AngularJS $http

- ❑ AngularJS $http is a core service for reading data from web servers using AJAX using browser's XMLHttpRequest object or via JSONP.
- ❑ The $http API is based on the deferred/promise APIs exposed by the $q service.
- ❑ Since the returned value of calling the $http function is a promise, you can also use the then method to register callbacks, and these callbacks will receive a single argument – an object representing the response.
- ❑ We can use either $http() or one of its shortcut methods:

  - ✓ $http.get
  - ✓ $http.head
  - ✓ $http.post
  - ✓ $http.put
  - ✓ $http.delete
  - ✓ $http.jsonp
  - ✓ $http.patch

```
$http.get('/someUrl').
  success(function(data, status, headers, config) {
  }).
  error(function(data, status, headers, config) {
  });
```

```
<table ng-controller=”customerController”>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
<script>
function customersController($scope,$http){
  $http.get("url")
   .success(function(response) {$scope.names = response;});
}
</script>
```

```
$http(config)
```

For better control on Ajax request, use config object with $http service as follows:

```
var req = {
 method: 'POST',
 url: 'http://www.example.com',
 headers: {
   'Content-Type': undefined
 },
 data: { test: 'test'},
}

$http(req).success(function(){...}).error(function(){...});
```

Object describing the request to be made and how it should be processed. The object has following properties:

- **method** – {string} – HTTP method (e.g. 'GET', 'POST', etc)
- **url** – {string} – Absolute or relative URL of the resource that is being requested.
- **params** – {Object.<string|Object>} – Map of strings or objects which will be turned to?key1=value1&key2=value2 after the url. If the value is not a string, it will be JSONified.
- **data** – {string|Object} – Data to be sent as the request message data.
- **headers** – {Object} – Map of strings or functions which return strings representing HTTP headers to send to the server. If the return value of a function is null, the header will not be sent.
- **cache** – {boolean|Cache} – If true, a default $http cache will be used to cache the GET request, otherwise if a cache instance built with $cacheFactory, this cache will be used for caching.
- **timeout** – {number|Promise} – timeout in milliseconds, or promise that should abort the request when resolved.
- **withCredentials** - {boolean} - whether to set the withCredentials flag on the XHR object.
- **responseType** - {string}

What it returns?

- ❑ It returns a promise object with the standard then method and two http specific methods: success and error.
- ❑ The **then** method takes two arguments a success and an error callback which will be called with a response object.
- ❑ The **success** and **error** methods take a single argument - a function that will be called when the request succeeds or fails respectively.
- ❑ The arguments passed into these functions are destructured representation of the response object passed into the then method.
- ❑ The response object has these properties:

  - ✓ **data** – {string|Object} – The response body transformed with the transform functions.
  - ✓ **status** – {number} – HTTP status code of the response.
  - ✓ **headers** – {function([headerName])} – Header getter function.
  - ✓ **config** – {Object} – The configuration object that was used to generate the request.
  - ✓ **statusText** – {string} – HTTP status text of the response.

Dependency Injection

- Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.
- The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.
- DI is pervasive throughout Angular.
- Components such as services, directives, filters, and animations are defined by an injectable factory method or constructor function.
- Controllers are defined by a constructor function, which can be injected with any of the "service" and "value" components as dependencies.

Controllers

Controllers are "classes" or "constructor functions" that are responsible for providing the application behavior that supports the declarative markup in the template. The recommended way of declaring Controllers is using the array notation:

```
someModule.controller('MyController',
     ['$scope', 'dep1', 'dep2', function($scope, dep1, dep2) {
  ...
  $scope.aMethod = function() {
    ...
  }
  ...
}]);
```

Factory Methods

The way you define a directive, service, or filter is with a factory function. The factory methods are registered with modules. The recommended way of declaring factories is:

```
angular.module('myModule', [])
.factory('serviceId', ['depService', function(depService) {
  // ...
}])
.directive('directiveName', ['depService', function(depService) {
  // ...
}])
.filter('filterName', ['depService', function(depService) {
  // ...
}]);
```

The simplest way to get hold of the dependencies is to assume that the function parameter names are the names of the dependencies.

```
someModule.controller('MyController', function($scope, greeter) {
});
```

Note: If you plan to minify your code, your service names will get renamed and break your app.

In the above example, $scope and greeter are two services which need to be injected into the function.

