new List<Country>();

            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

            using (SqlConnection con = new SqlConnection(cs))

            {

                SqlCommand cmd = new SqlCommand("Select \* from tblCountry;Select \* from tblCity", con);

                SqlDataAdapter da = new SqlDataAdapter(cmd);

                DataSet ds = new DataSet();

                da.Fill(ds);

                DataView dataView = new DataView(ds.Tables[1]);

                foreach (DataRow countryDataRow in ds.Tables[0].Rows)

                {

                    Country country = new Country();

                    country.Id = Convert.ToInt32(countryDataRow["Id"]);

                    country.Name = countryDataRow["Name"].ToString();

                    dataView.RowFilter = "CountryId = '" + country.Id + "'";

                    List<City> listCities = new List<City>();

                    foreach (DataRowView cityDataRowView in dataView)

                    {

                        DataRow cityDataRow = cityDataRowView.Row;

                        City city = new City();

                        city.Id = Convert.ToInt32(cityDataRow["Id"]);

                        city.Name = cityDataRow["Name"].ToString();

                        city.CountryId = Convert.ToInt32(cityDataRow["CountryId"]);

                        listCities.Add(city);

                    }

                    country.Cities = listCities;

                    listCountries.Add(country);

                }

            }

            JavaScriptSerializer js = new JavaScriptSerializer();

            Context.Response.Write(js.Serialize(listCountries));

        }

    }

}

**Step 7 :** Add a new folder to the project. Name it Scripts. Add a new JavaScript file to the Scripts folder. Name it Script.js. Copy and paste the following code.

/// <reference path="angular.js" />

var demoApp = angular.module("demoApp", [])

                     .controller("countryController",

                        function ($scope, $location, $anchorScroll, $http) {

                         $http.get("CountryService.asmx/GetData")

                              .then(function (response) {

                                  $scope.countries = response.data;

                              });

                         $scope.scrollTo = function (countryName) {

                             $location.hash(countryName);

                             $anchorScroll();

                         }

                     });

**Step 8 :** Add a new stylesheet to the project. Name it Styles.css. Copy and paste the following styles in it. 

body {

    font-family: Arial;

}

div {

    display: block;

    font-size: xx-large;

    height: 350px;

    width: 400px;

    border: 1px solid black;

    padding: 10px;

    overflow-y: scroll;

}

**Step 9 :** Add an HTML page to the ASP.NET project. Copy and paste the following HTML and Angular code

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="demoApp">

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-controller="countryController">

    <span ng-repeat="country in countries">

        <button ng-click="scrollTo(country.Name)">{{country.Name}}</button>

    </span>

    <br /><br />

    <div class="containerDiv">

        <fieldset ng-repeat="country in countries" id="{{country.Name}}">

            <legend>{{country.Name}}</legend>

            <ul>

                <li ng-repeat="city in country.Cities">

                    {{city.Name}}

                </li>

            </ul>

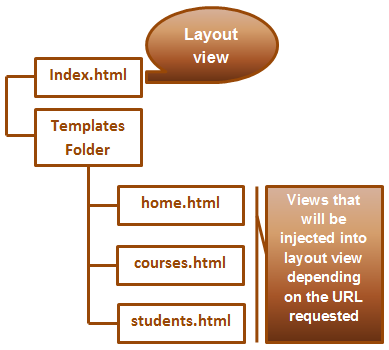
        </fieldset>

    </div>

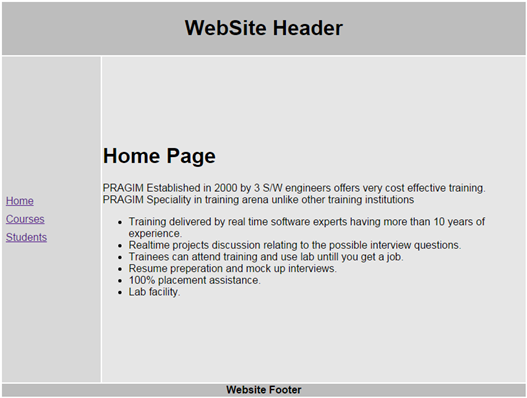
</body>

</html>

### AngularJS routing tutorial

In general, as the application becomes complex you will have more than one view in the application. Let's say you are building a single page application for a training institute and you have the following views  
**- Home**  
**- Courses**  
**- Students**   
  
We can take advantage of the Angular routing feature, to have a single layout page, and then inject and swap out different views depending on the URL the end user has requested.   
  
**So in our application we will have the following views**  
   
  
**index.html** is the layout view  
**home.html**, **courses.html** & **students.html** will be injected into the layout view(**index.html**) depending on the URL the end user has requested   
  
For example, if the user has requested http://localhost:51983/home, then home.html will be injected into the layout view (index.html). Similarly if the user has requested http://localhost:51983/courses, then courses.html will be injected into the layout view (index.html).   
  
**Preparing the angular application application to use routing :** The AngularJS Route module is present in a separate JavaScript file. You can either download it from AngularJs.org and use it in the application or you can use the CDN link.   
  
In our next video, we will discuss **creating the main layout view**.

### Angular layout template

The layout page for our example should be as shown below.   
   
  
**Here are the steps**  
**Step 1 :** Copy and paste the following HTML in the body section of the page.

<table style="font-family: Arial">

    <tr>

        <td colspan="2" class="header">

            <h1>

                WebSite Header

            </h1>

        </td>

    </tr>

    <tr>

        <td class="leftMenu">

            <a href="#/home">Home</a>

            <a href="#/courses">Courses</a>

            <a href="#/students">Students</a>

        </td>

        <td class="mainContent">

            <ng-view></ng-view>

        </td>

    </tr>

    <tr>

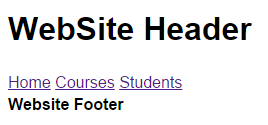
        <td colspan="2" class="footer">

            <b>Website Footer</b>

        </td>

    </tr>

</table>

**Please note :**  
1. The partial templates (home.html, courses.html & students.html) will be injected into the location where we have ng-view directive.   
2. For linking to partial templates we are using # symbol in the href attribute. This tells the browser not to navigate away from index.html. In a later video we will discuss how to get rid of the # symbol.   
  
