**AngularJS Applications**

To start the Angular JS Application we should download the source file from angularjs.com and reference it in script **or** we should declare the CDN into our script <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"> </script>

AngularJS **modules** define AngularJS applications, **ng-app** directive defines the application.

var app = angular.module('myApp', []);

<div ng-app="myApp">...</div>

The ng-app directive also tells AngularJS that the <div> element is the "owner" of the AngularJS application, Now we can add controllers, directives, filters, and more, to your AngularJS application.

AngularJS **controllers** control AngularJS applications, **ng-controller** directive defines the controller.

app.controller('myCtrl', function($scope) {  
    $scope.firstName= "John";  
    $scope.lastName= "Doe";  
});

AngularJS **model** will bind the value of html control(input, select, textarea) **ng-model** directive defines the value of the UI control.

<input type="text" ng-model="name"><br>

<input type="number" ng-model="children"><br>

<p>{{name }} is having {children} childrens<p>

Note: To make your page HTML valid, we can use **data-ng-**, instead of **ng-**, like :data-ng-controller, data-ng-bind

Lets use them in our HTML

<div ng-app="**myApp**" ng-controller="**myCtrl**">  
  
First Name: <input type="text" ng-model="firstName"><br>  
Last Name: <input type="text" ng-model="lastName"><br>  
<br>  
Full Name: {{firstName + " " + lastName}}  
  
</div>

AngularJS expressions can be written inside double braces: {{ expression }}.

AngularJS expressions can also be written inside a directive: ng-bind="expression".

AngularJS will resolve the expression, and return the result exactly where the expression is written. ex:{{5+7}} o/p 12

Note:1)If we remove the ng-app directive, HTML will display the expression as it is, without solving it.

2) If we are not define module name in script file then we can directly call them in Html By saying <div ng-app>

**ng-init** directive is used to initializes application data, The ng-init directive defines **initial values** for an AngularJS application.

<div ng-app="" ng-init="firstName='John';lastName='Doe'">  
  
<p>The full name is {{ firstName + " " + lastName }}</p>

<p>The full name is <span ng-bind="firstName + ' ' + lastName"></span></p>   
  
</div>

O/P is : The full name is: John Doe

The full name is: John Doe

<div ng-app="" ng-init="quantity=1;cost=5">  
  
<p>Total in dollars: <span ng-bind="quantity \* cost"></span></p>  
  
</div>

O/P is :Total in dollers: 5

## AngularJS Objects:

## AngularJS objects are like JavaScript objects:

## <div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}"> <p>The name is {{ person.lastName }}</p>

## <p>The name is <span ng-bind="person.lastName"></span></p> </div>

## o/p: The name is Doe

## AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

<div ng-app="" ng-init="points=[1,15,19,2,40]">  
  
<p>The third result is {{ points[2] }}</p>

<p>The third result is <span ng-bind="points[2]"></span></p>  
  
</div>

o/p: The third result is 19

## AngularJS Expressions vs. JavaScript Expressions

Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.

Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.

AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.

AngularJS expressions support filters, while JavaScript expressions do not.

## Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

For a full reference, visit https://www.w3schools.com/angular/angular\_ref\_directives.asp

In addition you can use the module to add your own directives to your applications:

<script>   
var app = angular.module("myApp", []);  
  
app.directive("w3TestDirective", function() {  
    return {  
        template : "I was made in a directive constructor!"  
    };  
});  
</script>

To Make use this

<div ng-app="myApp" w3-test-directive></div>

o/p: I was made in a directive constructor!

but not with ng- w3-test-directive

Element name: <w3-test-directive></w3-test-directive>

Attribute: <div w3-test-directive></div>

Class: <div class="w3-test-directive"></div>

Comment: <!-- directive: w3-test-directive -->

## Creating and Using the functions in AngularJS Controller

## A controller can also have methods (variables as functions):

## <body>

## <div ng-app="myApp" ng-controller="personCtrl"> First Name: <input type="text" ng-model="firstName"><br> Last Name: <input type="text" ng-model="lastName"><br> <br> Full Name: {{fullName()}} </div>

