Desarrollo Web Front-End con AngularJS

Important AngularJS Components and their usage

- **ng-app** attribute sets the scope of a module
- **ng-controller** attribute applies a controller to the view
- **$scope** service passes data from controller to the view
- **angular.module()** defines a module
- **$filter** service uses a filter

Important AngularJS Components and their usage

- Module.controller() defines a controller
- Module.directive() defines a directive
- Module.filter() defines a filter
- Module.service() or **Module.factory()** or Module.provider() defines a service
- Module.value() defines a service from an existing object Module

Bootstrapping AngularJS application

- Bootstrapping in HTML:

** **

- Manual bootstrapping:

angular.bootstrap(document,[“moduleName”])

Expressions

- {{ 4+5 }} -> yields 9
- {{ name }} -> Binds value of name from current

scope and watches for changes to name

- {{ ::name }} -> Binds value of name from current

scope and doesn’t watch for change (Added in AngularJS 1.3)

Most used built-in directives

- **ng-app:** To bootstrap the application
- **ng-controller:** To set a controller on a view
- **ng-view:** Indicates the portion of the page to be updated when route changes
- **ng-show / ng-hide:** Shows/hides the content within the directive based on boolean equivalent of value assigned
- **ng-if:** Places or removes the DOM elements under this directive based on boolean equivalent of value assigned

Most used built-in directives

- **ng-model:** Enables two-way data binding on any input controls and sends validity of data in the input control to the enclosing form
- **ng-class:** Provides an option to assign value of a model to CSS, conditionally apply styles and use multiple models for CSS declaratively
- **ng-repeat:** Loops through a list of items and copies the HTML for every record in the collection
- **ng-options:** Used with HTML select element to render options based on data in a collection

Most used built-in directives

- **ng-href:** Assigns a model as hyperlink to an anchor element
- **ng-src:** Assigns a model to source of an image element
- **ng-click:** To handle click event on any element
- **ng-change:** Requires ng-model to be present along with it. Calls the event handler or evaluates the assigned expression when there is a change to value of the model
- **ng-form:** Works same as HTML form and allows nesting of forms

Most used built-in directives

- **ng-non-bindable:** Prevents AngularJS from compiling or binding the contents of the current DOM element
- **ng-repeat-start** and **ng-repeat-end:** Repeats top-level attributes
- **ng-include:** Loads a partial view
- **ng-init:** Used to evaluate an expression in the current scope
- **ng-switch** conditionally displays elements
- **ng-cloak** to prevent Angular HTML to load before bindings are applied.

Dependency Injection

- AngularJS has a built-in dependency injector that keeps track of all components (services, values, etc.) and returns instances of needed components using dependency injection.
- Angular's dependency injector works based on names of the components.

```
myModule.controller("MyController", ["$scope",
"$window", "myService",
function($scope, $window, myService){
}]);
```

Factory

- A factory is a function that returns an object.
- The members that are not added to the returning object, remain private

to the factory.

- The factory function is executed once and the result is stored.
- Whenever an application asks for a factory, the application returns the same object.
- This behavior makes the factory a singleton.

Run block

```
angular.module("myModule").run(function(<any
services, factories>){
console.log("Application is configured. Now inside run
block");
});
```

Run block is used to initialize certain values for further use, register global events and anything that needs to run at the beginning of the application. Run block is executed after config block, and it gets access to services, values and factories. Run block is executed only once in the lifetime of an Angular application.

Filters

```
angular.module("myModule").
filter("dollarToRupeee", function(){
return function(val){
return "Rs. " + val * 60;
};
});
```

Usage:

```
<span>{{price | dollarToRupee}}</span>
```

- Filters are used to extend the behavior of binding expressions and directives.
- They are used to format values or to apply certain conditions.
- They are executed whenever the value bound in the binding expression is updated.

