Usage Guidelines

Do not forward this document to any non-Infosys mail ID. Forwarding this document to a non-Infosys mail ID may lead to disciplinary action against you, including termination of employment.

Contents of this material cannot be used in any other internal or external document without explicit permission from ETA@infosys.com.

Angular JS

Day 4

Education, Training and Assessment

We enable you to leverage knowledge anytime, anywhere!

Infosys® | Building Tomorrow's Enterprise

Copyright Guideline

Confidential Information

- This Document is confidential to Infosys Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").
- Confidential Information includes, but is not limited to, the following:
  - Corporate and Infrastructure information about Infosys
  - Infosys' project management and quality processes
  - Project experiences provided included as illustrative case studies
- Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.
- Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.
- Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.
- This document also contains third party confidential and proprietary information. Such third party information has been included by Infosys after receiving due written permissions and authorizations from the party/ies. Such third party confidential and proprietary information shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.

Course Information

Course Code : LA1211

Course Name : AngularJS

Version Number : 1.2

Day 4 Objectives

- Modularizing Angular Applications
  - Need of modules
  - Module loading
- Routing in Angular
- Custom components in Angular
  - Custom directives
  - Custom filters
  - Custom services

References

- Brad Green, Shyam Seshadri, AngularJS , O'Reilly Media, 2013.
- http://docs.Angularjs.org/guide/
- https://github.com/Angular

Modularizing Angular Applications

Angular Modules

- For improving scalability of the application, we need to classify and segregate the entire application into different modules.
  - Service module, for service declaration
  - Directive module, for module declaration
  - Filter module, for filter declaration
  - An initialization module containing the code for initialization.

- AngularJS follows the module pattern due to many advantages:
  - Declaring modules is pretty easy and well as easier to understand.
  - Code can be packaged as reusable modules.
  - Testing of modules one by one

Need of modules

- Angular code can be written more efficiently by creating modules

```
var myModule = Angular.module(‘myModule’,[])s
```

- The module can now be set as the app module:

```
<html ng-app = 'myModule' >
```

- This approach prevents pollution of global namespace
- An application can consist of multiple modules working together
- A module can request for another module via dependency injection

```
var module2 =
Angular.module(‘module2’,['myModule‘]);
```

module2 receives **myModule** as a dependency

- The naming convention for a module is camelCased name with small first letter.

Module Loading

- A module in Angular is a collection of configurations and block which will be executed accordingly.
- Modules are applied during the bootstrap process.
- Modules basically contain two kind of blocks:
  - Configuration blocks
  - Run blocks
- Configuration Blocks:
    - Executed when the provider registrations happen and also during the configuration phase.
    - We insert only providers and constants in this block. This prevents services from getting accidentally installed before complete configuration is done.
    - Except registered constants, these blocks are applied in the order in which these have been registered.

Module Loading

- Run Blocks:
  - Executed after the injector is created.
  - Kick starts the application.
  - Instances and constants alone injected in this block. This prevents further system configurations which might be applied during run time of the application.
  - Executed after configuring all services
  - Its better to declare run blocks in separate modules as this will help us ignore these run blocks during testing.

Module Loading

```
angular.module('mymodule', []).
        config(function(injectables) { //provider-injector
        //Config block example
        //can have as many as needed
        //can inject only providers to config blocks.
        }).

        run(function(injectables) { //instance-injector
        //run block example
        //can have as many as needed
        //can inject only instances to run blocks
        });
```

Module Loading: ModuleLoading.html


```
<!doctype html>

<html ng-app="app">

<head>

 <style type="text/css">

        @import "styles/style_casestudy.css";

      </style>


<script src="lib/Angular/angular.js"></script>


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


</head>

<body>
```

```
<center><header><h2>AngularJS Config and Run blocks</h2></header>


<div ng-controller="Ctrl">

   Config time: {{timeConfig}}<br>

   Run time: {{timeRun}}

</div>

</center></body>

</html>
```

Module Loading: controllers_ModuleLoading.js

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


app.config(function($provide){
    $provide.value("timeConfig", new Date());
    $provide.value("timeRun", new Date());
    for(var x=0;x<500000000;x++)
    {
     var y=Math.sqrt(Math.log(x));
    }
});


app.run(function(timeConfig, timeRun){
    timeRun.setTime(new Date().getTime());


