ATTUNE

AngularJS

By Juned Laliwala

Short description of AngularJS

- What is AngularJS?
- A Complete Client-Side solution
- The Zen of Angular

ATTUNE

Contents

1. Preface

1.1. About This Guide

The main intense for making this guide is to make a better understanding of the new framework named “AngularJS”. AngularJS is nothing but the framework for dynamic web Apps. During this whole guide, we will be more focussing on the general syntax and the coding of the framework needed to complete our task.

1.2. Intended Audience

This guided is intended for the developers or the beginners who want to have basic understanding of the framework which are in demand on date. This can be helpful to the students who are just entered into these coding techniques.

1.3. Revision History

This is the first book in the series of AngularJS 1.5.
This will cover the overall features and functionalities of AngularJS 1.5.

2. Introduction-AngularJS

AngularJS is a structural framework for making dynamic web apps. Along with basic HTML, it helps you to make HTML as a language templates and using HTMl syntax we can express our application's component successfully. In AngularJS there is one great functionality of data-binding which will eliminate much of the code you would otherwise have to write . And it all happens within the browser, making it an ideal partner with any server technology.

AngularJS is nothing but same as HTML. In HTML we are making dynamic web pages. In addition to it does not contain much more creative way to work with application so AngularJS will resolve this issue. In the coming sections we will see the reasons for the above issue.

2.1. Features of AngularJS

AngularJS is a great JavaScript framework that have numerous number of functionalities for not only developers, but designers as well!. In this guide, we will see all the basic functionalities and the also see the practical implementation of our basic features which are included as follws:

2.1.1. MVC Architecture

The first important feature, which comes to my mind, is the MVC or Model-View-Controller architecture. The main reason for making MVC is to split the web application into a more manageable structure. The MVC architecture comprises of three important elements, the model, view and controller.

Model

The model is as the name suggest, we have to place our data into the model. This particular data can be static or dyanlic, which can be tucked into a server that is miles far away from the client, and this can be possessed using JSON.

A model comprises of a simple JavaScript object called the scope. Tied to a controller, the model object receives the data (from the source) and delivers it to a view (HTML).

```
myApp.controller(
   'myController',
   ['$scope', function ($greet) {
      $greet.greetings = function () {
         $greet.theguest = 'Hello ' + $greet.name;
      }
   } ]
);
```

In the above script, the $scope is an object in the model. See how it is encapsulated inside the controller() method.

View

In Angular, the view comprises of HTML elements and directives. View is nothing but the display that we want to see in our browser is view. Other than markups, every view has as an expression (with curly braces), which is tied up to the scope object.

```
<div ng-app="myApp" ng-controller="myController">

  <p>Enter your name  <input type="text" ng-model="name" /></p>
  <p><input type="button" value="Click Me" ng-click="greetings()" /></p>

  <p> {{ theguest }} </p>

</div>
```

The above piece of markup, also known as Template in Angular, when bound, complied and loaded on the browser is then called the view.

Controller

The controller actually controls the entire business logic of your application. The initial stage is set here, that is, it initializes and registers the application by creating a new Angular Module.

```
(function (){
var app = angular.module('myapp ', [ ]);
app.controller('mycontroller',function(){
}
}();
```

In the above example, we can see that we have made one module named "myapp". Inside this module we have created one controlled named "mycontroller" which is made using "app.controller".

2.1.2. Two-way Data Binding

Angular Data-Binding creates a link between model and view.

The two-way Data Binding is an extraordinary feature ever integrated in a JavaScript framework. In two-way data binding, any change made in the view will reflect in model, similarly changes made in the model will reflect in the view. It is a two way process.

In Angular, we need to use the ng-model directive to create a two-way data binding. This directive will bind the model to the view. We'll do an example to understand the process better.

```
<div ng-app="">
      <p>
      <label>Current Bid Value</label>
    <input type="text" ng-model="bid" ng-init="bid ='75'" />
   </p>
   <p> ${{bid}} </p>

</div>
```

2.1.3. Templates

In Angular, a template usually means a view with HTML elements attached to Angular Directives , add markup for data binding using expressions (with curly braces).

```
<div ng-app="BiddingApp">
<input type="text" ng-model="bid" />

   <p> ${{bid}} </p>
</div>
```

An Angular template is exactly the same as HTML, except for its attributes. If we want to make it dynamic we have to work with model and controller.
Now let us see the main elements and the successive attributes which basically makes up the template.

a) Directive – In the above example, the ng-app and ng-model are directives.

b) Markup – Binding the view with a model using the curly braces {{ bid }} (expressions) is the markup.

c) Filters – Filters are useful for formatting the value in an expression. It is very useful. I have explained about filters later in this article.

d) Form Controls – We can use Angular Form Controls to validate user inputs. Let us assume the form has an input field, which accepts email ids only. The form control will validate the input and display a message if the value is invalid.

2.1.4. Directives

Angular Directives are attributes applied in the View. Attached to HTML elements, the directives augment the elements with new behaoviors. Did you see the above examples; each element has directives, prefixed with ng-. Whenever we attach a directive with an element, it tells AngularJS that it is a part of Angular app and Angular treats it that way.

Here in this example, I have used useful directive, called the ng-include, to add the content of an .html file.

```
<!DOCTYPE html>
<html>
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.8/angular.min.js"><
/script>
</head>
<body>
  <div ng-app>
    <div ng-include="'hello-world.htm'"></div>
  </div>
</body>
</html>
```

The example has two distinct directives. The first is the ng-app to initialize the app. This will tell Angular about the app and its features. Next is the ng-include directive, which is useful for displaying data extracted from an external file.

2.1.5. Expressions

Angular Expressions are JavaScript like expressions, however with lots of difference. Written inside two curly braces, these expressions will bind Angular application data to HTML elements.

2.1.6. Modules

In the section of the MVC, we have seen the architecture which helps in designing Angular application by splitting the app into little manageable structures. The Modules are pillar of this architecture. A module creates a well-defined structure, which will keep everything organized, at one place.

A single application may more than one module. By creating a new module, each application is first initialized and registered.

```
<script>
  // INITIALIZE FIRST APP.
  var mailApp = angular.module('myMails', []);

  // INITIALIZE THE SECOND APP.
  var bullionApp = angular.module('myMoney', []);
</script>
```

2.1.7. Scope

A Scope is a JavaScript object that sits inside the model, and delivers data to the view. The view's expression will receive the value(s) from the $scope and display the result exactly where the expression is located.

```
<div ng-app="myMoney" ng-controller="moneyController">
  <p>Dollar Today: {{ rate }} </p>
</div>

<script>
  var bullionApp = angular.module('myMoney', []);

  bullionApp.controller(
        'moneyController',
        ['$scope', function ($currency) {
            $currency.rate = '62.15';
        } ]
    );
</script>
```

Output
Dollar Today: 62.15

See, how the expression property rate in the view is accessed inside the controller using $scope. The variable $currency is passed to the controller and assigns the value to the property rate.

2.1.8. Filters

An Angular Filter modifies the data before presenting it to the user. We can use these filters with expressions and directives. A filter is usually a predefined keyword, used with the symbol "|" (a pipe).
While explaining about Template in this article (see the template section above), I have used an expression to display the bid amount.

```
<div ng-app="BiddingApp">
    <input type="text" ng-model="bid" />

    <p> ${{bid}} </p>
</div>
```

The expression has a text inside the curly braces {{bid}} and is prefixed with the "$" (dollar) sign. The Angular Filter currency would spare me from using the "$" sign. Simply add a "|" pipe after the text bid and add currency.

```
<p> {{bid  | currency}} </p>
```

Filters are case-sensitive. Therefore, be careful while adding it in your application.Some commonly used filters are…

a) currency b) number c) uppercase d) lowercase e) date f) orderBy g) filter

2.2. Advantages and Disadvantages

Advantages	Disadvantages
No need to use observable functions; Angular analyses the page DOM and builds the bindings based on the Angular-specific element attributes. That requires less writing, the code is cleaner, easier to understand and less error prone.	Angular is big and complicated. With multiple ways to do the same thing it is hard to tell which way is better for particular task. Mastering Angular over the "Hello world" level requires considerable efforts. Different developers' coding styles and habits might complicate integration of different components into a whole solution.
Angular modifies the page DOM directly instead of adding inner HTML code. That is faster.	The lifecycle of Angular application is complex, and to master it you really need to read the code. Compile and link are not intuitive, and specific cases can be confusing (recursion in compile, collisions between directives etc.).
Data binding occurs not on each control or value change (no change listeners) but at particular points of the JavaScript code	As the project grows with time, you most likely will need to throw away existing implementations and create new versions

execution. That dramatically improves performance as a single bulk Model/View update replaces hundreds of cascading data change events.	using different approaches. Angular implementations scale poorly.
Quite a number of different ways to do the same things, thus accommodating to particular development styles and tasks.	More than 2000 watchers can severely lag the UI. That limits the complexity of your Angular forms, especially big data grids and lists.
Extended features such as dependency injection, routing, animations, view orchestration, and more.	
Supported by IntelliJ IDEA and Visual Studio .NET IDEs.	
Supported by Google and a great development community.	