Registering services

Angular provides us three ways to create and register our own Services – using Factory, Service, and Provider. They are all used for the same purpose. Here's the syntax for all the three:

- Service: module.service('serviceName', function);
- Factory: module.factory('factoryName', function);
- Provider: module.provider('providerName', function);

Some useful utility functions

- **angular.copy -** Creates a deep copy of source
- **angular.extend -** Copy methods and properties from one object to another
- **angular.element -** Wraps a raw DOM element or HTML string as a jQuery element
- **angular.equals -** Determines if two objects or two values are equivalent

Some useful utility functions

- **angular.forEach -** Enumerate the content of a collection
- **angular.toJson -** Serializes input into a JSON-formatted string
- **angular.fromJson -** Deserializes a JSON string
- **angular.identity -** Returns its first argument
- **angular.isArray -** Determines if a reference is an Array
- **angular.isDate -** Determines if a value is a date

Some useful utility functions

- **angular.isDefined -** Determines if a reference is defined
- **angular.isElement -** Determines if a reference is a DOM element
- **angular.isFunction -** Determines if a reference is a Function
- **angular.isNumber -** Determines if a reference is a Number
- **angular.isObject -** Determines if a reference is an Object
- **angular.isString -** Determines if a reference is a String
- **angular.isUndefined -** Determines if a reference is undefined
- **angular.lowercase -** Converts the specified string to lowercase
- **angular.uppercase -** Converts the specified string to uppercase

$http

$http is Angular's wrapper around XmlHttpRequest. It provides a set of high level APIs and a low level API to talk to REST services. Each of the API methods return $q promise object. Following are the APIs exposed by $http:

- **$http.$get(url):** Sends an HTTP GET request to the URL specified
- **$http.post(url, dataToBePosted):** Sends an HTTP POST request to the URL specified
- **$http.put(url, data):** Sends an HTTP PUT request to the URL specified
- **$http.patch(url, data):** Sends an HTTP PATCH request to the URL specified

$http

• **$http.delete(url):** Sends an HTTP DELETE request to the URL specified

• **$http(config):** It is the low level API. Can be used to send any of the above request types and we can also specify other properties to the request. Following are the most frequently used config options:

o **method:** HTTP method as a string, e.g., 'GET', 'POST', 'PUT', etc.

o **url:** Request URL

o **data:** Data to be sent along with request

o **headers:** Header parameters to be sent along with the request

o **cache:** caches the response when set to true

$http

Following is a small snippet showing usage of $http:

```
$http.get('/api/data').then(function(result){
return result.data;
}, function(error){
return error;
});
```

Using AngularJS functions

Wherever possible, use AngularJS versions of JavaScript functionality. So instead of setInterval, use the $interval service. Similarly instead of setTimeout use the $timeout service. It becomes easier to mock them or write unit tests . Also make sure to clean it up when you have no use for it. Use the $destroy event to do so:

```
$scope.$on("$destroy", function (event) {
$timeout.cancel(timerobj);
});
```

Deferred and Promise

The $q service provides deferred objects/promises.

- **$q.all([array of promises]) -** creates a Deferred object that is resolved when all of the input promises in the specified array are resolved in future
- **$q.defer() -** creates a deferred object with a promise property that can be passed around applications, especially in scenarios where we are integrating with a 3rd-party library

Deferred and Promise

$q.reject(reason) - Creates a promise that is resolved as rejected with the specified reason. The return value ensures that the promise continues to the next error handler instead of a success handler.

- **deferredObject.resolve -** Resolves the derived promise with the value
- **deferredObject.reject -** Rejects the derived promise with the reason and triggers the failure handler in the promise.

HTTP Interceptors

- Any HTTP request sent through $http service can be intercepted to perform certain operation at a given state.
- The state may be one of the following: before sending request, on request error, after receiving response and on response error.

HTTP Interceptors

```
myModule.config(function ($provide) {
$provide.factory('myHttpInterceptor', function () {
return {
request: function (req) {
//logic before sending request
},
response: function (res) {
//logic after receiving response
},
requestError: function () {
//logic on request error
},
responseError: function () {
//logic on response error
}
};
});
});
```