});
```

```
app.controller('Ctrl',['$scope','timeConfig','timeRun',
    function($scope, timeConfig, timeRun) {
      $scope.timeConfig=timeConfig;
      $scope.timeRun=timeRun;
    }
])
```

Routing in Angular

Router

- Application Routing based on url fragments in Angular is done using the **$routeProvider**, which provides the **$route** Service
- Using the service we will be able to wire together view templates,controllers and the current url location in browser
- This feature also allows us **deep linking & history support,** which is use of the browser's history (back/forward functionality)
- The $routeProvider exposes API such as the when() and otherwise() methods which allow us to define routes for our application

Routing

- Angular has a service called $ngRoute and to declare the application routes we use $routeProvider.
- This service helps in binding the controllers, view templates, and the current URL location in the browser.

Main module's config block

Satisfied when browser url is baseurl#/main

When browser url is baseurl#/main, Login template is displayed

Controller function to be used in Login template

```
bagApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
  when('/main', {
      templateUrl: 'Login',
      controller: 'LoginCtrl'
    }).
  when('/loginsuccess', {
      templateUrl: 'Home.html'
    }).
    when('/viewbaglist', {
      templateUrl: 'partials/bag-list.html',
      controller: 'BagListCtrl'
    }).
  otherwise({
      redirectTo: '/main'
    });
  }]);
```

Templates

- Templates are used in UI for decoupling markup from data and keep the html free of script logic
- Templates are of great help in building highly reusable UI where the look and feel of the application is subject to frequent changes
- Templates make the code much more simpler and readable and also help in RWD for UI
- In Angular templates are represented in the form of expressions.
- We can display multiple views within one main page using "partials" – segments of template located in separate HTML files or as script templates.
- ngView directive can be used to load partials based on configuration passed to the $route service.

Routing-Case Study

Shop Your Bags Online

Angular SPA

localhost:8090/AngularJS_Demos/Day4/Index.html#/main

Shop Your Bags Online

Username infy

Password infy

Login

Angular SPA

localhost:8090/AngularJS_Demos/Day4/Index.html#/login

Shop Your Bags Online

Welcome infy

View Bag List

In all the screens, base url remains the same

Angular SPA

localhost:8090/AngularJS_Demos/Day4/Index.html#/viewbaglist

Shop Your Bags Online

Search: Sort by: Newest

- Velvet Emb. Bag

  Code:d-023

- AB-03 All Over Print Jute Bag

  Code:d-108

- Golden Ring Bag

  Code:d-070

Home

Angular SPA

localhost:8090/AngularJS_Demos/Day4/Index.html#/bags/d-108

Shop Your Bags Online

Detailed view

Id: d-108

Image:

Home BagList

style.css

```
body
{
background-color:#EFFBFB;
}
header
{
background-color:#0101DF;
 color: white
}

footer
{
background-color:#0101DF;
 color: white
}
```

```
.button
{
border-radius:10px;
}
.input
{
border-radius: 10px;
}
.select
{
border-radius: 10px;

}
```

```
.LoginFormDiv
{
    Border-radius: 10px;
    background-color: #FAFAFA;
    width:40%;
    box-shadow: -1px -1px 2px 15px #FBFBEF;
    -webkit-box-shadow:-5px -5px 5px 9px #E6E6E6;
    -moz-box-shadow:0px 0px 3px 3px #FBFBEF;
}
```

controllers.js (1 of 2)


```
var bagControllers = angular.module('bagControllers',
[]);

bagControllers.controller('LoginCtrl', ['$scope',
'$location', '$rootScope',

 function ($scope, $location, $rootScope) {

$scope.validate=function()

    {

      if($scope.username===$scope.password)

      { $rootScope.username=$scope.username;

        $location.path( "/login" );

      }

      else

      {alert("enter valid data");

        $location.path( "/main" )

      }

} }]);
```

Declaring a controller module named bagControllers

Declaring a controller function named LoginCtrl inside the controller module

Validating login credentials

Assigning valid username to a model in rootScope

Changing the path to /login on success

Changing the path to /main on failure

controllers.js(2 of 2)


```
bagControllers.controller('BagListCtrl', ['$scope',
'$http',

 function ($scope, $http) {

  $http.get('data/bags.json').success(function(data) {

   $scope.bags = data;

  });

  $scope.orderProp = 'age';

 }]);


bagControllers.controller('BagDetailCtrl', ['$scope',
'$routeParams',

 function($scope, $routeParams) {

  $scope.bagId = $routeParams.code;

 }]);
```

Declaring a controller function named BagListCtrl inside the controller module

Ajax call made to bags.json to fetch the bag details

Declaring a controller function named BagDetailCtrl inside the controller module

Retrieving the route parameter 'code' and assigning its value to a model

app.js


```
var bagApp = angular.module('bagApp', [
    'bagControllers'
    ]);
bagApp.config(['$routeProvider',
    function($routeProvider) {
        $routeProvider.
        when('/main', {
            templateUrl: 'Login',
            controller: 'LoginCtrl'
        }).
        when('/login', {
            templateUrl: 'Home'
        }).
```