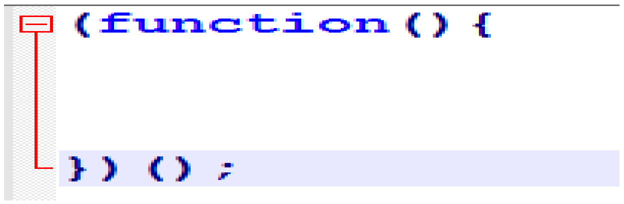
3. Implementation with AngularJS

3.1 Basic Syntax

In this section, we are going to see the general syntax of the Javascript file, modules and controller.

3.1.1 app.js

This JavaScript file plays a vital role in our angularjs projects. This file is the javascript of the application. Whenever we are starting with our application the main important work is that the developers will have to start making their functionalities of the project using the syntax as shown below.



```
(function() {


})();
```

Fig: General Syntax of app.js file

The syntax shown above depicts that app.js file includes one function and that function is nothing but the functionalities of the application.
As we can see that function() is included inside the two round-brackets "( )". So we should remember this point.

3.1.2 Creating a Module

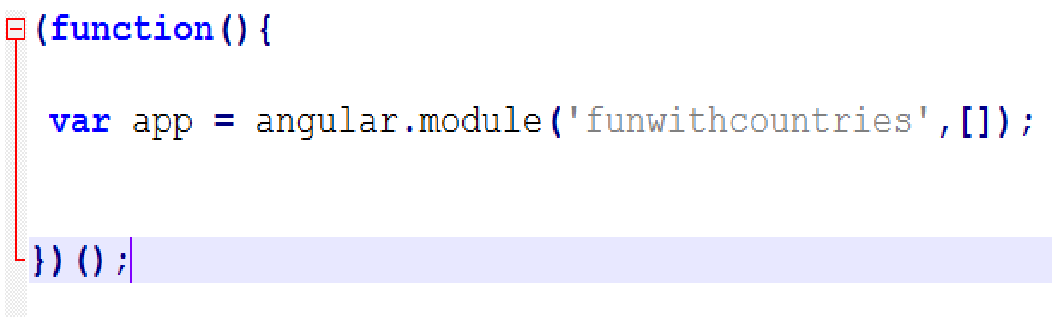
Module in general term is a container of different parts like controllers, services, filters, directives, etc..
Most applications have a main method that instantiates and wires together the different parts of the application.

Angular apps don't have a main method. Instead modules declaratively specify how an application should be bootstrapped. There are several advantages to this approach:

- The declarative process is easier to understand.

- You can package code as reusable modules.
- The modules can be loaded in any order (or even in parallel) because modules delay execution.
- Unit tests only have to load relevant modules, which keeps them fast.
- End-to-end tests can use modules to override configuration.

The module is made in app.js file as :



```
(function(){

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


})();
```

Fig: Creating a Module

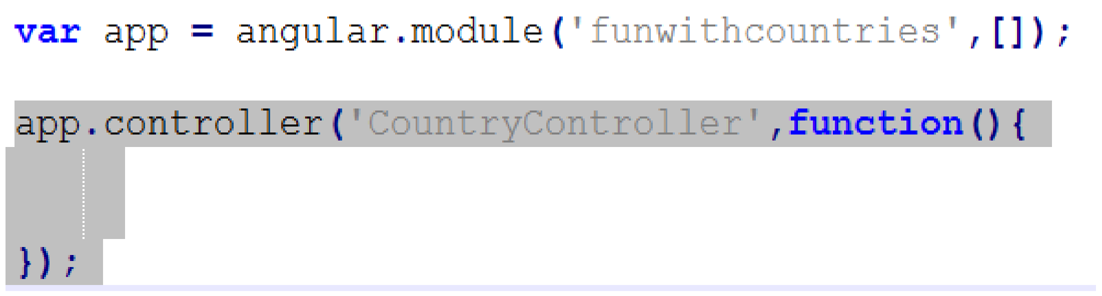
As we can see from the above figure that the name of the module is “funwithcountries”. The module is created with angular.module.

3.1.3 Creating a Controller

In Angular, a Controller is defined by a JavaScript constructor function that is used to augment the Angular Scope.

When a Controller is attached to the DOM via the ng-controller directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function. A new child scope will be created and made available as an injectable parameter to the Controller's constructor function as $scope.

If the controller has been attached using the controller as syntax then the controller instance will be assigned to a property on the new scope.



```
var app = angular.module('funwithcountries',[]);

app.controller('CountryController',function(){

});
```

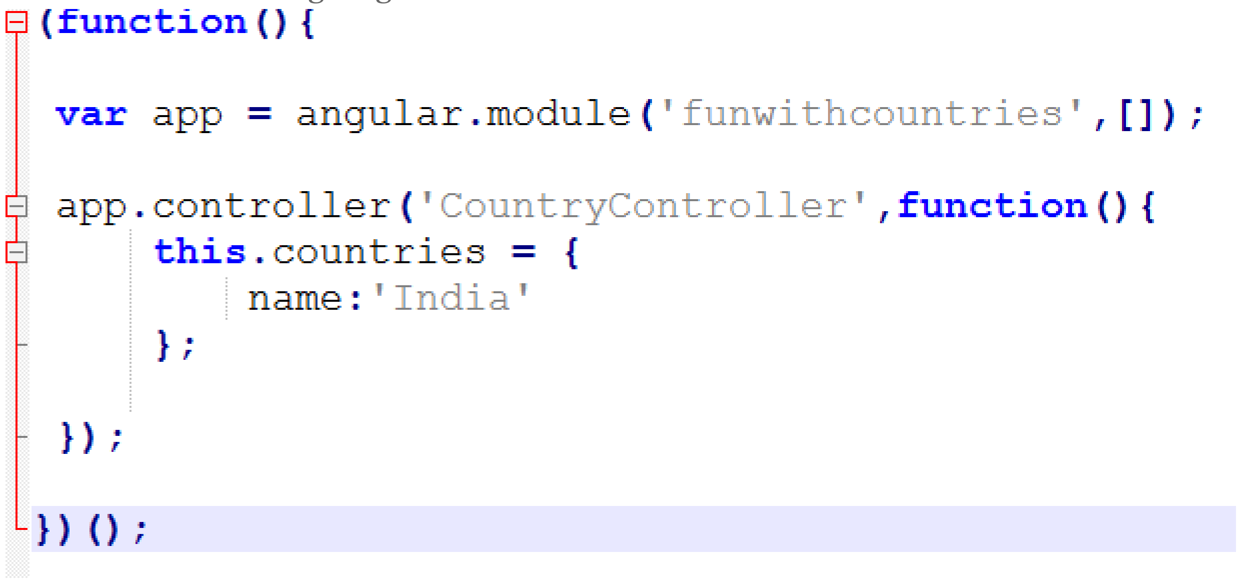
Fig: Creating a Controller

As from the diagram we can say that the name of the controller is "CountryController". Controller is created with "app.controller".

3.2 Data Binding Concept

In the previous section we have almost seen the basic meaning of the term data binding. In this section we will see the practical implementation of the data binding.

