Talk

Overview of AngularJS, Angular, React and other UI frameworks.

- They are UI frameworks
- What is a framework?
- What do they achieve
    - Over Javascript and JQuery
    - Utilizing Frontend power
    - Fluid Experience
    - Binding
    - New HTML UI tags
    - HTML5 SPA
    - Backend is just REST service because of multiple frontend
    - Modular
    - Tree structure module
    - Not webpage but App
    - Client side rendering (Also talk about why Server side is better than Client side)
    - Sanity in frontend code and Advanced programming language

Why AngularJS

- MVC, DI, Services
- Has been out since 2010
- Google
- A lot of applications have been developed on it
- I enjoyed it. Not too many concept to understand for person like me coming from JQuery JS times.
- I also enjoyed simple functions for Directive, Controller, Services/Providers, Config...
    - Unlike React that make us extend it classes
    - Unlike Angular that have complicated annotations/decorators

Future

- I don't think google will abandon it for next coming foreseeable future.
- Example of CIOX. Companies that started using it do not want to let go and keep growing in it.

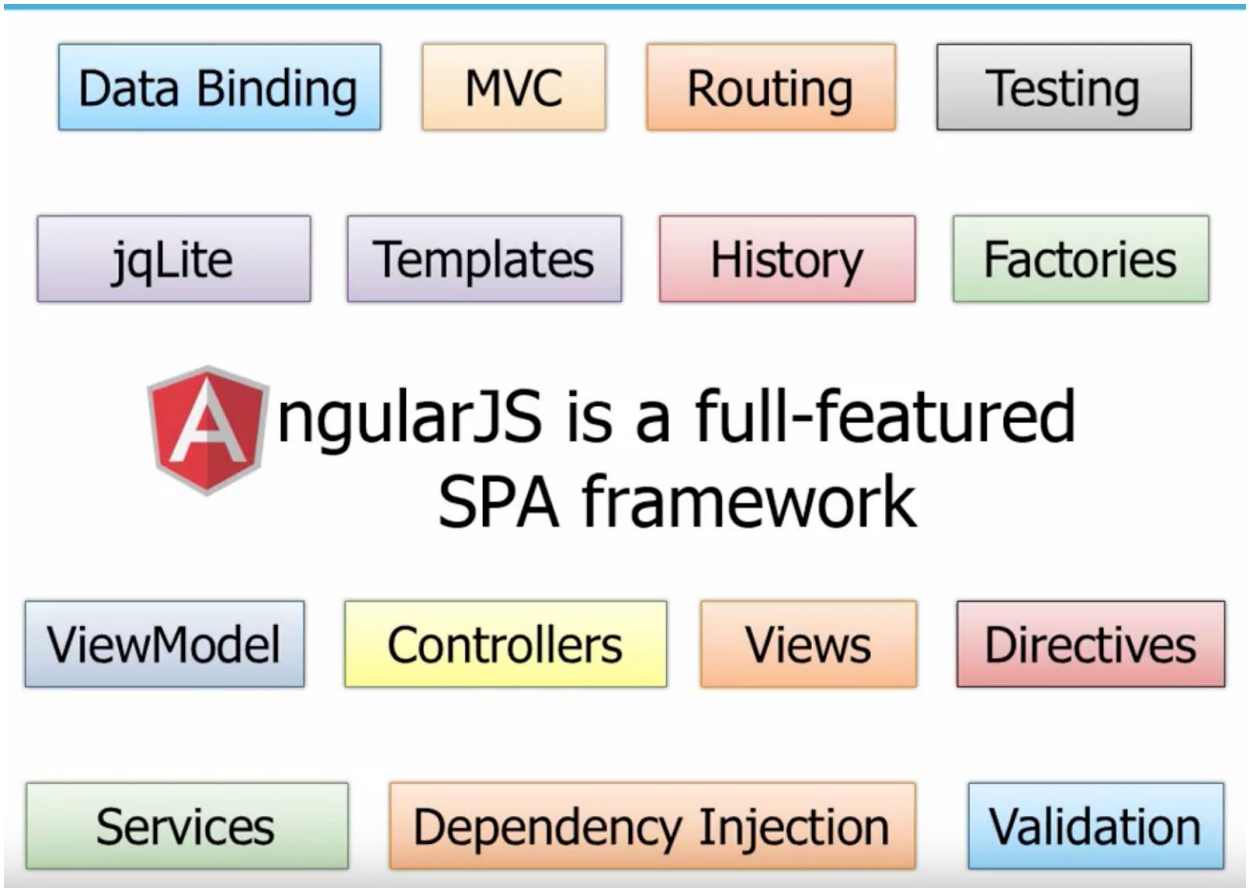
After this course we should be able to

- Build SPA
- Use 3rd party angular extensions/modules
- build production/dev application (Need gulp, webpack 2 and 3.)
- Unit Test UI code. (Need Unit Testing sessions before doing this)

AngularJS 1.x

Angularjs is a frontend MVC framework to build:

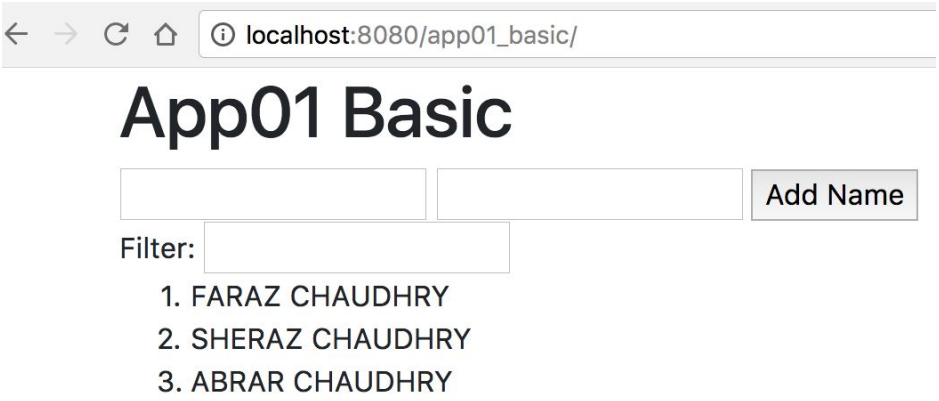
- Dynamic Web Application
- Single Page Application (SPA)



Basics - Learn by example 1

Below example:

- Adds first and last name to a order list
- Before adding it capitalize alphabets
- Has a filter box. When user types in it filters the list




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/c
ss/bootstrap.min.css">
</head>
  <body>
  <div ng-app="myAngularApp" class="container">
      <h1>App01 Basic</h1>
      <div ng-controller="myAppController">
          <input ng-model="firstName"/>
          <input ng-model="lastName"/>
          <button ng-click="addName()">Add Name</button>
          <br/>
          Filter: <input ng-model="nameFilter">
          <ol>
              <li ng-repeat="name in allNames | filter:
nameFilter">{{name.fName}} {{name.lName}}</li>
          </ol>
      </div>
  </div>

  <script src="../lib/angular.min.js"></script>
  <script>
      function myController($scope, myAppService) {
          $scope.firstName = "";
          $scope.lastName = "";
```

```
30.          $scope.allNames = [];
31.
32.          $scope.addName = function(event) {
33.              $scope.allNames.push({
34.                      fName:
    myAppService.makeUpper($scope.firstName),
35.                      lName:
    myAppService.makeUpper($scope.lastName)
36.                  });
37.          }
38.      }
39.
40.      function myService() {
41.          this.makeUpper = function(str) {
42.              return str.toUpperCase();
43.          };
44.      }
45.
46.      var myAngularApp = angular.module("myAngularApp", []);
47.      myAngularApp.controller("myAppController",
    myController);
48.      myAngularApp.service("myAppService", myService);
49.  </script>
50.  </body>
51.  </html>
```

Line 25: Loading angular.js script

Line 11: Root Module - Define angular application UI container

`ng-app="myAngularApp"` directive defines UI boundaries for angular application. Optionally but highly recommended we can give application a name. We gave it "myAngularApp"

Line 46: Handle to application/root module in Script

We pass `angular.module("myAngularApp", [])` function, application name that we created in `ng-app="myAngularApp` and it returns handle to angular application. We can use `var myAngularApp` to add all sorts of features to our application.

Line 27, 40 & 48: Services

Line 40: Define Service Function

Services in angular are regular javascript function.

Line 48: Add Service Function to App Module

Once a service is added to the App Module and given a name `"myAppService"`. Then this name can be dependency injected in any other controller or service.

Line 27: Dependency Injection: Services into controller

```
function myController($scope, myAppService){...}
```

Injecting service function in controller. The variable name should be the same as given when adding it to App Module on Line 48.

Line 13, 27 & 47: Controller

Line 27: Controller Definition Function

```
function myController($scope, myAppService){...}
```

Controller in angular are regular javascript function.

`$scope`

We can optionally dependency inject `$scope` variable as function argument. Any variable or function added to `$scope` will be available in the UI. Like variable and functions are added from line 28 to 32.

`myAppService`

Dependency inject service function. Note how service function's function are being used on line 34 and 35.

Line 47: Add Controller Function to App Module

Once a controller is added to the App Module and given a name `"myAppController"`. Then this name can be used in any HTML element

Line 13: Using Controller in UI

Controllers can be added to any HTML element by using directive

```
ng-controller="myAppController"
```

The name should be the same as given to the App Module on line 47.
Any variables added to `$scope` added can be used in the element where ng-controller is used and its children HTML elements.

Line 14, 15 and 18: ng-modal - Form elements model binding

Line 20: ng-repeat - Loop over Array

```
ng-repeat="name in allNames | filter: nameFilter"
```

Line 16: ng-click - Event handling

```
ng-click="addName()"
```

Directives

They are HTML elements, attributes, classes or comments that teach HTML new ticks.
Here is the full list of all angularjs built-in directives.
https://docs.angularjs.org/api/ng/directive
Most of angularjs built in directives start with ng- prefix.

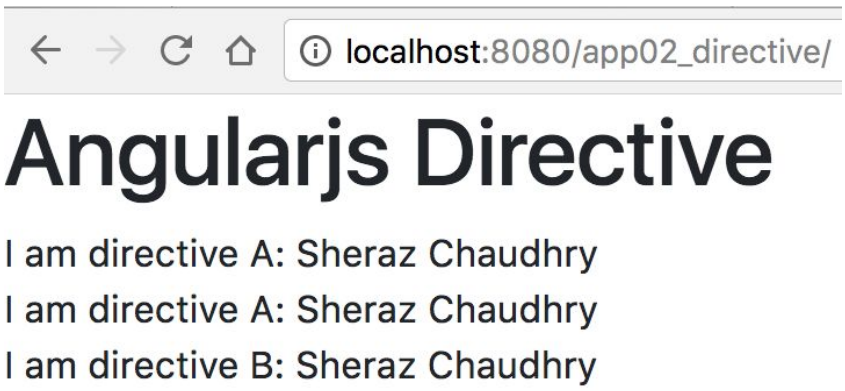
Directives - Learn by example 2

Below example show 2 directives.
directive-a:

- could be used as attribute or as an element
- has in-line template, HTML

directive-b

- could only be used as element
- has a separate HTML template file



index.html