## <script src="personController.js"></script>

## </body>

## In External JS File personController.js will be

## <script> var app = angular.module('myApp', []); app.controller('personCtrl', function($scope) {     $scope.firstName = "John";     $scope.lastName = "Doe";     $scope.fullName = function() {         return $scope.firstName + " " + $scope.lastName;     }; }); </script>

**AngularJS Filters**

AngularJS provides filters to transform data:

* currency Format a number to a currency format.
* date Format a date to a specified format.
* filter Select a subset of items from an array.
* json Format an object to a JSON string.
* limitTo Limits an array/string, into a specified number of elements/characters.
* lowercase Format a string to lower case.
* number Format a number to a string.
* orderBy Orders an array by an expression.
* uppercase Format a string to upper case.

## Adding Filters to Expressions

Filters can be added to expressions by using the pipe character |, followed by a filter.

<div ng-app="myApp" ng-controller="personCtrl">  
  
<p>The name is {{ lastName | uppercase }}</p>  
  
</div>

<div ng-app="myApp" ng-controller="namesCtrl">  
  
<ul>  
  <li ng-repeat="x in names | orderBy:'country'">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
  
</div>

<script>

angular.module('myApp', []).controller('namesCtrl', function($scope) {

$scope.names = [

{name:'Jani',country:'Norway'},{name:'Carl',country:'Sweden'},

{name:'Margareth',country:'England'},{name:'Hege',country:'Norway'},

{name:'Joe',country:'Denmark'},{name:'Gustav',country:'Sweden'},

{name:'Birgit',country:'Denmark'},{name:'Mary',country:'England'},

{name:'Kai',country:'Norway'}

];

});

</script>

The filter filter selects a subset of an array.

The filter filter can only be used on arrays, and it returns an array containing only the matching items.

Return the names that contains the letter "i":

<div ng-app="myApp" ng-controller="namesCtrl">  
  
<ul>  
  <li ng-repeat="x in names | filter : 'i'">  
    {{ x }}  
  </li>  
</ul>  
  
</div>

## If you want to return the names based on the user input then we can pass it from model data

## <p><input type="text" ng-model="test"></p> <ul>   <li ng-repeat="x in names | filter : test">     {{ x }}   </li> </ul>

## Custom Filters

You can make your own filters by registering a new filter factory function with your module:

Make a custom filter called "myFormat":

<ul ng-app="myApp" ng-controller="namesCtrl">  
    <li ng-repeat="x in names">  
        {{x | **myFormat**}}  
    </li>  
</ul>  
  
<script>

var app = angular.module('myApp', []);  
app.filter('**myFormat**', function() {  
    return function(x) {  
        var i, c, txt = "";  
        for (i = 0; i < x.length; i++) {  
            c = x[i];  
            if (i % 2 == 0) {  
                c = c.toUpperCase();  
            }  
            txt += c;  
        }  
        return txt;  
    };  
});  
app.controller('namesCtrl', function($scope) {  
    $scope.names = ['Jani', 'Carl', 'Margareth', 'Hege', 'Joe', 'Gustav', 'Birgit', 'Mary', 'Kai'];  
});

</script>

# AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services

## The $http Service

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Use the $http service to request data from the server:

<div ng-app="myApp" ng-controller="myCtrl">   
  
<p>Today's welcome message is:</p>  
<h1>{{myWelcome}}</h1>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
    $http.get("welcome.htm")  
    .then(function(response) {  
        $scope.myWelcome = response.data;  
    });  
});

</script>

To handle errors, add one more functions to the .then method that is on failure:

**Methods**

The example above uses the .get method of the $http service.

The .get method is a shortcut method of the $http service. There are several shortcut methods:

* .delete()
* .get()
* .head()
* .jsonp()
* .patch()
* .post()
* .put()

The methods above are all shortcuts of calling the $http service:

 $http({  
        method : "GET",  
        url : "welcome.htm"  
    }).then(onSuccess Funcition, onFailure Function);

## JSON

The data you get from the response is expected to be in JSON format.

JSON is a great way of transporting data, and it is easy to use within AngularJS, or any other JavaScript.

Example: On the server we have a file that returns a JSON object containing 15 customers, all wrapped in array called records.

