

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 freely used, changed, and shared by anyone.

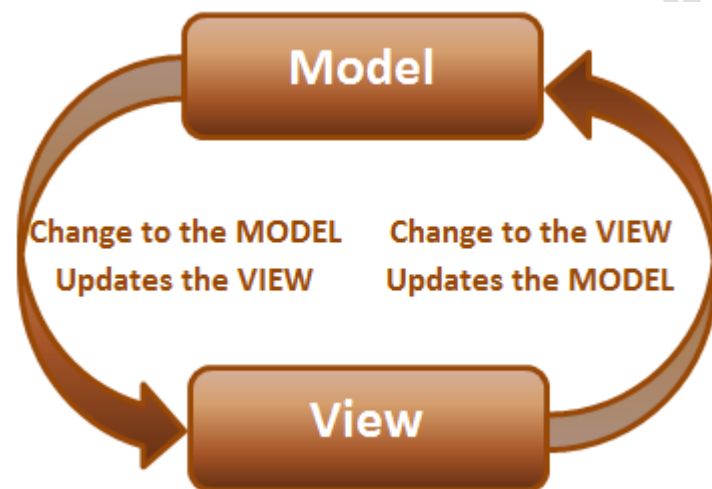
AngularJS is an excellent framework for building both Single Page Applications (SPA) and Line of Business Applications. Many companies are using Angular today, and there are many public facing web sites that are built with angular.

There is a website, <https://www.madewithangular.com> 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 later.

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 behavior of DOM elements using **directives** and the flexibility

that **angular filters** provide.

We will discuss directives, filters, modules, routes etc with examples later.

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

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

1. You can download the angular script file
2. Various resources to learn angular - Here you will find videos, Free courses, Tutorials and Case Studies. You will also find API reference which is extremely 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.

```
<!doctype html>
<html ng-app='myModule'>
<head>
    <script src="scripts/angular.js"></script>
</head>
<body>
```

```
<div>
    10+20 = {{ 10 + 20 }}
</div>
<div>
    40+10 ={{ 40+10}}
</div>
</body>
</html>
```

Output:

10+20 = 30
40+10 = 50

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.

```
<!doctype html>
<html >
<head>
    <script src="scripts/angular.js"></script>
</head>
<body>
    <div ng-app='myModule'>
        10+20 = {{ 10 + 20 }}
    </div>
    <div>
        40+10 ={{ 40+10}}
    </div>
</body>
</html>
```

Output:

10+20 = 30

40+10 = {{40+10}}

All the following are valid expressions in angular

{{ 1 == 1 }} - Evaluates to true

{{ { name: 'John', age : '25' }.name }} - Returns the name property value

{{ ['Ben', 'David', 'Chris'][2] }} - Returns the 2nd element from the array

What is a module in AngularJS

A module is a container for different parts of your application i.e controllers, services, directives, filters, etc.

Why is a module required

You can think of a module as a Main() method in other types of applications 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 later.

How to create a module?

Creating a module in angular is straight 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 later. 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. Within 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";  
});
```

Now 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 to it.

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: 'Ben',  
        lastName: 'Stokes',  
        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

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: 'Ben',
      lastName: 'Stokes',
      gender: 'Male'
    };
    $scope.employee = employee;
  });
```

AngularJS ng-src directive

We will discuss the use of **ng-src directive in AngularJS**

Let us understand this with an example : We want to display the name of the country, capital and it's flag.

Country : United States of America

Capital : Washington, D.C.



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>
            
        </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.

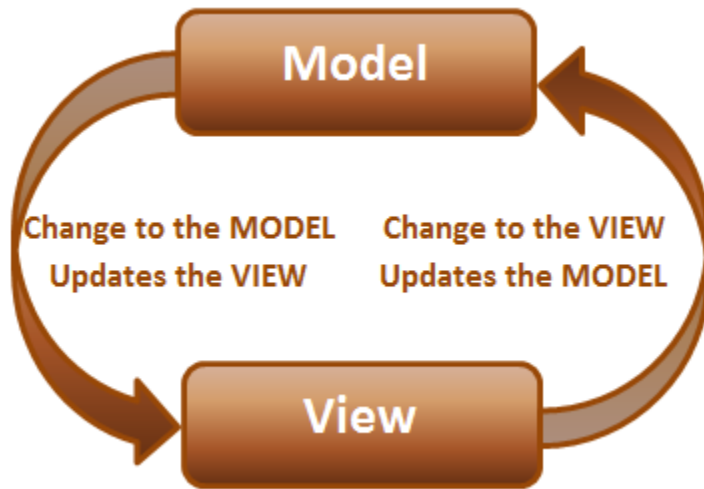
Let's now understand the reason for the 404 error

As soon as the DOM is parsed, an attempt is made 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

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 >
<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>
<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: "Stokes",
      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>
<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

we will discuss

1. ng-repeat directive in Angular
2. Nesting ng-repeat with an example

ng-repeat is similar to foreach loop in PHP.

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

```
var employees = [
  { firstName: "Ben", lastName: "Stokes", gender: "Male", salary:55000 },
  { firstName: "David", lastName: "Warner", gender: "Male", salary:65000 },
  { firstName: "Chris", lastName: "Morris", gender: "Male", salary:40000 },
  { firstName: "Alex", lastName: "Blackwell", gender: "Female", salary:35000 },
  { firstName: "Amy", lastName: "Jones", gender: "Female", salary:38000 }
]
```

First Name	Last Name	Gander	Salary
Ben	Stokes	Male	55000
David	Warner	Male	65000
Chris	Morris	Male	40000
Alex	Blackwell	Female	35000
Amy	Jones	Female	38000

This can be achieved very easily using **ng-repeat directive**

Script.js : The controller 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: "Stokes", gender: "Male", salary:55000 },
            { firstName: "David", lastName: "Warner", gender: "Male", salary:65000 },
            { firstName: "Chris", lastName: "Morris", gender: "Male", salary:40000 },
            { firstName: "Alex", lastName: "Blackwell", gender: "Female", salary:35000 },
            { firstName: "Amy", lastName: "Jones", gender: "Female", salary:38000 },
        ];

        $scope.employees = employees;
    });
```

HtmlPage1.html : In the view, we are using ng-repeat directive which loops through each employee in employees array. For each employee, we 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>
<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>First Name</th>
                    <th>Last Name</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.

- UK
 - London
 - Birmingham
 - Manchester
- USA
 - Los Angeles
 - Chicago
 - Houston
- India
 - Hyderabad
 - Chennai
 - Mumbai

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 through each country in the model. The inner ng-repeat directive loops through each city of a given country.

```
<!DOCTYPE html>
<html>
<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 retrieve the index of the elements from a nested collection

```
<!DOCTYPE html>
<html>
<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

we will discuss how to handle events in AngularJS.

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

Name	Likes	Dislikes	Like/Dislike	
PHP	16	1	Like	Dislike
Android	3	1	Like	Dislike
SQL	16	2	Like	Dislike
AngularJS	25	0	Like	Dislike

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: "PHP", likes: 0, dislikes: 0 },
            { name: "Android", 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 corresponding 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>
<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

we will discuss **filters in AngularJS**

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

Rows to display :

Name	Date of Birth	Gender	Salary (number filter)	Salary (currency filter)
BEN	23/11/1980	Male	55,000.79	£55,000.8
DAVID	05/05/1970	Female	68,000.00	£68,000.0
CHRIS	15/08/1974	Male	57,000.00	£57,000.0
ALEX	27/10/1979	Female	53,000.00	£53,000.0
AMY	30/12/1983	Male	60,000.00	£60,000.0

Script.
js

var ap
p =

angular

```
.module("myModule", [])
.controller("myController", function ($scope) {

    var employees = [
        {
            name: "Ben", dateOfBirth: new Date("November 23, 1980"),
            gender: "Male", salary: 55000.788
        },
        {
            name: "David", dateOfBirth: new Date("May 05, 1970"),
            gender: "Male", salary: 68000
        },
        {
            name: "Chris", dateOfBirth: new Date("August 15, 1974"),
            gender: "Male", salary: 57000
        },
        {
            name: "Alex", dateOfBirth: new Date("October 27, 1979"),
            gender: "Female", salary: 53000
        },
        {
            name: "Amy", dateOfBirth: new Date("December 30, 1983"),
            gender: "Female", salary: 60000
        }
    ];

    $scope.employees = employees;
    $scope.rowCount = 3;
});
```

HtmlPage1.html

```
<!DOCTYPE html>
<html>
<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>
                </tr>
            </thead>
        </table>
    </div>
</body>
</html>
```

```
<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;
}
```

Sorting data in AngularJS

we will discuss **how to implement sorting in AngularJS**.

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.

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: "David", dateOfBirth: new Date("May 05, 1970"),
                gender: "Male", salary: 68000
            },
            {
                name: "Chris", dateOfBirth: new Date("August 15, 1974"),
                gender: "Male", salary: 57000
            },
            {
                name: "Alex", dateOfBirth: new Date("October 27, 1979"),
                gender: "Female", salary: 53000
            },
            {
                name: "Amy", dateOfBirth: new Date("December 30, 1983"),
                gender: "Female", 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>
<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

we will discuss **how to implement bidirectional sort in Angular JS**.

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: "David", dateOfBirth: new Date("May 05, 1970"),
      gender: "Male", salary: 68000
    },
    {
      name: "Chris", dateOfBirth: new Date("August 15, 1974"),
      gender: "Male", salary: 57000
    },
    {
      name: "Alex", dateOfBirth: new Date("October 27, 1979"),
      gender: "Female", salary: 53000
    },
    {
      name: "Amy", dateOfBirth: new Date("December 30, 1983"),
      gender: "Female", 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

we will discuss, **how to implement search in Angular** using search filter.

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: "David", gender: "Male", salary: 68000, city: "Chennai" },
    { name: "Chris", gender: "Male", salary: 57000, city: "London" },
    { name: "Alex", gender: "Female", salary: 53000, city: "Chennai" },
    { name: "Amy", gender: "Female", salary: 60000, city: "London" }
  ];

  $scope.employees = employees;
});
```

HtmlPage1.html :

```
<!DOCTYPE html>
<html>
<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

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: "David", gender: "Male", salary: 68000, city: "Chennai" },  
      { name: "Chris", gender: "Male", salary: 57000, city: "London" },  
      { name: "Alex", gender: "Female", salary: 53000, city: "Chennai" },  
      { name: "Amy", gender: "Female", salary: 60000, city: "London" },  
    ];  
  
    $scope.employees = employees;  
  });
```