```
1. <!DOCTYPE html>
2. <html lang="en" ng-app="myApp">
3. <head>
4.     <meta charset="UTF-8">
5.     <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
6.     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7.     <title>Angular</title>
8.     <link rel="stylesheet"
   href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/c
   ss/bootstrap.min.css">
9. </head>
10.  <body>
11.     <h1>Angularjs Directive</h1>
12.     <div ng-controller="myController">
13.         <span directive-a></span>
14.         <br/>
15.         <directive-a></directive-a>
16.         <br/>
17.         <directive-b></directive-b>
18.         <br/>
19.     </div>
20.     <script src="../lib/angular.min.js"></script>
21.     <script src="./script.js"></script>
22.  </body>
23.  </html>
```

Line 13: Used directive-a as attribute

Line 15: Used directive-a as element

Line 17: Used directive-b as element

scripts.js

```
1. var app = angular.module("myApp", []);
2.
3. app.controller("myController", function($scope) {
```

```
4.     $scope.fName = "Sheraz";
5.     $scope.lName = "Chaudhry"
6. });
7.
8. app.directive("directiveA", directiveA);
9. app.directive("directiveB", directiveB);
10.
11.   function directiveA() {
12.       return {
13.           restrict: "EA",
14.           template: "I am directive A: {{fName}} {{lName}}"
15.       }
16.   }
17.
18.   function directiveB() {
19.       return {
20.           restrict: "E",
21.           templateUrl: "./directiveBTemplate.html"
22.       }
23.   }
```

Line 11: First example Directive Function

- Directive functions return an object with directive configurations
- It returns `restrict: "EA"` that tells angular, it could be used as element and attribute
- It returns `template:` which is in-line HTML that will be outputted

Line 18: Second example Directive Function

- It returns `restrict: "E"` that tells angular, it could only be used as element
- It returns `templateUrl:` which is path to HTML that will be outputted

Line 8 and 9: Assign Directive Function a name

- directiveA function is given name "directiveA" and directiveB function is given name "directiveB"
- since HTML is case insensitive and we have used camelcase in our directive name so we will have to use "directiveA" as "directive-a" and "directiveB" as "directive-b" in HTML. Read Matching Directive https://docs.angularjs.org/guide/directive

directiveBtemplate.html

templateUrl file could contain HTML that we write with-in HTML body. It could even contain angular code. Our example just contain one line:

```
1. I am directive B: {{fName}} {{lName}}
```

Directive link and two way data binding

