<AngularJS>

What is AngularJS

- **AngularJS is a framework that extends HTML by teaching it new syntax, making it suitable for developing really great web applications. With AngularJS you can introduce new HTML elements and custom attributes that carry special meaning.**
  - For example, you can create a new HTML element <date-picker/>, say, which adds a date picker widget.
- **Why is it Called AngularJS?**
  - In case you are wondering why the name of the framework is AngularJS: Well, HTML uses angle brackets. Hence, the name!

The Power Features of AngularJS

- **Magical two-way data binding:**
  - The two-way data binding is probably the coolest and most useful feature in AngularJS. Data binding is automatic synchronization of data between your view (HTML) and model (simple JavaScript variables) that means, update the view and model gets updated automatically and update model and view gets update automatically.
- **Routing Support:**
  - Supports Single Page Apps (SPAs). In SPA, we don't want to redirect our users to a new page every time they click on something. Instead, we want to load the content asynchronously on the same page and just change the URL in the browser to reflect it. It makes the user feel as if they are interacting with a desktop app.

The Power Features of AngularJS

- **Templating done right with HTML:**
  - AngularJS uses plain old HTML as the templating language. The workflow becomes much simpler with plain HTML as the templating language, as the designers and developers don't have to depend on each other. Designers can create UIs in usual way and developers can use declarative binding syntax to tie different UI components with data models very easily.
- **Enhanced user experience with form validation:**
  - Forms are the most important part of any CRUD (Create, Read, Update, Delete) app.
- **Teach HTML new syntax with directives:**
  - A directive in AngularJS is what tricks HTML into doing new things that are not supported natively. This is done by introducing new elements/attributes and teaching the new syntax to HTML.

Powered by Google and an active community:

- Whenever we adopt a new technology we always look for a good community support. AngularJS is currently being maintained by the awesome developers at Google. Being open source software the code is released under the MIT license and is available for download at GitHub.

Download and Installation

- **Installing via CDN**
  - Just put the following script in the <head> of your HTML to get started:
  - <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.16/angular.js"></script>
- **Download and Installation**
  - Many companies do it because the CDNs might be blocked in the user's network.
  - Download the script from https://angularjs.org/ to your server and serve it from there
  - <script type="text/javascript" src="js/angular/angular.js"></script>

AngularJS support libs

- **angular.js/angular.min.js**
  - Base file for angularjs.
- **angular-route.js:**
  - Adds routing support. As we will be building a real-world single-page app.
- **angular-animate.js:**
  - This is useful for creating nice CSS3 animations.
- **angular-mocks.js & angular-scenarios.js:**
  - This is useful for testing purposes.
- **angular-resource.js:**
  - Provides better support for interacting with REST APIs.

Components of AngularJS:

- **Model:**
  - The thing that define our data. The model data are simple POJOs (Plain Old JavaScript Objects).
- **View:**
  - This is what the users see when they visit your page, that is to say after the raw HTML template involving directives and expressions is compiled and linked with correct scope.
- **Controller:**
  - The business logic that drives your application.
- **Scope:**
  - A context that holds data models and functions. A controller usually sets these models and functions in the scope. It is glue between model and view.
- **Directives:**
  - Something that teaches HTML new syntax. It extends HTML with custom elements and attributes.
- **Expressions:**
  - Expressions are represented by {{}} in the HTML. They are useful for accessing scope models and functions.
- **Template:**
  - HTML with additional markup in the form of directives (<drop-zone/>) and expressions {{}}.

When Not To Use AngularJS

- **AngularJS was designed with data-driven apps in mind. It's also an excellent choice if your app interacts with a lot of RESTful web services.**
- **But if your app needs heavy DOM manipulations where model data is not the focus, you might want to use a library like jQuery instead.**

Modules, Controllers & Data Binding

- **The reusable components called modules, can be placed in different js files like, controllers.js, directives.js, filters.js , services.js, etc. In the HTML we tell AngularJS to start bootstrapping our app using ‘myApp’ module as the starting point. This is done by writing ng-app='myApp'in the HTML.**
- **Built-in Modules**
  - AngularJS has several built-in modules. The ng module defines many directives, filters, services for our use. We have already used some of the directives from this module such as ng-model, ng-controller, etc. Similarly, you can also create your own module and use it in your app.