HtmlPage1.html :

```
<!DOCTYPE html>  
<html>  
<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: "David", gender: "Male", salary: 68000, city: "Chennai" },
        { name: "Chris", gender: "Male", salary: 57000, city: "London" },
        { name: "Alex", gender: "Female", salary: 53000, city: "Chennai" },
        { name: "Amy", gender: "Female", 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

we will discuss **how to 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 anonymous 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: "David", gender: 2, salary: 68000 },
{ name: "Chris", gender: 1, salary: 57000 },
{ name: "Alex", gender: 2, salary: 53000 },
{ name: "Amy", 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>
<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 have 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: "David", gender: 2, salary: 68000 },
            { name: "Chris", gender: 1, salary: 57000 },
            { name: "Alex", gender: 2, salary: 53000 },
            { name: "Amy", 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

we will discuss **ng-hide and ng-show directives in Angular with examples**

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: "David", gender: "Male", city: "Chennai", salary: 68000 },  
        { name: "Mark", gender: "Male", city: "Chicago", salary: 57000 },  
        { name: "Alex", gender: "Female", city: "London", salary: 53000 },  
        { name: "Amy", gender: "Female", 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>  
<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 through 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>
<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: 'David', gender: 'Male', city: 'Chennai' },
    { name: 'Chris', gender: 'Male', city: 'Chicago' },
    { name: 'Alex', gender: 'Female', city: 'London' },
    { name: 'Amy', 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>
<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: "David", gender: "Male", salary: 68000 },
  { name: "Chris", gender: "Male", salary: 57000 },
  { name: "Alex", gender: "Female", salary: 53000 },
  { name: "Amy", gender: "Female", 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>
<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: "David", gender: "Male", salary: 68000 },
      { name: "Chris", gender: "Male", salary: 57000 },
      { name: "Alex", gender: "Female", salary: 53000 },
      { name: "Amy", 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>
    </ul>
  </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: "David", gender: "Male", salary: 68000 },
            { name: "Chris", gender: "Male", salary: 57000 },
            { name: "Alex", gender: "Female", salary: 53000 },
            { name: "Amy", gender: "Female", 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>
<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 PHP Web Service in AngularJS using \$http

we will discuss how to consume an PHP Web Service in an AngularJS application. Let us understand this step by step from creating an PHP web service to consuming it in an Angular application.

Here is what we want to do

1. Create an PHP Web service. This web service retrieves the data from SQL Server database table, returns it in JSON format.
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 `angular`.`tblEmployees` ( `Id` INT NOT NULL AUTO_INCREMENT , `Name` VARCHAR(50) NOT NULL , `Gender` VARCHAR(10) NOT NULL , `Salary` INT NOT NULL , PRIMARY KEY (`Id`))
```

```
INSERT INTO tblEmployees values ('', 'Ben', 'Male', 55000)
INSERT INTO tblEmployees values ('', 'David', 'Male', 68000)
INSERT INTO tblEmployees values ('', 'Chris', 'Male', 57000)
INSERT INTO tblEmployees values ('', 'Alex', 'Female', 53000)
INSERT INTO tblEmployees values ('', 'Amy', 'Female', 60000)
```

Step 2 : Create rest API to fetch data from database and output in form of JSON:

Employees.php

```
<?php
/*
A domain Class to demonstrate RESTful web services
*/
Class Employees {

    public function db_connect(){
        $con = mysql_connect("localhost", "root", "");
        mysql_select_db("angular", $con);
        return $con;
    }

    public function getAllEmployees(){
        $i=0;
        $con = $this->db_connect();
        $sql = "select * from tblEmployees";
        $result = mysql_query($sql, $con);
        $data = array();
        while($row = mysql_fetch_array($result))
        {
            $data[$i]['id'] = $row['Id'];
            $data[$i]['name'] = $row['Name'];
            $data[$i]['gender'] = $row['Gender'];
        }
    }
}
```

```
        $data[$i]['salary'] = $row['Salary'];
        $i++;
    }
    return $data;
}

public function getEmployee($id){

    $i=0;
    $con = $this->db_connect();
    $sql = "select * from tblEmployees where Id = $id";
    $result = mysql_query($sql, $con);
    if(mysql_num_rows($result)>0)
    {
        $data = array();
        while($row = mysql_fetch_array($result))
        {
            $data[$i]['id'] = $row['Id'];
            $data[$i]['name'] = $row['Name'];
            $data[$i]['gender'] = $row['Gender'];
            $data[$i]['salary'] = $row['Salary'];
            $i++;
        }
        return $data;
    }else{
        return false;
    }
}
}
?>
```

EmployeesRestHandler.php

```
<?php
require_once("SimpleRest.php");
require_once("Employees.php");

class EmployeesRestHandler extends SimpleRest {

    function getAllEmployees() {

        $employee = new Employees();
        $rawData = $employee->getAllEmployees();

        if(empty($rawData)) {
            $statusCode = 404;
            $rawData = array('error' => 'No employees found!');
        } else {
            $statusCode = 200;
        }

        $requestContentType = 'application/json';
        $this ->setHttpHeaders($requestContentType, $statusCode);
    }
}
```

```
$response = $this->encodeJson($rawData);
echo $response;
}

public function encodeHtml($responseData) {
    $htmlResponse = "<table border='1'>";
    foreach($responseData as $res) {
        foreach($res as $key=>$value)
        {
            $htmlResponse .= "<tr><td>". $key. "</td><td>". $value. "</td></tr>";
        }
    }
    $htmlResponse .= "</table>";
    return $htmlResponse;
}

public function encodeJson($responseData) {
    $jsonResponse = json_encode($responseData);
    return $jsonResponse;
}

public function encodeXml($responseData) {
    // creating object of SimpleXMLElement
    $xml = new SimpleXMLElement('<?xml version="1.0"?><employee></employee>');
    foreach($responseData as $key=>$value) {
        $xml->addChild($key, $value);
    }
    return $xml->asXML();
}

public function getEmployee($id) {

    $employee = new Employees();
    $rawData = $employee->getEmployee($id);

    if(empty($rawData)) {
        $statusCode = 404;
        $rawData = array('error' => 'No employees found!');
    } else {
        $statusCode = 200;
    }

    $requestContentType = 'application/json';
    $this->setHttpHeaders($requestContentType, $statusCode);
    $response = $this->encodeJson($rawData);
    echo $response;
}
}
?>
```

RestController.php

```
<?php
require_once("EmployeesRestHandler.php");

$view = "";
if(isset($_GET["view"]))
    $view = $_GET["view"];
/*
controls the RESTful services
URL mapping
*/
switch($view){

    case "all":
        // to handle REST Url /mobile/list/
        $mobileRestHandler = new EmployeesRestHandler();
        $mobileRestHandler->getAllEmployees();
        break;

    case "single":
        // to handle REST Url /mobile/show/<id>/
        $mobileRestHandler = new EmployeesRestHandler();
        $mobileRestHandler->getEmployee($_GET["id"]);
        break;

    case "" :
        //404 - not found;
        break;

}
?>
```

SimpleRest.php

```
<?php
/*
A simple RESTful webservice base class
Use this as a template and build upon it
*/
class SimpleRest {

    private $httpVersion = "HTTP/1.1";

    public function setHttpHeaders($contentType, $statusCode){

        $statusMessage = $this -> getHttpStatusMessage($statusCode);

        header($this->httpVersion. " ". $statusCode . " ". $statusMessage);
        header("Content-Type:". $contentType);
    }

    public function getHttpStatusMessage($statusCode){
        $httpStatus = array(
```

```
100 => 'Continue',
101 => 'Switching Protocols',
200 => 'OK',
201 => 'Created',
202 => 'Accepted',
203 => 'Non-Authoritative Information',
204 => 'No Content',
205 => 'Reset Content',
206 => 'Partial Content',
300 => 'Multiple Choices',
301 => 'Moved Permanently',
302 => 'Found',
303 => 'See Other',
304 => 'Not Modified',
305 => 'Use Proxy',
306 => '(Unused)',
307 => 'Temporary Redirect',
400 => 'Bad Request',
401 => 'Unauthorized',
402 => 'Payment Required',
403 => 'Forbidden',
404 => 'Not Found',
405 => 'Method Not Allowed',
406 => 'Not Acceptable',
407 => 'Proxy Authentication Required',
408 => 'Request Timeout',
409 => 'Conflict',
410 => 'Gone',
411 => 'Length Required',
412 => 'Precondition Failed',
413 => 'Request Entity Too Large',
414 => 'Request-URI Too Long',
415 => 'Unsupported Media Type',
416 => 'Requested Range Not Satisfiable',
417 => 'Expectation Failed',
500 => 'Internal Server Error',
501 => 'Not Implemented',
502 => 'Bad Gateway',
503 => 'Service Unavailable',
504 => 'Gateway Timeout',
505 => 'HTTP Version Not Supported';
return ($httpStatus[$statusCode]) ? $httpStatus[$statusCode] : $status[500];
```

```
}
?>
```

.htaccess

```
# Turn rewrite engine on
Options +FollowSymlinks
RewriteEngine on
```

```
# map neat URL to internal URL
```

```
RewriteRule ^employees/list/$ RestController.php?view=all [nc,qs]
RewriteRule ^employees/list/([0-9]+)/$ RestController.php?view=single&id=$1 [nc,qs]
```

Step 3 : Add a new JavaScript file to the Scripts folder. Name it **Script.js**. Copy and paste the following code.

```
var app = angular
    .module("myModule", [])
    .controller("myController", function ($scope, $http) {

        $http.get("EmployeeService.asmx/GetAllEmployees")
            .then(function (response) {
                $scope.employees = response.data;
            });
    });
```

Step 4 : 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 : Create an HTML page. Copy and paste the following HTML and Angular code

```
<!DOCTYPE html>
<html>
<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. 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

We will answer all these questions little later. 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 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: 'service/employees/list'
});
```

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](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('service/employees/list/')`

\$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('service/employees/list/');
```

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('service/employees/list/').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('service/employees/list/').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('service/employees/list/').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('service/employees/list/').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('service/employees/list/').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

1. What is a service in Angular
2. Why do we need services in an angular application
3. What are the benefits of using 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.. \$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.

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 discussion, we had 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 within 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 its 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 within 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.

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**".

Your String	<input type="text" value="AngularVideoTutorial"/>
Result	<input type="text" value="Angular Video Tutorial"/>
	<input type="button" value="Process String"/>

Let us first see, how to achieve this without using a custom service.

HtmlPage1.html :

```
<!DOCTYPE html>
<html>
<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>
    </table>
  </div>
</body>
</html>
```

```
</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 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>

<div>

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 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>, 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.

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.

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

To get the script file visit <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 of jQuery tutorial.

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 extremely useful.