Services

- Application developers are free to define their own services by registering the service's name and service factory function, with an Angular module.
- The service factory function generates the single object or function that represents the service to the rest of the application.
- The object or function returned by the service is injected into any component (controller, service, filter or directive) that specifies a dependency on the service.

Typically you use the Module's factory API to register a service:

Note: You are not registering a service instance, but rather a factory function that will create this instance when called.

```
app.factory('TimeService',
 // factory function that returns an instance of
 //  service instance
 function () {
        return {
            'hours': new Date().getHours(),
            'minutes': new Date().getMinutes(),
            'seconds': new Date().getSeconds(),
             getTime : function() {
                return new Date().toString();
            }
        }
 }
);
```

```
factory (name, providerFunction)
```

Param	Type	Details
name	string	service name
providerFunction	Function	Function for creating new instance of the service.

```
service(name, constructor)
```

Param	Type	Details
name	string	service name
constructor	Function	A constructor function that will be instantiated.

Service provided by Angular

The following are services provided by AngularJS.

Name	Description
$anchorScroll	When called, it checks the current value of $location.hash() and scrolls to the related element, according to the rules specified in the Html5 spec.
$animate	The $animate service provides rudimentary DOM manipulation functions to insert, remove and move elements within the DOM, as well as adding and removing classes. This service is the core service used by the ngAnimate $animator service which provides high-level animation hooks for CSS and JavaScript.
$cacheFactory	Factory that constructs Cache objects and gives access to them.
$compile	Compiles an HTML string or DOM into a template and produces a template function, which can then be used to link scope and the template together.
$controller	$controller service is responsible for instantiating controllers.
$document	A jQuery or jqLite wrapper for the browser's window.document object.
$exceptionHandler	Any uncaught exception in angular expressions is delegated to this service. The default implementation simply delegates to $log.error which logs it into the browser console.
$filter	Filters are used for formatting data displayed to the user.

$http	The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.
$interpolate	Compiles a string with markup into an interpolation function. This service is used by the HTML $compile service for data binding.
$interval	Angular's wrapper for window.setInterval. The fn function is executed every delay milliseconds.
$locale	$locale service provides localization rules for various Angular components. As of right now the only public api is:
$location	The $location service parses the URL in the browser address bar (based on the window.location) and makes the URL available to your application. Changes to the URL in the address bar are reflected into $location service and changes to $location are reflected into the browser address bar.
$log	Simple service for logging. Default implementation safely writes the message into the browser's console (if present).
$parse	Converts Angular expression into a function.
$rootElement	The root element of Angular application. This is either the element where ngApp was declared or the element passed into angular.bootstrap. The element represents the root element of application. It is also the location where the application's $injector service gets published, and can be retrieved using $rootElement.injector().
$rootScope	Every application has a single root scope. All other scopes are descendant scopes of the root scope. Scopes provide separation between the model and the view, via a mechanism for watching the model for changes. They also provide an event emission/broadcast and subscription facility.

$timeout	Angular's wrapper for window.setTimeout. The fn function is wrapped into a try/catch block and delegates any exceptions to $exceptionHandler service.
$window	A reference to the browser's window object. While window is globally available in JavaScript, it causes testability problems, because it is a global variable. In angular we always refer to it through the $window service, so it may be overridden, removed or mocked for testing.

Using Restful Services

- The RESTful functionality is provided by Angular in the **ngResource** module, which is distributed separately from the core Angular framework.
- Download angular-resouce.js and include in your project
- The ngResource module provides interaction support with RESTful services via the $resource service.

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

$resource

- A factory which creates a resource object that lets you interact with RESTful server-side data sources.
- The returned resource object has action methods which provide high-level behaviors without the need to interact with the low level $httpservice.

```
$resource(url, [paramDefaults], [actions], options);
```

Param	**Details**
url	A parameterized URL template with parameters prefixed by : as in /user/:username. If you are using a URL with a port number (e.g. http://example.com:8080/api), it will be respected.
paramDefaults (optional)	Default values for url parameters. These can be overridden in actions methods. If any of the parameter value is a function, it will be executed every time when a param value needs to be obtained for a request (unless the param was overridden). Given a template /path/:verb and parameter {verb:'greet', salutation:'Hello'} results in URL /path/greet?salutation=Hello. If the parameter value is prefixed with @ then the value for that parameter will be extracted from the corresponding property on the data object (provided