Creating Our First Module

- **To create a module, we call the angular.module() function.**
  - The first argument to the function is the module name.
  - The second argument is an array that specifies the additional modules upon which this module depends. If there's no dependency you just pass an empty array.
    - angular.module('firstModule',[]);

      //defines a module with no dependencies
    - angular.module('firstModule', ['moduleA','moduleB']);

      //defines a module with 2 dependencies
    - Once you have a module, you can attach different components to it. Controller, Directives, Services, Factories, etc.

- **Define module with variable.**
  - var firstModule=angular.module('firstModule',[]);

```
    //define a module
firstModule.controller('FirstController',function($scope){
    // register a controller
    //your rocking controller
});
firstModule.directive('FirstDirective',function(){
    // register a directive
    return {
    };
});
```

Modular Programming Best Practices

- **Modularization by layers**
  - The following five source files, each representing a module:
    - app.js
    - controllers.js
    - directives.js
    - filters.js
    - services.js
  - This is called modularization by layers. Each type of component goes into a particular module. Finally, the main app module is defined in app.js, which depends on the other four modules. This is specified by passing a dependency list to angular.module() as the second argument. You just refer to this main module in ng-app in HTML.

Modular Programming Best Practices

- **Modularization by features**
  - When your app grows in terms of size and complexity it may not be good to modularize by layer. Instead you can achieve this by feature.
    - For example, your website might have a login module, a comment module and a posts module. All these modules exist independently and are loosely coupled. Usually, you'd create a separate directory for each module and place all the relevant JavaScript files inside the relevant folders.

Modularization by features

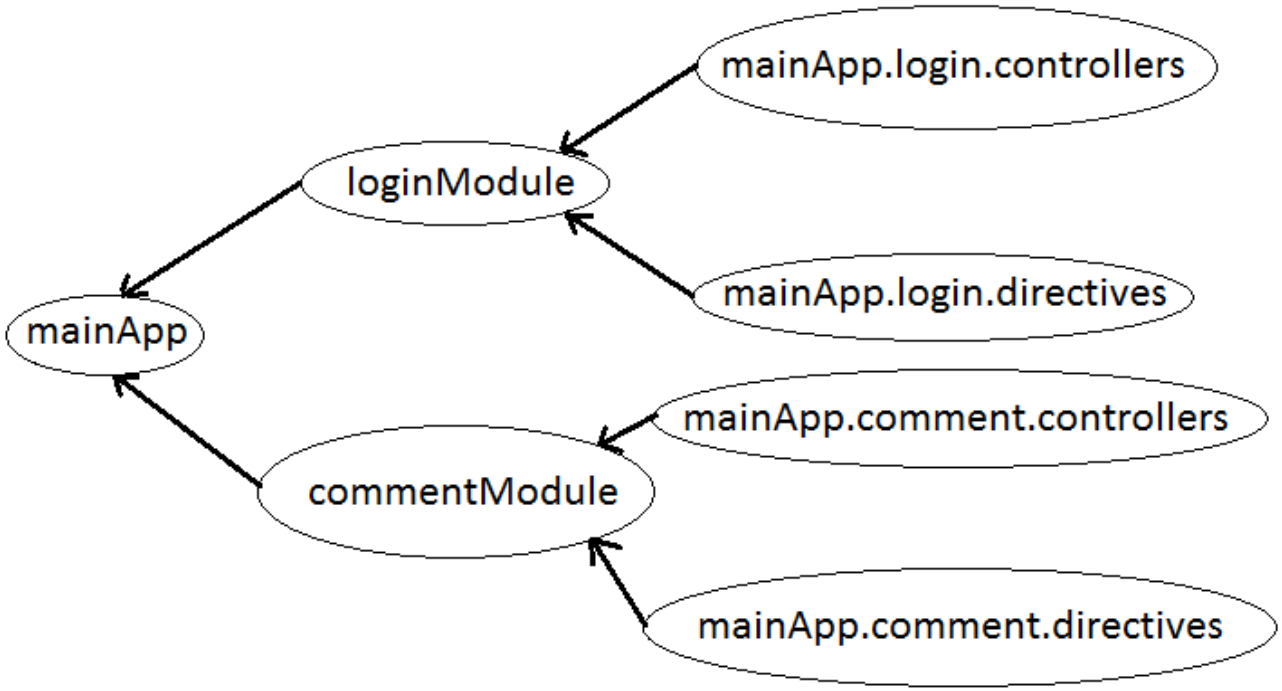
```
/app
    /img -- application level images
    /css -- application level css
    /js
        app.js -- the main app module
    /modules
        /login
            /js
                controllers.js  --controllers for login module
                directives.js   --directives for login module
            /views -- views for login module
            /css
            /img
            loginModule.js -- Main login module definition
        /comment
            /js
                controllers.js  --controllers for login module
                directives.js   --directives for login module
            /views -- views for comment module
            /css
            /img
            commentModule.js -- Main comment module definition

        ...

        ...
    index.html
```