At this point, if you navigate to index.html, the page looks as shown below. This is because we do not have the styles applied yet.   
   
  
**Step 2 :** Add a stylesheet to your project. Name it styles.css. Copy and paste the following. 

.header {

    width: 800px;

    height: 80px;

    text-align: center;

    background-color: #BDBDBD;

}

.footer {

    background-color: #BDBDBD;

    text-align: center;

}

.leftMenu {

    height: 500px;

    background-color: #D8D8D8;

    width: 150px;

}

.mainContent {

    height: 500px;

    background-color: #E6E6E6;

    width: 650px;

}

a{

    display:block;

    padding:5px

}

**Step 3 :** Finally add a reference to **styles.css** in index.html page. At this point the HTML in the layout page (index.html) should be as shown below. 

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="Demo">

<head>

    <title></title>

    <script src="Scripts/angular.min.js"></script>

    <script src="Scripts/angular-route.min.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body>

    <table style="font-family: Arial">

        <tr>

            <td colspan="2" class="header">

                <h1>

                    WebSite Header

                </h1>

            </td>

        </tr>

        <tr>

            <td class="leftMenu">

                <a href="#/home">Home</a>

                <a href="#/courses">Courses</a>

                <a href="#/students">Students</a>

            </td>

            <td class="mainContent">

                <ng-view></ng-view>

            </td>

        </tr>

        <tr>

            <td colspan="2" class="footer">

                <b>Website Footer</b>

            </td>

        </tr>

    </table>

</body>

</html>

In our next video, we will discuss **creating the partial templates** i.e **home.html**, **courses.html** and **students.html**

### Angularjs partial templates

One important thing to keep in mind is that, since we have all the surrounding HTML (i.e html, head, body etc) in the layout view (index.html), there is no need to include that same surrounding HTML again in the partial templates.   
  
All our **partial templates** are going to be in **Templates**folder. So first add Templates folder to the project's root folder.   
  
**home.html :** Right click on the Templates folder and add a new HTML file. Name it home.html. Copy and paste the following. The homeController will set the message property on the $scope object. We will discuss creating the homeController in a later video. 

<h1>{{message}}</h1>

<div>

    PRAGIM Established in 2000 by 3 S/W engineers offers very cost effective training. PRAGIM Speciality in training arena unlike other training institutions

</div>

<ul>

    <li>Training delivered by real time software experts having more than 10 years of experience.</li>

    <li>Realtime projects discussion relating to the possible interview questions.</li>

    <li>Trainees can attend training and use lab untill you get a job.</li>

    <li>Resume preperation and mock up interviews.</li>

    <li>100% placement assistance.</li>

    <li>Lab facility.</li>

</ul>

**courses.html :** The coursesController will set the courses property on the $scope object. We will discuss creating the coursesController in a later video. 

<h1>Courses we offer</h1>

<ul>

    <li ng-repeat="course in courses">

        {{course}}

    </li>

</ul>

**students.html :** The students data is going to come from a database table. So here are the steps    
  
**Step 1 :** Create the database table (tblStudents) and populate it with test data. 

Create table tblStudents

(

    Id int primary key identity,

    Name nvarchar(50),

    Gender nvarchar(10),

    City nvarchar(20)

)

Go

Insert into tblStudents values ('Mark', 'Male', 'London')

Insert into tblStudents values ('John', 'Male', 'Chennai')

Insert into tblStudents values ('Sara', 'Female', 'Sydney')

Insert into tblStudents values ('Tom', 'Male', 'New York')

Insert into tblStudents values ('Pam', 'Male', 'Los Angeles')

Insert into tblStudents values ('Catherine', 'Female', 'Chicago')

Insert into tblStudents values ('Mary', 'Female', 'Houston')

Insert into tblStudents values ('Mike', 'Male', 'Phoenix')

Insert into tblStudents values ('Rosie', 'Female', 'Manchester')

Insert into tblStudents values ('Sasha', 'Female', 'Hyderabad')

Go

**Step 2 :** Include the following configuration in web.config file. Notice we have a connection string to the database. We also have enabled HttpGet protocol for ASP.NET web services. 

<?xml version="1.0" encoding="utf-8"?>

<configuration>

  <connectionStrings>

    <add name="DBCS"

         connectionString="server=(local);database=SampleDB; integrated security=true"

         providerName="System.Data.SqlClient" />

  </connectionStrings>

  <system.web>

    <webServices>

      <protocols>

        <add name="HttpGet" />

      </protocols>

    </webServices>

  </system.web>

</configuration>

**Step 3 :** Add a class file to the project. Name it Student.cs. Copy and paste the following code.

namespace Demo

{

    public class Student

    {

        public int id { get; set; }

        public string name { get; set; }

        public string gender { get; set; }

        public string city { get; set; }

    }

}

**Step 4 :** Add an ASMX web service to the project. Name it StudentService.asmx. Copy and paste the following code.

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Web.Script.Serialization;

using System.Web.Services;

namespace Demo

{

    [WebService(Namespace = "http://tempuri.org/")]

    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

    [System.ComponentModel.ToolboxItem(false)]

    [System.Web.Script.Services.ScriptService]

    public class StudentService : System.Web.Services.WebService

    {

        [WebMethod]

        public void GetAllStudents()

        {

            List<Student> listStudents = new List<Student>();

            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

            using (SqlConnection con = new SqlConnection(cs))

            {

                SqlCommand cmd = new SqlCommand("Select \* from tblStudents", con);

                con.Open();

                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())

                {

                    Student student = new Student();

                    student.id = Convert.ToInt32(rdr["Id"]);

                    student.name = rdr["Name"].ToString();

                    student.gender = rdr["Gender"].ToString();

                    student.city = rdr["City"].ToString();

                    listStudents.Add(student);

                }

            }

            JavaScriptSerializer js = new JavaScriptSerializer();

            Context.Response.Write(js.Serialize(listStudents));

        }

    }

}

**Step 5 :** Right click on the Templates folder and add a new HTML file. Name it students.html. Copy and paste the following. The studentsController will set the students property on the $scope object. We will discuss creating the studentsController in a later video.