To get started with angular

1. Add a reference to the angular script

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

```
var demoApp = angular.module("demoApp", [])
    .controller("demoController", function
        ($scope, $location, $anchorScroll) {
        $scope.scrollTo = function (scrollLocation) {
            $location.hash(scrollLocation);
            $anchorScroll.yOffset = 20;
            $anchorScroll();
        }
    });
```

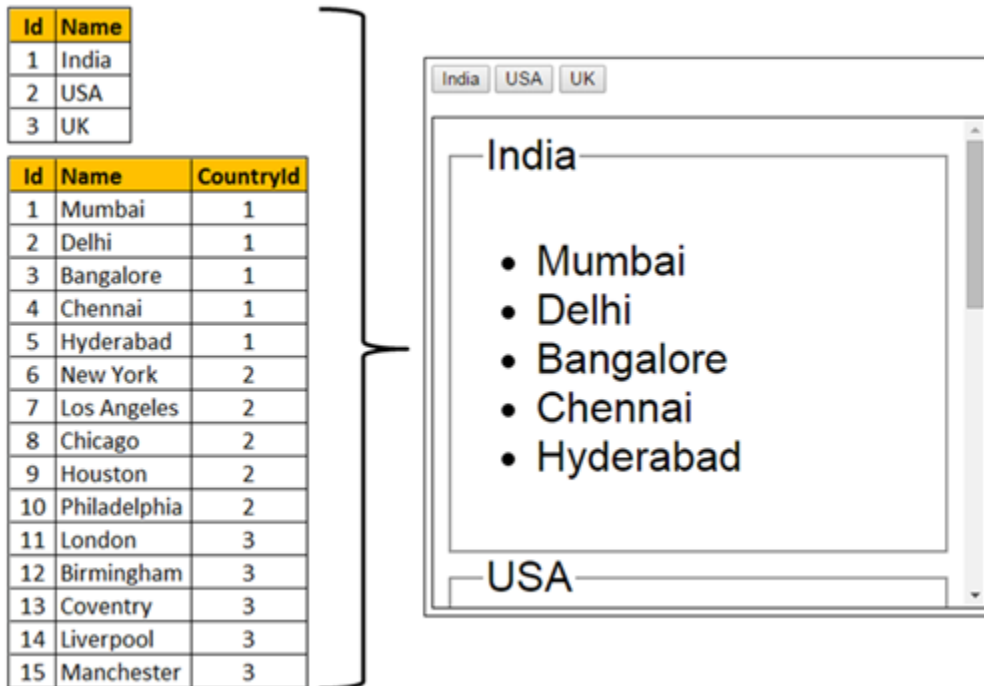
Styles.css

```
div {
    width: 400px;
    border: 1px solid black;
    font-family: Arial;
    font-size: large;
    padding: 5px;
}
```


Angular anchorscroll with database data (Case Study)

In last part we discussed using anchorscroll service with hardcoded data. Now we will discuss using angular anchorscroll service 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.



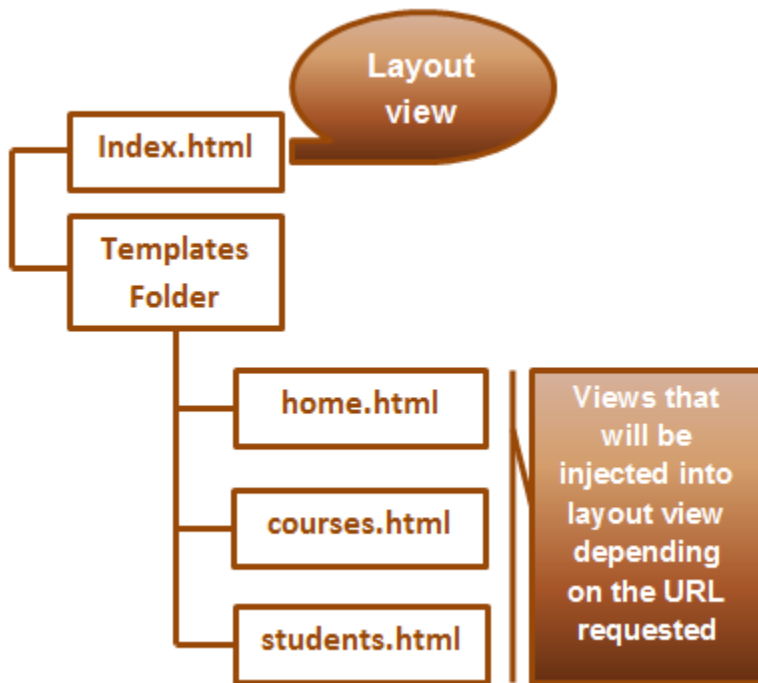
AngularJS routing tutorial

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

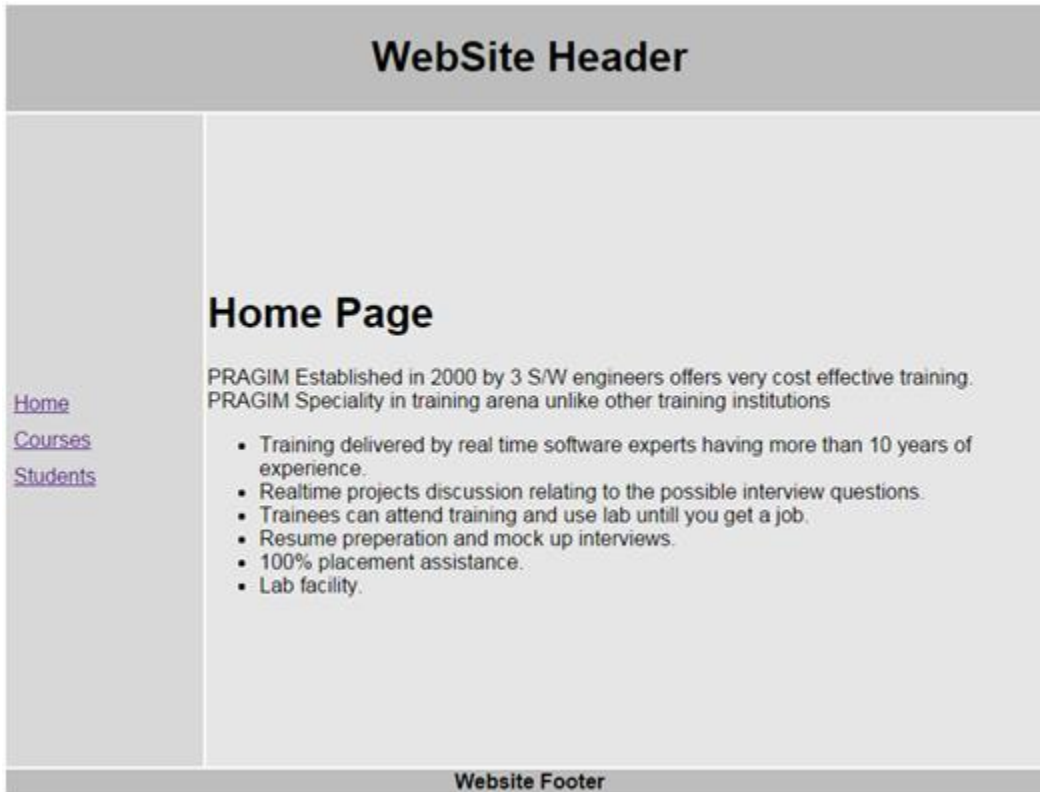
home.html, **courses.html** & **students.html** will be injected into the layout view(**index.html**) depending on the URL the end user has requested

For example, if the user has requested <http://localhost/spa/home>, then **home.html** will be injected into the layout view (**index.html**). Similarly if the user has requested <http://localhost/spa/courses>, then **courses.html** will be injected into the layout view (**index.html**).

Preparing the angular 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.

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.

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.

WebSite Header

[Home](#) [Courses](#) [Students](#)

Website Footer

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 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>
```

Angularjs partial templates

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

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 : Create 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 short while.

```
<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>

Training delivered by real time software experts having more than 10 years of experience.

Realtime projects discussion relating to the possible interview questions.

Trainees can attend training and use lab until you get a job.

Resume preparation and mock up interviews.

100% placement assistance.

Lab facility.

courses.html : The coursesController will set the courses property on the \$scope object. We will discuss creating the coursesController shortly.

<h1>Courses we offer</h1>

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

{{course}}

students.html : The students data is going to come from a database table. 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 short while.

<h1>List of Students</h1>

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

{{student.name}}

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.

```
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"
    })
})
.config(['$locationProvider', function($locationProvider) {
    $locationProvider.hashPrefix("");
}])
.controller("homeController", function ($scope) {
    $scope.message = "Home Page";
})
.controller("coursesController", function ($scope) {
    $scope.courses = ["PHP", "Android", "Angular JS", "SQL Server"];
})
.controller("studentsController", function ($scope, $http) {
    $http.get("service/student/list/")
        .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

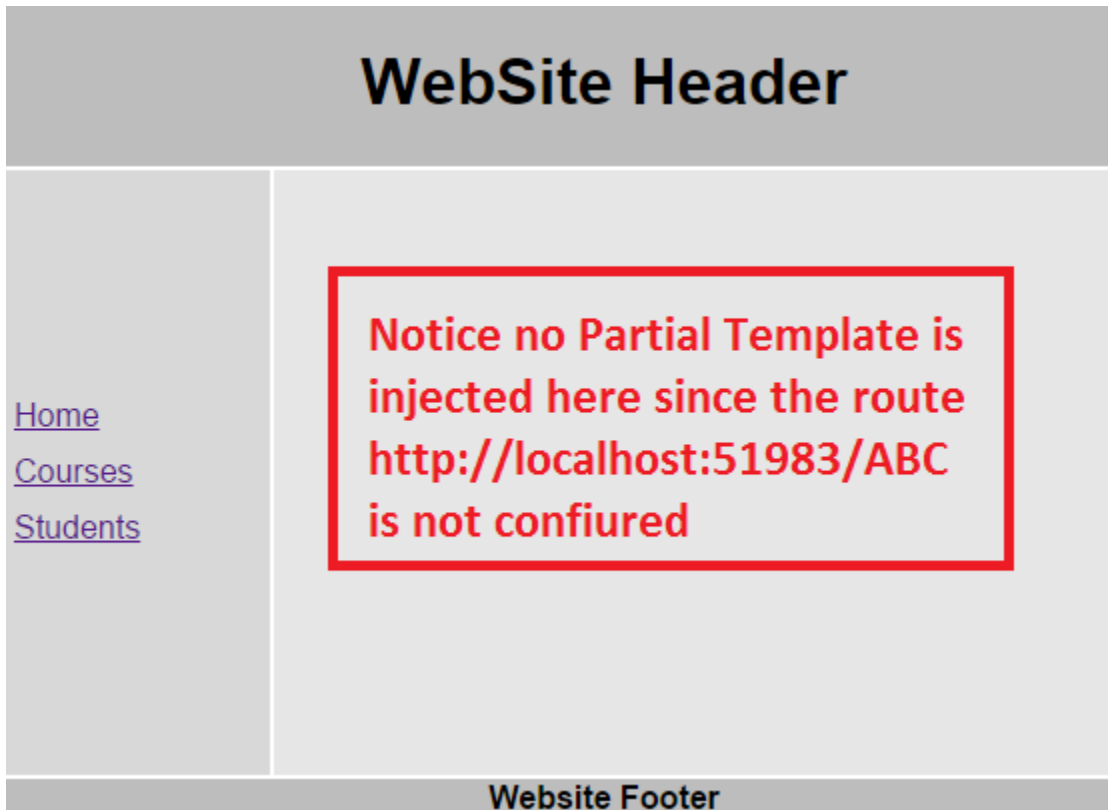
```
<html ng-app="Demo">
```

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.

AngularJS default route

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/spa/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/spa/>. 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"
    })
})
```