Modularization by features

- **Inside each of these files, we define our modules. For the login module, the definitions are as follows:**
  - /app/modules/login/js/controllers.js
    angular.module('mainApp.login.controllers',[]);
  - /app/modules/login/js/directives.js
    angular.module('mainApp.login.directives',[]);
  - /app/modules/login/loginModule.js
    angular.module('loginModule',['mainApp.login.controllers', 'mainApp.login.directives']);
  - For the comment module, the definitions are as follows:
  - /app/modules/comment/js/controllers.js
    angular.module('mainApp.comment.controllers',[]);
  - /app/modules/comment/js/directives.js
    angular.module('mainApp.comment.directives',[]);
  - /app/modules/comment/commentModule.js
    angular.module('commentModule',['mainApp.comment.controllers', 'mainApp.comment.directives']);
  - The main module is defined in /app/app.js like this:
    angular.module('mainApp',['loginModule','commentModule']);
  - Finally, inside index.html we bootstrap the app by writing:
    ng-app='mainApp'

Modularization by features

- **Using Dot Notation in Module Names**
  - The dot notation used in the module names is there simply to mimic namespaces, and doesn't hold any special meaning in AngularJS.



Controllers

- **The Role of a Controller**
  - The task of a controller is to augment the scope by attaching models and functions to it that are subsequently accessed in the view. A controller is nothing but a constructor function which is instantiated by AngularJS when it encounters ng-controller directive in HTML.
    - Attaching Properties and Functions to Scope

```
angular.module('myApp.controllers', []).controller('RegisterController',
function($scope){
    $scope.changeName=function(){   //
        alert("Hi "+$scope.firstName+" "+$scope.lastName);
        $scope.firstName= prompt("", "Suresh");
        $scope.lastName= prompt("", "Kane");
    }
});
```

What Controller should do/ should not do:

- **What a controller should not do:**
  - No DOM manipulation. This should be done in directives.
  - Don't format model values in controllers. Filters exist for this purpose.
  - Don't write repeatable code in controllers. Rather encapsulate them in services.
- **Controllers should be used to:**
  - Set the initial state of a scope by attaching models to it.
  - Set functions on the scope that perform some tasks.
- **Naming Controllers**
  - use a full name, like **demoController** and not **demoCtrl**. This makes your code more readable.