```
yearAppUiModule.directive("calendarMonth", function() {
    var link = function(scope, element, attrs) {
        scope.monthName = "#monthName#";
        console.log(scope.dates);
    };
    return {
        restrict: "E",
        scope: {
            dates: '='
        },
        templateUrl: "calendar-month.html",
        link: link
    };
});
```

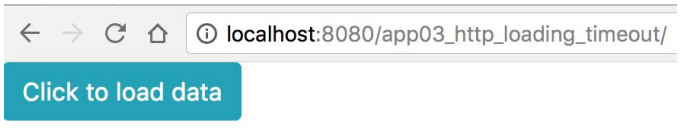
$http, loading screen, and $timeout - Learn by example 3

This example has a button. On clicking it button it timeout for 2 seconds and show loading screen. After 2 seconds it calls a HTTP GET service that returns array of contact. Our application list those contacts.

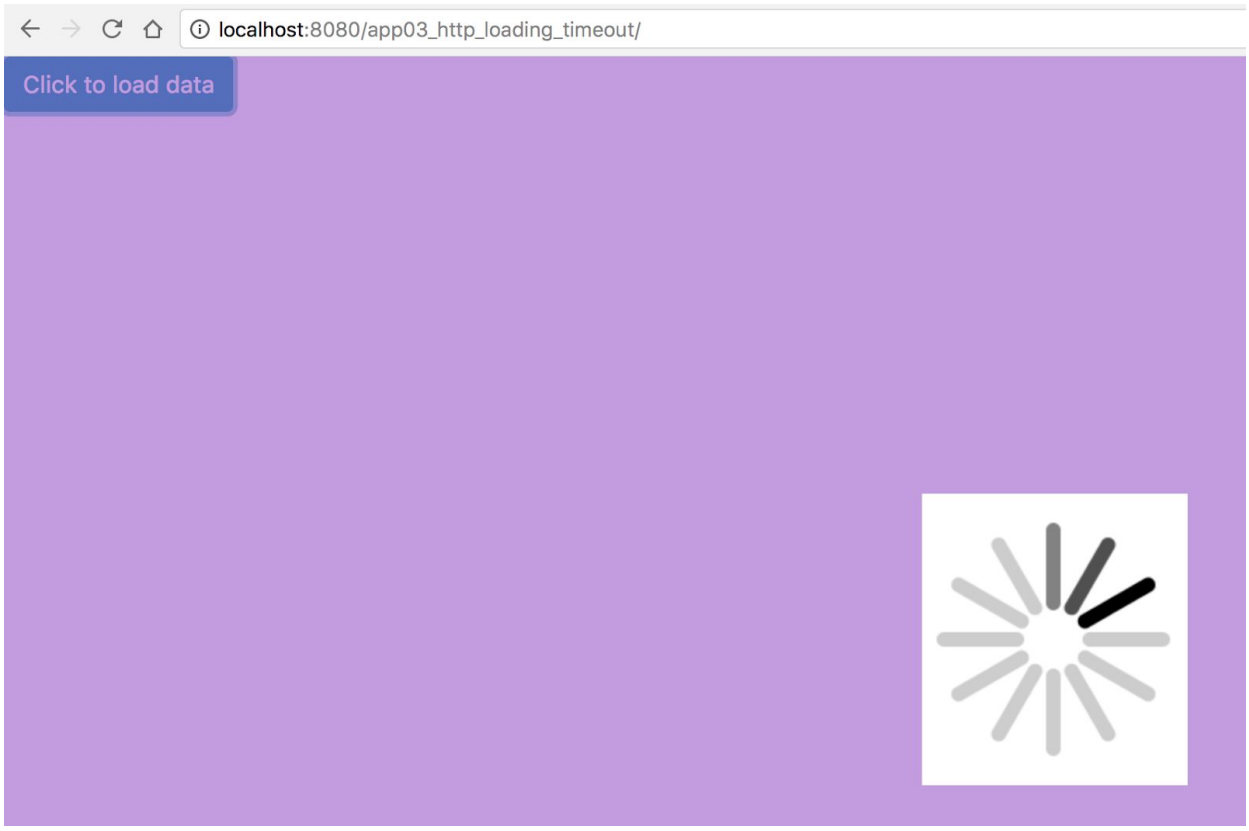
Timeout is used to mimic service is taking some time to respond.

NOTE: $timeout, and $http are angular services. We can add that as parameter in any controller, or service and angular will take care of injecting them.

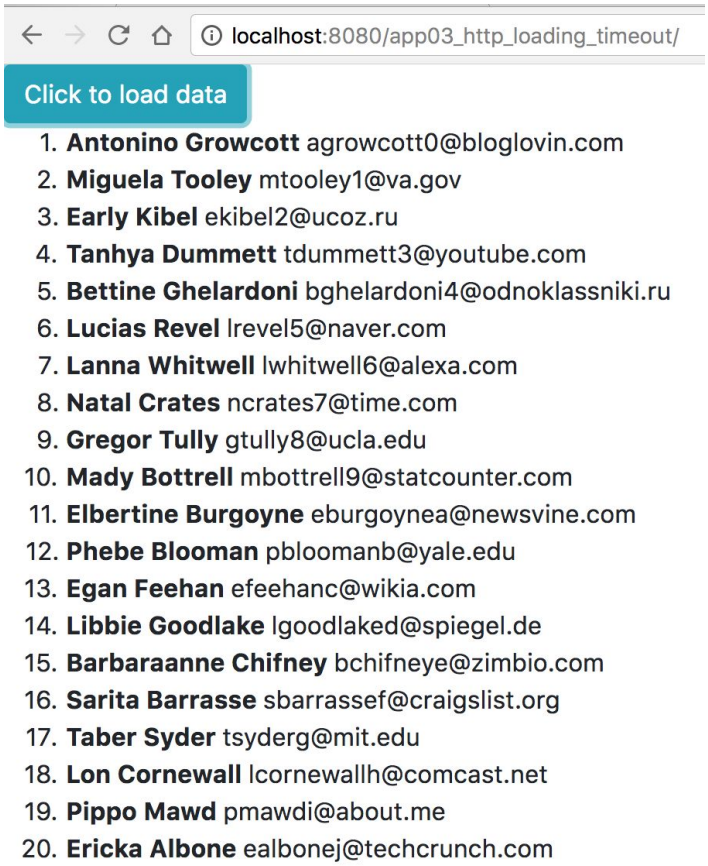
Step 1



Step 2


localhost:8080/app03_http_loading_timeout/
Click to load data

Step 3



index.html

```
1. <!DOCTYPE html>
2. <html lang="en" ng-app="myApp">
3. <head>
4.     <meta charset="UTF-8">
5.     <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
6.     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7.     <title>Angular</title>
8.     <link rel="stylesheet"
   href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/c
   ss/bootstrap.min.css">
9. </head>
10.   <body>
11.       <div ng-controller="myController">
```