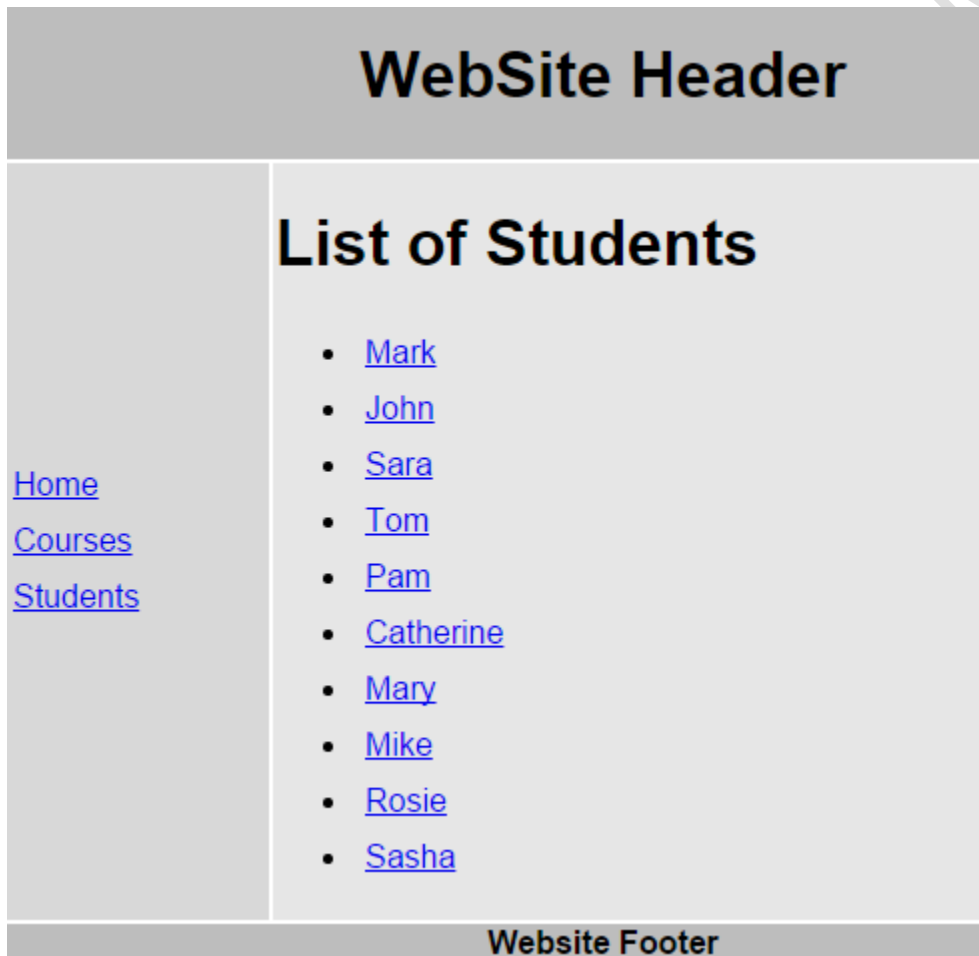
```
.otherwise({
  redirectTo: "/home"
})
$locationProvider.html5Mode(true);
})
```

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

AngularJS routeparams example

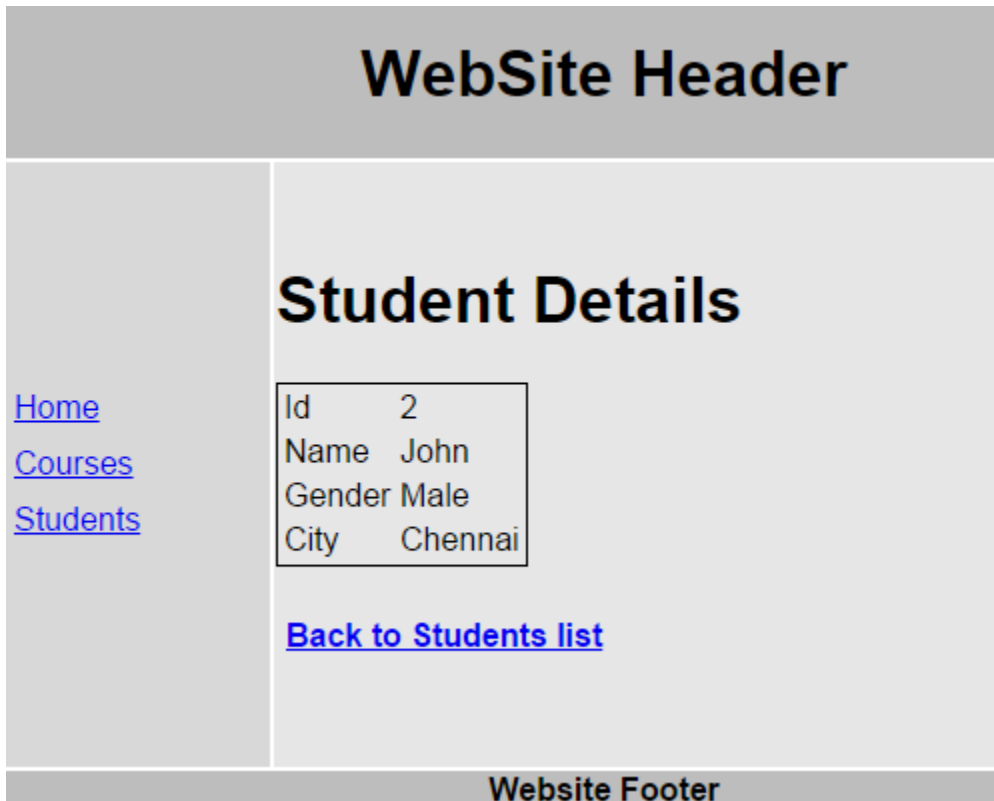
Here we will discuss **how to pass and retrieve parameters from URL in AngularJS**.

Here is what we want to do : When we navigate to [/students](#), the list of student names must be displayed as hyperlinks.



The screenshot displays a web application layout. At the top is a grey header bar with the text "WebSite Header". Below the header is a main content area with a light grey background. On the left side of this area is a vertical sidebar containing three blue hyperlinks: "Home", "Courses", and "Students". The "Students" link is currently selected. The main content area has the title "List of Students" in bold black text. Below the title is a bulleted list of ten student names, each preceded by a blue dot and followed by a blue underline: "Mark", "John", "Sara", "Tom", "Pam", "Catherine", "Mary", "Mike", "Rosie", and "Sasha". At the bottom of the page is a grey footer bar with the text "Website Footer".

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 : 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 2 : 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.

```
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) {
  $scope.courses = ["PHP", "Android", "Angular JS", "SQL Server"];
})
.controller("studentsController", function ($scope, $http) {
  $http.get("service/student/list/ ")
    .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**.

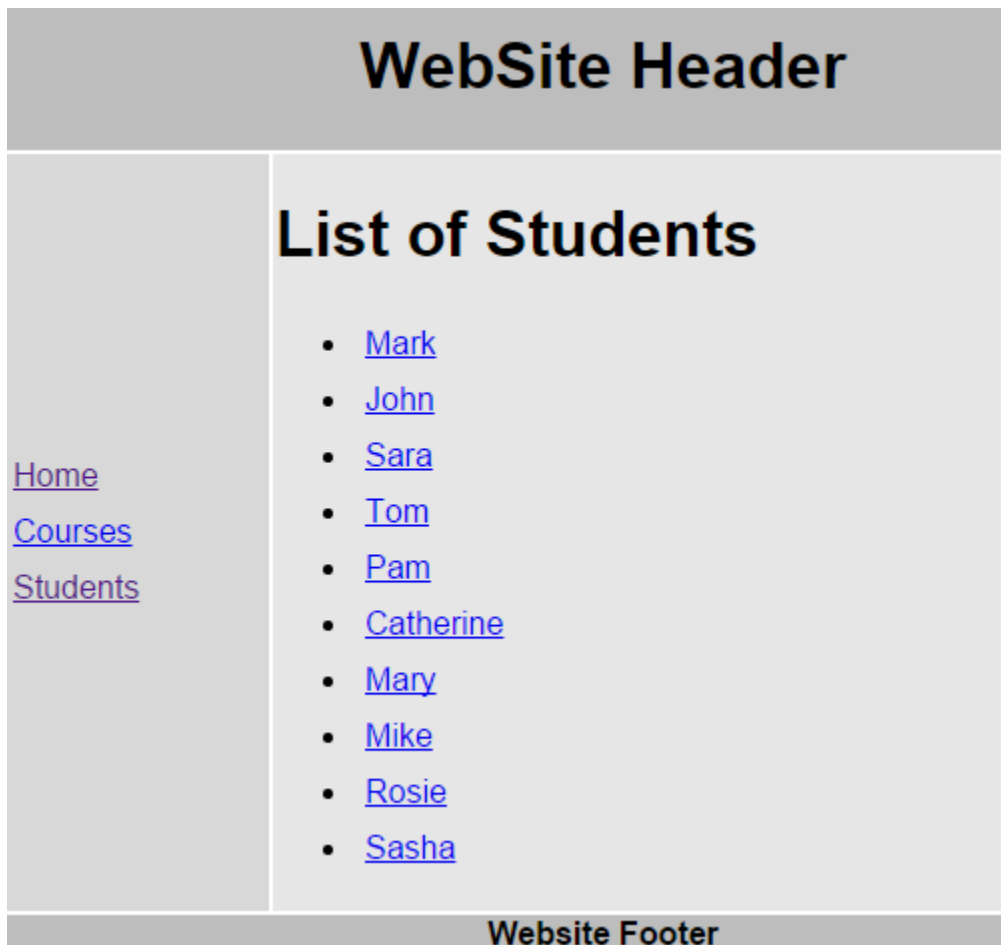
```
var app = angular
    .module("Demo", ["ngRoute"])
    .config(function ($routeProvider, $locationProvider) {
        $routeProvider
            .when("/home", {
                templateUrl: "Templates/home.html",
                controller: "homeController"
            })
            .when("/courses", {
                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.aspx/GetAllStudents")
        .then(function (response) {
            $scope.students = response.data;
        })
})
.controller("studentDetailsController", function ($scope, $http, $routeParams) {
    $http({
        url: "StudentService.aspx/GetStudent",
        method: "get",
        params: { id: $routeParams.id }
    }).then(function (response) {
        $scope.student = response.data;
    })
})
})
```

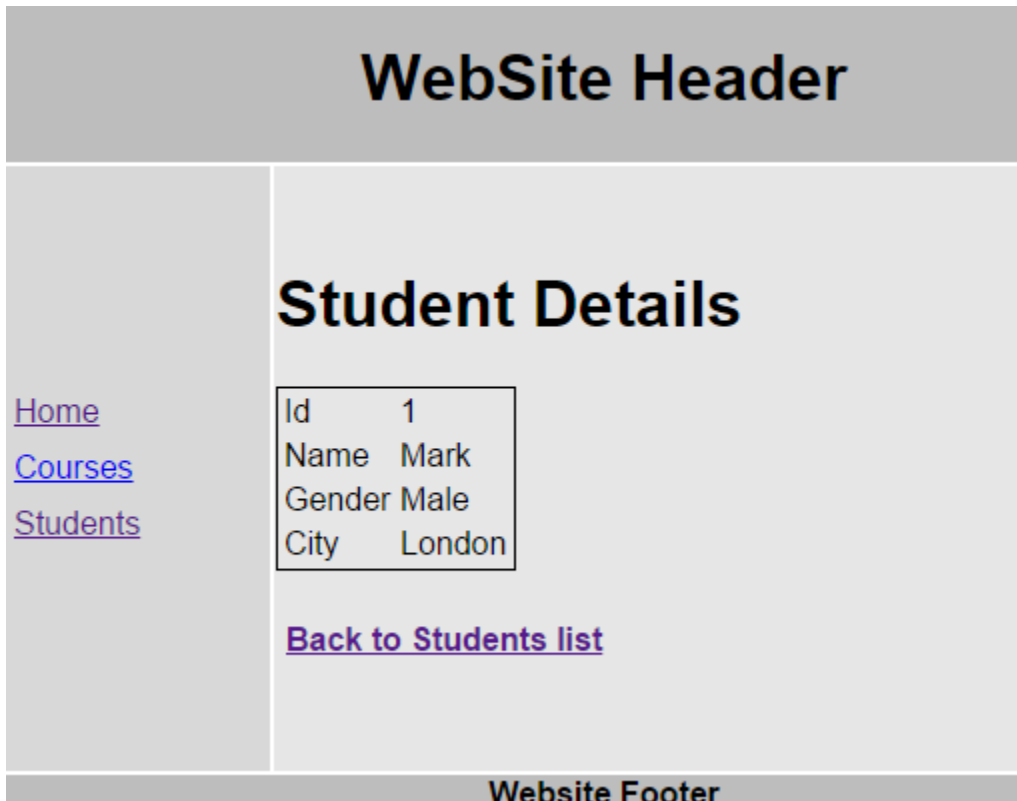
AngularJS page refresh problem

we will discuss page refresh issue that may arise when you refresh the page by pressing **CTRL + F5** or **CTRL + R**.

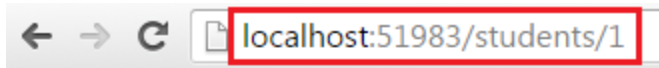
What is the issue : When you navigate to <http://localhost/spa/students>, you will see list of students as shown below



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/spa/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 rendered as shown below.



WebSite Header

[Home](#) [Courses](#) [Students](#)

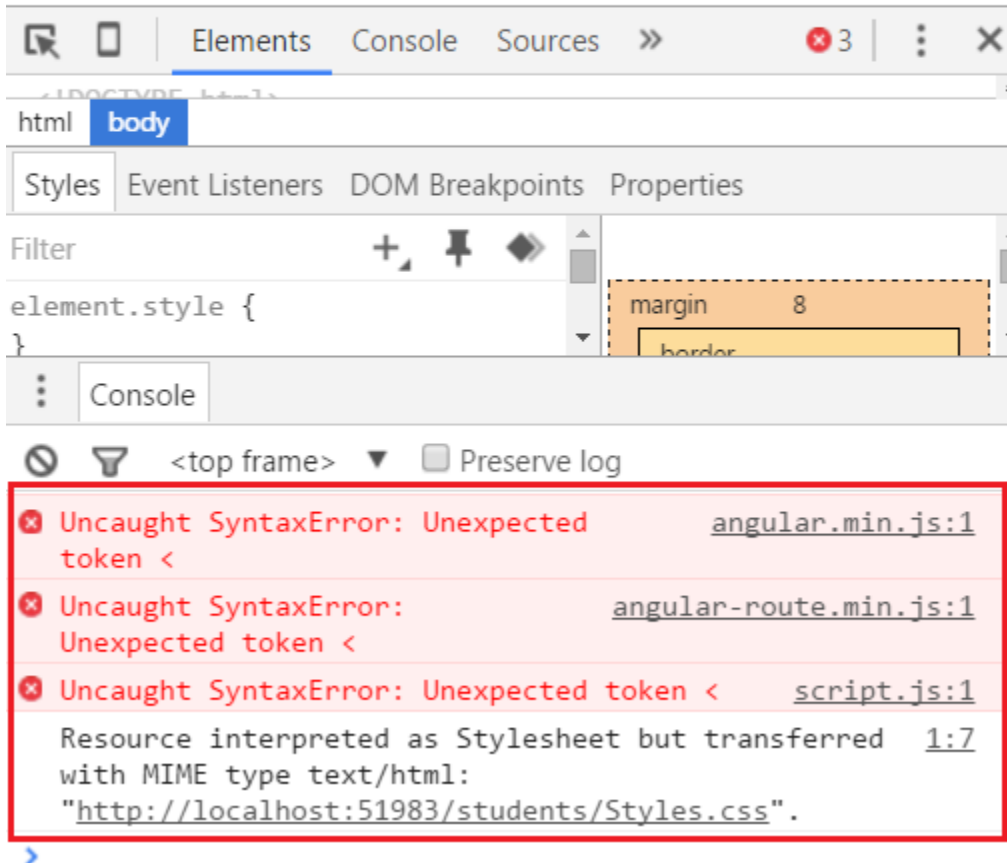
Website Footer

To see the errors, launch the **Browser Developer Tools** by pressing **F12**.

WebSite Header

[Home](#) [Courses](#) [Students](#)

Website Footer



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

we will discuss **AngularJS controller as syntax**.

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 \$scope** object 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](#).

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("service/student/list/")  
        .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);
})
```