<h1>List of Students</h1>

<ul>

    <li ng-repeat="student in students">

        {{student.name}}

    </li>

</ul>

In our next video we will discuss **configuring angularjs routes** and creating the controllers i.e homeController, coursesController and studentsController.

### Angularjs route configuration

Right click on the **Scripts**folder and add a **new JavaScript file**. Name it **script.js**. Copy and paste the following code. 

/// <reference path="angular.min.js" />

var app = angular

            .module("Demo", ["ngRoute"])

            .config(function ($routeProvider) {

                $routeProvider

                    .when("/home", {

                        templateUrl: "Templates/home.html",

                        controller: "homeController"

                    })

                    .when("/courses", {

                        templateUrl: "Templates/courses.html",

                        controller: "coursesController"

                    })

                    .when("/students", {

                        templateUrl: "Templates/students.html",

                        controller: "studentsController"

                    })

            })

            .controller("homeController", function ($scope) {

                $scope.message = "Home Page";

            })

            .controller("coursesController", function ($scope) {

                $scope.courses = ["C#", "VB.NET", "ASP.NET", "SQL Server"];

            })

             .controller("studentsController", function ($scope, $http) {

                 $http.get("StudentService.asmx/GetAllStudents")

                                        .then(function (response) {

                                            $scope.students = response.data;

                                        })

             })

**2 Changes to index.html**   
  
1. Add a reference to the script.js file in the layout template i.e index.html.

<script src="Scripts/script.js"></script>

2. Set **ng-app="Demo"** on the root html element

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="Demo">

At this point, depending on the URL, the respective partial template will be injected into the layout template in the location where we have ng-view directive. For example if you have index.html#/home, then home.html is injected into index.html. Similarly if you are on index.html#/courses, then course.html is injected into index.html.

### Remove # from URL AngularJS

There are 4 simple steps to **remove # from URLs in Angular**.   
  
**Step 1 :**Enable html5mode routing. To do this inject $locationProvider into config() function in script.js and call html5Mode() method passing true as the argument value. With this change the config function will now look as shown below. 

.config(function ($routeProvider, $locationProvider) {

    $routeProvider

        .when("/home", {

            templateUrl: "Templates/home.html",

            controller: "homeController"

        })

        .when("/courses", {

            templateUrl: "Templates/courses.html",

            controller: "coursesController"

        })

        .when("/students", {

            templateUrl: "Templates/students.html",

            controller: "studentsController"

        })

    $locationProvider.html5Mode(true);

})

**Step 2 :** In index.html, remove # symbols from all the links. The links in index.html should look as shown below. 

<a href="home">Home</a>

<a href="courses">Courses</a>

<a href="students">Students</a>

**Step 3 :** Include the following URL rewrite rule in web.config. This rewrite rule, rewrites all urls to index.html, except if the request is for a file, or a directory or a Web API request. 

<system.webServer>

  <rewrite>

    <rules>

      <rule name="RewriteRules" stopProcessing="true">

        <match url=".\*" />

        <conditions logicalGrouping="MatchAll">

          <add input="{REQUEST\_FILENAME}" matchType="IsFile" negate="true" />

          <add input="{REQUEST\_FILENAME}" matchType="IsDirectory" negate="true" />

          <add input="{REQUEST\_URI}" pattern="^/(api)" negate="true" />

        </conditions>

        <action type="Rewrite" url="/index.html" />

      </rule>

    </rules>

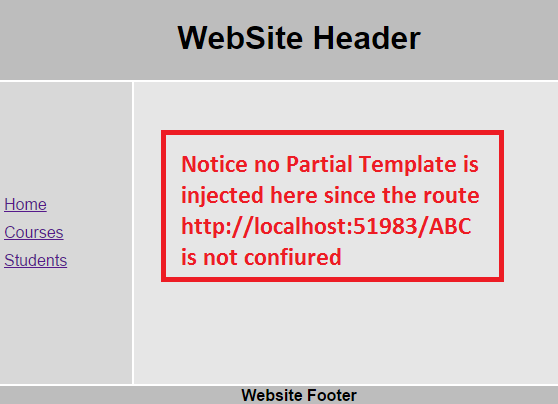
  </rewrite>

</system.webServer>

**Step 4 :** Set the base href to the location of your single page application. In the head section of index.html include the following line.

<base href="/" />

### AngularJS default route

We will continue with the example that we worked with in [Part 27](http://csharp-video-tutorials.blogspot.com/2016/02/remove-from-url-angularjs.html).    
  
At the moment the problem is that, if you try to navigate to a route that is not configured, you will see only the layout page without any partial template injected into it.   
  
For example if you navigate to http://localhost:51983/ABC, since ABC is not a configured route you will see the layout page (index.html) as shown below.   
  
   
  
You will also have this same problem if you navigate to the root of the site i.e http://localhost:51983. The reason angular is displaying the empty layout template in both these cases, is because it does not know what partial template to inject. We want angular to redirect to default route if the user is trying to navigate to a route that is not configured.   
  