Consider the following diagram:

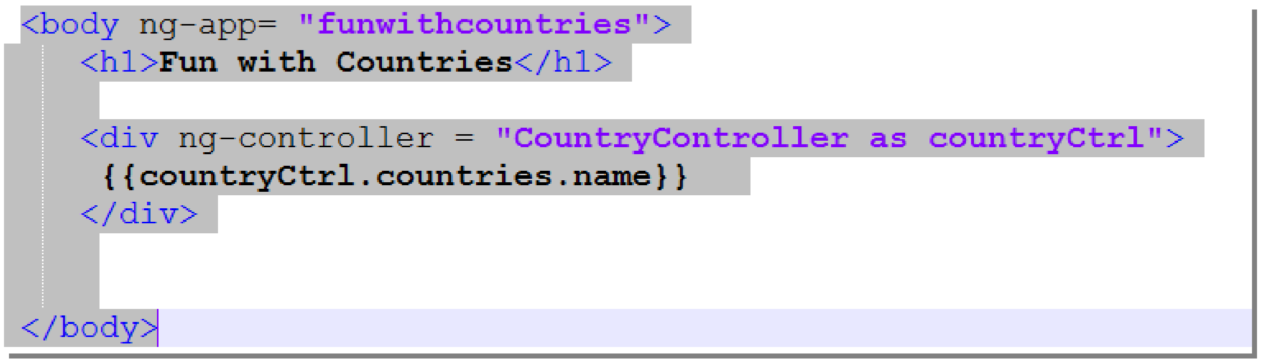


```
(function(){

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

 app.controller('CountryController',function(){
      this.countries = {
          name:'India'
      };

 });

})();
```

Fig: Controller with some function

From the above diagram we can say that inside the CountryController we are including one function and this function is recognized with variable "countries". Inside the countries method we have one parameter named "name" which specifies that the name of the country is India.

Now if we reward back to our main web page where our file is made to call our method. Here we are using HTML file coding.



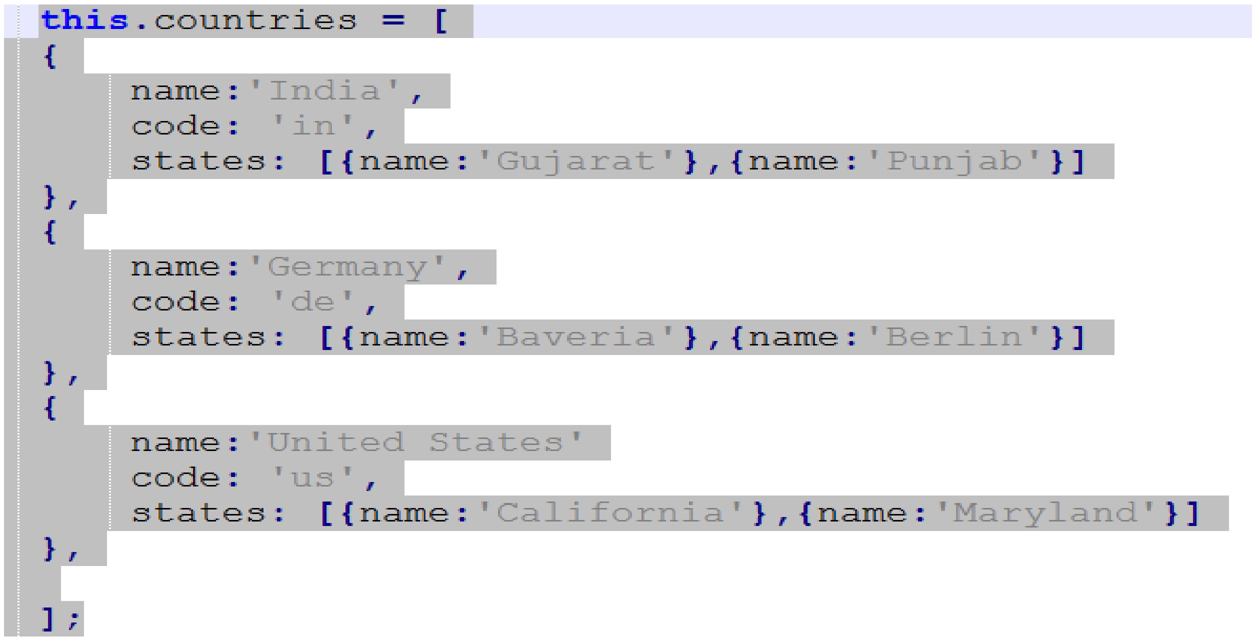
```
<body ng-app= "funwithcountries">
   <h1>Fun with Countries</h1>

   <div ng-controller = "CountryController as countryCtrl">
    {{countryCtrl.countries.name}}
   </div>


</body>
```

Fig: Data Binding

In the above diagram we have one HTML file, in which there is one division part. Inside that division part we are calling our controller named “CountryController” using a directive named ng-controller.

{{countryCtrl.countries.name}} indicates that we are binding the parameter named “name” which is included in “countries” variable which is inside the “CountryController” Controller.

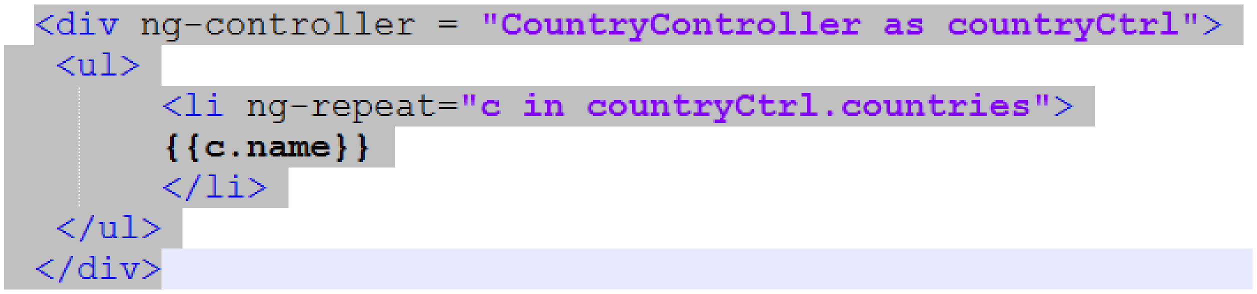
Remember that we have given the short name of CountryController as countryCtrl in our div part.



```
this.countries = [
{
    name:'India',
    code: 'in',
    states: [{name:'Gujarat'},{name:'Punjab'}]
},
{
    name:'Germany',
    code: 'de',
    states: [{name:'Baveria'},{name:'Berlin'}]
},
{
    name:'United States'
    code: 'us',
    states: [{name:'California'},{name:'Maryland'}]
},

];
```

Fig: Controller with more parameters

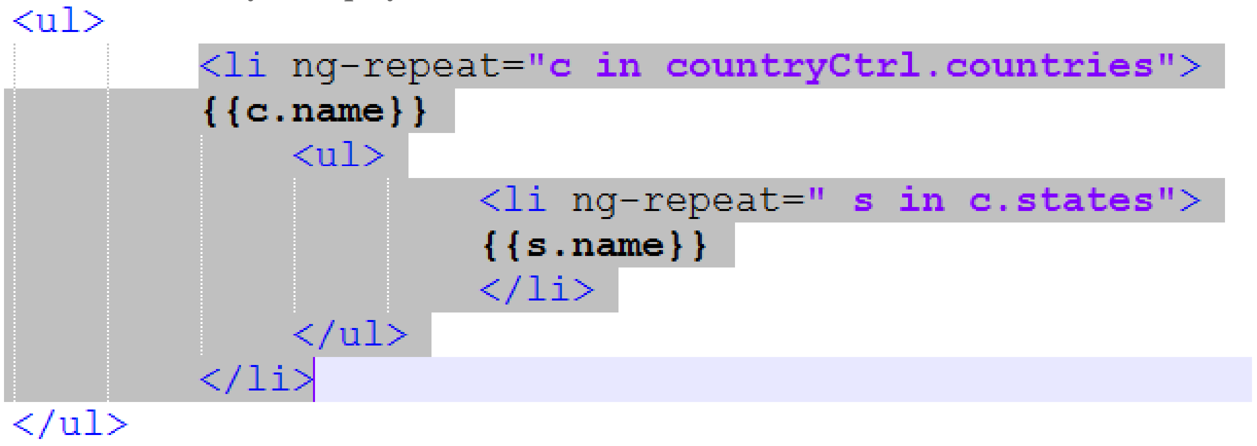
From the above figure we can see that we have included more parameters. Apart from name, we have also included code and various states of the countries. So we have different countries with different parameters.
Suppose we want to list only the names of the country. So we will use the



```
<div ng-controller = "CountryController as countryCtrl">
  <ul>
          <li ng-repeat="c in countryCtrl.countries">
          {{c.name}}
          </li>
  </ul>
</div>
```

Fig: Controller with only Name of Country

We have used "c" as short name for countries part of CountryController. So we are just binding the values using {{c.name}} which means the name of the countries will only is displayed.