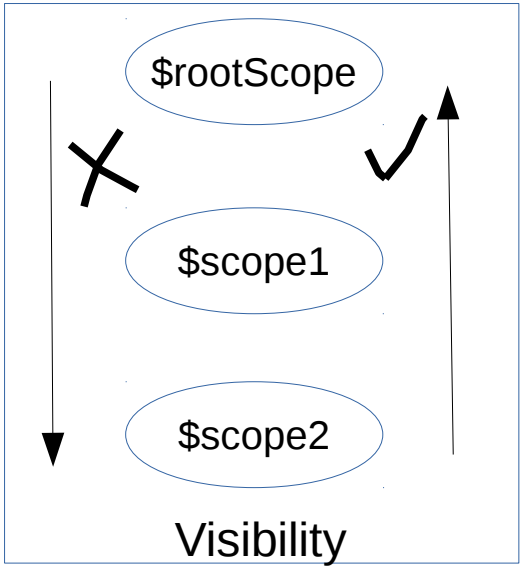
AngularJS Scope

- **Essentially, a scope is nothing but a plain old JavaScript object, and that is just a container for key/value pairs, as follows:**
  - var myObject={name:'AngularJS', creator:'Misko'}
- **A scope (like other objects) can have properties and functions attached to it. The only difference is that we don't usually construct a scope object manually. Rather, AngularJS automatically creates and injects one for us.**
- **In the Angular world, a scope object is useful for holding model values, which are presented by the view.**
- **Every AngularJS application has at least one scope called $rootScope. This is created as a result of attaching the ng-app directive to any HTML element.**
- **when you attach ng-controller to any element, it creates a new child scope, which prototypally inherits from the $rootScope.**
- **$rootScope is the parent of all scopes. $rootScope has a function called $new() that is used to create child scopes.**

Inheritance of Scopes

- **You can nest scopes by using an ng-controller directive inside another ng-controller . Take a look at following code.**

```
<div ng-app=””> <!-- creates a $rootScope -->
    <div ng-controller="OuterController"> <!--creates a scope (call it scope 1) that inherits from $rootScope-->
        <div ng-controller="InnerController"> <!-- Creates a child scope (call it scope 2) that inherits from scope 1 -->
        </div>
    </div>
</div>
```



- **Here is how AngularJs handles scope hierarchies.**
  - AngularJS finds ng-app and therefore creates a $rootScope object.
  - It encounters ng-controller and finds that it points to OuterController . So, it calls $rootScope.$new(), which creates a child scope(let's call it $scope1).
  - When AngularJS encounters another ng-controller directive which points to InnerController, it creates another child scope(call it $scope2).
- **The inner scope always has access to the properties defined by the outer scope.**

Filters

- **The most common usage of filters can be found in view templates. Here is how we apply a filter to an expression:**
  - {{ countryCode | uppercase }}
- **Here, countryCode is the scope model and uppercase is the name of the filter (in this example, uppercase is a built-in filter in AngularJS). If you have an expression you'd like to display in uppercase, you simply need to add the uppercase filter to it.**
- **Expression is separated from the filter by a vertical pipe character (|).**
- **Filters can take arguments too! Let's take a look at another built-in filter in AngularJS, the date filter:**
  - {{ currentDate | date:'yyyy-MM-dd' }}
- **You can make your own filters.**
  - angular.module('myFilter', []).filter('signFilter', function(){

```
    return function(is_true){
        return is_true ? '\u2713' : '\u2718';
    }
});
```

Directives

- **The Directive is one of the coolest features of AngularJS. A directive is something that attaches special behavior to a DOM element and teaches it awesome tricks.**
- **There are many built in directives, we already used some of them like ng-app, ng-controller, ng-model, etc.**
- **You can create your own directives.**
- **Directives come in three type, as follows:**
  - As elements(tags): <date-picker></date-picker>
  - As attributes: <input type="text" date-picker/>
  - As classes: <input type="text" class="date-picker"/>
- **Here is one example of making a custom directive.**
  - angular.module('myApp', []).directive('helloWorld', function() {

```
    return {
        restrict: 'AEC',
        replace: true,
        template: '<h3>Hello, World!</h3>'
    };
});
```