The ng-repeat directive is perfect for looping through an array:

<div ng-app="myApp" ng-controller="customersCtrl">   
<ul>  
  <li ng-repeat="x in myData">  
    {{ x.Name + ', ' + x.Country }}  
  </li>  
</ul>  
</div>

<script>

var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
    $http.get("customers.php").then(function(response) {

//First Function Handles on Success  
        $scope.myData = response.data.records;  
    },  function(response) {  
        //Second function handles on Failure  
        $scope.content = "Something went wrong";  
    });  
});

</script>

$log Service: The $log service is used to log the response of the object to the console, it is very usefull to debug from the browser, it is having several properties

app.controller('customersCtrl', function($scope, $http, $log) {  
    $http.get("customers.php").then(function(response) {

$log.info(response) ;  
        $scope.myData = response.data.records;  
    },  function(response) {  
        $log.info(response) ;  
        $scope.content = "Something went wrong";  
    });  
});

## $timeout Service: The $timeout service is AngularJS' version of the window.setTimeout function.

## function($scope, $timeout) {     $scope.myHeader = "Hello World!";     $timeout(function () {         $scope.myHeader = "How are you today?";     }, 2000);

## The $interval service is AngularJS' version of the window.setInterval function.

## function($scope, $interval) {     $scope.theTime = new Date().toLocaleTimeString();     $interval(function () {         $scope.theTime = new Date().toLocaleTimeString();     }, 1000);

## Create Your Own Service

To create your own service, connect your service to the module:

Create a service named hexafy:

app.service('hexafy', function() {  
    this.myFunc = function (x) {  
        return x.toString(16);  
    }  
});  
To use your custom made service, add it as a dependency when defining the filter:

Use the custom made service named hexafy to convert a number into a hexadecimal number:

app.controller('myCtrl', function($scope, **hexafy**) {  
    $scope.hex = **hexafy**.myFunc(255);  
});

## Create Your Own Service

To create your own service, connect your service to the module:

Create a service named hexafy:

app.service('hexafy', function() {  
    this.myFunc = function (x) {  
        return x.toString(16);  
    }  
});

To use your custom made service, add it as a dependency when defining the filter:

Use the custom made service named hexafy to convert a number into a hexadecimal number:

app.controller('myCtrl', function($scope, **hexafy**) {  
    $scope.hex = **hexafy**.myFunc(255);  
});

You can use the filter when displaying values from an object, or an array:

Create a service named hexafy:

<ul>  
<li ng-repeat="x in counts">{{x | myFormat}}</li>  
</ul>

## Creating a Select Box or DropdownList Using ng-options

If you want to create a dropdown list, based on an object or an array in AngularJS, you should use the ng-options directive:

<div ng-app="myApp" ng-controller="myCtrl">  
  
<select ng-model="hello" ng-options="x for x in names">  
</select>  
  
<h1>You selected: {{selectedCar}}</h1>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.names = ["Emil", "Tobias", "Linus"];  
});

</script>

Assume we have an array of objects:

$scope.cars = [  
    {model : "Ford Mustang", color : "red"},  
    {model : "Fiat 500", color : "white"},  
    {model : "Volvo XC90", color : "black"}  
];

The ng-repeat directive has its limitations, the selected value must be a string:

<select ng-model="selectedCar" ng-options="x.model for x in cars">  
</select>  
  
<h1>You selected: {{selectedCar.model}}</h1>  
<p>Its color is: {{selectedCar.color}}</p>

//using ng-repeat

<select ng-model="selectedCar">  
<option ng-repeat="x in cars" >{{x.model}}</option>  
</select>

<h1>You selected: {{selectedCar}}</h1>

Here we can observe the differences

1) while we using **ng-repeat** we have to add option tag use this directive as a attribute to the option tag, where as

by using **ng-options** we can directly declare ng-options directive as an attribute to the select tag

2) while using **ng-repeat** each member treated as **string**, where as **ng-options** will be an **objects**

## The Data Source as an Object

## Assume you have an object with key-value pairs:

## $scope.cars = {     car01 : "Ford",     car02 : "Fiat",     car03 : "Volvo" };

The expression in the ng-options attribute is a bit different for objects:

Using an object as the data source, x represents the key, and y represents the value:

<select ng-model="selectedCar" ng-options="**x for (x, y) in cars**">  
</select>  
  
<h1>You selected: {{selectedCar}}</h1>

The selected value will always be the **value** in a key-**value** pair.