	when calling an action method). For example, if the defaultParam object is {someParam: '@someProp'} then the value of someParam will bedata.someProp.
actions (optional)	Hash with declaration of custom actions that should extend the default set of resource actions. The declaration should be created in the format of $http.config: {action1: {method:?, params:?, isArray:?, headers:?, ...}, action2: {method:?, params:?, isArray:?, headers:?, ...}, ...} Where: action – {string} – The name of action. This name becomes the name of the method on your resource object. method – {string} – Case insensitive HTTP method (e.g. GET, POST, PUT, DELETE, JSONP, etc). params – {Object=} – Optional set of pre-bound parameters for this action. If any of the parameter value is a function, it will be executed every time when a param value needs to be obtained for a request (unless the param was overridden). isArray – {boolean=} – If true then the returned object for this action is an array, seereturns section. .

It returns resource "class" object with methods for the default set of resource actions optionally extended with custom actions. The default set contains these actions:

```
{ 'get':    {method:'GET'},
  'save':   {method:'POST'},
  'query':  {method:'GET', isArray:true},
  'remove': {method:'DELETE'},
  'delete': {method:'DELETE'} };
```

```
var Github = $resource
         ("https://api.github.com/users/srikanthpragada", {});
$scope.user = Github.get();
```

Angular Methods

Following are methods of angular object.

Name	Description
angular.lowercase	Converts the specified string to lowercase.
angular.uppercase	Converts the specified string to uppercase.
angular.forEach	Invokes the iterator function once for each item in obj colle which can be either an object or an array. The iterator func invoked with iterator (value, key, obj), where value is the v an object property or an array element, key is the object pr key or array element index and obj is the obj itself. Specify a context for the function is optional.
angular.extend	Extends the destination object dst by copying own enumera properties from the src object(s) todst. You can specify multiple src objects. If you want to preserve original objects can do so by passing an empty object as the target: var object = angular.extend({}, object1, object2).
angular.merge	Deeply extends the destination object dst by copying own enumerable properties from the srcobject(s) to dst. You car multiple src objects. If you want to preserve original objects can do so by passing an empty object as the target:var object = angular.merge({}, object1, object2).

angular.noop	A function that performs no operations. This function can be when writing code in the functional style.
	function foo(callback) {
	  var result = calculateResult();
	  (callback || angular.noop)(result);
	}

angular.identity	A function that returns its first argument. This function is us when writing code in the functional style.
angular.isUndefined	Determines if a reference is undefined.
angular.isDefined	Determines if a reference is defined.
angular.isObject	Determines if a reference is an Object. Unlike typeof in JavaScript, nulls are not considered to be objects. Note that JavaScript arrays are objects.
angular.isString	Determines if a reference is a String.
angular.isNumber	Determines if a reference is a Number.

angular.isDate	Determines if a value is a date.
angular.isArray	Determines if a reference is an Array.
angular.isFunction	Determines if a reference is a Function.
angular.isElement	Determines if a reference is a DOM element (or wrapped jQ element).
angular.copy	Creates a deep copy of source, which should be an object o array.
angular.equals	Determines if two objects or two values are equivalent. Sup value types, regular expressions, arrays and objects.
angular.bind	Returns a function which calls function fn bound to self (self becomes the this for fn). You can supply optional args that are prebound to the function. This feature known as partial application, as distinguished from function currying.
angular.toJson	Serializes input into a JSON-formatted string. Properties wit leading $$ characters will be stripped since angular uses thi notation internally.
angular.fromJson	Deserializes a JSON string.
angular.bootstrap	Use this function to manually start up angular application.
angular.injector	Creates an injector object that can be used for retrieving se as well as for dependency injection (see dependency injecti
angular.element	Wraps a raw DOM element or HTML string as a jQuery elem
angular.module	The angular.module is a global place for creating, registerin retrieving Angular modules. All modules (angular core or 3r that should be available to an application must be registere this mechanism.

Creating Custom Directives

- ❑ Much like you create controllers and services, you can create your own directives for Angular to use.
- ❑ In AngularJS, "compilation" means attaching directives to the HTML to make it interactive.

In the following example, we say that the `<input>` element **matches** the `ngModel` directive.

```
<input ng-model="foo">
```

The following `<input>` element also **matches** ngModel:

```
<input data-ng-model="foo">
```

And the following `<person>` element **matches** the `person` directive:

```
<person>{{name}}</person>
```

Normalization

Angular **normalizes** an element's tag and attribute name to determine which elements match which directives. We typically refer to directives by their case-sensitive camelCase **normalized** name (e.g. `ngModel`). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using dash-delimited attributes on DOM elements (e.g. `ng-model`).