The Watchers in AngularJS

- **A watcher monitors model changes and takes action in response. The $scope object has a function $watch() that's used to register a watcher.**
  - $scope.$watch('someModel', function(newValue, oldValue){
    if(condition){
    //do something
    }
    }
- **The first parameter is watchExpression .This can be the model name or a function. The second parameter is a callback function that's called whenever AngularJS detects a change in the model value or the return value of the passed function.**
- **This callback function takes two parameters. The first is the new value of the model, while the second is the old value.**

Multiple Views and Routing

- **Every web app you'll build in future will have different views associated with it.**
  - To display contents of different section of web site.
  - Can be separate views for different CRUD operations.
- **As we're developing Single Page Apps, it's important to map each route to a particular view(template).**
- **This logically divides the app into manageable parts and keeps it maintainable.**
- **Each AngularJS route is composed of a URL, a template, and a controller.**
  - When someone first lands on web page, it's a full-page load. Subsequently, when the user clicks a link and navigates to a different URL, the route changes and content of the route are dynamically loaded into the page via an AJAX call.
    - Advantage - less network traffic.

Displaying HTML Content with <ng-view> Hashbang Mode

```
<html>
        :
    <ul class="menu">
      <li><a href="#/view1">view1</a>
    </li>
      <li><a href="#/view2">view2</a>
    </li>
    </ul>
        :
     <ng-view></ng-view>

</html>
```

The route name is prefixed with # in the URL. For example, when you click on hyperlink view1, the URL in the address bar changes to index.html#/view1. This is called hashbang mode, which is the default in AngularJS.

- **The most important point to note here is that our main view, index.html, has an element <ng-view>.**
- **This acts as a container.**
- **When the user clicks on a link (for example #/view1), the route changes to index.html#/view1**
  - AngularJS searches for the template mapped with the current route.
  - Once the template is found the content is fetched and included in <ng-view> and compiled against the $scope so that the expressions ({{}}) are evaluated.
  - The same $scope is passed as an argument to the controller associated with the particular route.
  - Note that there should be only one <ng-view> in your page.
- **You can also use ng-view as an attribute or class.**
  -
  - <div ng-view>

Creating Multiple Views

- **Views in a web app are regular HTML files. As a result, we need to create a separate HTML file for each view. These Views are Just Partials. Please note that the views view1.html and view2.html are not complete HTML files—they're just partials.**
- **The syntax to configure the routes is as follows:**
  - angular.module('myApp').config(function($routeProvider) {
    $routeProvider.when('/view1', {
    controller: 'Controller1',
    templateUrl: 'partials/view1.html',
    }).when('/view2', {
    controller: 'Controller2',
    templateUrl: 'partials/view2.html',
    });
    });

Understanding Routing

- **angular.module() has a function config(), which lets us configure a module. We inject $routeProvider as an argument to the function.**
- **$routeProvider is a built-in AngularJS object. When we declare a function with the parameter $routeProvider, AngularJS automatically injects a $routeProvider object while calling it—this is another example of Dependency Injection.**
- **The when() function of $routeProvider can be used to configure your routes. This particular function takes two parameters: The first parameter is the route name, and is a string representing the route; the second parameter is the route definition object. The most commonly use the following two properties:**
  - controller: The controller which will be associated with the view. After all, we need a scope against which the template will be compiled.
  - templateUrl: The HTML template to be used.

Using $routeParams in the Controller

- **Sometimes it's necessary to pass some parameters to the controller associated with your view.**
  - userId, productId, etc.
- **These parameters passed in the route are called route parameters and are exposed by a integrated AngularJS service called $routeParams.**
  - angular.module('myApp').config(function($routeProvider, $locationProvider){
    $routeProvider.when('/view1',{
    controller:'Controller1',
    templateUrl:'/partials/view1.html'
    }).when('/view2/:firstname/:lastname',{
    controller: 'Controller2',
    templateUrl: '/partials/view2.html'
    }).otherwise({redirectTo:'/view1'});
    $locationProvider.html5Mode(true);
    });