The **value** in a key-**value** pair can also be an object:

The selected value will still be the **value** in a key-**value** pair, only this time it is an object:

$scope.cars = {  
car01 : {brand : "Ford", model : "Mustang", color : "red"},  
car02 : {brand : "Fiat", model : "500", color : "white"},  
car03 : {brand : "Volvo", model : "XC90", color : "black"}  
};

The options in the dropdown list does not have to be the **key** in a **key**-value pair, it can also be the value, or a property of the value object:

<select ng-model="selectedCar" ng-options="**y.brand** for (x, y) in cars">  
</select>

## Toggle, True/False

If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

<div ng-app="myApp" ng-controller="myCtrl">  
  
<button ng-click="myFunc()">Click Me!</button>  
  
<div ng-show="showMe">  
    <h1>Menu:</h1>  
    <div>Pizza</div>  
    <div>Pasta</div>  
    <div>Pesce</div>  
</div>  
  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.showMe = false;  
    $scope.myFunc = function() {  
        $scope.showMe = !$scope.showMe;  
    }  
});  
</script>

Here observe The showMe variable starts out as the Boolean value false.

The myFunc function sets the showMe variable to the opposite of what it is, by using the ! (not) operator.

## $event Object

You can pass the $event object as an argument when calling the function.

The $event object contains the browser's event object:

### Example

<div ng-app="myApp" ng-controller="myCtrl">  
  
<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>  
  
<p>Coordinates: {{x + ', ' + y}}</p>  
  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.myFunc = function(myE) {  
        $scope.x = myE.clientX;  
        $scope.y = myE.clientY;  
    }  
});  
</script>

**Example to ReSET form Fields**

<div ng-app="myApp" ng-controller="formCtrl">  
  <form novalidate>  
    First Name:<br>  
    <input type="text" ng-model="user.firstName"><br>  
    Last Name:<br>  
    <input type="text" ng-model="user.lastName">  
    <br><br>  
    <button ng-click="reset()">RESET</button>  
  </form>  
  <p>form = {{user}}</p>  
  <p>master = {{master}}</p>  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('formCtrl', function($scope) {  
    $scope.master = {firstName: "John", lastName: "Doe"};  
    $scope.reset = function() {  
        $scope.user = angular.copy($scope.master);  
    };  
    $scope.reset();  
});  
</script>

Here We can observe this $scope.master = {firstName: "John", lastName: "Doe"};  
if we change this to $scope.master = {firstName: "", lastName: ""}; then when we click on the reset it will set to the empty controls

## The ng-click directive invokes the reset() method, only if the button is clicked. Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

* $untouched The field has not been touched yet
* $touched The field has been touched
* $pristine The field has not been modified yet
* $dirty The field has been modified
* $invalid The field content is not valid
* $valid The field content is valid

They are all properties of the input field, and are either true or false.

Forms have the following states:

* $pristine No fields have been modified yet
* $dirty One or more have been modified
* $invalid The form content is not valid
* $valid The form content is valid
* $submitted The form is submitted

They are all properties of the form, and are either true or false.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

Show an error message if the field has been touched AND is empty:

<input name="myName" ng-model="myName" required>  
<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The name is required.</span>

## CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

* ng-untouched The field has not been touched yet
* ng-touched The field has been touched
* ng-pristine The field has not been  modified yet
* ng-dirty The field has been modified
* ng-valid The field content is valid
* ng-invalid The field content is not valid
* ng-valid-key One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
* ng-invalid-key Example: ng-invalid-required

The following classes are added to, or removed from, forms:

* ng-pristine No fields has not been modified yet
* ng-dirty One or more fields has been modified
* ng-valid The form content is valid
* ng-invalid The form content is not valid
* ng-valid-key One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
* ng-invalid-key Example: ng-invalid-required

The classes are removed if the value they represent is false.

Add styles for these classes to give your application a better and more intuitive user interface.