Angular nested scopes and controller as syntax

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

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.

India
Maharashtra
Mumbai

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.

India
India - Maharashtra
India - Maharashtra - Mumbai

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"** keyword.

```
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

we will discuss the **difference between \$scope and CONTROLLER AS syntax.**

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 these techniques. Both have their own uses. For example, CONTROLLER AS syntax makes your code more readable when working with nested scopes..
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>
```

AngularJS caseInsensitiveMatch and Inline Templates

we will discuss 2 simple but useful features in Angular

- `caseInsensitiveMatch`
- 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"
})
```

AngularJS route reload

we will discuss **angular route service reload()** method.

We will continue with the same example that we have been working with in the previous part.

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 is 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

Here we will discuss the **difference between \$scope and \$rootScope**. The main difference is that, **\$rootScope** is available globally (for all controllers), whereas **\$scope** is only available to the controller that has created it and it's children.

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.

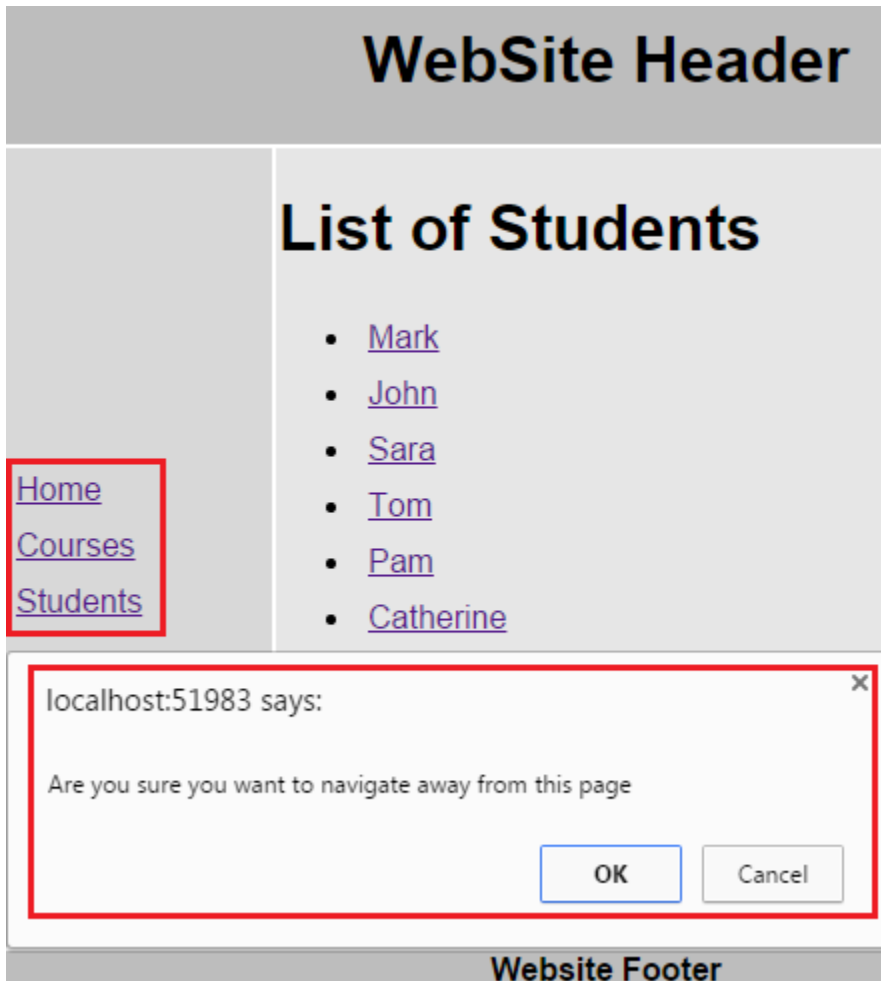
Root Scope Colour : I am Root Scope Colour
Red Colour Controller : I am Red Colour
Green Colour Controller : **greenColour is undefined**

Root Scope Colour : I am Root Scope Colour
Green Colour Controller : I am Green Colour
Red Colour Controller : **redColour is undefined**

AngularJS cancel route change

we will discuss, **how to cancel route change in Angular with an example**. This is extremely useful if you want to warn a user when they are navigating away from a page with unsaved changes.

For our example, let us assume that the user is currently on this students page. When he click on any other links (Home or Courses on the left), we want to present the user with a confirmation box (**Are you sure you want to navigate away from this page**). If the user clicks OK, the user will be redirected to the new route. If cancel is clicked, route navigation should be cancelled and the user should stay on the students page.



Handle `$routeChangeStart` event in `studentsController` function

```
.controller("studentsController", function ($http, $route, $scope) {  
    var vm = this;  
  
