

Table of Contents

<i>Introduction</i>	<i>3</i>
Diagram of AngularJS.....	3
Single-Page Applications	3
Bootstrapping the Application.....	3
Dependency Injection	4
AngularJS Route.....	4
HTML5 Mode	4
Model View Controller (MVC).....	4
Views.....	4
Model.....	5
Controller.....	5
<i>AngularJS Project</i>	<i>5</i>
Edit HTML (Regular AngularJS)	7
OnTrack Version	7
Edit JavaScript (Regular AngularJS).....	8
Create Templates (Regular AngularJS)	9
Testing AngularJS	9
<i>MVC and Angular JS.....</i>	<i>10</i>
Server-side page scripting	10
Old Way	10
New Way	10
<i>AngularJS Controllers</i>	<i>11</i>
Initializing Example.....	11
Adding Behavior with Controllers	13
Controller Business Logic	14
Form Submission	14
Using Submitted Form Data	15
Conclusion	15
<i>AngularJS Views and Bootstrap.....</i>	<i>16</i>
Two-way databinding	16
Exercises	16
<i>AngularJS and REST Services</i>	<i>17</i>

<i>Reference</i>	22
-------------------------------	-----------

Introduction

AngularJS support has officially ended as of January 2022.

Diagram of AngularJS

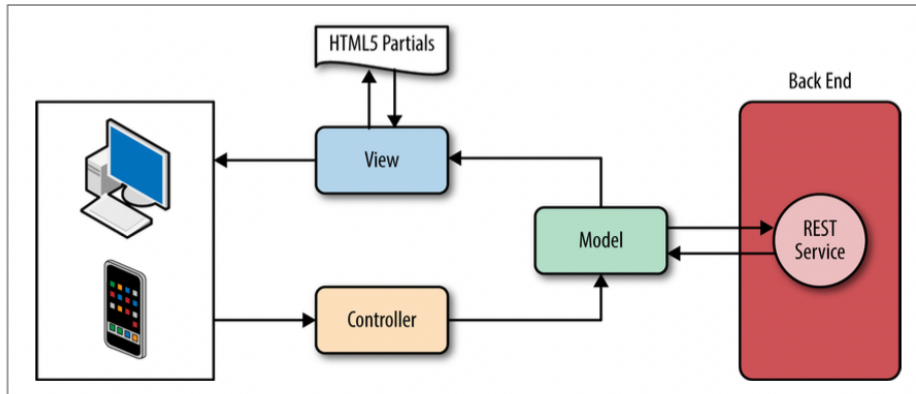


Figure 1-1. Diagram of an AngularJS MVC application

More Diagram and explanation [here](#)

Single-Page Applications

AngularJS is most often used to build applications that conform to the single-page application (SPA) concept. SPAs are applications that have one entry point HTML page; all the application content is dynamically added to and removed from that one page.

Bootstrapping the Application

Bootstrapping AngularJS is the process of loading AngularJS when an application first starts. Loading the AngularJS libraries in a page will start the bootstrap process. The index.html file is analyzed, and the parser looks for the **ng-app** tag.

```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
  </head>
  ...
</html>
```

Dependency Injection

[Dependency injection](#) (DI) is a design pattern where dependencies are defined in an application as part of the configuration. Dependency injection helps you avoid having to manually create application dependencies.

```
'use strict';

/* App Module */

var helloWorldApp = angular.module('helloWorldApp', [
    'ngRoute',
    'helloWorldControllers'
]);
```

AngularJS Route

AngularJS routes are defined through the `$routeProvider` API. Routes are dependent on the `ngRoute` module, and that's why it is a requirement when the application starts.

```
helloWorldApp.config(['$routeProvider', '$locationProvider',
    function($routeProvider, $locationProvider) {
        $routeProvider.
            when('/', {
                templateUrl: 'partials/main.html',
                controller: 'MainCtrl'
            }).when('/show', {
                templateUrl: 'partials/show.html',
                controller: 'ShowCtrl'
            });

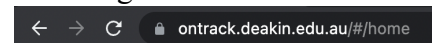
        $locationProvider.html5Mode(false).hashPrefix('!');
    }]);
```

If a user clicks on a link in the application specified as `www.someDomainName/show`, the `/show` route will be followed, and the content associated with that URL will be displayed.