**How to configure the default route in Angular :**Well that is straight forward. All you need is the following line in config() function in script.js file   
  
.otherwise({  
    redirectTo: "/home"  
})   
  
With the above change the code in config() function should be as shown below. 

.config(function ($routeProvider, $locationProvider) {

    $routeProvider

        .when("/home", {

            templateUrl: "Templates/home.html",

            controller: "homeController"

        })

        .when("/courses", {

            templateUrl: "Templates/courses.html",

            controller: "coursesController"

        })

        .when("/students", {

            templateUrl: "Templates/students.html",

            controller: "studentsController"

        })

        .otherwise({

            redirectTo: "/home"

        })

    $locationProvider.html5Mode(true);

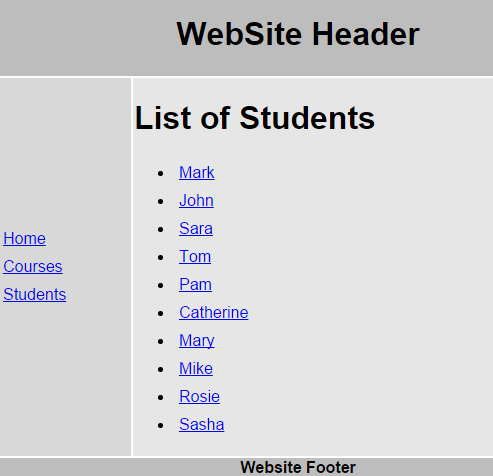
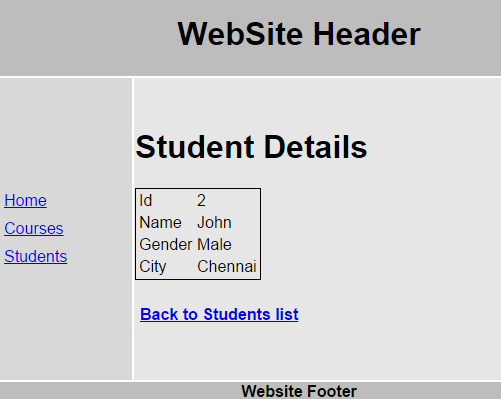
})

With this change if the user tries to navigate to a route that is not configured (http://localhost:51983/ABC) or just to the rooot URL (http://localhost:51983), the user will be automatically redirected to http://localhost:51983/home.

### AngularJS intellisense in visual studio

In the previous videos if you have noticed as we were typing the angular code in Script.js file we were getting some intellisense but it definitely is not good though. For example when we type $routeProvider and then a . (DOT) we were not getting intellisense.   
  
**3 simple steps to get better intellisense for angular in visual studio**   
  
**Step 1 :**Download AngularJS extension for Visual Studio from the following link. The link displays the script in a web page.   
<https://raw.githubusercontent.com/jmbledsoe/angularjs-visualstudio-intellisense/master/src/Scripts/angular.intellisense.js>  
  
**Step 2 :** Copy and paste the script in a new notepad. Name it angular.intellisense.js and save it to the following folder on your computer  
C:\Program Files (x86)\Microsoft Visual Studio 12.0\JavaScript\References   
  
**Step 3 :** Now drag and drop the following 2 files from Scripts folder onto script.js file  
angular.min.js  
angular-route.min.js  
  
Visual Studio will automatically add references to the above 2 files in script.js  
/// <reference path="angular.min.js" />  
/// <reference path="angular-route.min.js" />  
  
At this point when we type $routeProvider and then a . (DOT) in the script.js file, we get great intellisense.

### AngularJS routeparams example

**Here is what we want to do :** When we navigate to /students, the list of student names must be displayed as hyperlinks.   
  
   
  
When we click on any student name, the respective student details should be displayed as shown below. When we click on "Back to Students list" it should take us back to students list page.   
  
   
  
**Here are the steps to achieve this**  
  
**Step 1 :** The first step is to add a Web Method to our ASP.NET web service, which will retrieve student by their id. Add the following Web Method in StudentService.asmx.cs

[WebMethod]

public void GetStudent(int id)

{

    Student student = new Student();

    string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

    using (SqlConnection con = new SqlConnection(cs))

    {

        SqlCommand cmd = new

            SqlCommand("Select \* from tblStudents where id = @id", con);

        SqlParameter param = new SqlParameter()

        {

            ParameterName = "@id",

            Value = id

        };

        cmd.Parameters.Add(param);

        con.Open();

        SqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())

        {

            student.id = Convert.ToInt32(rdr["Id"]);

            student.name = rdr["Name"].ToString();

            student.gender = rdr["Gender"].ToString();

            student.city = rdr["City"].ToString();

        }

    }

    JavaScriptSerializer js = new JavaScriptSerializer();

    Context.Response.Write(js.Serialize(student));

}

After adding the above method, the complete code in **StudentService.asmx.cs** should be as shown below.