```
when('/viewbaglist', {
        templateUrl: 'partials/bag-list.html',
        controller: 'BagListCtrl'
    }).
    when('/bags/:code', {
        templateUrl: 'partials/bag-detail.html',
        controller: 'BagDetailCtrl'
    }).
    otherwise({
        redirectTo: '/main'
```

Declaring the main module 'bagApp'

Configuring routes

On encountering baseurl#/main,

On encountering baseurl#/login, Home template is rendered

If none of the routes match, redirects to '/main'

On encountering baseurl#/viewbaglist,

On encountering baseurl#/bags/<bagid>, content of bag-detail.html is rendered

bag-list.html


```
<div class="container-fluid">

    <div class="row-fluid">

        <div class="span2">

            <!--Sidebar content-->



            Search: <input ng-model="query"
class="input">

            Sort by:

            <select ng-model="orderProp" class="select">

                <option
value="name">Alphabetical</option>

                <option value="age">Newest</option>

            </select>



        </div>
```

```
 <div class="span10">

            <!--Body content-->



            <ul class="bags">

                <li ng-repeat="bag in bags | filter:query |
orderBy:orderProp">

<a href="#/bags/{{bag.code}}">{{bag.name}}</a>

<p>Code:{{bag.code}}</p>

                </li>

            </ul>



        </div>

    </div>

</div>

<center><a href="#/login">Home</a></center>
```

bag-detail.html


```
<center><div class="LoginFormDiv" align="center">
<br>
<h2>Detailed view</h2>
<center>
<table >
<tr>
<td> Id:</td>
<td>{{bagId}}</td>
</tr>
<tr>
<td> Image:</td>
<td><img ng-src="imgs/{{bagId}}_b.jpg" height="25%" width="50%"></td>
</tr>
</table>
</center>
<br></br>
</div></center>

<center><a href="#/login">Home</a>  
<a href="#/viewbaglist">BagList</a>  
</center>
```

index.html(1 of 2)


```
<!doctype html>
<html ng-app="bagApp">
  <head>      <title>Angular SPA</title>
<style type="text/css">
@import "styles/style.css";
      </style>
      <script src="lib/Angular/angular.js"></script>
      <script src="js/controllers.js"></script>
      <script src="js/app.js"></script>
  </head>
  <body>
```

Home Template

```
<script type="text/ng-template" id="Home">
        <center>  Welcome {{username}} <br>
        <a href="#/viewbaglist">  View Bag  List  </a>
  </center>     </script>
```

Login Template

```
<script type="text/ng-template" id="Login">
<center>
<div class="LoginFormDiv" align="center"><br>
<form> <table>
<tr>
<td>Username</td>
<td><input  ng-model="username" class="input"></td>
</tr><tr>
<td>Password</td>
<td><input  ng-model="password" class="input"></td>
</tr>  </table>
<input type="submit" class="button" value="Login" ng-click="validate()"/>
</form><br>
</div>
</center>    </script>
```

index.html (2 of 2)


```
<center>
    <div  ng-include="'Header.html'"></div>
 </center>
   <div ng-view></div>

   </body>
</html>
```

Including the content of Header.html

Using ng-view to dynamically load the respective view as per route received

Case Study – MobileStore

Problem Statement

A web application has to be built for a mobile store with very minimal coding. It must contain 3 tabs (mobile, tablet and contact page) on click of which we need to display different information. But the template or structure of all 3 pages are the same i.e. every page has a title, a description and an image.

The application must also be routed based on URL fragments added to the current URL . The information for each page is obtained from a JSON file

Some space should also be reserved for advertisements and this information must remain static for every page.

Mobile Store

Solution

Use Angular.js to create an app which makes use of –

- A partial page which can be rendered on a view. The partial page can use a some standard template for standardizing the display.
- A module where the application routes for routing and their respective partials can be configured.
- An appController for obtaining data from the JSON file and for listening to events in the application
- A routeController which will be in charge of binding the appropriate page content to the view

Code

- Lets start creating this application. First let us take a look at the JSON file (data) that we have. This may also be a typical response from a web service.
- pages.json contains the following data.