HTML5 Mode

```
$locationProvider.html5Mode(false).hashPrefix('!');
```

HTML5 mode also gives the application pretty URLs like `/someAppName/blogPost/5` instead of the standard AngularJS URLs like `/someAppName/#!/blogPost/5` that use the `#!`, known as the hashbang.



HTML5 mode can provide pretty URLs, but it does require configuration changes on the web server in most cases. (AJAX crawling scheme is deprecated, and no hashtags should be used in URLs as they are bad for the search.)¹

Model View Controller (MVC)

Views

AngularJS pulls in all the templates defined for an application and builds the views in the document object model ([DOM](#)) for you. Therefore, the only work you need to do to build the views is to create the templates.

¹ [websiteseochecker.com](http://www.websiteseochecker.com), n.d.

AngularJS Note (OnTrack)

Model

AngularJS has a **\$scope** object that is used to store the application model. Scopes are attached to the DOM. The way to access the model is by using data properties assigned to the \$scope object.

Controller

AngularJS controllers are the tape that holds the models and views together. Business logic should almost always be placed in backend REST services whenever possible, however we can place all business logic specific to a particular view when it's not possible to place the logic inside a REST service.

Business Logic

In computer software, business logic or domain logic is the part of the program that encodes the real-world business rules that determine how data can be created, stored, and changed. It is contrasted with the remainder of the software that might be concerned with lower-level details of managing a database or displaying the user interface, system infrastructure, or generally connecting various parts of the program.

The following code shows the contents of the controllers.js file. At the start of the file we define the helloWorldController module. We then define two new controllers, MainCtrl and ShowCtrl, and attach them to the helloWorldController module.

```
/* chapter1/controllers.js */

'use strict';
/* Controllers */

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

helloWorldControllers.controller('MainCtrl', ['$scope', function MainCtrl($scope) {

    $scope.message = "Hello World";
}]);

helloWorldControllers.controller('ShowCtrl', ['$scope', function ShowCtrl($scope) {

    $scope.message = "Show The World";
}]);
```

The line `$scope.message = "Hello World"` in the MainCtrl controller is used to create a property named message that is added to the scope. . We then use the double curly braces markup (`{{}}`) inside the main.html template to gain access to and display the value assigned to `$scope.message`:

```
<!-- chapter1/main.html -->

<div>{{message}}</div>
```

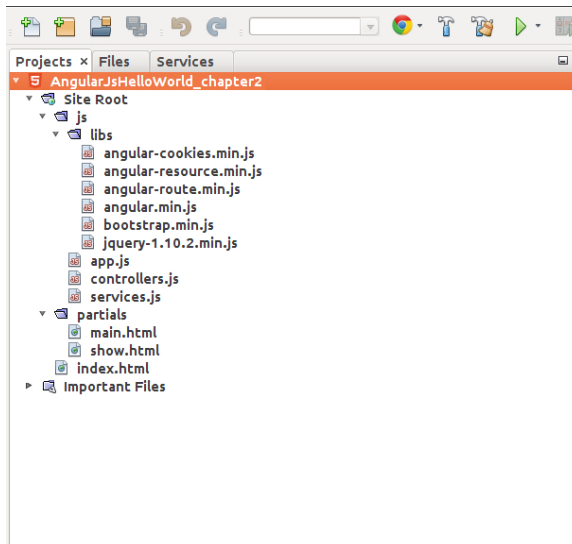
AngularJS Project

- angular.min.js (main libs)
- angular-route.min.js (routing libs)
- angular-cookies.min.js (cookie libs)
- angular-resource.min.js (REST service libs)

AngularJS Note (OnTrack)

There are some necessary files that needed to create such as:

- app.js (where the application is defined)
- controllers.js (where controllers are defined)
- services.js (where services are defined)
- main.html under the partials folder
- show.html under the partials folder
- index.html under the Site Root folder



More detail in the project structure [here](#).

