Session 4

Custom Directives, Scope, and Services

Session Objectives

- ✓ Outline how to create a custom directive
- ✓ Use a custom directive
- ✓ Describe the concept of scope in AngularJS
- ✓ Explain services in AngularJS

Custom Directives

1-2

- AngularJS allows us to create our own application specific, custom directives.

```
var app = angular.module('myApp', []);
app.directive('myCustomDirective', function() {
return {
restrict: 'AE',
template: '<h3>Hello AngularJS!!</h3>
<p>I was made inside a Custom directive<p>'
}
```

A sample code using custom directive

Custom Directives

- To call a custom directive in HTML, simply use it as an Element.

```
<body ng-app= "myApp">
<my-custom-directive></my-custom-directive>
</body>
```

A sample code using custom directive as an element

```
<body ng-app= "myApp">
<div my-custom-directive></div>
</body>
```

A sample code using custom directive as an attribute

Invoking a Custom Directive

- In AngularJS, restrict values restrict use of custom directive as an Element or as an Attribute

Allowed restrict values are:

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

Example - Custom Directives

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
<body ng-app="myApp">
<w3-custom-directive></w3-custom-directive>
<div w3-custom-directive></div>
<script>
var app = angular.module("myApp", []);
app.directive('w3CustomDirective', function() {
return {
restrict: 'A',
template: '<h3>Hello AngularJS!!</h3><p>I was made inside a Custom directive
</p>'
       };
   });
</script>
</body>
</html>
```

Scopes

- The scope of AngularJS is the model.
- It is a JavaScript object with properties and methods available for both view and controller.
- It gives execution context for expressions used in the application.
- Three types of scopes are:

Shared scope	Inherited scope	Isolated scope

Scope Hierarchies

1-2

- All applications have a `$rootScope` which is scope created on HTML element containing `ng-app`.
- `$rootScope` is available in the entire application.
- When a variable has same name in both current scope and in `$rootScope`, the application makes use of variable in current scope.

Scope Hierarchies

2-2

$rootScope and $scope Example- Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Scope Demo</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js"></script>
</head>
<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';
                 });
```

Code Continued...

```
app.controller('myCtrl', function($scope)
{
                 $scope.color = "red";
                 });
</script>
<p>Notice that controller's color variable
does not overwrite the rootScope's color
value.</p>
</body>
</html>
```

Nested Scopes and Controllers

1-3

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Nested Scope Demo</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
<script src="script.js"></script>
</head>
<body ng-app="myApp">
<div>
<h3>Nested controllers with model variables defined directly on the scopes</h3>
    (typing on an input field, with a data binding to the model, overrides the same variable of a
parent scope)
</div>
<div ng-controller="firstControllerScope">
<h3>First controller</h3>
<strong>First name:</strong> {{firstName}}<br />
<br />
<label>Set the first name: <input type="text" ng-model="firstName"/></label><br />
<br />
<div ng-controller="secondControllerScope">
<h3>Second controller (inside First)</h3>
```

Nested Scopes and Controllers Example- HTML Code

Nested Scopes and Controllers 2-3

```
<strong>First name (from First):</strong> {{firstName}}<br />
<strong>Last name (new variable):</strong> {{lastName}}<br />
<strong>Full name:</strong> {{getFullName()}}<br />
<br />
<label>Set the first name: <input type="text" ng-model="firstName"/></label><br />
<label>Set the last name: <input type="text" ng-model="lastName"/></label><br />
<br />
<div ng-controller="thirdControllerScope">
<h3>Third controller (inside Second and First)</h3>
<strong>First name (from First):</strong> {{firstName}}<br />
<strong>Middle name (new variable):</strong> {{middleName}}<br />
<strong>Last name (from Second):</strong> {{$parentlastName}}<br />
<strong>Last name (redefined in Third):</strong> {{lastName}}<br />
<strong>Full name (redefined in Third):</strong> {{getFullName()}}<br />
<br />
<label>Set the first name: <input type="text" ng-model="firstName"/></label><br />
<label>Set the middle name: <input type="text" ng-model="middleName"/></label><br />
<label>Set the last name: <input type="text" ng-model="lastName"/></label>
</div>
</div>
</div>
</body>
</html>
```

Nested Scopes and Controllers Example- HTML Code

Nested Scopes and Controllers 3-3

```
var app = angular.module('myApp', [ ]);
app.controller('firstControllerScope', function($scope){
  // Initialize the model variables
  $scope.firstName = "Chris";
});
app.controller('secondControllerScope', function($scope){
  // Initialize the model variables
  $scope.lastName = "Hemsworth";
  // Define utility functions
  $scope.getFullName = function ()
  {
return $scope.firstName + " " + $scope.lastName;
  };
});
app.controller('thirdControllerScope', function($scope){
  // Initialize the model variables
  $scope.middleName = "Whitelaw";
  $scope.lastName = "Pine";
  // Define utility functions
  $scope.getFullName = function () {
return $scope.firstName + " " + $scope.middleName + " " + $scope.lastName;
  };
});
```

Nested Scopes and Controllers Example-JavaScript Code

Services

- Refer to simple objects that perform some sort of work.
- Are JavaScript functions and are responsible to do a specific task only.
- Are injected using dependency injection mechanism of AngularJS.

Some built-in services provided by AngularJS:

$http

$window

$route

$location

$http Service

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>$http service demo</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
</head>
<body>
          <div ng-app="myApp" ng-controller="myCtrl">
                    <p>Today's welcome message is:</p>
                    <h1>{{myWelcome}}</h1>
          </div>
          <p>The $http service requests a page on the server, and the response is set as
the value of the "myWelcome" variable.</p>
          <script>
                    var app = angular.module('myApp', []);
                    app.controller('myCtrl', function($scope, $http) {
                              $http.get("welcome.html")
                              .then(function(response) {
                                        $scope.myWelcome = response.data;
                              });
                    });
          </script>
</body>
</html>
```

We use
http service for
reading data
from remote servers

$location Service

1-2

- Has methods which return information about the location of the current Web page.
- Also keeps itself and the URL in synchronization.
- Any modification made to $location is passed to the URL.
- Whenever the URL changes (such as when a new route is loaded) the $location service updates itself.
- $location updates the browser's URL to navigate to a different route or to watch for changes in $location.

$location Service

```
<!DOCTYPE html>
<html>
<script src = https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
    <h4>$location Service Example</h4>
    <div ng-app="app" ng-controller="LocController">
        <div>
            Current absolute URL: {{currentURL}}
        </div>
        <br />
    </div>
<script>
       var app = angular.module("app", []);
    app.controller("LocController", function ($scope, $location) {
        $scope.currentURL = $location.absUrl();
    });
       </script>
</body>
</html>
```

Summary

- Developers can create new directives using .directive method.
- Allowed restrict values for a custom directive include E for Element name,
  A for Attribute, C for Class, and M for Comment
- The custom directive is used in the view by separating camel case name with a hyphen/dash.
- The $rootScope is available in the entire application.
- In a scenario where a variable has the same name in both current scope and in $rootScope, the application uses the variable defined in the current scope.
- Services are JavaScript functions and are responsible to perform only a specific task.
- Services are injected using the dependency injection mechanism of AngularJS.