```
12.         <loading-directive
    ng-show="loadingData"></loading-directive>
13.         <button ng-click="loadData()" class="btn
    btn-info">Click to load data</button>
14.         <ol>
15.             <li ng-repeat="contact in contacts">
16.                 <b>
17.                     {{contact.first_name}}
18.                     {{contact.last_name}}
19.                 </b>
20.                 {{contact.email}}
21.             </li>
22.         </ol>
23.     </div>
24.   <script src="../lib/angular.min.js"></script>
25.   <script>
26.     var myApp = angular.module("myApp", []);
27.     myApp.controller("myController", myController);
28.     myApp.directive("loadingDirective", loadingDirective);
29.
30.     function myController($scope, $timeout, $http) {
31.         $scope.loadingData = false;
32.         $scope.contacts = [];
33.         $scope.loadData = function() {
34.             $scope.contacts = [];
35.             $scope.loadingData = true;
36.             $timeout(function(){
37.                 $http({method: "GET",url: "./data.json"})
38.                     .then(function success(response) {
39.                         console.log(response);
40.                         $scope.contacts = response.data;
41.                         $scope.loadingData = false;
42.                     }, function error(response) {});
43.             }, 2000);
44.         }
45.     }
46.
47.     function loadingDirective() {
48.         return {
49.             templateUrl: "loading_screen.html"
50.         }
51.     }
52.
53.   </script>
54.   </body>
```

```
55.  </html>
```

Line 12, 28, 47-51. Loading screen directive

This is a directive that show contents of loading_screen.html if $scope.loadingData is true.

Line 31, 35, 41. Show/Hide Loading screen directive

Line 15-21, 32. Listing content of $scope.contacts array

Line 35, 43. $timeout

https://docs.angularjs.org/api/ng/service/$timeout
https://docs.angularjs.org/api/ng/service/$interval
$timeout is a wrapper around window.setTimeout() function

Line 37. $http

https://docs.angularjs.org/api/ng/service/$http
$http function expects object that would be used to create request and returns server response in a promise.

Line 40. Reading server response body.

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.
- **xhrStatus** – `{string}` – Status of the XMLHttpRequest (`complete`, `error`, `timeout` or `abort`).

In our example we are only interested in response.body

Create Helper Objects in AngularJS

Other than Controller, there are 5 ways to create helper objects in Angularjs.

1. **Value:** Simple value that could be used across application

```
var myApp = angular.module('myApp', []);
myApp.value('clientId', 'a12345654321x');
```

2. **Constant:** Constants are just like values. But unlike value it can be injected anywhere even in configuration. e.g.

```
var app = angular.module('app', []);
app.constant('PI', 3.14159265359);

// PI can be injected here in the config block
app.config(function(PI){
   var radius = 4;
   var perimeter = 2 * PI * radius;
});
```

3. **Service:** Object with helper stateless methods. Methods can have dependencies injected to them. e.g.

```
var myServicesModule = angular.module("myServicesModule",
[]);
myServicesModule.service("myService", function() {
   this.add = function(a, b) {
       return a + b;
   }
});
```

4. **Factory:** Same as services but, if an object requires code to run before initializing then we use Factories.