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Web.Script.Serialization;

using System.Web.Services;

namespace Demo

{

    [WebService(Namespace = "http://tempuri.org/")]

    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

    [System.ComponentModel.ToolboxItem(false)]

    [System.Web.Script.Services.ScriptService]

    public class StudentService : System.Web.Services.WebService

    {

        [WebMethod]

        public void GetAllStudents()

        {

            List<Student> listStudents = new List<Student>();

            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

            using (SqlConnection con = new SqlConnection(cs))

            {

                SqlCommand cmd = new SqlCommand("Select \* from tblStudents", con);

                con.Open();

                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())

                {

                    Student student = new Student();

                    student.id = Convert.ToInt32(rdr["Id"]);

                    student.name = rdr["Name"].ToString();

                    student.gender = rdr["Gender"].ToString();

                    student.city = rdr["City"].ToString();

                    listStudents.Add(student);

                }

            }

            JavaScriptSerializer js = new JavaScriptSerializer();

            Context.Response.Write(js.Serialize(listStudents));

        }

        [WebMethod]

        public void GetStudent(int id)

        {

            Student student = new Student();

            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

            using (SqlConnection con = new SqlConnection(cs))

            {

                SqlCommand cmd = new

                    SqlCommand("Select \* from tblStudents where id = @id", con);

                SqlParameter param = new SqlParameter()

                {

                    ParameterName = "@id",

                    Value = id

                };

                cmd.Parameters.Add(param);

                con.Open();

                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())

                {

                    student.id = Convert.ToInt32(rdr["Id"]);

                    student.name = rdr["Name"].ToString();

                    student.gender = rdr["Gender"].ToString();

                    student.city = rdr["City"].ToString();

                }

            }

            JavaScriptSerializer js = new JavaScriptSerializer();

            Context.Response.Write(js.Serialize(student));

        }

    }

}

**Step 2 :** Change the HTML in students.html partial template as shown below. Notice student name is now wrapped in an anchor tag. This will display student name as a hyperlink. If you click on a student name with id = 1, then we will be redirected to /students/1

<h1>List of Students</h1>

<ul>

    <li ng-repeat="student in students">

        <a href="students/{{student.id}}">

            {{student.name}}

        </a>

    </li>

</ul>

**Step 3 :** Now let's create an angularjs route with parameters. Add the following route in **script.js file**. Our next step is to create **studentDetails.html**partial template and **studentDetailsController**.

.when("/students/:id", {

    templateUrl: "Templates/studentDetails.html",

    controller: "studentDetailsController"

})

With the above change, the code in **script.js** should now look as shown below. Please pay attention to the code highlighted in yellow.

/// <reference path="angular.min.js" />

/// <reference path="angular-route.min.js" />

var app = angular

            .module("Demo", ["ngRoute"])

            .config(function ($routeProvider, $locationProvider) {

                $routeProvider

                    .when("/home", {

                        templateUrl: "Templates/home.html",

                        controller: "homeController"

                    })

                    .when("/courses", {

                        templateUrl: "Templates/courses.html",

                        controller: "coursesController"

                    })

                    .when("/students", {

                        templateUrl: "Templates/students.html",

                        controller: "studentsController"

                    })

                    .when("/students/:id", {

                        templateUrl: "Templates/studentDetails.html",

                        controller: "studentDetailsController"

                    })

                    .otherwise({

                        redirectTo: "/home"

                    })

                $locationProvider.html5Mode(true);

            })

            .controller("homeController", function ($scope) {

                $scope.message = "Home Page";

            })

            .controller("coursesController", function ($scope) {

                $scope.courses = ["C#", "VB.NET", "ASP.NET", "SQL Server"];

            })

             .controller("studentsController", function ($scope, $http) {

                 $http.get("StudentService.asmx/GetAllStudents")

                                        .then(function (response) {

                                            $scope.students = response.data;

                                        })

             })

**Step 4 :**Right click on Templates folder in solution explorer and add a new HTMLpage. Name it **studentDetails.html**. Copy and paste the following HTML. 

<h1>Student Details</h1>

<table style="border:1px solid black">

    <tr>

        <td>Id</td>

        <td>{{student.id}}</td>

    </tr>

    <tr>

        <td>Name</td>

        <td>{{student.name}}</td>

    </tr>

    <tr>

        <td>Gender</td>

        <td>{{student.gender}}</td>

    </tr>

    <tr>

        <td>City</td>

        <td>{{student.city}}</td>

    </tr>

</table>

<h4><a href="students">Back to Students list</a></h4>

**Step 5 :** Add **studentDetailsController** in **script.js**which calls GetStudent web method and returns the requested student data.

.controller("studentDetailsController", function ($scope, $http, $routeParams) {

    $http({

        url: "StudentService.asmx/GetStudent",

        method: "get",

        params: { id: $routeParams.id }

    }).then(function (response) {

        $scope.student = response.data;

    })

})

With the above change, the code in script.js should now look as shown below. Please pay attention to the code highlighted in yellow.

/// <reference path="angular.min.js" />

/// <reference path="angular-route.min.js" />

var app = angular

            .module("Demo", ["ngRoute"])

            .config(function ($routeProvider, $locationProvider) {

                $routeProvider

                    .when("/home", {

                        templateUrl: "Templates/home.html",

                        controller: "homeController"

                    })

                    .when("/courses", {

                        templateUrl: "Templates/courses.html",

                        controller: "coursesController"

                    })

                    .when("/students", {

                        templateUrl: "Templates/students.html",

                        controller: "studentsController"

                    })

                    .when("/students/:id", {

                        templateUrl: "Templates/studentDetails.html",

                        controller: "studentDetailsController"

                    })

                    .otherwise({

                        redirectTo: "/home"

                    })

                $locationProvider.html5Mode(true);

            })

            .controller("homeController", function ($scope) {

                $scope.message = "Home Page";

            })

            .controller("coursesController", function ($scope) {

                $scope.courses = ["C#", "VB.NET", "ASP.NET", "SQL Server"];

            })

             .controller("studentsController", function ($scope, $http) {

                 $http.get("StudentService.asmx/GetAllStudents")

                                        .then(function (response) {

                                            $scope.students = response.data;

                                        })

             })