    $scope.$on("$routeChangeStart", function (event, next, current) {  
        if (!confirm("Are you sure you want to navigate away from this page")) {  
            event.preventDefault();  
        }  
    });  
  
    vm.reloadData = function () {  
        $route.reload();  
    }  
  
    $http.get("StudentService.aspx/GetAllStudents")  
        .then(function (response) {  
            vm.students = response.data;  
        })  
})
```

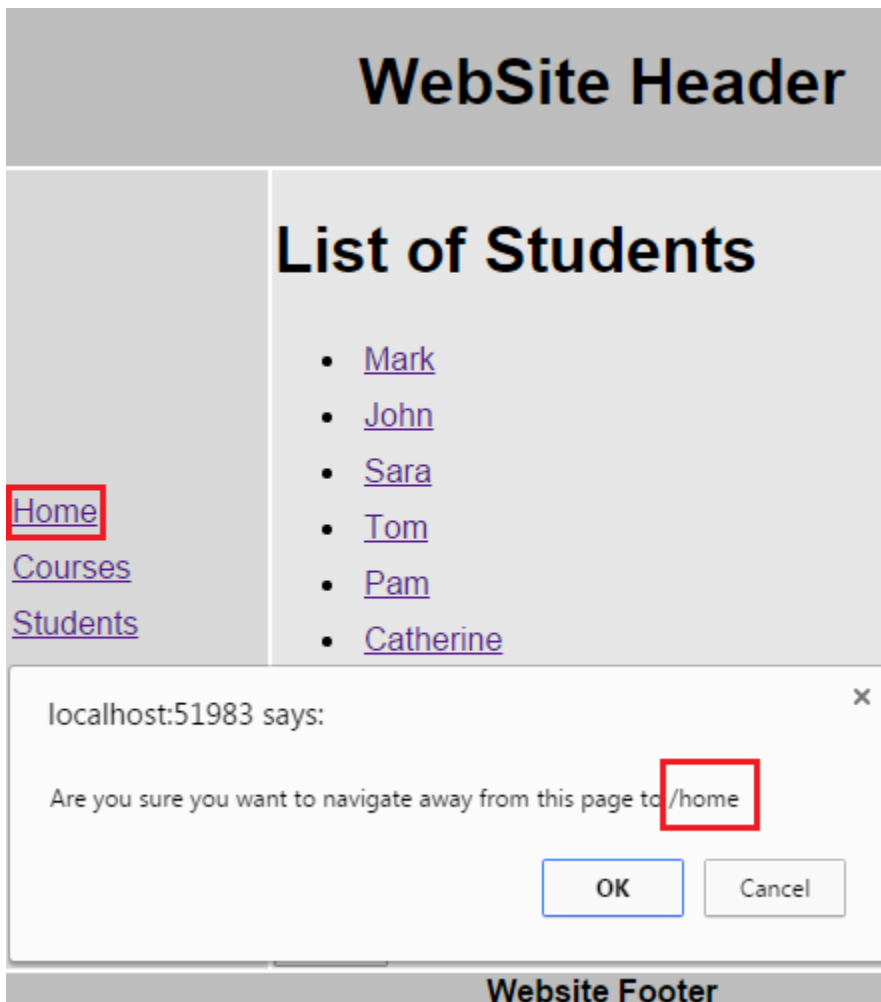
Notice

1. We are injecting \$scope object into the controller function
2. With in \$routeChangeStart event handler function, we are using confirm() function to display the confirmation dialog box
3. When the confirmation dialog box is displayed, If the user clicks Cancel, event.preventDefault() is called and it cancels the route change, so the user stays on the same page
4. On the other hand if the user clicks OK, event.preventDefault() is not called and the route is allowed to change.

If you want to include the route that the user is trying to navigate to in the confirmation message you can do so by using **next parameter** of the **\$routeChangeStart** event handler function as shown below.

```
.controller("studentsController", function ($http, $route, $scope) {  
    var vm = this;  
  
    $scope.$on("$routeChangeStart", function (event, next, current) {  
        if (!confirm("Are you sure you want to navigate away from this page to "  
                    + next.$$route.originalPath))  
        {  
            event.preventDefault();  
        }  
    });  
  
    vm.reloadData = function () {  
        $route.reload();  
    }  
  
    $http.get("StudentService.asmx/GetAllStudents")  
        .then(function (response) {  
            vm.students = response.data;  
        })  
    })
```

With this change in place and while you are on students page and if you click on Home link on the left menu, you will get the following message.



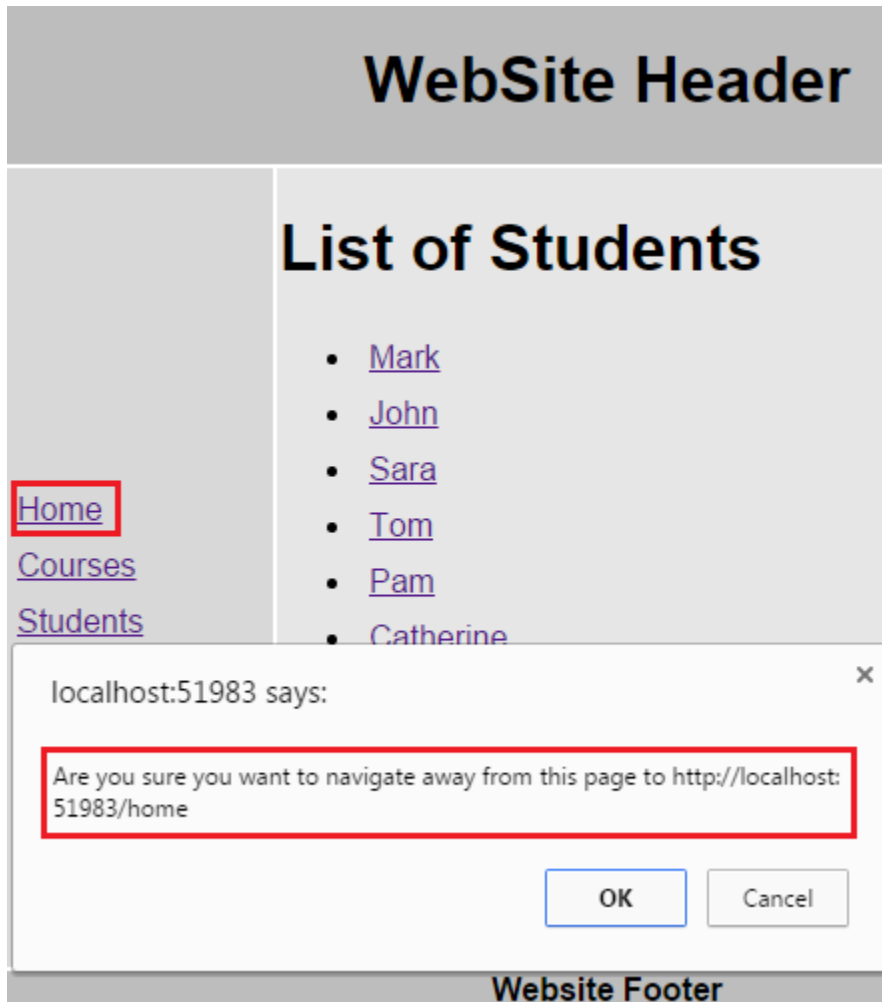
At this point, you might be thinking, how do I know **next** parameter has **\$\$route.originalPath** property. Well you can log the next parameter and see all that properties it has.

You can also cancel route change by handling `$locationChangeStart` event

```
.controller("studentsController", function ($http, $route, $scope) {  
    var vm = this;  
  
    $scope.$on("$locationChangeStart", function (event, next, current) {  
        if (!confirm("Are you sure you want to navigate away from this page to " + next)) {  
            event.preventDefault();  
        }  
    });  
  
    vm.reloadData = function () {  
        $route.reload();  
    }  
  
    $http.get("StudentService.aspx/GetAllStudents")  
        .then(function (response) {  
            vm.students = response.data;  
        })  
    });
```

```
}}
```

The **next** and **current** parameters of **\$locationChangeStart** event handler has the complete next and current URLs. So when you click on home while on students page, you will see the following message.



AngularJS route change events

we will discuss

1. Different events that are triggered when a route change occurs in an angular application
2. Logging the events and event handler parameters to inspect their respective properties

When route navigation occurs in an Angular application, the following events are triggered

1. `$locationChangeStart`
2. `$routeChangeStart`
3. `$locationChangeSuccess`
4. `$routeChangeSuccess`

The following code proves the above point

```
.controller("studentsController", function ($http, $route, $rootScope, $log) {  
    var vm = this;  
  
    $rootScope.$on("$locationChangeStart", function () {  
        $log.debug("$locationChangeStart fired");  
    });  
  
    $rootScope.$on("$routeChangeStart", function () {  
        $log.debug("$routeChangeStart fired");  
    });  
  
    $rootScope.$on("$locationChangeSuccess", function () {  
        $log.debug("$locationChangeSuccess fired");  
    });  
  
    $rootScope.$on("$routeChangeSuccess", function () {  
        $log.debug("$routeChangeSuccess fired");  
    });  
  
    vm.reloadData = function () {  
        $route.reload();  
    }  
  
    $http.get("StudentService.aspx/GetAllStudents")  
        .then(function (response) {  
            vm.students = response.data;  
        })  
    })
```

Please note that we are injecting **\$log** service into the controller function to log the events.

Previously we had used **\$\$route.originalPath** property to get the route that the user is navigating to. **How do we know next parameter has \$\$route.originalPath property.** Well the easiest way is to log and inspect their properties. The following code does exactly the same thing.

```
.controller("studentsController", function ($http, $route, $scope, $log) {  
    var vm = this;  
  
    $scope.$on("$routeChangeStart", function (event, next, current) {  
        $log.debug("RouteChangeStart Event");  
        $log.debug(event);  
        $log.debug(next);  
        $log.debug(current);  
    });  
  
    $scope.$on("$locationChangeStart", function (event, next, current) {  
        $log.debug("LocationChangeStart Event");  
        $log.debug(event);  
        $log.debug(next);  
        $log.debug(current);  
    });  
    })
```

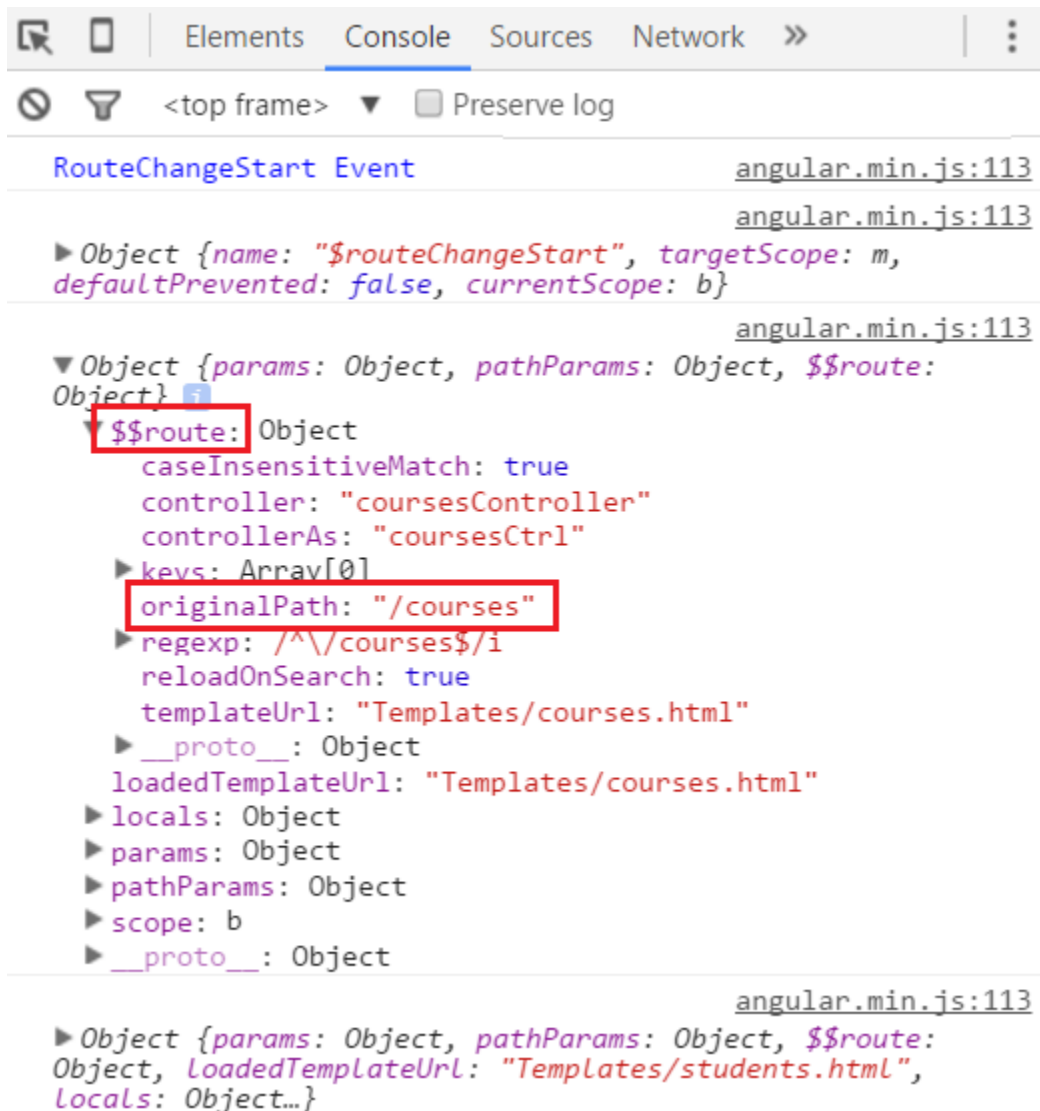
```
});