```
var myApp = angular.module('myApp', []);
myApp.factory('apiService', function
buildApiService(clientId, encrypter, apiCaller) {
   var encryptedClientId = encrypter(clientId);
   return {
       callServer: function() {
           apiCaller(encryptedClientId);
       }
   };
});
```

5. **Provider**

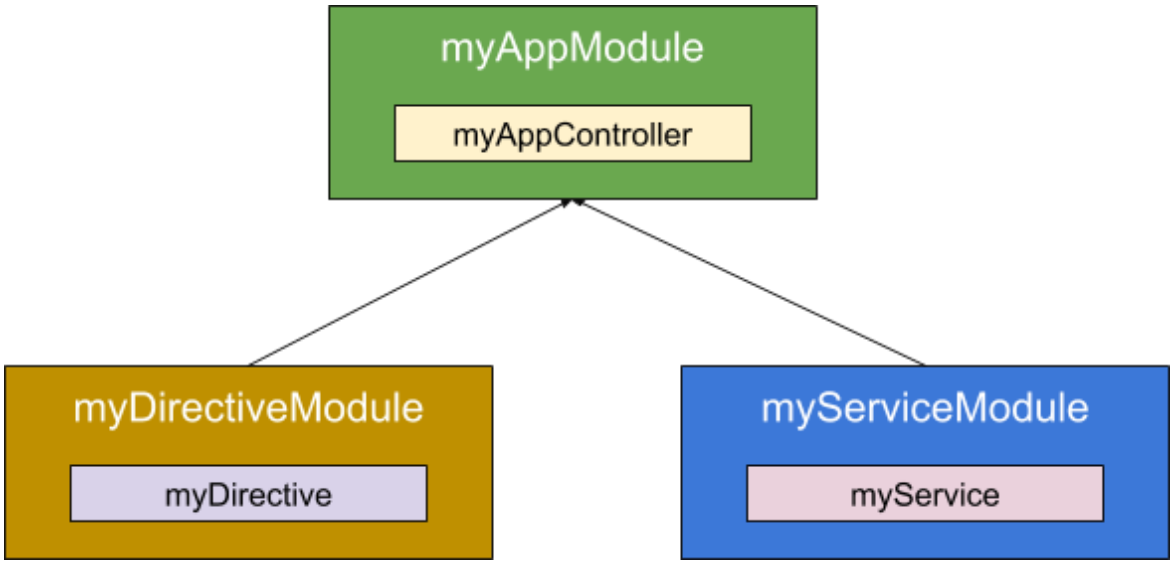
Features / Recipe type	Factory	Service	Value	Constant	Provider
can have dependencies	yes	yes	no	no	yes
uses type friendly injection	no	yes	yes*	yes*	no
object available in config phase	no	no	no	yes	yes**
can create functions	yes	yes	yes	yes	yes
can create primitives	yes	no	yes	yes	yes

* at the cost of eager initialization by using `new` operator directly

** the service object is not available during the config phase, but the provider instance is (see the `unicornLauncherProvider` example above).

Combining modules - Learn by example 4

In angular we could create a hierarchy of modules. Each module could contain controller, directives, and helper objects (value, constant, Service, Factory, Provider and ...). Module helper object could be shared across modules by using dependency injection.

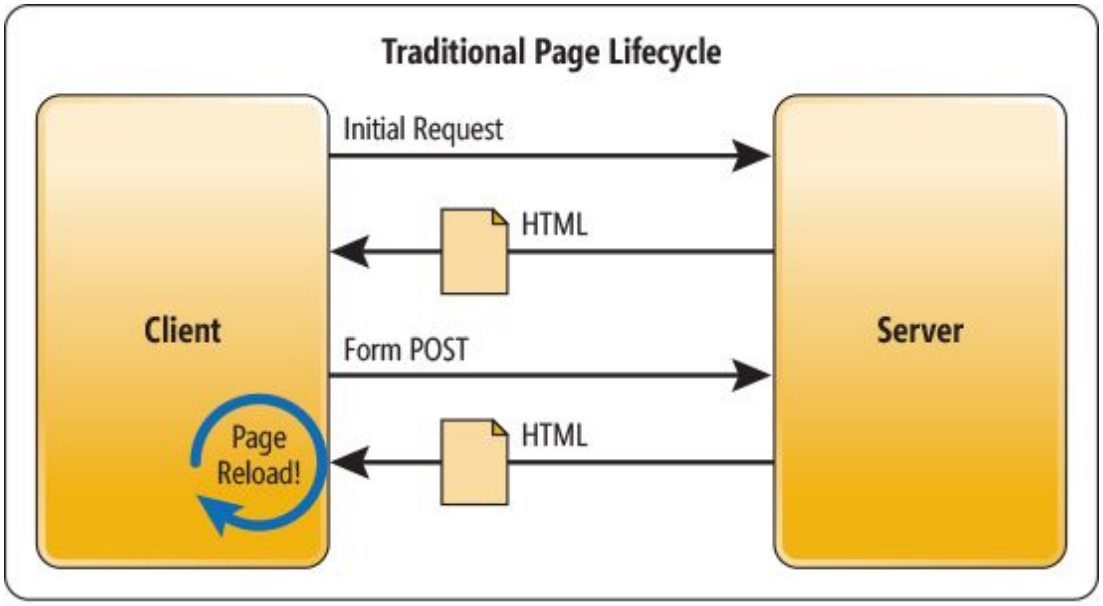


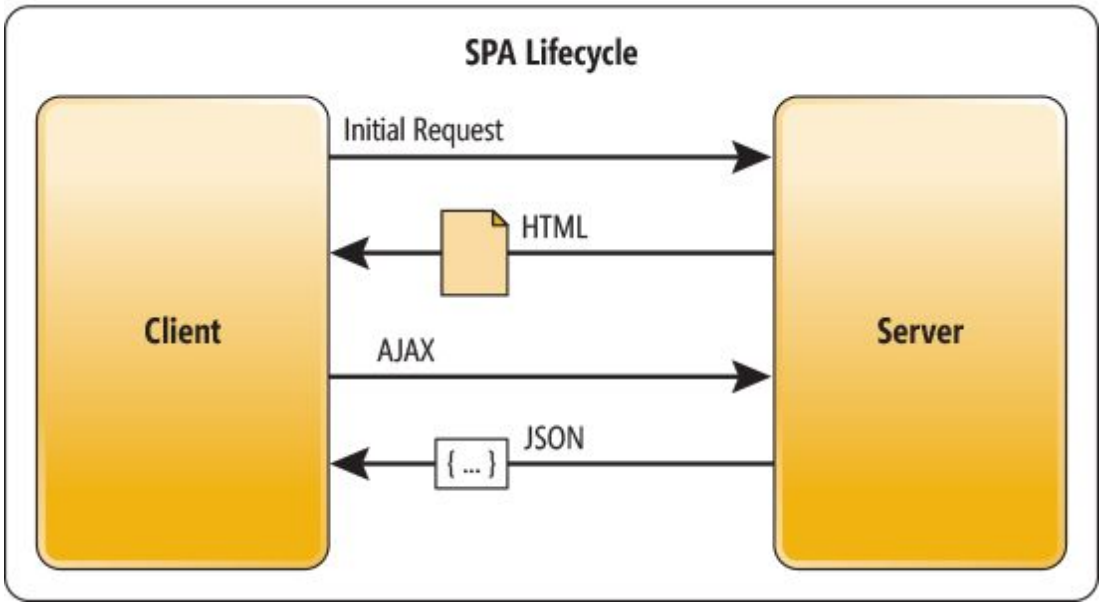

```
1. var myServicesModule = angular.module("myServicesModule", []);
2. myServicesModule.service("myService", function() {
3.     this.add = function(a, b) {
4.         return a + b;
5.     }
6. });
7.
```