```
{
  "mobile": {
    "title": "Mobile store ",
    "content": "Welcome to the world of mobile phones”,
    "img": "imgs/Mobiles.jpg"
  },
  "tablet": {
    "title": "Tablet store",
    "content": "Welcome to the world of tablets ",
    "img": "imgs/Tablets.jpg"
  },
 "contact": {
    "title": "Contact Page",
    "content": "Call us : 111234445",
    "img": "imgs/Contacts.jpg"
  }
}
```

style.css

```
body
{
background-color:#EFFBFB;
}
header
{
background-color:#0101DF;
 color: white
}

footer
{
background-color:#0101DF;
 color: white
}
```

```
.button
{
border-radius:10px;
}
.input
{
border-radius: 10px;
}
.select
{
border-radius: 10px;

}
```

```
.LoginFormDiv
{
    Border-radius: 10px;
    background-color: #FAFAFA;
    width:40%;
    box-shadow: -1px -1px 2px 15px #FBFBEF;
    -webkit-box-shadow:-5px -5px 5px 9px #E6E6E6;
    -moz-box-shadow:0px 0px 3px 3px #FBFBEF;
}
```

Index.html

```
<!DOCTYPE HTML>
<html lang="en-US" ng-app="Site">
  <head>
  <link href="styles/style.css" rel="stylesheet">
  <script src="lib/Angular/angular.js"></script>
  <script src="js/SPA_controllers.js"></script>
  </head>
<body ng-controller="AppController">
<center>
<header>
<h2> {{ 'Example for SPA' }} </h2>
</header>
```

Main Module

Including js file

Controller function

```
h4>
href="#/page/mobile">Mobile store</a>  
<a href="#/page/tablet">Tablet store</a>  
<a href="#/page/contact">Contact Page</a>
</h4>
<
dynamic content -->
  <ng-view></ng-view>
  <br>
<footer>
  <center>
        Copyright @ Infosys
  </center>
  </footer>
```

url mappings

Including dynamic views

SPA_controllers.js


```
var Site = angular.module('Site', []);

function AppController($scope, $rootScope, $http) {
  $http.get('data/pages.json').success(function(data) {
    $rootScope.pages = data;
    console.log(data.mobile);
  }) .error(function(data, status){alert(data+ "
"+status);
  });
}

Site.config(function($routeProvider) {
  $routeProvider.when('/page/:id', {
    templateUrl: 'partials/page.html',
    controller: 'RouteController'
  }).otherwise({
    redirectTo: '/page/mobile'
  });
```

```
function RouteController($scope, $rootScope,
$routeParams) {
  // Getting the id from $routeParams
  var id = $routeParams.id;
  $scope.page = $rootScope.pages[id];
}
```

Main module

Controller function to be used in main page

Configuring the routes

Controller function to be used in page.html

page.html


```
<center>
<div class="LoginFormDiv">
<h2>{{page.title}}</h2>

{{page.content}}<br><br>

<img src="{{page.img}}" width="50%" height="50%">

<br><br><br>

</div>
</center>
```

Custom components in AngularJS

Customizing Angular

- Custom directives
- Creating custom filters
- Custom services

Custom Directives

- A directive is an Angular component which is generally used as an attribute for an html tag or at times even as elements
- While customizing Angular we can choose how the directive can be included
- There are four different types of restrictions on the directive
  - Element – denoted with letter ‘E’
  - Attribute – denoted with letter ‘A’
  - Class – denoted with letter ‘C’
  - Comment – denoted with letter ‘M’

Sr. No.	Usage	restrict
1	**<div** my-directive**> **	A (attribute)*
2	**<my-directive></my-directive>**	E (element)
3	**<div** class="my-directive"**> **	C (class)
4	*<!-- directive: my-directive -->*	M (comment)

Structure of Directives

- Custom directives can be defined using the directive() function of a module
- A simple directive may have the following structure:

```
module.directive('myDirective', function() {
 return {
    // directive definition object goes below
                restrict: //define the directive type(e.g. 'E','A','EC' etc)
                link:  function(scope, element, attrs)
            {
                    //logic goes here.
}     };});
```

  - **restrict** : subset of EACM (Element,Attribute,Class,Comment) defines how the directive can be used.
  - **Link** : The directive logic is placed here. This function registers DOM listeners and updates the DOM accordingly. It is executed once the template is cloned.

Custom Directives- as element

- Lets modify the ‘Shop Your Bags Online’ Case study to include custom directives.
- Lets modify app.js file to add custom directive with restriction ‘E’.
- Add the below code to app.js file

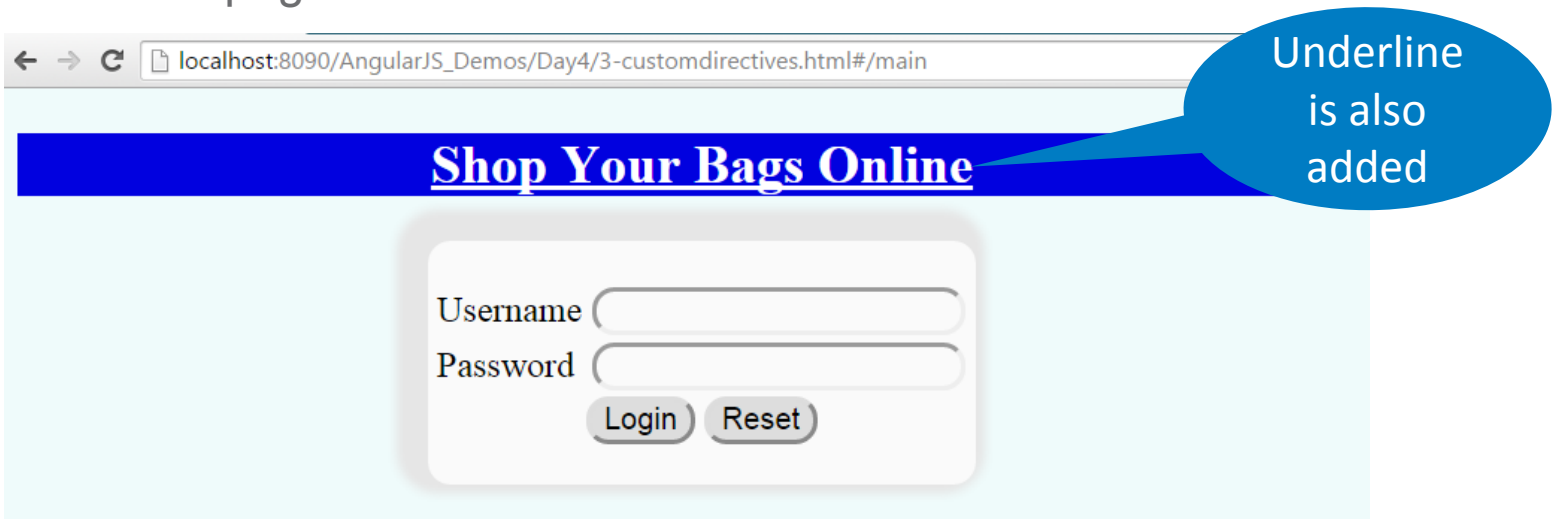
```
//custom directive with restriction 'E'
bagApp.directive('headerline',function(){
    return {
        restrict : 'E',
        scope    : {
          'title'  : '=title'
        },
        template : '<header>'+
                    '<h2> <u>{{ title }} </u></h2>'+
                   '</header>'
      }
```

Custom Directives- as element

- Modify index.html.
- Replace the line in index.html with the below code.

```
<headerline title="'Shop Your Bags Online'"></headerline>
```

- The index.html page will look as below.



Custom Directives- as attribute

- Lets modify the custom directive *headerline* created earlier.

```
bagApp.directive('headerline',function(){

   return {

      restrict : 'EA',

      scope    : {

        'title'  : '=title'

      },       template : '<header>'+

                  '<h2> {{ title }} </h2>'+

                 '</header>' ,

    link: function($scope,element,attrs){
              element.bind("mouseenter", function() {

                                          element.css("font-style", "italic");

                                          });

    }    }});
```

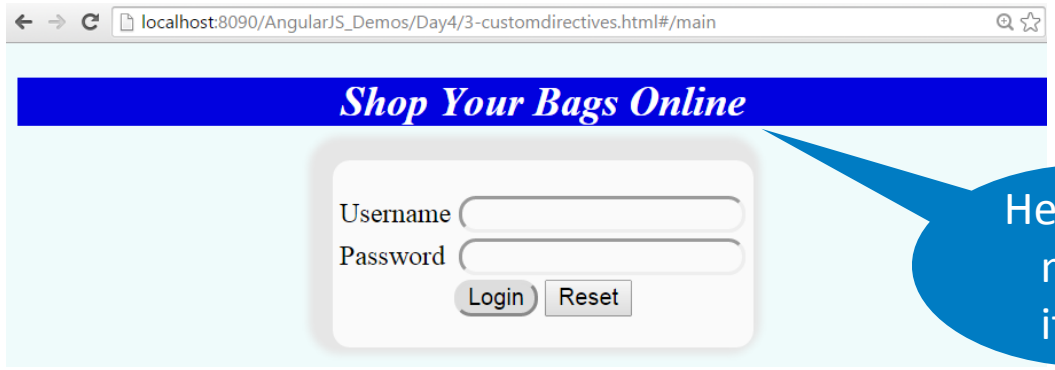
Custom Directives- as attribute

- Modify index.html.
- Replace the line <headerline title="'Shop Your Bags Online'" ></headerline> in index.html with the below code.