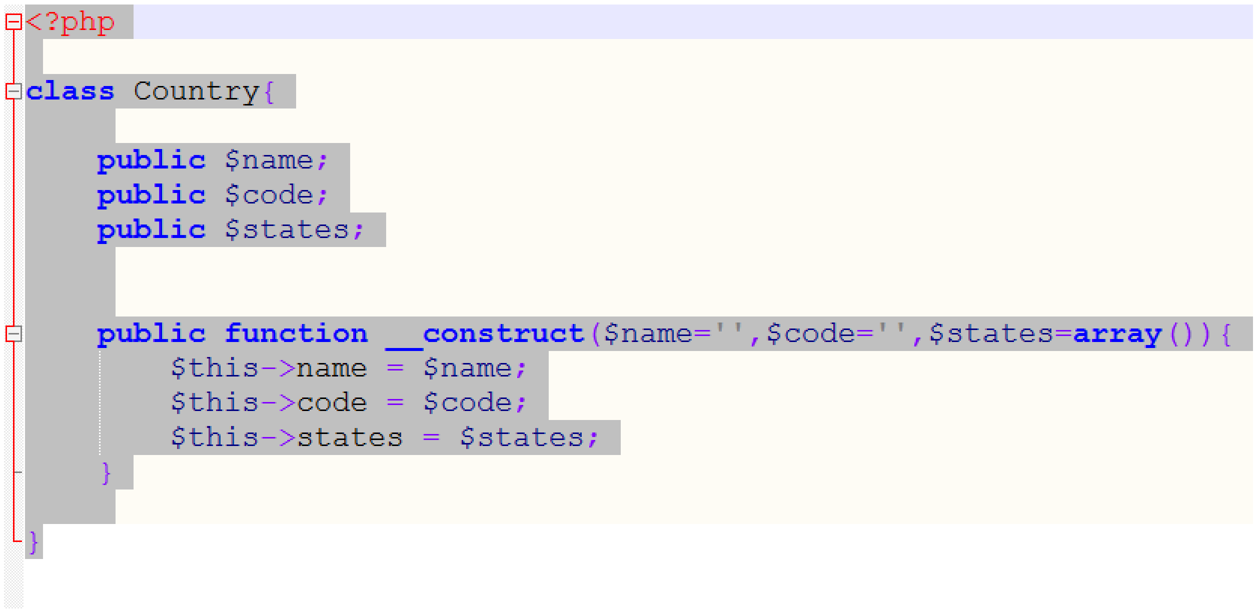
```
<ul>
        <li ng-repeat="c in countryCtrl.countries">
        {{c.name}}
            <ul>
                <li ng-repeat=" s in c.states">
                {{s.name}}
                </li>
            </ul>
        </li>
</ul>
```

Fig: Controller with States

This diagram shows that we have listed the states of different countries in the page.

3.3 Working with Class

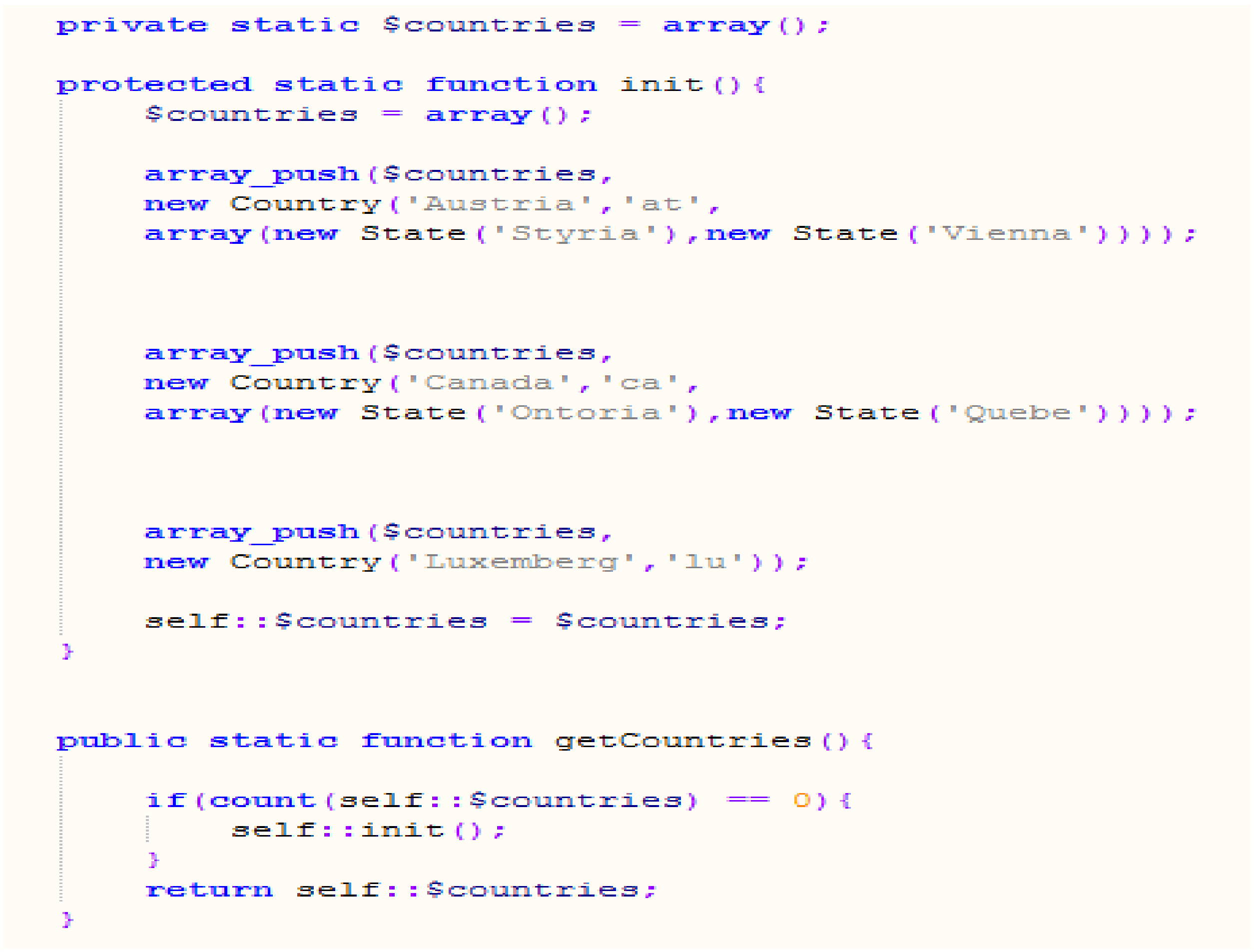
In this guide we are referring to the word "class", which means that we are referring to store some definite amount of values into our application. Apart from the database we will enter the values statically to show how the application works. So now we will create a Country class and State Class to point to the country and state.



```php
<?php

class Country{

    public $name;
    public $code;
    public $states;


    public function __construct($name='',$code='',$states=array()){
        $this->name = $name;
        $this->code = $code;
        $this->states = $states;
    }

}
```