The **normalization** process is as follows:

1. Strip `x-` and `data-` from the front of the element/attributes.
2. Convert the `:`, `-`, or `_`-delimited name to `camelCase`.

For example, the following forms are all equivalent and match the `ngBind` directive:

```
<div ng-controller="Controller">
  Hello <input ng-model='name'> <hr/>
  <span ng-bind="name"></span> <br/>
  <span ng:bind="name"></span> <br/>
  <span ng_bind="name"></span> <br/>
  <span data-ng-bind="name"></span> <br/>
  <span x-ng-bind="name"></span> <br/>
</div>
```

Note: Prefer using the dash-delimited format (e.g. ng-bind for ngBind). If you want to use an HTML validating tool, you can instead use the data-prefixed version (e.g. data-ng-bind for ngBind). The other forms shown above are accepted for legacy reasons but we advise you to avoid them.

Creating Directive

- Much like controllers, directives are registered on modules.
- To register a directive, you use the module.directive API, which takes the normalized directive name followed by a factory function. This factory function should return an object with the different options to tell $compile how the directive should behave when matched.

```
angular.module('M1', [])
.controller('C1', ['$scope', function($scope) {
  $scope.customer = {
    name: 'Srikanth',
    city: 'Vizag'
  };
}])
.directive('myCustomer', function() {
  return {
    template : 'Name: {{customer.name}} City: {{customer.city}}'
  };
});
```

```
<div ng-controller="C1">
  <div my-customer></div>
</div>
```

We can use templateUrl to specify the file that contains template.

```
.directive('myCustomer', function() {
  return {
    templateUrl: 'my-customer.html'
  };
});
```

my-customer.html

```
Name: {{customer.name}} Address: {{customer.address}}
```

Property **templateUrl** can also be a function which returns the URL of an HTML template to be loaded and used for the directive.

Angular will call the templateUrl function with two parameters: the element that the directive was called on, and an attr object associated with that element.

Note: You do not currently have the ability to access scope variables from the templateUrl function, since the template is requested before the scope is initialized.

When you create a directive, it is restricted to attribute and elements only by default. In order to create directives that are triggered by class name, you need to use the restrict option.

The **restrict** option is typically set to:

- ✓ 'A' - only matches attribute name
- ✓ 'E' - only matches element name
- ✓ 'C' - only matches class name
- ✓ 'M' - only matches comment

These restrictions can all be combined as needed. For example, 'AEC' - matches either attribute or element or class name

```
.directive('myCustomer', function() {
  return {
    restrict: 'E',
    templateUrl: 'my-customer.html'
  };
});
```

Note: Use an element when you are creating a component that is in control of the template. The common case for this is when you are creating a Domain-Specific Language for parts of your template. Use an attribute when you are decorating an existing element with new functionality.

Routing

- Routing allows you to navigate to different pages in your application without page reloading
- Used in Single Page Application (SPA)
- Routing is provided by AngularJS in the ngRoute module
- ngRoute module is present in angular-route.js file, so include this script file in page where ngRoute module is used
- The ngRoute module routes application to different pages without reloading the entire application.
- $routeProvider is used to specify which view is to be used for which url.
- We need to provide a container in which different views are loaded based on url. Container is provided using ng-view directive.
- For each url $routeProvider specifies templateUrl (view to be used) and controller.

The following code shows how $routeProvider is used:

```
var app = angular.module('routedemo', ['ngRoute']);
app.config(
    ['$routeProvider',
     function ($routeProvider)
     {
      $routeProvider.
        when('/first', {
            templateUrl: 'first.html',
            controller: 'FirstController'
        }).
        when('/second', {
            templateUrl: 'second.html',
            controller: 'SecondController'
        }).
        otherwise({
            redirectTo: '/first'
        });
     }
    ]
 );
```

The following is the main page which hosts view container that shows different views based on url provider.

```
<!DOCTYPE html>
<html ng-app="routedemo">
<head>
    <title>Routing Demo</title>
</head>
<body>
        <h1>Routing Demo</h1>
        <div class="menu">
              <a href="#first">First Book </a> |
              <a href="#second">Second Book</a>
        </div>

        <div ng-view=""></div>


    <script src="../angular.js"></script>
    <script src="../angular-route.js"></script>
    <script src="app.js"></script>


</body>
</html>
```