```
<headerline title="'Shop Your Bags Online'" mousemove></headerline>
```

- On mouseovering the header, the index.html page will look as below

Added attribute



Header is made italics

Custom Directives- as comment

- Lets add the below code to app.js.

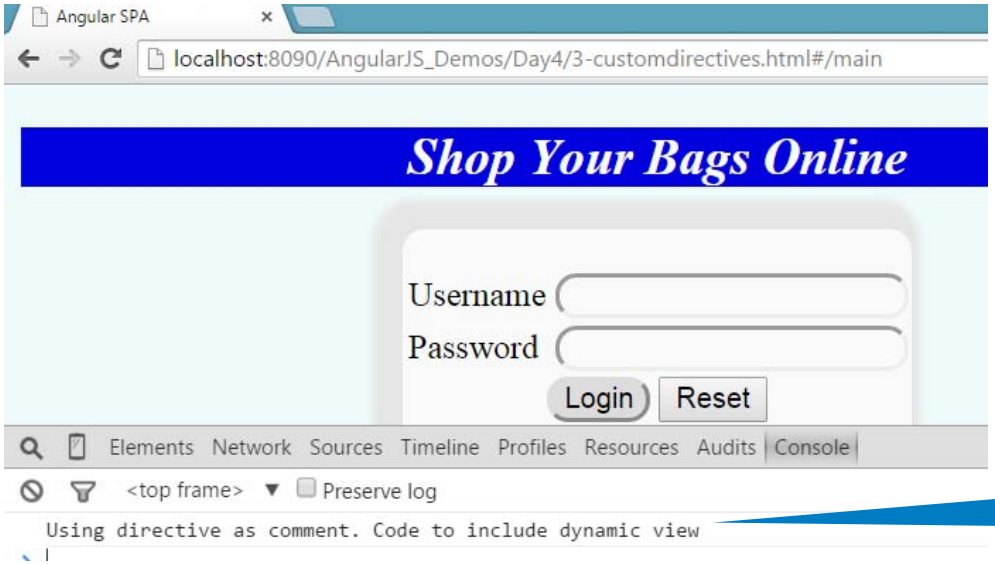
```
//custom directive with restriction 'M'

bagApp.directive('mycommentdir', function() {

return  {

restrict:"M",

link: function(){

console.log("Using directive as comment. Code to include dynamic view");

}

};

});
```

Custom Directives- as comment

- Modify index.html.
- Add the below line before the line <div ng-view class="myclassdir"> in index.html.

```
<!-- directive: mycommentdir -->
```

- On viewing index.html page, the below will be shown.



Comment in the console.

Custom Directives- as class

- Lets add the below code to app.js.

```
//custom directive with restriction 'C'
bagApp.directive('myclassdir', function() {
        return  {
restrict:"C",
link: function($scope,element){
                element.bind("mouseenter", function() {
                        element.css("border-radius", "10px");
                        });
}
};
});
```

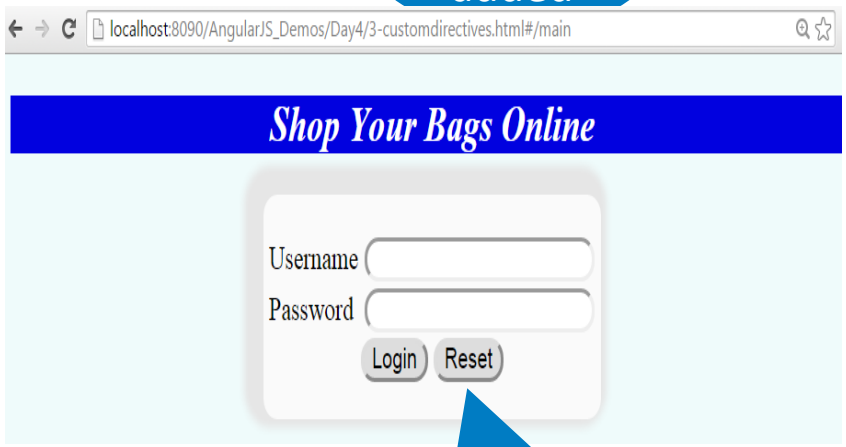
Custom Directives- as class

- Modify the Login template inside index.html.
- Add the below line after the line <input type="submit" class="button" value="Login" ng-click="validate()"/> in the Login template.