Apply styles, using standard CSS:

<style>

input.ng-invalid {  
    background-color: pink;  
}  
input.ng-valid {  
    background-color: lightgreen;  
}

</style>

Forms can also be styled:

Apply styles for unmodified (pristine) forms, and for modified forms:

<style>

form.ng-pristine {  
    background-color: lightblue;  
}  
form.ng-dirty {  
    background-color: pink;  
}

</style>

## Custom Validation

To create your own validation function is a bit more tricky. You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Create your own directive, containing a custom validation function, and refer to it by using my-directive.

The field will only be valid if the value contains the character "e":

<form name="myForm">  
<input name="myInput" ng-model="myInput" required my-directive>  
</form>  
  
<script>

var app = angular.module('myApp', []);  
app.directive('myDirective', function() {  
  return {  
    require: 'ngModel',  
    link: function(scope, element, attr, mCtrl) {  
      function myValidation(value) {  
        if (value.indexOf("e") > -1) {  
          mCtrl.$setValidity('charE', true);  
        } else {  
          mCtrl.$setValidity('charE', false);  
        }  
        return value;  
      }  
      mCtrl.$parsers.push(myValidation);  
    }  
  };  
});

</script>

### Example Explained:

In HTML, the new directive will be referred to by using the attribute my-directive.

In the JavaScript we start by adding a new directive named myDirective.

Remember, when naming a directive, you must use a camel case name, myDirective, but when invoking it, you must use - separated name, my-directive.

Then, return an object where you specify that we require  ngModel, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, mCtrl, is the ngModelController,

Then specify a function, in this case named myValidation, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either true or false.

At last, mCtrl.$parsers.push(myValidation); will add the myValidation function to an array of other functions, which will be executed every time the input value changes.

## Validation Example

<!DOCTYPE html>  
<html>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>  
<body>  
  
<h2>Validation Example</h2>  
  
<form  ng-app="myApp"  ng-controller="validateCtrl"  
name="myForm" novalidate>  
  
<p>Username:<br>  
  <input type="text" name="user" ng-model="user" required>  
  <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">  
  <span ng-show="myForm.user.$error.required">Username is required.</span>  
  </span>  
</p>  
  
<p>Email:<br>  
  <input type="email" name="email" ng-model="email" required>  
  <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">  
  <span ng-show="myForm.email.$error.required">Email is required.</span>  
  <span ng-show="myForm.email.$error.email">Invalid email address.</span>  
  </span>  
</p>  
  
<p>  
  <input type="submit"  
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||  
  myForm.email.$dirty && myForm.email.$invalid">  
</p>  
  
</form>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('validateCtrl', function($scope) {  
    $scope.user = 'John Doe';  
    $scope.email = 'john.doe@gmail.com';  
});  
</script>  
  
</body>  
</html>

The HTML form attribute **novalidate** is used to disable default browser validation.

### Example Explained

The AngularJS directive **ng-model** binds the input elements to the model.

The model object has two properties: **user** and **email**.

Because of **ng-show**, the spans with color:red are displayed only when user or email is **$dirty** and **$invalid**.

## AngularJS Global API

The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

* Comparing objects
* Iterating objects
* Converting data

The Global API functions are accessed using the angular object.

Below is a list of some common API functions:

API Description

angular.lowercase():Converts a string to lowercase

angular.uppercase():Converts a string to uppercase

angular.isString(): Returns true if the reference is a string

angular.isNumber(): Returns true if the reference is a number

### angular.lowercase()

<div ng-app="myApp" ng-controller="myCtrl">  
<p>{{ x1 }}</p>  
<p>{{ x2 }}</p>  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
$scope.x1 = "JOHN";  
$scope.x2 = angular.lowercase($scope.x1);

$scope.x3 = angular.IsNumber($scope.x2);

$scope.x4 = angular.isString($scope.x2);  
});  
</script>

Note: We cannot call directly without $scope like $scope.x4 = angular.isString(x2);

# AngularJS Routing

## What do I Need?

The ngRoute module helps your application to become a Single Page Application.

To make your applications ready for routing, you must include the AngularJS Route module:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-route.js"></script>

Then you must add the ngRoute as a dependency in the application module::

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

Use the $routeProvider to configure different routes in your application:

## What is Routing in AngularJS?

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module.