vm.reloadData = function () {
  $route.reload();
}

$http.get("StudentService.asmx/GetAllStudents")
  .then(function (response) {
    vm.students = response.data;
  })
})
```

In this example we have logged just \$routeChangeStart & \$locationChangeStart events parameters. In a similar way, you can also log \$routeChangeSuccess & \$locationChangeSuccess events parameters.

Now, launch the browser developer tools and you can see the properties available



AngularJS ui-router tutorial

1. It will provide an introduction to ui-router
2. Where to get ui-router module from and including it, in an angular application

What is ui-router

The UI-Router is a third party routing framework for AngularJS. It provides everything that the ngRoute module provides plus many additional features. These additional features are very useful for larger applications. We will discuss some of these additional features with examples shortly.

ui-router implements routing based on the state of the application where as ngRoute implements routing based on the route URL. This means with ngRoute, views and routes are tied to the application URL where as with ui-router views and routes are not tied to the application URL, which means parts of the site can be changed even if the URL does not change. If this does not make sense at the moment, don't worry it will be clear in our upcoming topics. Working with ui-router is similar to working with with ngRoute.

CDN link for ui-router : At the following link you can find both minified and non-minified versions ui-router module file

<https://cdnjs.com/libraries/angular-ui-router>

There are 3 simple steps to include ui-router in your angular application

1. Reference the ui-router CDN link
2. Add ui.router as module dependency
3. Add ui-view directive in the layout page

AngularJs ui-router configuring states

We will discuss **configuring states in an angular application**. According to angular documentation, a state corresponds to a "place" in the application.

To configure a state use **state()** method of **\$stateProvider service**. state() method has 2 parameters

- i) **name** - A unique state name, e.g. "home", "courses", "students"
- ii) **stateConfig** - State configuration object

State configuration object has several properties. Some of them are listed below.

template
templateUrl
controller
controllerAs
resolve
url

Let us now configure states for the sample application that we have been working with. We will configure home, courses and students states. Here are the steps

Step 1 : Modify the code in **config()** function in **script.js** as shown below. Notice we are using **state()** method

of **\$stateProvider** service to configure states.

```
var app = angular
    .module("Demo", ["ui.router"])
    .config(function ($stateProvider) {

        $stateProvider
            .state("home", {
                url: "/home",
                templateUrl: "Templates/home.html",
                controller: "homeController",
                controllerAs: "homeCtrl"
            })
            .state("courses", {
                url: "/courses",
                templateUrl: "Templates/courses.html",
                controller: "coursesController",
                controllerAs: "coursesCtrl"
            })

            .state("students", {
                url: "/students",
                templateUrl: "Templates/students.html",
                controller: "studentsController",
                controllerAs: "studentsCtrl",
                resolve: {
                    studentslist: function ($http, $location) {
                        return $http.get("StudentService.aspx/GetAllStudents")
                            .then(function (response) {
                                return response.data;
                            })
                    }
                }
            })
    })
```

Step 2 : Modify **studentsController** function in script.js as shown below. Notice instead of **\$route** service we are injecting **\$state** service. We used **\$route** service **reload()** method to reload just the current route instead of the entire app. **\$route** service is in **ngRoute** module. Since we removed **ngRoute** module from our application, we cannot use **\$route** service. Instead let's use **\$state** service. Notice to reload just the current state we are using **reload()** method of the **\$state** service.

```
.controller("studentsController", function (studentslist, $state, $location) {
    var vm = this;

    vm.studentSearch = function () {
        if (vm.name)
            $location.url("/studentsSearch/" + vm.name)
        else
            $location.url("/studentsSearch")
    }

    vm.reloadData = function () {
```

```
$state.reload();  
}  
  
vm.students = studentslist;  
})
```

Step 3 : Modify the links in index.html (layout view) as shown below. Notice we are using **ui-sref** to create links with UI-Router. **ui-sref** points to the state of the application. **sref** stands for state reference. You may also use **href** to create links and activate a state. Yet another method available to activate a state is by using state service **go()** method. We will discuss different ways to activate state later. For now let us use **ui-sref** attribute.

```
<a ui-sref="home">Home</a>  
<a ui-sref="courses">Courses</a>  
<a ui-sref="students">Students</a>
```

AngularJS ui router parameters

we will discuss **how to use url parameters with ui router in angular**.

At the moment when you click on any student name on the students page, the student details are not displayed as expected.

WebSite Header

[Home](#)
[Courses](#)
[Students](#)

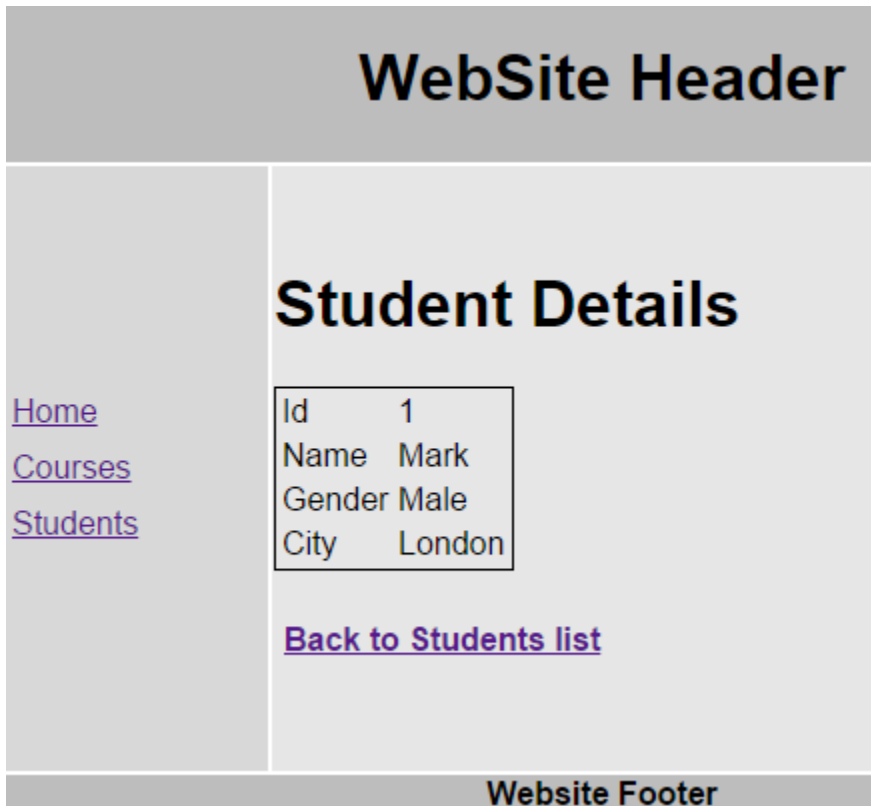
List of Students

Name :

- [Mark](#)
- [John](#)
- [Sara](#)
- [Tom](#)
- [Pam](#)
- [Catherine](#)
- [Mary](#)
- [Mike](#)
- [Rosie](#)
- [Sasha](#)

Website Footer

When we click on a student name, we want to pass student id as a URL parameter and on the subsequent page, we want to display all the details of that specific student.



There are 3 steps to use URL parameters with states in angular.

Step 1 - Define a state with URL parameters : To define a URL parameter use url property of the state. In the example below, id is the URL parameter

```
.state("studentDetails", {  
  url: "/students/:id",  
  templateUrl: "Templates/studentDetails.html",  
  controller: "studentDetailsController",  
  controllerAs: "studentDetailsCtrl"  
})
```

Step 2 - Create links to the state with URL parameters : To create links to the state with URL parameters, use ui-sref attribute on the anchor element. Notice the value of the ui-sref attribute is the name of the state and we are using it like a function. To the function we are passing a JavaScript object, which has a property (id) that matches the name of the state parameter defined in Step 1.

```
<ul>  
  <li ng-repeat="student in studentsCtrl.students">  
    <a ui-sref="studentDetails({id: student.id})">  
      {{student.name}}  
    </a>  
  </li>  
</ul>
```

For example, if we have a value of 1 for student.id and a value of Chris for student.name, the above code will generate an anchor element with href attribute value as

```
<a href="/#/students/1">Chris</a>
```

Please note that angular uses url property value of the state configuration object, to generate the correct href attribute value for the anchor link

Step 3 - Access URL parameters : To access URL parameters, use \$stateParams service. Notice the parameter name is used as a property on the \$stateParams service.

```
.controller("studentDetailsController", function ($http, $stateParams) {  
    var vm = this;  
    $http({  
        url: "StudentService.asmx/GetStudent",  
        method: "get",  
        params: { id: $stateParams.id }  
    }).then(function (response) {  
        vm.student = response.data;  
    })  
})
```

Finally to link back to list of students page, use ui-sref attribute on studentDetails.html as shown below.

```
<h4><a ui-sref="students">Back to Students list</a></h4>
```

Test the application : With all these changes if you click on a student name, the respective student id should be passed in the URL and on the subsequent page we should see that specific student details.

Case sensitivity with angularjs ui-router

we will discuss **how to make routes that are configured using ui-router case-insensitive**.

The routes that are configured using ui-router are **case sensitive by default**. For example, consider the state below. Notice the url (/home) is lowercase.

```
$stateProvider  
    .state("home", {  
        url: "/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, then you will see a blank layout page. This is because, by default routes are case-sensitive

<http://localhost:51983/#/HOME>

To make the routes case-insensitive inject **\$urlMatcherFactoryProvider** service into the **config()** function and call **caseInsensitive(true)** function passing a value of true.

```
var app = angular  
    .module("Demo", ["ui.router"])
```

```
.config(function ($urlMatcherFactoryProvider) {  
    $urlMatcherFactoryProvider.caseInsensitive(true);  
})
```

Angular ui router default route

we will discuss **how to set a default route when using ui-router in your angular application.**

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 a blank layout page

You will also have this same problem if you navigate to the root url (<http://localhost:51983>). In both these cases we want angular to redirect to default route if the user is trying to navigate to a state that is not configured.