            .controller("studentDetailsController", function ($scope, $http, $routeParams) {

                $http({

                    url: "StudentService.asmx/GetStudent",

                    method: "get",

                    params: { id: $routeParams.id }

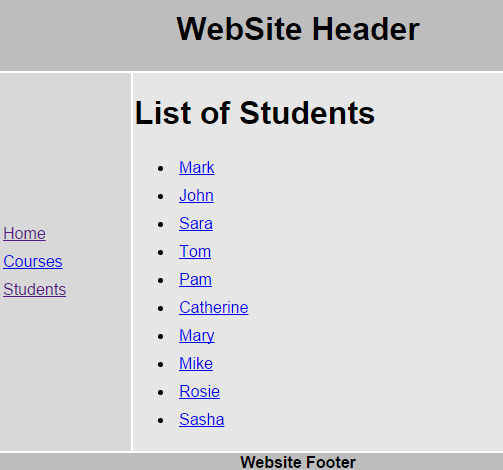
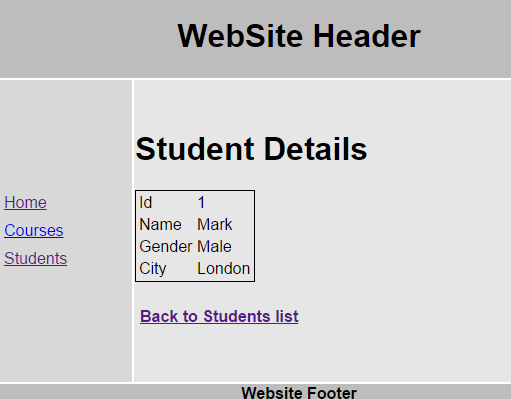
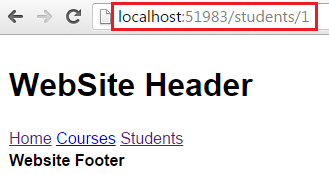
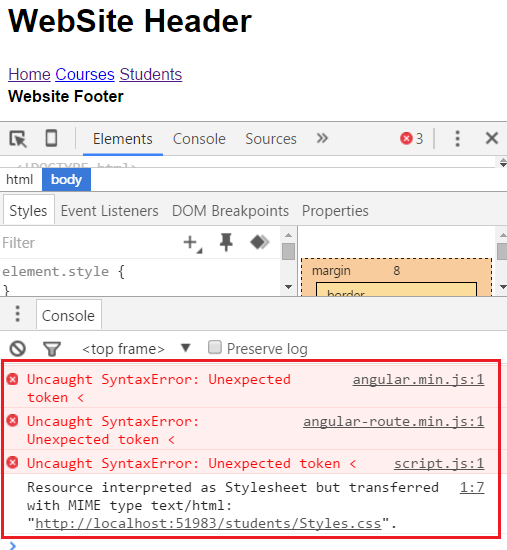
                }).then(function (response) {

                    $scope.student = response.data;

                })

            })

### AngularJS page refresh problem

**What is the issue :** When you navigate to http://localhost/students, you will see list of students as shown below   
  
   
  
Click on any student name. For example when you click on Mark, you will see mark details and the URL in the address bar is http://localhost/students/1  
  
  
   
  
At this point if you refresh the page by pressing **CTRL + F5** or **CTRL + R**, the student details disappear and the page will be renedered as shown below.   
  
   
  
To see the errors, launch the **Browser Developer Tools** by pressing **F12**.   
  
   
  
To fix this issue all you have to do is place the **<base href="/">** element just below the **<title>** element in the head section of index.html page.

### AngularJS controller as syntax

So far in this video series we have been using $scope to expose the members from the controller to the view.

app.controller("mainController", function ($scope) {

    $scope.message = "Hello Angular";

});

In the example above we are attaching message property to $scope, which is automatically available in the view.

<h1 ng-controller="mainController">**{{**message**}}**</h1>

Another way that is available to expose the members from the controller to the view, is by using **CONTROLLER AS** syntax. With this syntax, there is no need to inject $scopeobject in to the controller function, instead you simply use this keyword as shown below.

app.controller("mainController", function () {

    this.message = "Hello Angular";

});

In the view, you use, **CONTROLLER AS** **syntax**as shown below.

<h1 ng-controller="mainController as main">**{{**main.message**}}**</h1>

Now, let us see how we can use this **CONTROLLER AS syntax** in the example that we worked with in [Part 31](http://csharp-video-tutorials.blogspot.com/2016/03/angularjs-page-refresh-problem.html).  
  
**Code changes in script.js. Notice the changes highlighted in yellow.**  
1. In the when() function, notice that we are using CONTROLLER AS syntax  
2. With in each controller() function we are using this keyword to set the properties that we want to make available in the view  
3. Notice in studentsController and studentDetailsController we are assigning this keyword to variable vm. vm stands for ViewModel. You can give it any meaningful name you want.  
4. If you use this keyword in then() function as shown below, you would not get the result you expect. That's because 'this' keyword points to the window object when the control comes to then() function. 

.controller("studentsController", function ($http) {

    $http.get("StudentService.asmx/GetAllStudents")

                        .then(function (response) {

                            this.students = response.data;

                        })

})

At this point we also need to modify all our partial templates  
  
**Changes in home.html :** Use **homeCtrl**object to retrieve the message property that the homeController has set.

<h1>{{homeCtrl.message}}</h1>

<div>

    PRAGIM Established in 2000 by 3 S/W engineers offers very cost effective training. PRAGIM Speciality in training arena unlike other training institutions

</div>

<ul>

    <li>Training delivered by real time software experts having more than 10 years of experience.</li>

    <li>Realtime projects discussion relating to the possible interview questions.</li>

    <li>Trainees can attend training and use lab untill you get a job.</li>

    <li>Resume preperation and mock up interviews.</li>

    <li>100% placement assistance.</li>

    <li>Lab facility.</li>

</ul>

**Changes in courses.html :** Use coursesCtrl object to retrieve the courses property that the coursesController has set.