The ngRoute module routes your application to different pages without reloading the entire application.

Navigate to "red.htm", "green.htm", and "blue.htm":

<body ng-app="myApp">  
  
<p><a href="#/">Main</a></p>  
  
<a href="#red">Red</a>  
<a href="#green">Green</a>  
<a href="#blue">Blue</a>  
  
<div ng-view></div>  
  
<script>  
var app = angular.module("myApp", ["ngRoute"]);  
app.config(function($routeProvider) {  
    $routeProvider  
    .when("/", {  
        templateUrl : "main.htm"  
    })  
    .when("/london", {  
        templateUrl : "london.htm",  
        controller : "londonCtrl" // Adding Controllers  
    })  
    .when("/paris", {  
        templateUrl : "paris.htm",  
        controller : "parisCtrl"// Adding Controllers  
    })

.otherwise({  
        template : "<h1>None</h1><p>Nothing has been selected</p>" // Adding Html directly  
    });  
});  
app.controller("londonCtrl", function ($scope) {  
    $scope.msg = "I love London";  
});  
app.controller("parisCtrl", function ($scope) {  
    $scope.msg = "I love Paris";  
});  
</script>

london.htm , paris.htm are the two different html pages where we want to redirect based on url.

## Where Does it Go?

Your application needs a container to put the content provided by the routing.This container is the ng-view directive.

There are three different ways to include the ng-view directive in your application:

1 <div ng-view></div>

2 <ng-view></ng-view>

3 <div class="ng-view"></div>

Applications can only have one ng-view directive, and this will be the placeholder for all views provided by the route.

## AngularJS Application

<script>

var app = angular.module("myShoppingList", []);   
app.controller("myCtrl", function($scope) {  
    $scope.products = ["Milk", "Bread", "Cheese"];  
    $scope.addItem = function () {  
**$scope.errortext = "";  
        if (!$scope.addMe) {return;}  
        if ($scope.products.indexOf($scope.addMe) == -1) {**            $scope.products.push($scope.addMe); // Adding Items  
**} else {  
            $scope.errortext = "The item is already in your shopping list.";** // Error Handling **}**    }  
    $scope.removeItem = function (x) {  
**$scope.errortext = "";**        $scope.products.splice(x, 1);// Removing Item  
    }    
});

</script>  
  
<div ng-app="myShoppingList" ng-controller="myCtrl">  
    <ul>  
        <li ng-repeat="x in products">  
            {{x}}            <span ng-click="removeItem($index)">&times;</span>  
</li>  
    </ul>  
    <input ng-model="addMe">  
    <button ng-click="addItem()">Add</button>  
    **<p>{{errortext}}</p>**  
</div>

## Q and A

## Q. When to load the Angular JS library ? and why?

## A. AngularJS library either in the <head> or at the start of the <body>. This is because calls to angular.module can only be compiled after the library has been loaded.

## Q. What is Binding in Anjular JS?

## A. the Automatic synchronization between the model and view is called as binding

## Q. How to bind the data in Angular JS?

## A. By putting the expression in double curly braces{{}} or using ng-bind directive

## Q. How Functions can Pollute the Global Namespace?

## A. Functions can easily be overwritten or destroyed by other scripts, thus Global functions should be avoided in JavaScript

## Q. How the Angular JS Overcome Global Namespace Problem

## A. AngularJS modules reduces this problem, by keeping all functions local to the module.

## Q. How can you restrict your directives to only be invoked by some of the methods.

A: The legal restrict values are:

* E for Element name
* A for Attribute
* C for Class
* M for Comment

## By default the value is EA, meaning that both Element names and attribute names can invoke the directive.

## EX:By adding a restrict property with the value "A", the directive can only be invoked by attributes:

var app = angular.module("myApp", []);  
app.directive("w3TestDirective", function() {  
    return {  
        restrict : "A",  
        template : "<h1>Made by a directive!</h1>"  
    };  
});

Q: What is Two Way Binding? How it can be done?