Edit HTML (Regular AngularJS)

Now we must edit the index.html file to create bootstrapping for the application.

```
<!-- chapter2/index.html -->
<!DOCTYPE html>

<html lang="en" ng-app="helloWorldApp"> <head>

<title>AngularJS                                Hello                                World</title>
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"> <script src="js/libs/jquery-
1.10.2.min.js"></script>
<script src="js/libs/angular.min.js"></script>
<script src="js/libs/angular-route.min.js"></script>
<script src="js/libs/angular-resource.min.js"></script>
<script src="js/libs/angular-cookies.min.js"></script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/services.js"></script>

</head>

<body>
<div ng-view></div> </body>

</html>
```

OnTrack Version

However, in the OnTrack it is a bit different, The controller is at the very top of the html, which is the <body> and the view is a child of the body which is <div ui-view="main">. I think it will execute the **main.ts** and bootstrapping the AngularJS module and Angular.

```
<!--/src/index.html--!>

<body ng-controller="AppCtrl">
  <app-header></app-header>
  <div ui-view="main" class="container-fluid"></div>
  <alert-list> </alert-list>
  <noscript>Please enable JavaScript to continue using this application.</noscript>
</body>
```

src/main.ts

```
DoubtfireAngularJSModule.config(['$urlServiceProvider', ($urlService: UrlService) =>
$urlService.deferIntercept()]);
```

src/app/doubtfire-angularjs.module.ts

```
// If the user enters a URL that doesn't match any known URL (state), send them to `/home`
const otherwiseConfigBlock = [
  '$urlRouterProvider',
  '$locationProvider',
  ($urlRouterProvider: any, $locationProvider: any) => {
    $locationProvider.hashPrefix("");
    $urlRouterProvider.otherwise('/home');
  },
];
DoubtfireAngularJSModule.config(otherwiseConfigBlock);
```

After the main.ts being construct, the client will be automatically redirecting to the /home which another controller may involve taking control for validating the user.

[Edit JavaScript \(Regular AngularJS\)](#)

```
/* chapter2/app.js */

'use strict'; /* App Module */

var helloWorldApp = angular.module('helloWorldApp', [
  'helloWorldControllers'

]);

helloWorldApp.config(['$routeProvider', '$locationProvider', function($routeProvider, $locationProvider) {

  $routeProvider.
    when('/', {
      templateUrl: 'partials/main.html',
      controller: 'MainCtrl'
    }).when('/show', {
      templateUrl: 'partials/show.html',
      controller: 'ShowCtrl'
    });

  $locationProvider.html5Mode(false).hashPrefix('');
}]);
```

```
/* chapter2/controllers.js */

'use strict';
/* Controllers */

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

helloWorldControllers.controller('MainCtrl', ['$scope', '$location', '$http', function MainCtrl($scope, $location, $http) {
  $scope.message = "Hello World";
}]);

helloWorldControllers.controller('ShowCtrl', ['$scope', '$location', '$http', function ShowCtrl($scope, $location, $http) {
  $scope.message = "Show The World";
}]);
```


Create Templates (Regular AngularJS)

All that is left is to create the templates main.html && show.html.

```
<!-- chapter2/main.html -->
```

```
<div>{{message}}</div>
```

```
<!-- chapter2/show.html -->
```

```
<div>{{message}}</div>
```

Testing AngularJS

There are two types of tests that are used for testing AngularJS applications. The first type of test is the unit test. Unit testing is usually the first place where issues with the code are found, through testing small units of code. The second type of test is end-to-end (E2E) testing. E2E testing helps to identify software defects by testing how components connect and interact together as a whole. *Please follow the **testing guideline** on Microsoft Teams.*