```
<input type="reset" class="myclassdir" value="Reset"/>
```

Class added

- On viewing index.html page, the below will be shown.



On mouseover of reset button, the style gets applied

Using Directives for Binding Events

- Lets add the below code to app.js.

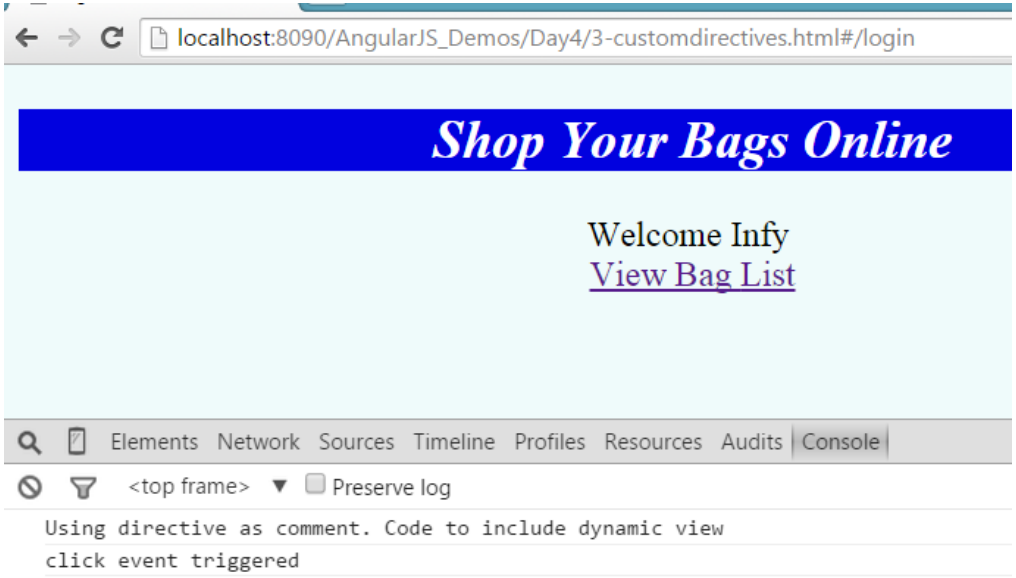
```
//custom directive for binding events on a button
bagApp.directive("buttonclick", function() {
        return  {
                restrict: 'A',
                link: function(s,e) {
                        e.bind("click", function(){
                                console.log('click event triggered');
                        }
                        );
                }
        }
        });
```

Using Directives for Binding Events

- Modify the Login template inside index.html.
- Modify the line <input type="submit" class="button" value="Login“ ng-click="validate()"/> in the Login template as below.

```
<input type="submit" class="button" value="Login" ng-click="validate()" buttonclick/>
```

- On clicking on the Login button, the text is displayed on the console.



Custom Filters

- Creating custom filters are very easy.
- Registering a new filter factory function with the module will create the new filter..
- The factory function needs to return a new filter function. The function has to take the input value as the initial argument.
- All other arguments will be passed in as additional arguments to the filter function.

Custom Filters

- Filters as the name appropriately indicate a reusable logic apart from the functional requirement in a module
- In general programming terminology a broader meaning for a filter would be a non-functional requirement
- The reverse message functionality implemented in the presentation before is a best use case for a filter which could be reused across multiple modules

Custom filter-example

- Lets add the below code to app.js.

```
//custom filter which will reverse the string passed to it
bagApp.filter("reverseString", function() {
        return function(message) {
                return message.split("").reverse().join("");
        }
        });
```

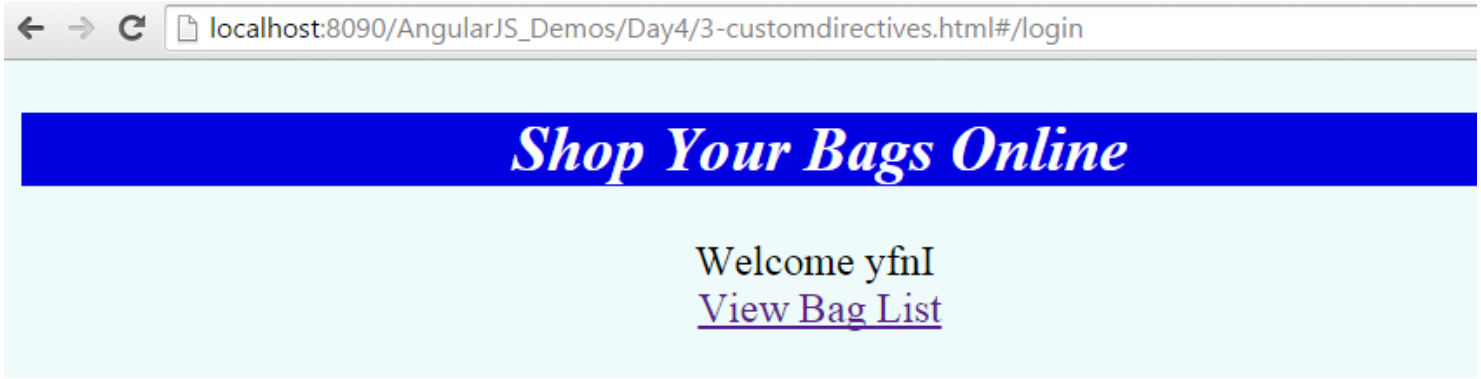
Custom filter example

- Modify the Home template inside index.html.