```
8. var myDirectiveModule = angular.module("myDirectiveModule",
   []);
9. myDirectiveModule.directive("myDirective", ["myService",
   function(myService) {return {
10.      template: "{{result}}",
11.      link: function (scope, element, attributes) {
12.          scope.result = scope.num1 + " + " + scope.num2
13.              + " = " + myService.add(scope.num1, scope.num2);
14.      }
15.  }}]);
16.
17.  var myAppModule = angular.module("myAppModule",
   ["myServicesModule", "myDirectiveModule"]);
18.  myAppModule.controller("myAppController", ["$scope",
   function($scope) {
19.      $scope.num1 = 3;
20.      $scope.num2 = 5;
21.  }]);
22.
```

What is SPA (Single Page Application)

https://msdn.microsoft.com/en-us/magazine/dn463786.aspx





AngularJS Router and SPA

We use routers when there comes a situation in our application when we have to replace a big portion of our page, lets say by main navigation links.

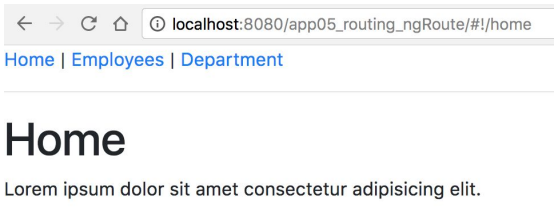
For AngularJS there 2 popular Routers.

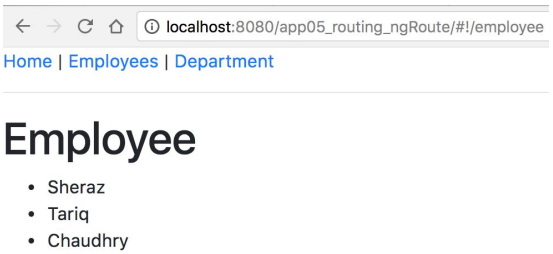
- **ngRouter** https://docs.angularjs.org/api/ngRoute
- **UI-Router** https://ui-router.github.io/

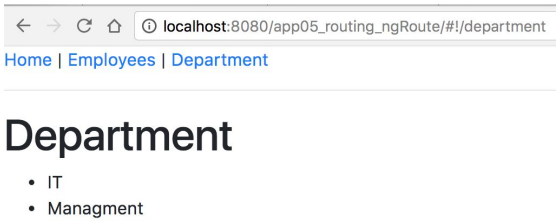
AngularJS Router - Learn by Example

Create an app with 3 pages. "home","employee", "department"
If no view link is specified then router should redirect to "home".
Make # hash base link that are compatible with HTML4
http://localhost:8080/app05_routing_ngRoute
http://localhost:8080/app05_routing_ngRoute/#!/home
http://localhost:8080/app05_routing_ngRoute/#!/employee
http://localhost:8080/app05_routing_ngRoute/#!/department

NOTE: To make HTML5 history links we have to use
https://docs.angularjs.org/api/ng/provider/$locationProvider







Create Views and script

Use CDN or download **angular-route.js**

https://code.angularjs.org/
Include **angular-route.js** in **index.html**.

scripts.js

```
var myApp = angular.module("myApp", ["ngRoute"]);
myApp.config(function($routeProvider) {

    $routeProvider
        .when("/home", {
            templateUrl: "./partial_views/home.html"
        })
        .when("/employee", {
            templateUrl: "./partial_views/employee.html",
               controller: "employeeController"
          })
          .when("/department", {
               templateUrl: "./partial_views/department.html",
```

```
14.             controller: "departmentController"
15.
16.         })
17.         .otherwise({
18.             redirectTo: "/home"
19.         });
20.
21.  });
22.
23.  myApp.controller("employeeController", function($scope) {
24.     $scope.employees = ["Sheraz", "Tariq", "Chaudhry"];
25.  });
26.
27.  myApp.controller("departmentController", function($scope) {
28.     $scope.managements = ["IT", "Management"];
29.  });
```

Line 1. ngRoute

By including **angular-route.js** in **index.html**, we can include **ngRoute**

Line 2. application's config() function

Angular application's **config()** function can be used to execute initializing code.

Line 2. $routeProvider

ngRoute module makes **$routeProvider** available for dependency injection. We can include it in application **config()** function.

Line 5, 8, 12. $routeProvider.when()

Define controller and template/templateUrl for each route path

Line 17. $routeProvider.otherwise()

For handling not existing route. In the example we are re

index.html

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
    <title>Angular</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css
/bootstrap.min.css">
</head>
<body>
    <div>
        <div>
            <a href="#!/home">Home</a>
            |
            <a href="#!/employee">Employees</a>
            |
            <a href="#!/department">Department</a>
        </div>
        <hr/>
        <div ng-view></div>
    </div>
<script src="../lib/angular.js"></script>
<script src="../lib/angular-route.min.js"></script>
<script src="./script.js"></script>
</body>
</html>
```

partial_views/home.html

```
<div>
    <h1>Home</h1>
    Lorem ipsum dolor sit amet consectetur adipisicing elit.
</div>
```

partial_views/employee.html

```
<div>
    <h1>Employee</h1>
    <ul>
        <li ng-repeat="employee in employees">{{employee}}</li>
    </ul>
</div>
```

partial_views/department.html

```
<div>
    <h1>Department</h1>
    <ul>
```

```
        <li ng-repeat="managment in
managments">{{management}}</li>
    </ul>
</div>
==========
```