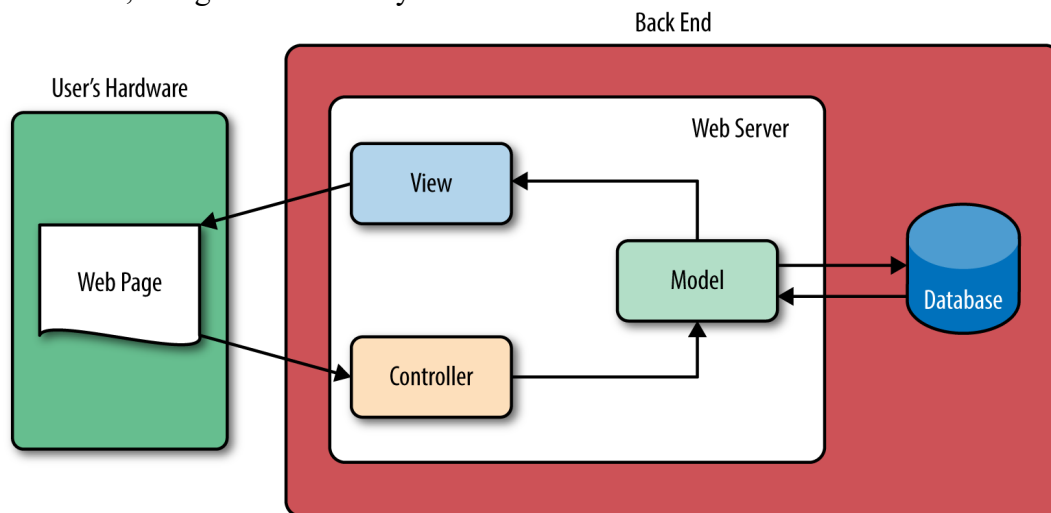
MVC and Angular JS

Server-side page scripting

Conventional web frameworks tend to run slower and be sluggish on mobile devices. And mobile users have a much lower tolerance for system delays and slow page loads than desktop users.

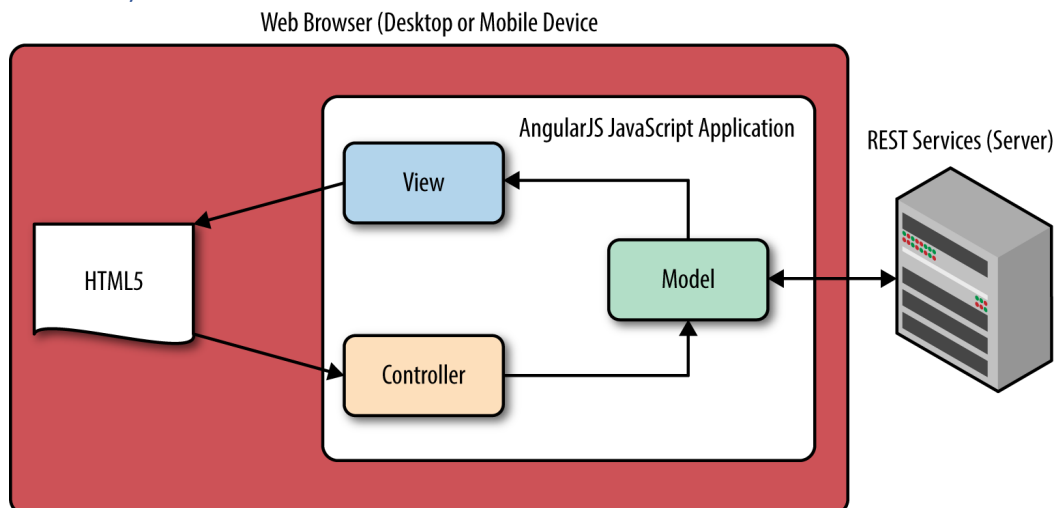
Old Way

All functions such as database access, business logic, display logic, and UI activities happen on the server, using server memory and resources in conventional web MVC framework.



Web applications and sites built with Ruby on Rails, the Zend Framework, Spring MVC, CakePHP, and other web frameworks are based on this design.

New Way



The approach that's taken frees the server or servers to handle nothing but business logic and data access. Using REST services that send and receive JSON helps to greatly simplify AngularJS application.

AngularJS Controllers

Initializing Example

In this code, we first create a new module named `addonsControllers` by making a call to the `module` method of `angular`. On the second line, we create a new controller named `AddonsCtrl` by calling the `controller` method of the `addonsControllers` module. Doing that attaches the new controller to that module. All controllers created in the `controllers.js` file will be added to the `addonsControllers` module.

The `[]` parameter in the module definition can be used to define dependent modules. Without the `[]` parameter, you are not creating a new module, but retrieving an existing one.

```
var addonsControllers = angular.module('addonsControllers', []);
addonsControllers.controller('AddonsCtrl',
    ['$scope', 'checkCreds', '$location', 'AddonsList', '$http', 'getToken',
    function AddonsCtrl($scope, checkCreds, $location, AddonsList, $http,
    getToken) {
        if (checkCreds() !== true) {
            $location.path('/loginForm');
        }
        $http.defaults.headers.common['Authorization'] =
            'Basic ' + getToken();
        AddonsList.getList({}, function success(response) {
            console.log("Success:" +
                JSON.stringify(response));
            $scope.addonsList = response;
        },
        function error(errorResponse) {
            console.log("Error:" +
                JSON.stringify(errorResponse));
        });
        $scope.addonsActiveClass = "active";
    }
]);
```