Fig: Country.php

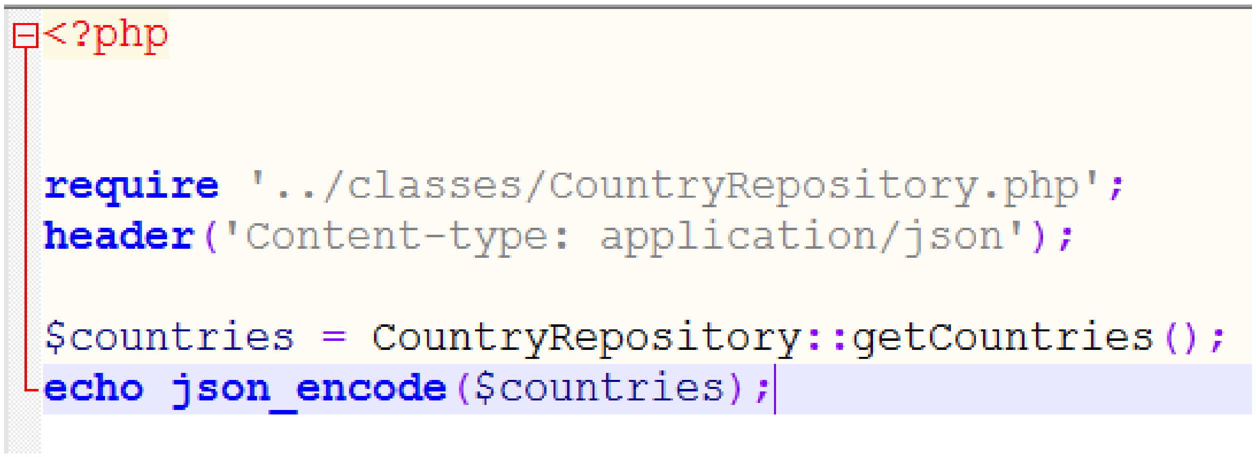
As we can see that Country Class have three parameters named name, code and state. In the next function named construct, we are importing our parameter to this parameters. This parameter basically points to our name, code and state of the country that we are importing.



```php
private static $countries = array();

protected static function init(){
    $countries = array();

    array_push($countries,
    new Country('Austria','at',
    array(new State('Styria'),new State('Vienna'))));


    array_push($countries,
    new Country('Canada','ca',
    array(new State('Ontoria'),new State('Quebe'))));


    array_push($countries,
    new Country('Luxemberg','lu'));

    self::$countries = $countries;
}


public static function getCountries(){

    if(count(self::$countries) == 0){
        self::init();
    }
    return self::$countries;
}
```

Fig: CountryRepository.php

We are referring to this file just for entering the values statically. As we can see that we have made one variable named “$countries” which initially holds an array variable. In the next function we are initiating the countries variable by using array_push function.
To call to this particular values we are referring to the **getCountries( )** named function.
getCountries( ) initially checks whether all the parameters within the countries variable are empty. If they are empty then filling of $countries variable starts and we will insert the values into the application.



```php
<?php


require '../classes/CountryRepository.php';
header('Content-type: application/json');

$countries = CountryRepository::getCountries();
echo json_encode($countries);
```

Fig: getCountries.php

This is the file to get the parameter of the countries variable.

3.4 Implementing HTTP Services

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.

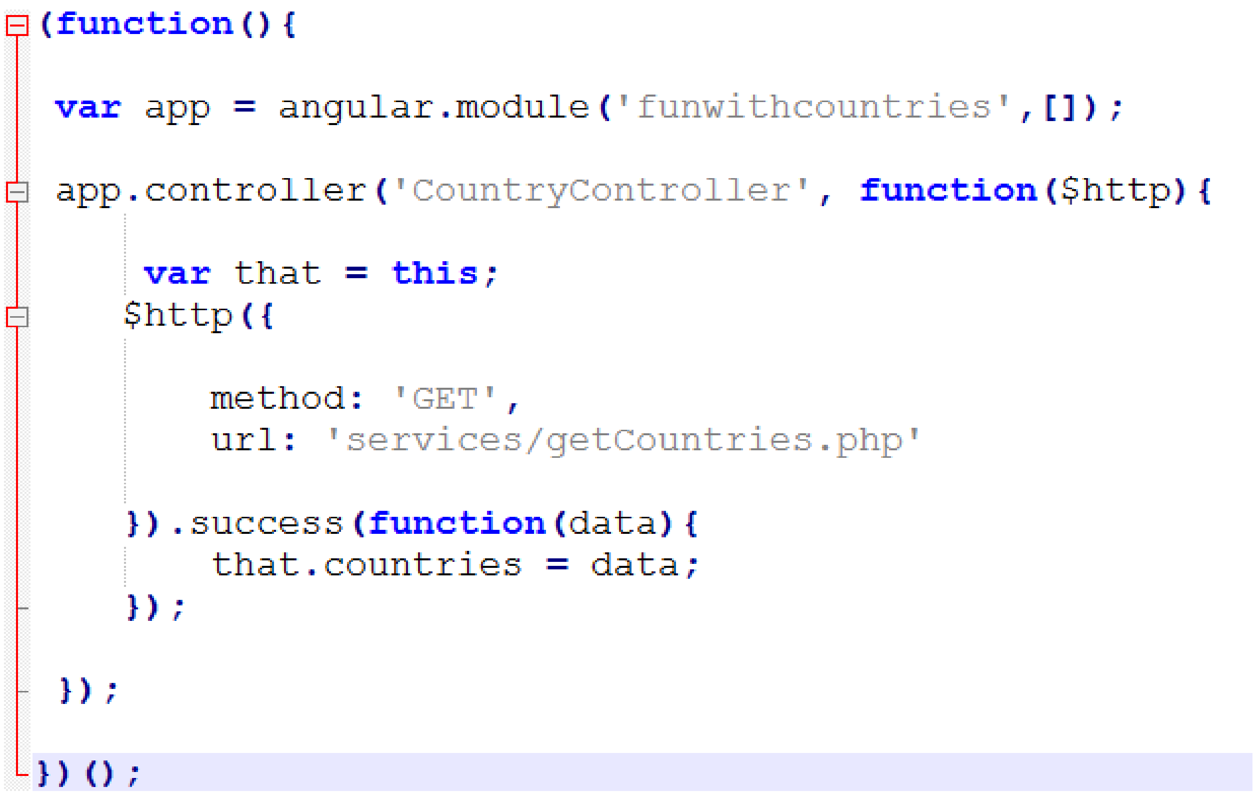
```
// Simple GET request example:
$http({
 method: 'GET',
 url: '/someUrl'
}).then(function successCallback(response) {
   // this callback will be called asynchronously
   // when the response is available
 }, function errorCallback(response) {
   // called asynchronously if an error occurs
   // or server returns response with an error status.
 });
```

Fig: General syntax of implementing $http services

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.

A response status code between 200 and 299 is considered a success status and will result in the success callback being called. Note that if the response is a redirect, XMLHttpRequest will transparently follow it, meaning that the error callback will not be called for such responses.



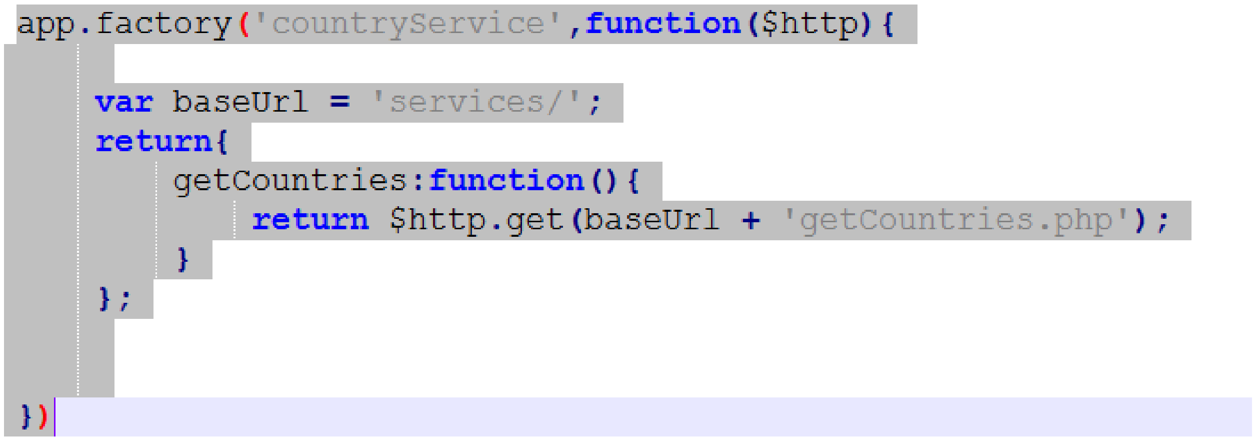
```
(function(){

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

 app.controller('CountryController', function($http){

     var that = this;
    $http({

        method: 'GET',
        url: 'services/getCountries.php'

    }).success(function(data){
        that.countries = data;
    });

 });

})();
```