<h1>Courses we offer</h1>

<ul>

    <li ng-repeat="course in coursesCtrl.courses">

        {{course}}

    </li>

</ul>

**Changes in students.html :**Use studentsCtrl object to retrieve the students property that the studentsController has set.

<h1>List of Students</h1>

<ul>

    <li ng-repeat="student in studentsCtrl.students">

        <a href="students/{{student.id}}">

            {{student.name}}

        </a>

    </li>

</ul>

**Changes in studentDetails.html :**Use studentDetailsCtrl object to retrieve the student property that the studentDetailsController has set.

<h1>Student Details</h1>

<table style="border:1px solid black">

    <tr>

        <td>Id</td>

        <td>{{studentDetailsCtrl.student.id}}</td>

    </tr>

    <tr>

        <td>Name</td>

        <td>{{studentDetailsCtrl.student.name}}</td>

    </tr>

    <tr>

        <td>Gender</td>

        <td>{{studentDetailsCtrl.student.gender}}</td>

    </tr>

    <tr>

        <td>City</td>

        <td>{{studentDetailsCtrl.student.city}}</td>

    </tr>

</table>

<h4><a href="students">Back to Students list</a></h4>

You can also use **CONTROLLER AS**syntax when defining routes as shown below

var app = angular

            .module("Demo", ["ngRoute"])

            .config(function ($routeProvider, $locationProvider) {

                $routeProvider

                    .when("/home", {

                        templateUrl: "Templates/home.html",

                        controller: "homeController",

                        controllerAs: "homeCtrl"

                    })

                    .when("/courses", {

                        templateUrl: "Templates/courses.html",

                        controller: "coursesController as coursesCtrl",

                        controllerAs: "coursesCtrl"

                    })

                    .when("/students", {

                        templateUrl: "Templates/students.html",

                        controller: "studentsController as studentsCtrl",

                        controllerAs: "studentsCtrl"

                    })

                    .when("/students/:id", {

                        templateUrl: "Templates/studentDetails.html",

                        controller: "studentDetailsController as studentDetailsCtrl",

                        controllerAs: "studentDetailsCtrl"

                    })

                    .otherwise({

                        redirectTo: "/home"

                    })

                $locationProvider.html5Mode(true);

            })

In our next video we will discuss, **how the CONTROLLER AS syntax can make our code more readable** as opposed to using $scope when working with nested scopes.

### Angular nested scopes and controller as syntax

**Working with nested scopes using $scope object :** The following code creates 3 controllers  - countryController, stateController, and cityController. All of these have set name property on the $scope object. 

var app = angular

            .module("Demo", [])

            .controller("countryController", function ($scope) {

                $scope.name = "India";

            })

            .controller("stateController", function ($scope) {

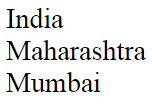
                $scope.name = "Maharashtra";

            })

            .controller("cityController", function ($scope) {

                $scope.name = "Mumbai";

            });

Now we want to display Country, State and City names as shown below.   
  
   
  
To get the output as shown above, we will have the following HTML in our view. name property retrieves the correct value as expected. However, the code is bit confusing. 

<div ng-controller="countryController">

    {{name}}

    <div ng-controller="stateController">

        {{name}}

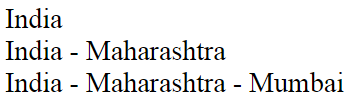
        <div ng-controller="cityController">

            {{name}}

        </div>

    </div>

</div>

Now let us say we want to display the names as shown below.   
  
   
  
To achieve this modify the HTML in the view as shown below. Notice we are using **$parent** to get the name property value of the immediate parent controller. To get the name property value of the grand parent, we are using **$parent.$parent**. This can get very confusing if you have many nested controllers and as a result the code gets less readable.

<div ng-controller="countryController">

    {{name}}

    <div ng-controller="stateController">

        {{$parent.name}} - {{name}}

        <div ng-controller="cityController">

            {{$parent.$parent.name}} - {{$parent.name}} - {{name}}

        </div>

    </div>

</div>

Let us see how things change when we use **CONTROLLER AS syntax**. First change the angular code to support CONTROLLER AS syntax. Notice we are not using $scope anymore with in our controllers, instead, we are using "this" keyowrd.

var app = angular

            .module("Demo", [])

            .controller("countryController", function () {

                this.name = "India";

            })

            .controller("stateController", function () {

                this.name = "Maharashtra";

            })

            .controller("cityController", function () {

                this.name = "Mumbai";

            });

With in the view, use CONTROLLER AS syntax. With this change, we are able to use the respective controller object and retrieve name property value. Now there is no need to juggle with $parent property. No matter how deep you are in the nested hierarchy, you can very easily get any controller object name property value. The code is also much readable now.

<div ng-controller="countryController as countryCtrl">

    {{countryCtrl.name}}

    <div ng-controller="stateController as stateCtrl">

        {{countryCtrl.name}} - {{stateCtrl.name}}

        <div ng-controller="cityController as cityCtrl">

            {{countryCtrl.name}} - {{stateCtrl.name}} - {{cityCtrl.name}}

        </div>

    </div>

</div>

### AngularJS controller as vs scope

There are 2 ways to expose the members from the controller to the view - **$scope and CONTROLLER AS.** The obvious question that comes to our mind at this point is - Why do we have 2 ways of doing the same thing. Which one to use over the other and what are the differences.   
  
**Here are the differences**  
1. CONTROLLER AS syntax is new and is officially released in 1.2.0. $scope is the old technique and is available since the initial version of angular is released.  
  
2. You can use either one of thes techniques. Both have their own uses. For example, CONTROLLER AS syntax makes your code more readable when working with nested scopes. We discussed this in our previous video.  
  
3. If you want to use $scope it has to be injected into controller function, where as with CONTROLLER AS syntax there is no need for such injection, unless you need it for something else.  
  