Animation

3rd party - community created angularjs modules

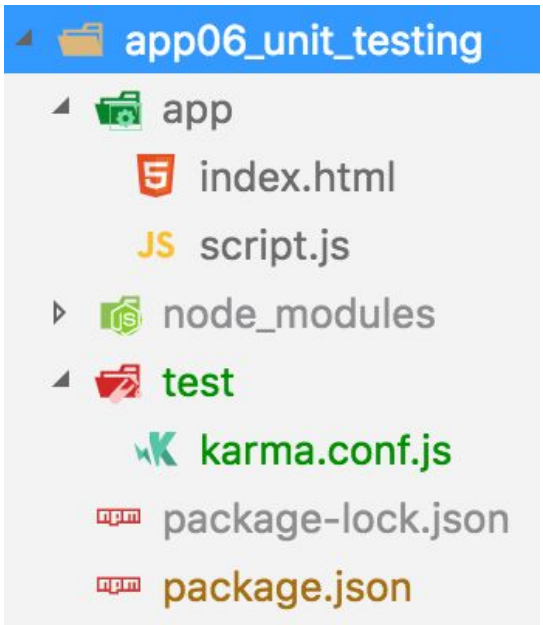
http://ngmodules.org/

Unit Testing

Install dependencies
$ npm install angular --save
$ npm install karma phantomjs jasmine -g
$ npm install karma jasmine angular-mocks --save-dev

Verify installation
$ karma --version
$ phantomjs --version

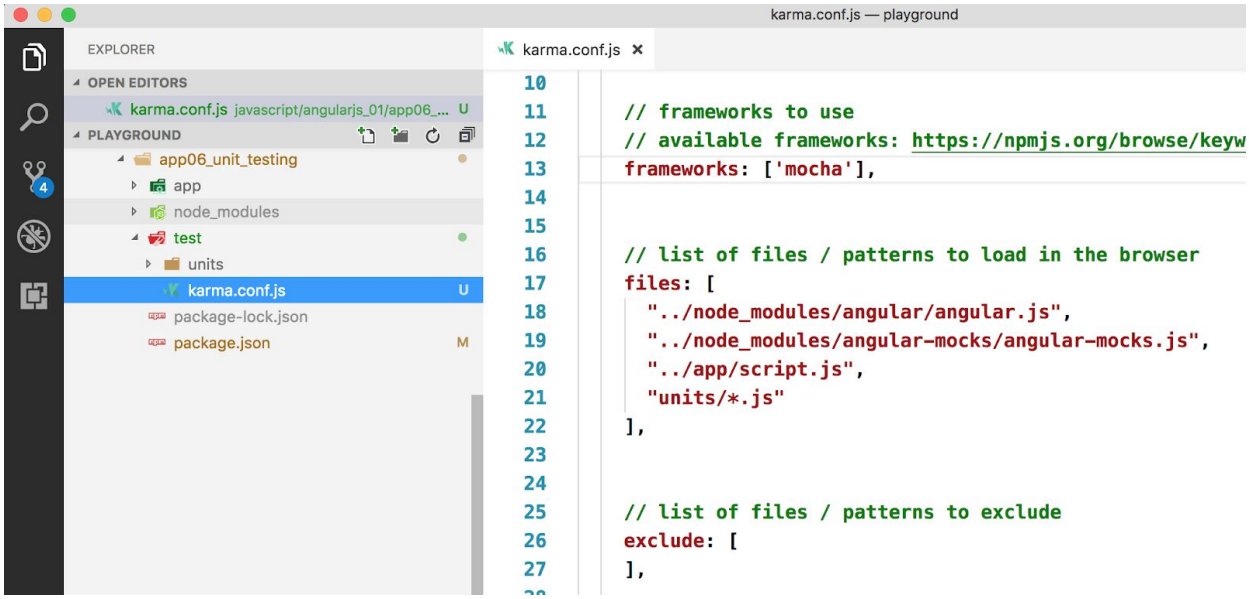
```
test — -bash — 204×60
Sherazs-MBP:test sheraz$ pwd
/Users/sheraz/dev/playground/javascript/angularjs_01/app06_unit_testing/test
Sherazs-MBP:test sheraz$ karma init

Which testing framework do you want to use ?
Press tab to list possible options. Enter to move to the next question.
> mocha

Do you want to use Require.js ?
This will add Require.js plugin.
Press tab to list possible options. Enter to move to the next question.
> no

Do you want to capture any browsers automatically ?
Press tab to list possible options. Enter empty string to move to the next question.
> PhantomJS
> Chrome
>

What is the location of your source and test files ?
You can use glob patterns, eg. "js/*.js" or "test/**/*Spec.js".
Enter empty string to move to the next question.
>

Should any of the files included by the previous patterns be excluded ?
You can use glob patterns, eg. "**/*.swp".
Enter empty string to move to the next question.
>

Do you want Karma to watch all the files and run the tests on change ?
Press tab to list possible options.
> yes

Config file generated at "/Users/sheraz/dev/playground/javascript/angularjs_01/app06_unit_testing/test/karma.conf.js".

Sherazs-MBP:test sheraz$
```



Add these scripts to the files array:

- angular,
- angular-mocks
- application script

- All unit test scripts



$ karma start karma.conf.js