Fig: Implementing $http in CountryController

3.4.1 Working with HTTP 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.

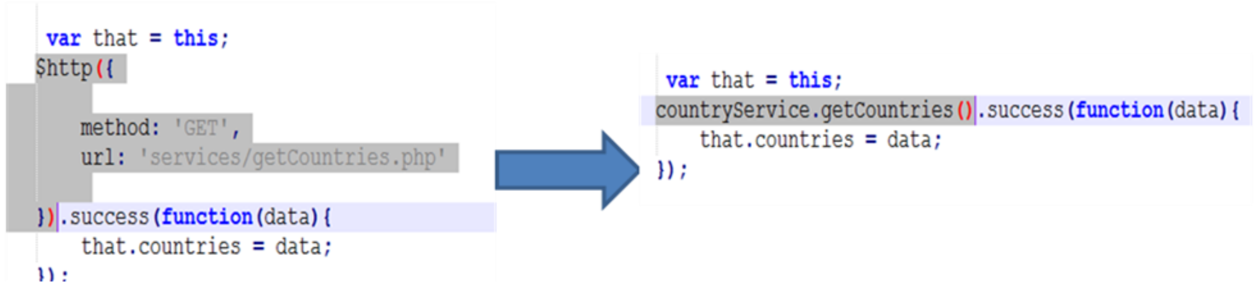


```
app.factory('countryService',function($http){

    var baseUrl = 'services/';
    return{
        getCountries:function(){
            return $http.get(baseUrl + 'getCountries.php');
        }
    };


})
```

Fig: Creating "countryService"

Inside the Country Controller we are creating new service named "countryService" using app.factory() method. The working of the countryService is simple as it is using the getCountries file.

Now if we recall back to our previous file that we were using $http method, instead of it we will now use countryService to get the getCountries file.



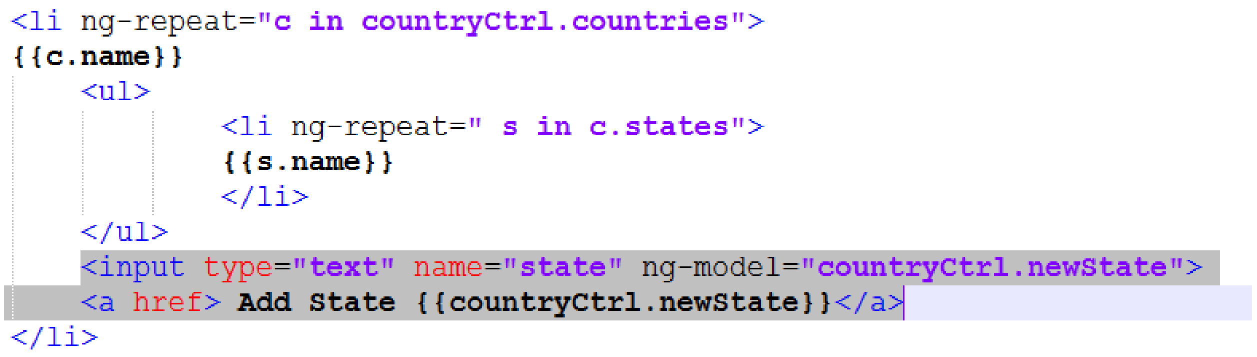
```
var that = this;
$http({

    method: 'GET',
    url: 'services/getCountries.php'

}).success(function(data){
    that.countries = data;
});
```

```
var that = this;
countryService.getCountries().success(function(data){
    that.countries = data;
});
```

Fig: Calling the countryService

3.5 Implementing Two-Way Data Binding

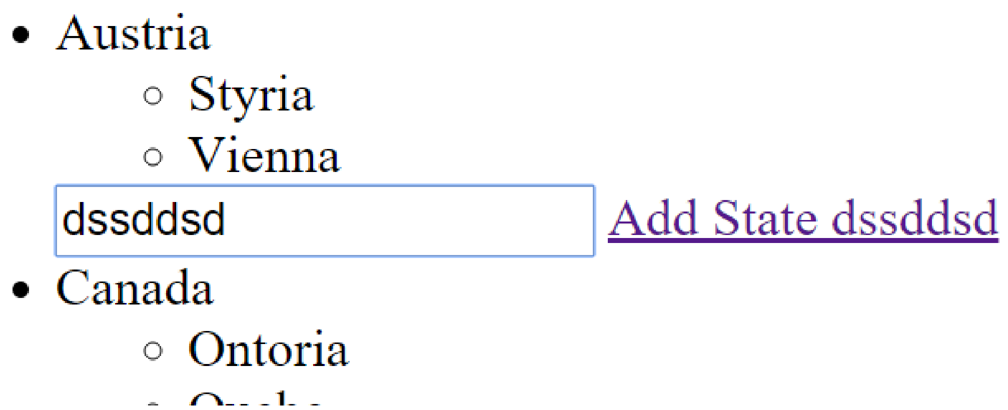
Two-way Data Binding concept is to input and output the same values at the same time. Here in this guide we will insert the value of a new state into a country by adding a new textbox as shown below.



```
<li ng-repeat="c in countryCtrl.countries">
{{c.name}}
    <ul>
            <li ng-repeat=" s in c.states">
            {{s.name}}
            </li>
    </ul>
    <input type="text" name="state" ng-model="countryCtrl.newState">
    <a href> Add State {{countryCtrl.newState}}</a>
</li>
```

Fig: Two-Way Data Binding

From the figure we can see that we have used ng-model to bind a value to our application. Inside the CountryController we have implemented a new method named “newState”.

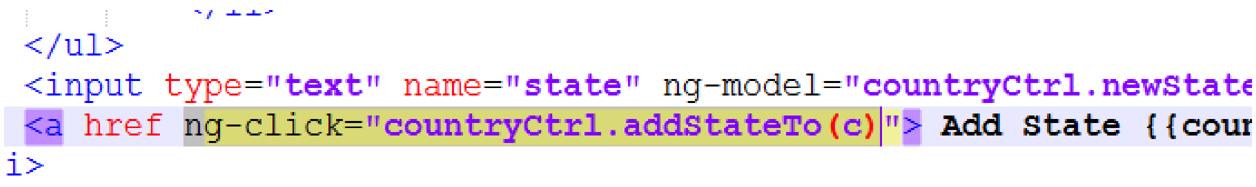


- Austria
  - Styria
  - Vienna

  dssddsd Add State dssddsd
- Canada
  - Ontoria

Fig: Output of Two-Way Data Binding

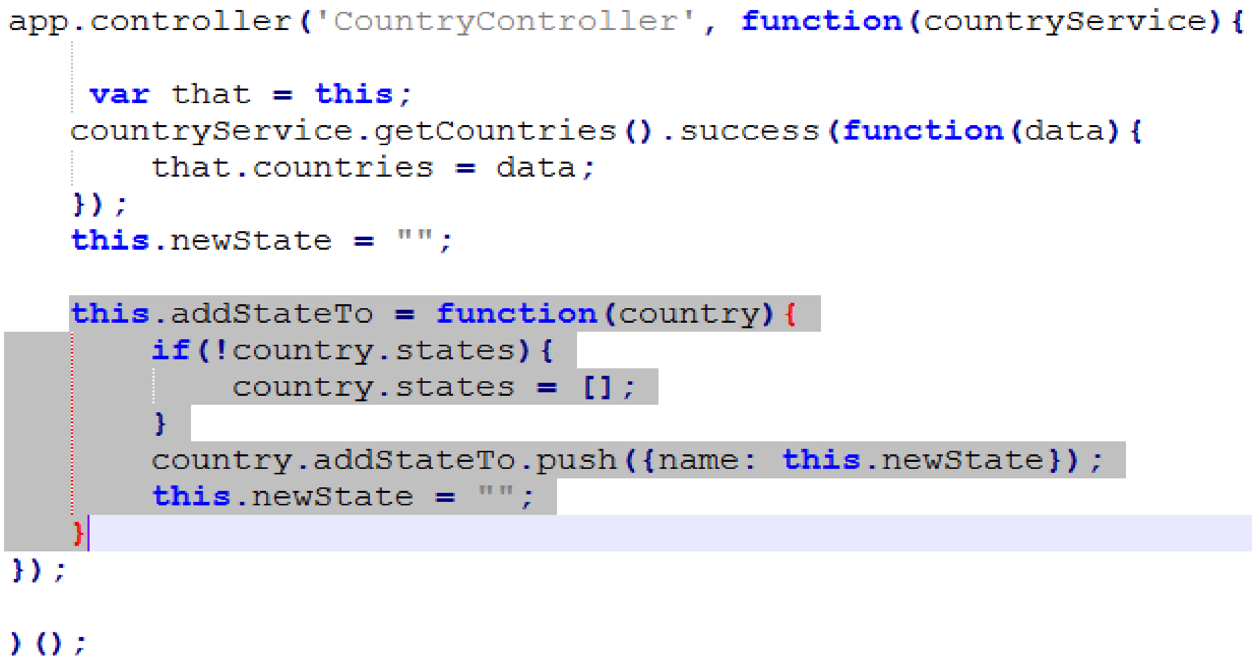
This data binding was just simply to show the process of doing the two-way binding. Now in the next section we will see the process to enter the value into the application.



```
</ul>
<input type="text" name="state" ng-model="countryCtrl.newState
<a href ng-click="countryCtrl.addStateTo(c)"> Add State {{cour
i>
```