Using $routeParams in the Controller

View-1

- ** **

  First name: <input type="text" ng-model ="firstname"/>

  Last name: <input type="text" ng-model ="lastname"/>

  <button ng-click="loadView2()">Load View2</button>

- ** **

View-2

- ** **

  First name: {{firstname}}

  Last name: {{lastname}}

- ** **

```
angular.module('myApp.controllers', []).controller('Controller1', function($scope,$location){
    $scope.loadView2=function(){
        $location.path('/view2/'+ $scope.firstname+'/'+ $scope.lastname);
    }}).controller('Controller2',function($scope,$routeParams){

$scope.firstname=$routeParams.firstname;
$scope.lastname=$routeParams.lastname;
});
```

Service

- **An AngularJS service encapsulates some specific business logic and exposes an API, to be used by other components. For example, the $http service encapsulates the logic required to interact with REST back-ends.We ask AngularJS to inject these services into our controllers so that we can utilize them. service is an injectable type.**
- **AngularJS Services are Always Singletons**
  - It's important to note that AngularJS services are always singletons. This means that, once AngularJS constructs a service object, the same instance is reused throughout your app.
- **Take a look at the following snippet:**
  - ```
    angular.module('myApp').controller('testController', function($http, $timeout){
      //use $http, $timeout here
    });
    ```
  - In the above snippet our controller testController declares dependency on two services: $http and $timeout. So, when the controller is instantiated, AngularJS passes the two services to it. Similarly, you can create your own services that encapsulate the business logic specific to your apps.
    ```

Define Custom Service:

service.html

```
<!DOCTYPE html>
<html ng-app="myApp">
   <head>
      <title ng-bind="title"></title>
      <script
src="../angular/js/angular.js"></script>
   </head>
   <body ng-controller="TestController">
      Date: <span>{{date |
date:"dd/MM/yyyy"}}</span>
   </body>


   <script
src="servfactprv/app.js"></script>
   <script
src="servfactprv/services.js"></script>
   <script
src="servfactprv/controllers.js"></script>
</html>
```

app.js

```
angular.module('myApp', ['myApp.controllers',
'myApp.services']);
```

Services.js

```
angular.module('myApp.services',
[]).service('helloService', function(){

    this.today = function(){ // define an instance
method

            return new Date();

     }

});
```

Controllers.js

```
angular.module('myApp.controllers',[])

      .controller('TestController', function($scope,
helloService){

$scope.date=helloService.today();

});
```

Important points to remember about services:

- A service is registered using the service() function of **angular.module()**.
- The second argument to service() is a **constructor function**. When we ask for the service as a dependency, AngularJS creates an object from this constructor function and injects it.
- A service is a **singleton**. AngularJS instantiates the service object only once and all other components share the same instance.
- Services are lazily instantiated. This means AngularJS instantiates the service only when it encounters a component that declares **dependency** on the service.

Factory:

- A factory is another injectable type. Effectively it's the same as a service. You register a factory by calling the factory() function on angular.module(). For example, we can implement helloService as a factory as follows:
- ```
  angular.module('myApp').factory('myFactory',function(){
      return {
            getCurrentDate: function(name){
            return new Date();
        }
      }
  });
  ```
- The above factory can be used in a controller in exactly the same way as services.
  ```

AngularJS Forms

- **What's a form? It's just a container that groups several related input controls together. A control is an HTML element (such as input, select, textarea) that lets users enter data. You already know AngularJS provides us with the very useful ngModel directive that keeps the input control value and scope model in sync.**
- **Forms in AngularJS provides data-binding and validation of input controls.**
- **Input Controls**
  - Input controls are the HTML input elements:
    - input elements
    - select elements
    - button elements
    - textarea elements
- **Data-Binding**

  Input controls provides data-binding by using the ng-model directive.

```
<input type="text" ng-model="firstname">
<script>
    var app = angular.module('myApp', []);
    app.controller('formCtrl', function($scope) {
        $scope.firstname = "John";
    });
</script>
```

Form Validation

- **AngularJS offers client-side form validation.**
  - 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
  - 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

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

CSS Classes:

- **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


Thank
You!