The following code is an excerpt of the previous file. It shows how we use dependency injection to add dependencies to the new controller. This code shows `$scope`, `check Creds`, `$location`, `AddonsList`, `$http`, and `getTokens` as dependencies for the new controller. We have already covered the `$scope` briefly. For now, it's not important what the other dependencies represent; you only need to understand they are required by the new controller:

```
['$scope', 'checkCreds', '$location', 'AddonsList', '$http', 'getToken',
function AddonsCtrl($scope, checkCreds, $location, AddonsList, $http,
getToken)
```

More example [here](#)

The screenshot illustrates the setup of a simple AngularJS application. It is divided into three main sections:

- Script.js:** A code block showing the JavaScript configuration for the application.

```
// Script.js
var myApp = angular.module("myModule", []);

myApp.controller("myController", function ($scope) {
    $scope.message = "AngularJS Tutorial";
});
```
- HTML Template:** A code block showing the HTML structure that uses the AngularJS module and controller.

```
<!doctype html>
<html ng-app="myModule">
<head>
    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/Script.js"></script>
</head>
<body>
    <div ng-controller="myController">
        {{ message }}
    </div>
</body>
</html>
```
- Result:** A browser window showing the rendered output of the application. The address bar indicates the file is running on localhost at port 33358. The page content displays "AngularJS Tutorial".

Yellow arrows in the original image point from the JavaScript code to the corresponding HTML attributes: from `myModule` to `ng-app="myModule"`, from `myController` to `ng-controller="myController"`, and from the `$scope.message` assignment to the `{{ message }}` interpolation in the HTML.

Adding Behavior with Controllers

The second primary use for controllers is adding behaviour to the \$scope object. We add behaviour by adding methods to the scope.

```
/* chapter4/controllers.js excerpt */
```

```
helloWorldControllers.controller('CustomerCtrl', ['$scope', function CustomerCtrl($scope) {  
  
    $scope.customerName = "Bob's Burgers";  
    $scope.customerNumber = 44522;  
    // add method to scope  
  
    $scope.changeCustomer = function(){ $scope.customerName = $scope.cName; $scope.customerNumber =  
    $scope.cNumber;  
  
}; }]);
```

chapter4/partials/customer.html

```
<div>  
  <b>Customer Name:</b> {{customerName}}  
</div>  
<div>  
  <b>Customer Number:</b> {{customerNumber}}  
</div>  
<form>  
  <div>  
    <input type="text" ng-model="cName" required />  
  </div>  
  <div>  
    <input type="number" ng-model="cNumber" required />  
  </div>  
  <div>  
    <button ng-click="changeCustomer();">Change Customer</button>  
  </div>  
</form>
```

Controller Business Logic

Controllers are used as just demonstrated to add business logic to an application. Business logic added in the controller, however, should be specific to the view associated with that one controller and used to support some display logic functionality of that one view. Any business logic that can be pushed off the client-side application should be implemented as a REST service and not actually inside the AngularJS application.

Form Submission

chapter4/partials/newCustomer.html (with controller as)

```
<form ng-submit="addCust.submit()" ng-controller="AddCustomerCtrl as addCust">
  <div>
    <input type="text" ng-model="addCust.cName" required />
  </div>
  <div>
    <input type="text" ng-model="addCust.cCity" required />
  </div>
  <div>
    <button id="f1" type="submit">Add Customer</button>
  </div>
</form>
```

chapter4/controllers.js

```
/* chapter4/controllers.js */
helloWorldControllers.controller('AddCustomerCtrl',
  ['$scope', '$location',
function AddCustomerCtrl($scope, $location) { $scope.submit = function(){
  $location.path('/addedCustomer/' + $scope.cName + "/" + $scope.cCity);
};
}]);
```