Fig: Adding a method inside Controller

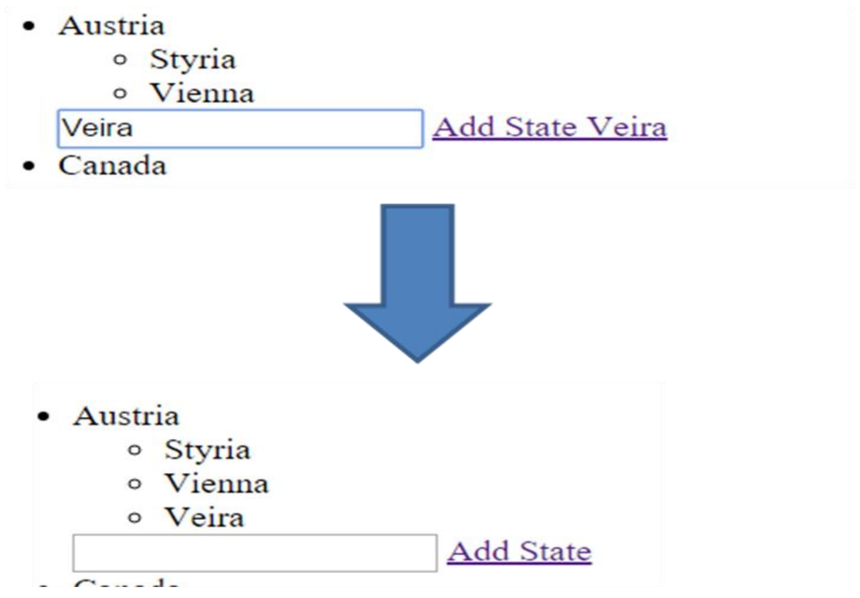
From the figure we can see that we are using ng-click to direct our call to addStateTo () function which is going to add the state in a particular country.



```
app.controller('CountryController', function(countryService){

    var that = this;
   countryService.getCountries().success(function(data){
       that.countries = data;
   });
   this.newState = "";

   this.addStateTo = function(country){
       if(!country.states){
           country.states = [];
       }
       country.addStateTo.push({name: this.newState});
       this.newState = "";
   }
});

)();
```

Fig: Calling a method inside controller

As we can see that a new function has been added to the controller. This function will basically add a state to the country. Here also two-way data binding will take place but the result will be stored into particular country.


- Austria
  - Styria
  - Vienna

  Veira [Add State Veira]
- Canada

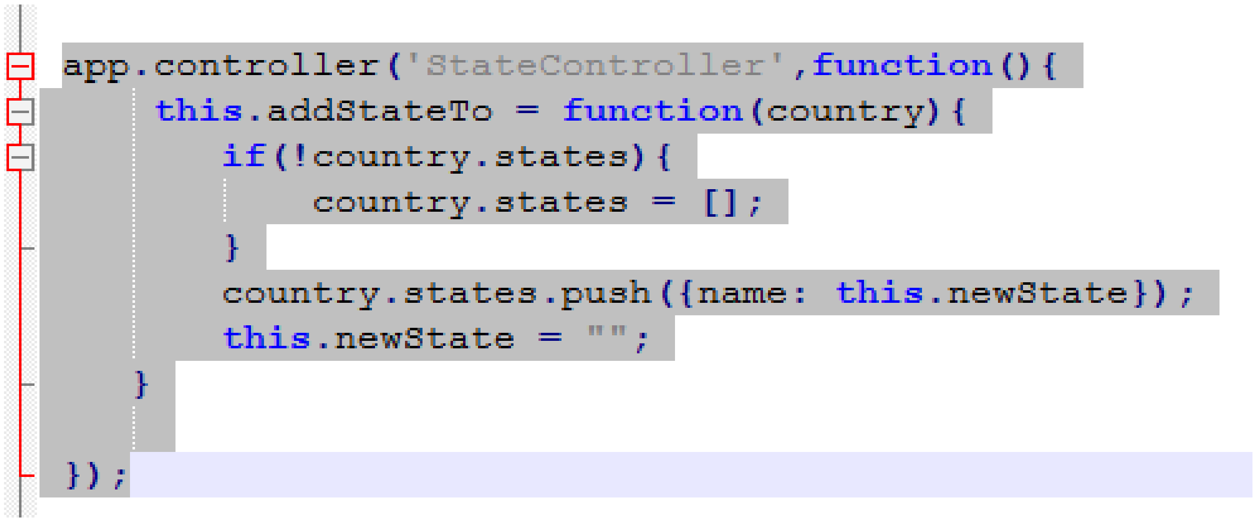
- Austria
  - Styria
  - Vienna
  - Veira

  [Add State]

Fig: Output of Two-way Data Binding

4. Creating a State Controller

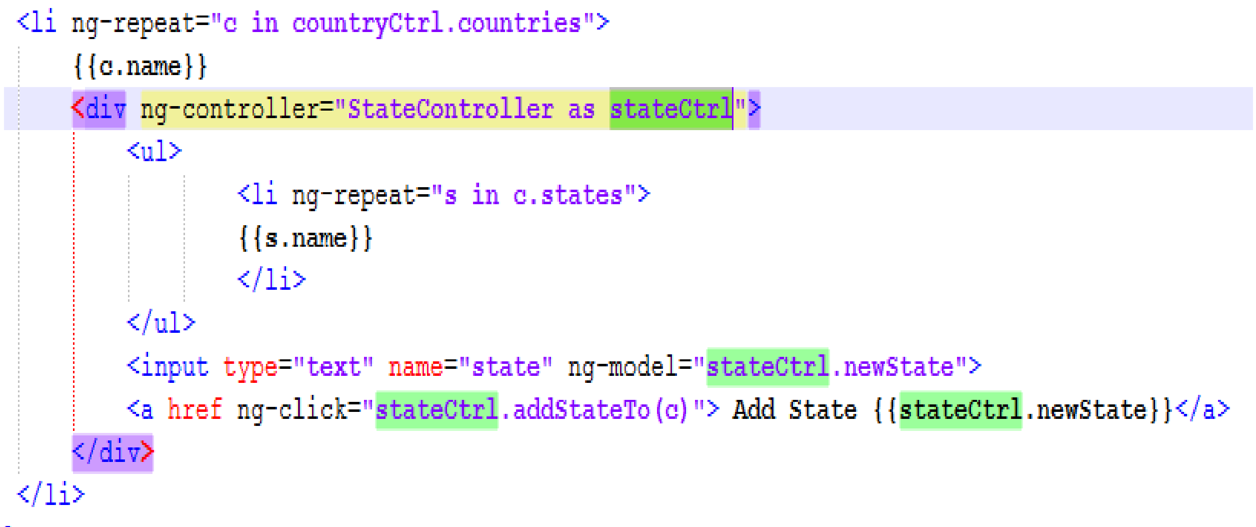
From the sections covered so far, we have seen that this application is being working on the two main blocks namely Country and State. So now what we will do, we will just make a new controller named "StateController".
This StateController will take the responsibility of the state part.



```
app.controller('StateController',function(){
    this.addStateTo = function(country){
        if(!country.states){
            country.states = [];
        }
        country.states.push({name: this.newState});
        this.newState = "";
    }

});
```