A. The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change its value:

<div ng-app="myApp" ng-controller="myCtrl">  
    Name: <input ng-model="name">  
    <h1>You entered: {{name}}</h1>  
</div>

Q: How to Validate User Input using AngularJS?

A. We can validate using ng-model attribute to check the application data(number, e-mail, required)

<form ng-app="" name="myForm">  
    Email:  
    <input type="email" name="myAddress" ng-model="text">  
    <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail address</span>  
</form>

In the example above, the span will be displayed only if the expression in the ng-show attribute returns true.

Note: If the property in the ng-model attribute does not exist, AngularJS will create one for you. here text is does not exist

Q. How the Application Status can be displayed in Angular JS

A. The ng-model directive can provide status for application data (invalid, dirty, touched, error):

<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">  
    Email:  
    <input type="email" name="myAddress" ng-model="myText" required>  
    <h1>Status</h1>  
    {{myForm.myAddress.$valid}}  
    {{myForm.myAddress.$dirty}}  
    {{myForm.myAddress.$touched}}  
</form>

Here we are checking the email validity status.

Valid: true (if true, the value meets all criteria).

Dirty: false (if true, the value has been changed).

Touched: false (if true, the field has been in focus).

Q. What is scope?

## A. The scope is a JavaScript object with properties and methods, which are available for both the view and the controller. the scope is the Model.

Q. What is rootscope?

## A. All applications have a $rootScope which is the scope created on the HTML element that contains the ng-app directive.

The rootScope is available in the entire application

If a variable has the same name in both the current scope and in the rootScope, the application use the one in the current scope.

<body ng-app="myApp">  
  
<p>The rootScope's favorite color:</p>  
<h1>{{color}}</h1>  
  
<div ng-controller="myCtrl">  
    <p>The scope of the controller's favorite color:</p>  
    <h1>{{color}}</h1>  
</div>  
  
<p>The rootScope's favorite color is still:</p>  
<h1>{{color}}</h1>  
  
<script>

var app = angular.module('myApp', []);  
app.run(function($rootScope) {  
    $rootScope.color = 'blue';  
});  
app.controller('myCtrl', function($scope) {  
    $scope.color = "red";  
});

</script>  
</body>

Q. What is the default order for orderby filter? How to Change it

A. Default is Ascending. we can change this by

Q. What is the difference between creating the dropdown list using ng-repeat and ng-options?

A. Dropdowns made with ng-options allows the selected value to be an **object**, while dropdowns made from ng-repeat has to be a string. So better is to use ng-options

Q. What is $routeProvider and how to define it?

A With the $routeProvider you can define what page to display when a user clicks a link. Define the $routeProvider using the config method of your application. Work registered in the configmethod will be performed when the application is loading.

Q. What are $routeProvider.When() and .otherwise() methods will do?

A. $routeProvider.When("provide url here") this will be used when the requested url is met with the given url, if it is not met with the requested url then .otherwise("provide url here") will be displayed

Q. What is Service in Angular JS?

A. A service in Anjgular is simply an object that provides some sort of service that can be reused within an angular application

Q. Why do we need services? and what are the benifits?

A. Services encapsulate reusable logic that doesnot belong anywhere else (i.e Directives, Filters, Views, Models and Controllers)

Benifits:

Reusability : In a service you usually have a logic that you want to reuse with in your entire application. For example, any time you want to make AJAX calls, you can use one of the built in angular service - $http, simply by injecting it into the object that needs that service. The application is also easier to maintain when the reusable components are encapsulated into their own services.  
  
Dependency Injection : Another benefit of services, is that, they can simply be injected into controllers or other services that need them.  
  
Testability : Since services are injected into controllers or other services that need them, it becomes very easy to test them. Depending on which service you are testing, you can pass mock implementations or real implementations

Q. Can you create a custom service? if so how?

A. 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>

Q. How to use $anchorScroll() Service in Angular App

var app = angular

        .module("myModule", [])

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

            $scope.scrollTo = function (inputId) {

$location.hash(inputId);

$anchorScroll();

            };

        });

<div ng-controller=" myController">

<button id="top" ng-click="scrollTo('bottom')">top</button>  
    <p>The scope of the controller's favorite color:</p>  
    <h1>Some more data</h1>

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

For SQl Records

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