Using Submitted Form Data

```
helloWorldApp.config(['$routeProvider', '$locationProvider', function($routeProvider, $locationProvider) {
    $routeProvider.
    when('/', {
        templateUrl: 'partials/main.html',
        controller: 'MainCtrl'
    }).when('/show', {
        templateUrl: 'partials/show.html',
        controller: 'ShowCtrl'
    }).when('/customer', {
        templateUrl: 'partials/customer.html',
        controller: 'CustomerCtrl'
    }).when('/addCustomer', {
        templateUrl: 'partials/newCustomer.html',
        controller: 'AddCustomerCtrl'
    }).when('/addedCustomer/:customer/:city', {
        templateUrl: 'partials/addedCustomer.html',
        controller: 'AddedCustomerCtrl'
    });
    $locationProvider.html5Mode(false).hashPrefix('!');
});
```

chapter4/controllers.js

```
/* chapter4/controllers.js excerpt */
helloWorldControllers.controller('AddedCustomerCtrl', ['$scope', '$routeParams',
function AddedCustomerCtrl($scope, $routeParams) {
    $scope.customerName = $routeParams.customer;
    $scope.customerCity = $routeParams.city;
}]);
```

You can see there are two path parameters, customer and city, for the addedCustomer route. The values are passed as arguments to a new controller, AddedCustomerCtrl.

Use AngularJS double curly braces to get access to and display both the customerName and customerCity properties in the view:

```
<div><b>Customer Name: </b> {{customerName}}</div>
<div><b>Customer City: </b> {{customerCity}}</div>
```

Conclusion

Unit testing AngularJS controllers allows us to validate the basic functionality of each controller. For now, our tests are very simple. Testing a controller that retrieves data from a REST service, for example, would be a more complex task.

End-to-end testing is a bit more involved and can be designed to completely exercise the entire application. For now, our E2E tests are also simple. E2E tests help to identify software defects early in the development process when used with CI build systems.

AngularJS Views and Bootstrap

AngularJS builds the views dynamically at runtime by merging the templates with the properties passed to the templates in the \$scope object. The result is pure HTML code bound to the ng-view directive.

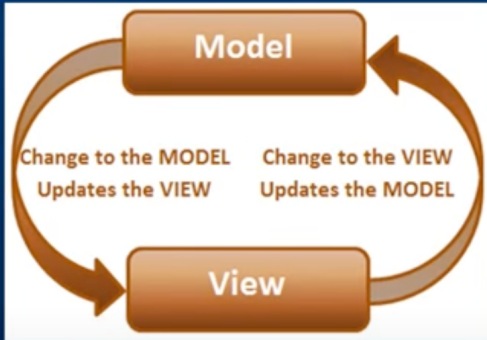
Two-way databinding

Two-way binding in AngularJS is the **synchronization** between the **view** and **model** (without any need to refresh the page or click a button). Any change in the model is reflected on the view and any change in the view is reflected on the model. Thus, this way of two-way binding ensures that your view and model are always **updated**. Also, the controller can remain separate from the view and focus on the model.²

More example [here](#).

If you are in need of the DVD with all the videos and PPT's, please visit
<http://pragimtech.com/order.aspx>

The Two Way Data-Binding, keeps the model and the view in sync at all times, that is a change in the model updates the view and a change in the view updates the model



```
graph TD; Model[Model] -- "Change to the MODEL Updates the VIEW" --> View[View]; View -- "Change to the VIEW Updates the MODEL" --> Model;
```

ng-model directive can be used with

- input
- select
- textarea

Binding expression updates the view when the model changes `{{ message }}`

ng-model directive updates the model when the view changes

```
<input type="text" ng-model="message" />
```

Exercises

Run the code provided [here](#), try to understand the workflow and answer:

1. The index.html is nothing but <div ng-view>, how the content being show?
2. How the menu is showing, modify the code such that the menu is show in any route location.

You should be able to have a basic understand of how the AngularJS works and make the effort to look at the OnTrack code to understand it. Try to Google it if the note is not mentioned some of the directive. Keep reading it if you want to know more in details.

² Stack Overflow, n.d.

AngularJS and REST Services

Reference

1. websiteseochecker.com. (n.d.). *Hashbang SEO In 2019 – A Complete Guide To Hash Bang URLs*. [online] Available at: <https://websiteseochecker.com/blog/hashbang-seo-in-2019-a-complete-guide-to-hash-bang-urls/> [Accessed 20 Sep. 2022].
2. Stack Overflow. (n.d.). angularjs - Two way databinding in angular js. [online] Available at: <https://stackoverflow.com/questions/39485905/two-way-databinding-in-angular-js> [Accessed 23 Sep. 2022].