```
<script type="text/ng-template" id="Home">
        <center>
        Welcome {{username | reverseString}} <br>
        <a href="#/viewbaglist">  View Bag  List  </a>
    </center>
```

Adding custom filter

- On entering valid username and password, the username displayed in the home page will look as below.



Custom Services

- To create a custom service, we use factory method of a root module.
- Where built-in services are always prefixed with $, care should be taken to make sure that custom services do not use $ and rather capitalize the first letter to indicate that they are user defined services.
- Custom services are created when a functional requirement on of a UI application needs to be reused
- Example:
  - Earlier we used $http service to make an Ajax call and retrieve the JSON response for bags
  - This is a functional requirement which can be part of multiple modules in Angular and hence it makes a better design choice to implement it as a custom service
  - The custom services are also implemented in many RESTful Webservice interactions with Angular

Custom Services

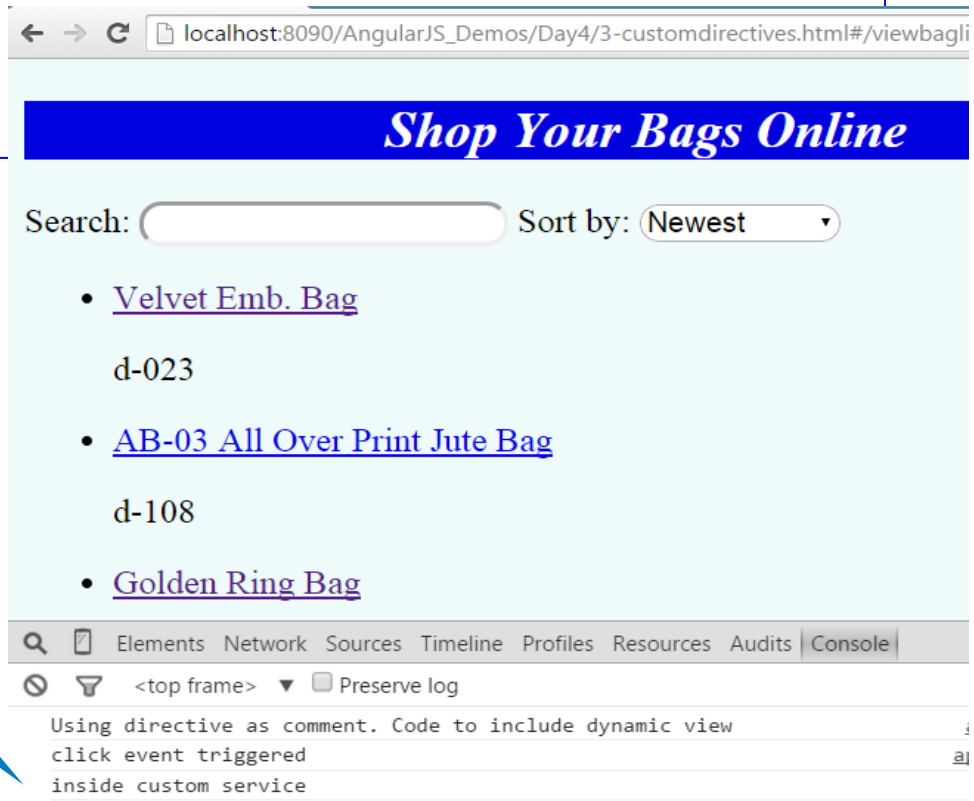
- A custom service Data is created inside the controllers.js file, to fetch JSON response using Ajax

```
//custom service
bagControllers.factory('Data', ['$http', function(http){
        return {
                getData: function() {
                console.log('inside custom service');
                return http.get('data/bags.json').then(function(result) {
                        return result.data;
                }
                );
        }
        };
}]);
```

Custom Services

- Using this custom service in the BagListCtrl controller function.

```
bagControllers.controller('BagListCtrl', ['$scope', '$http', 'Data',
  function ($scope, $http, Data) {
            $scope.bags = Data.getData();
   $scope.orderProp = 'age';
  }]);
```



Inside custom service is printed on the console.

Summary

- Modularizing Angular
- Routing in Angular
- Customizing Angular

Thank You

Infosys® | Building Tomorrow's Enterprise