To configure the default route when you are using ui-router inject **\$urlRouterProvider** service into the **config()** function and use **otherwise()** function passing it the default route.

```
.config(function ($urlRouterProvider) {  
    $urlRouterProvider.otherwise("/home");  
})
```

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

AngularJS ui router custom data

we will discuss **how to add custom data to a state in angular.**

To add custom data to a state use data property. In the example below, we have added a **"data"** a property to the **"home"** state. The value for this property is a Javascript object that contains our custom data. Along the same lines we have added custom data to the **"courses"** state.

```
$stateProvider  
    .state("home", {  
        url: "/home",  
        templateUrl: "Templates/home.html",  
        controller: "homeController",  
        controllerAs: "homeCtrl",  
        data: {  
            customData1: "Home State Custom Data 1",  
            customData2: "Home State Custom Data 2"  
        }  
    })  
    .state("courses", {  
        url: "/courses",
```

```
templateUrl: "Templates/courses.html",
controller: "coursesController",
controllerAs: "coursesCtrl",
data: {
  customData1: "Courses State Custom Data 1",
  customData2: "Courses State Custom Data 2"
}
})
```

The custom data is now available in all the controllers. To access state custom data from it's controller (i.e to access home state data from home controller) use `$state.current.data.customPropertyName`. To access state custom data from a different controller (i.e to access courses state data from home controller) use `$state.get("statename").data.customPropertyName`.

```
.controller("homeController", function ($state) {
  this.message = "Home Page";

  this.homeCustomData1 = $state.current.data.customData1;
  this.homeCustomData2 = $state.current.data.customData2;

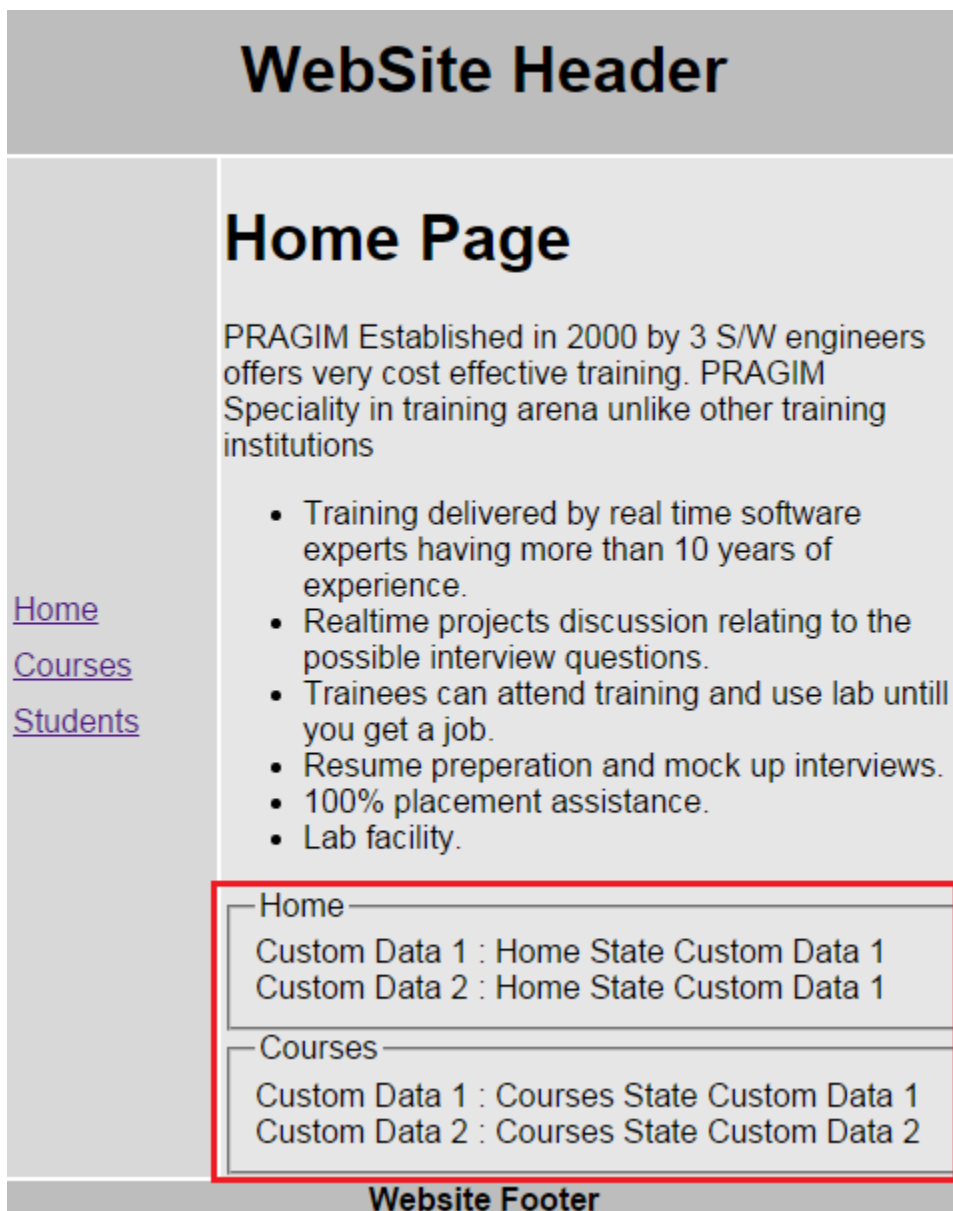
  this.coursesCustomData1 = $state.get("courses").data.customData1;
  this.coursesCustomData2 = $state.get("courses").data.customData2;
})
```

Modify **home.html** as shown below, if you want to display the custom data on the home view

```
<fieldset>
  <legend>Home</legend>
  Custom Data 1 : {{homeCtrl.homeCustomData1}}
  <br />
  Custom Data 2 : {{homeCtrl.homeCustomData2}}
</fieldset>

<fieldset>
  <legend>Courses</legend>
  Custom Data 1 : {{homeCtrl.coursesCustomData1}}
  <br />
  Custom Data 2 : {{homeCtrl.coursesCustomData2}}
</fieldset>
```

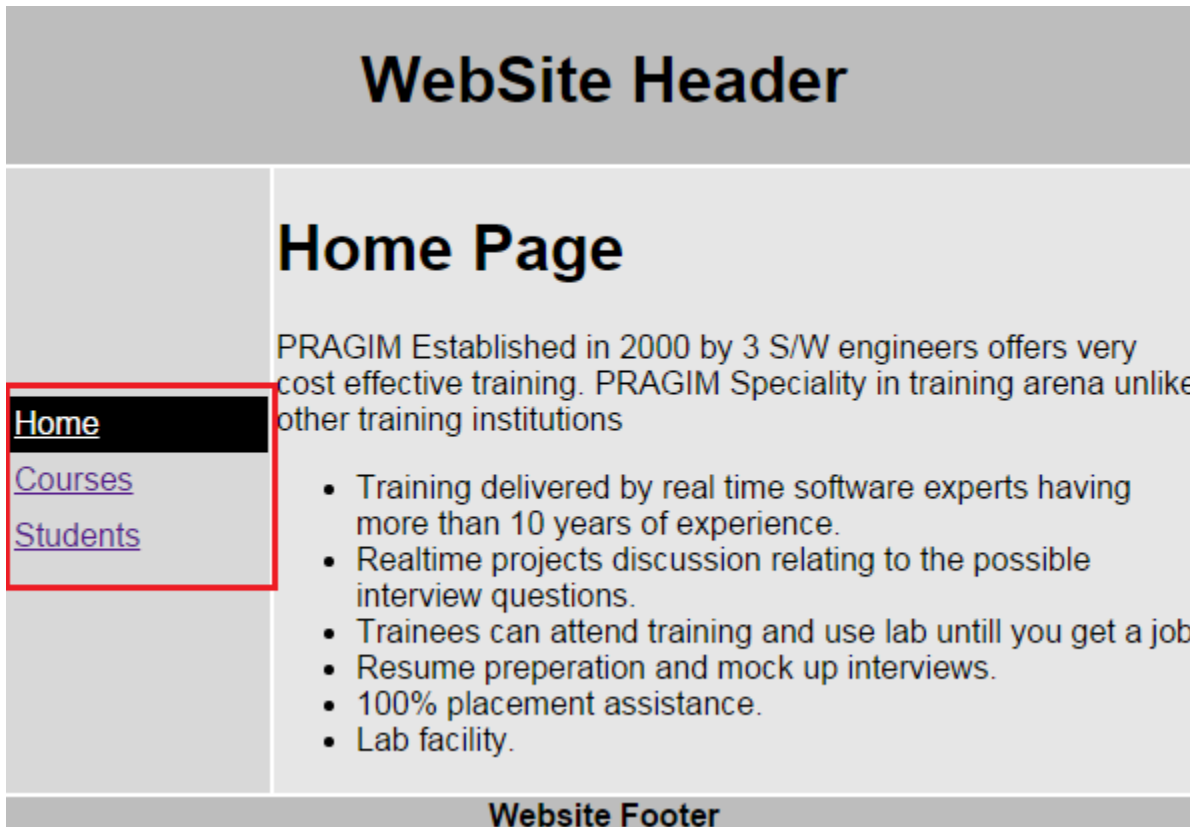
With all the above changes, the custom data will be displayed on the home view as shown below.



ui router active state css

we will discuss how to highlight the navigation menu item if the user is currently on that page

Let us understand this with an example. If the user is on the "**home page**", then we want to highlight "**Home**" menu item on the left. Similarly, if the user is on the "**courses**" page, highlight "**Courses**" and when on "**students**" page highlight "**Students**" menu item.



Here are the steps to achieve this

Step 1 : Create a CSS class that you want to be applied when a state is active. I named the css class `activeState`. You can give it any meaningful name. This class should be in `Styles.css`

```
.activeState{
  background-color:black;
  color:white
}
```

Step 2 : Modify links in `index.html` as shown below. `ui-sref-active` directive will automatically apply the specified css class when the respective state is active and removes the class when it is inactive.

```
<a ui-sref="home" ui-sref-active="activeState">Home</a>
<a ui-sref="courses" ui-sref-active="activeState">Courses</a>
<a ui-sref="students" ui-sref-active="activeState">Students</a>
```

angularjs ui-router nested views

we will discuss nested states and views.

One of the benefits of **ui-router** over **ngRoute** is that it provides support for nested states and views. Nested states have parent child relationship. This means properties of the parent state are available to the child state. There are several methods for nesting states. The following are the 2 common methods.

1. Using 'dot notation'. In the example below "students" is a child of "studentParent" state. Notice in the child state name "studentParent.students" we are using . (DOT)

```
.state("studentParent", {  
  // parent state config properties  
})  
.state("studentParent.students", {  
  // child state config properties  
})
```

2. Using the parent property with the parent name as string. In the example below, we are using parent property in the child state to specify who the parent is for this state.

```
.state("studentParent", {  
  // parent state config properties  
})  
.state("students", {  
  parent: "studentParent",  
  // child state config properties  
})
```

Difference between ngroute and ui-router

we will discuss the **difference between ngroute and ui-router**

ngRoute is developed by the Angular team where as **ui-router** is a 3rd-party module

ng-route implements routing based on the route URL where as **ui-router** implements routing based on the state of the application