Which one to use depends on your personal preference. Some prefer using $scope while others prefer using CONTROLLER AS syntax. One important thing to keep in mind is that, though you are using CONTROLLER AS syntax, behind the scenes angular is still using $scope. Angular takes the controller instance and adds it as a reference on the scope. 

<div ng-controller="cityController as cityCtrl">

    {{cityCtrl.name}}

</div>

In the above example since we are using CONTROLLER AS syntax, angular takes cityCtrl which is the instance of cityController and adds it as a reference on the scope. So in the binding expression, you can read it as $scope.cityCtrl.name

### AngularJS caseInsensitiveMatch and Inline Templates

Let us understand these 2 features with examples.   
  
**caseInsensitiveMatch :** The routes that are configured using config function are case sensitive by default. Consider the route below. Notice the route (/home) is lower case. 

$routeProvider

    .when("/home", {

        templateUrl: "Templates/home.html",

        controller: "homeController",

        controllerAs: "homeCtrl",

    })

If we type the following URL in the browser, we will see home.html as expected.  
http://localhost:51983/home  
  
If you type the following URL, the you will see a blank layout page. This is because, **by default routes are case-sensitive**  
http://localhost:51983/HOME  
  
To make the route case-insensitive set **caseInsensitiveMatch**property to true as shown below.

$routeProvider

    .when("/home", {

        templateUrl: "Templates/home.html",

        controller: "homeController",

        controllerAs: "homeCtrl",

        caseInsensitiveMatch: true

    })

**To make all routes case-insensitive set caseInsensitiveMatch property on $routeProvider** as shown below.  
  
$routeProvider.caseInsensitiveMatch = true;   
  
**Inline Templates :** The view content for the route (/home), is coming from a separate html file (home.html)

$routeProvider

    .when("/home", {

        templateUrl: "Templates/home.html",

        controller: "homeController",

        controllerAs: "homeCtrl",

    })

Should the view content always come from a separate html file. Not necessarily. You can also use an inline template. To use an inline template use template property as shown below.

$routeProvider

    .when("/home", {

        template: "<h1>Inline Template in action</h1>",

        controller: "homeController",

        controllerAs: "homeCtrl"

    })

At this point, when you navigate to http://localhost:51983/home, you should see Inline Template in action.

### AngularJS route reload

We will continue with the same example that we have been working with in the previous videos. When we navigate to http://localhost/students we see the list of students. Notice that when you click on the same students link, nothing happens. This means if we insert a new record in the database table and issue a request to the same route, you may not see the new records.   
  
One of the ways to see the new data is by clicking on the browser refresh button. The downside of this is that the entire app is reloaded. This means all the resources required to run your app will be reloaded. You can see all the resource requests made on the network tab in the browser developer tools.   
  
The other way is to reload just the current route. Here are the steps.   
  
**Step 1 :** Modify the studentsController function in script.js

.controller("studentsController", function ($http, $route) {

    var vm = this;

    vm.reloadData = function () {

        $route.reload();

    }

    $http.get("StudentService.asmx/GetAllStudents")

                .then(function (response) {

                    vm.students = response.data;

                })

})

**Please note :**  
1. $route service in injected in the controller function  
2. reloadData() function is attached to the view model (vm) which will be available for the view to call. This method simply calls reload() method of the route service.   
  
**Step 2 :**Modify the partial template (students.html). Please note that we have included a button which calls **reloadData()** method when clicked. 

<h1>List of Students</h1>

<ul>

    <li ng-repeat="student in studentsCtrl.students">

        <a href="students/{{student.id}}">

            {{student.name}}

        </a>

    </li>

</ul>

<button ng-click="studentsCtrl.reloadData()">Reload</button>

**At this point**  
1. Run the app  
2. Navigate to http://localhost/students. You should see list of students.  
3. Insert a new record in the database table  
4. Open browser developer tools  
5. Click the Reload button   
  
**There are 2 things to notice here**  
1. The newly added record will be shown on the view  
2. Only the resources required to reload the current route are requested from the server

### Difference between $scope and $rootScope

Let us understand this with an example.  
  
**Controller Code :** We have 2 controllers (redColourController & greenColourController). redColourController has set redColour property on $scope and rootScopeColour on $rootScope. This means redColour property cannot be used outside the redColourController, where as rootScopeColour that is set on $rootScope can be used anywhere. greenColourController has set greenColour property on $scope. This means greenColour property cannot be used outside the greenColourController

var app = angular

            .module("Demo", [])

            .controller("redColourController", function ($scope, $rootScope) {

                $rootScope.rootScopeColour = "I am Root Scope Colour";

                $scope.redColour = "I am Red Colour";

            })

            .controller("greenColourController", function ($scope) {

                $scope.greenColour = "I am Green Colour";

            })

**View HTML :**

<div ng-controller="redColourController">

    Root Scope Colour : {{rootScopeColour}} <br />

    Red Colour Controller : {{redColour}} <br />

    Green Colour Controller :

    <span style="color:red" ng-show="greenColour == undefined">

        greenColour is undefined

    </span>

</div>

<br />

<div ng-controller="greenColourController">

    Root Scope Colour : {{rootScopeColour}} <br />

    Green Colour Controller : {{greenColour}} <br />

    Red Colour Controller :

    <span style="color:red" ng-show="redColour == undefined">

        redColour is undefined

    </span>

</div>

**Output :** From the output it is clear that the rootScopeColour property that is set on $rootScope is available for both the controllers (redColourController & greenColourController). Where as redColour property set on $scope is available only for redColourController and not for greenColourController. Similarly, greenColour property set $scope is available only for greenColourController and not redColourController. Hope this example has made the difference between $rootScope and $scope clear.   
  