Fig: StateController

Now as we have made changes in our StateController , we will simultaneously have to make changes into our HTML file as :



```
<li ng-repeat="c in countryCtrl.countries">
    {{c.name}}
    <div ng-controller="StateController as stateCtrl">
        <ul>
                <li ng-repeat="s in c.states">
                {{s.name}}
                </li>
        </ul>
        <input type="text" name="state" ng-model="stateCtrl.newState">
        <a href ng-click="stateCtrl.addStateTo(c)"> Add State {{stateCtrl.newState}}</a>
    </div>
</li>
```

Fig:HTML file with StateController

5. Custom Directive

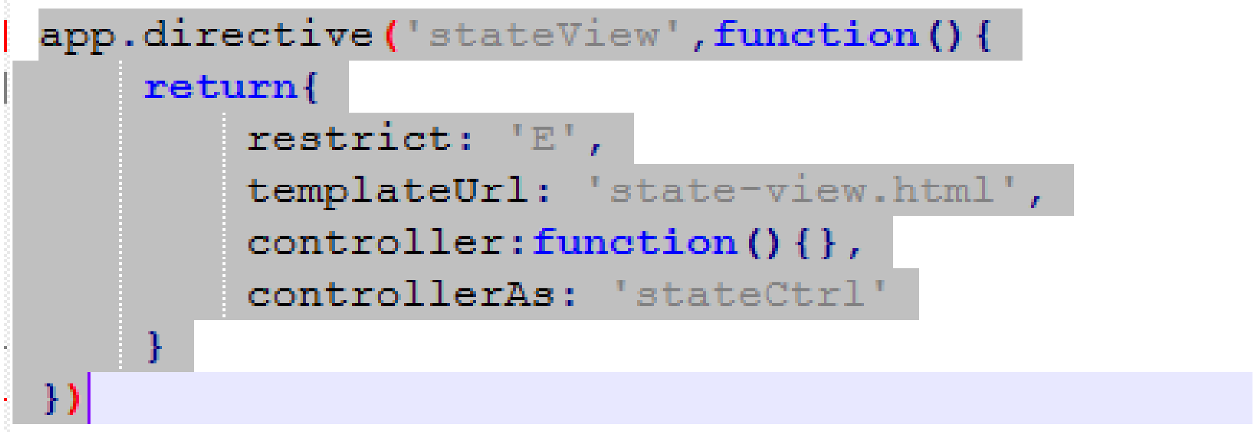
Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile() method of the custom directive then process the element using link() method of the custom directive based on the scope of the directive. AngularJS provides support to create custom directives for following type of elements.

Element directives – Directive activates when a matching element is encountered.

Attribute – Directive activates when a matching attribute is encountered.

CSS – Directive activates when a matching css style is encountered.

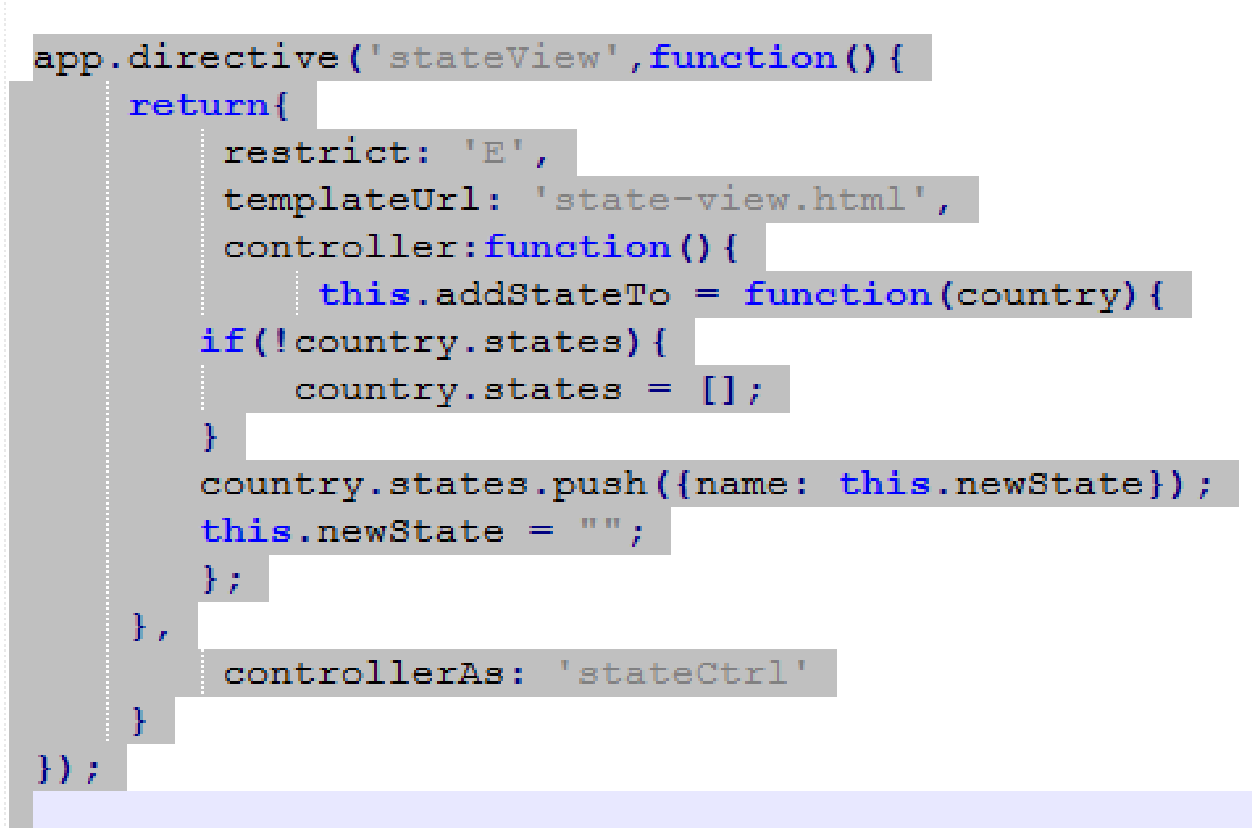
Comment – Directive activates when a matching comment is encountered.



```
app.directive('stateView',function(){
        return{
                restrict: 'E',
                templateUrl: 'state-view.html',
                controller:function(){},
                controllerAs: 'stateCtrl'
        }
})
```

Fig: General Syntax of Custom Directive

From the diagram we can depict that we are creating one new directive named "stateView" which are having different parameters namely:
Restrict: 'E': directive is Element directive
templateUrl: Path where to view
controller: The functionality of the controller that we have to use.
controllerAs: The name of the Controller to which we are focussing.

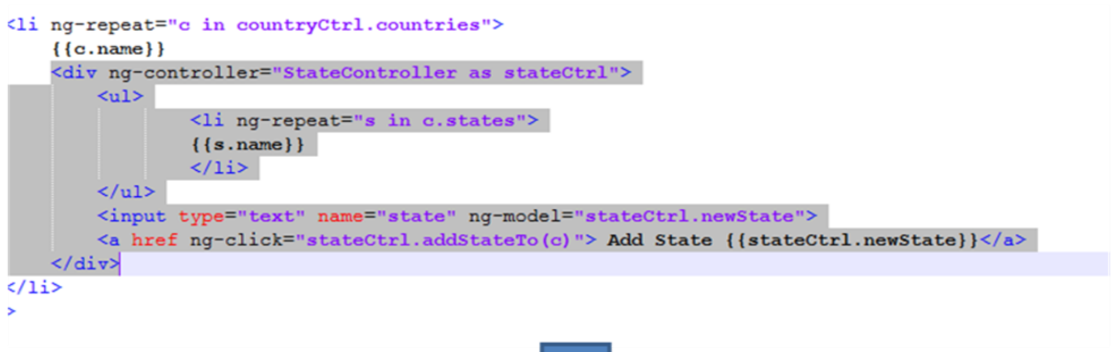


```
app.directive('stateView',function(){
    return{
        restrict: 'E',
        templateUrl: 'state-view.html',
        controller:function(){
            this.addStateTo = function(country){
       if(!country.states){
          country.states = [];
       }
       country.states.push({name: this.newState});
       this.newState = "";
       };
    },
        controllerAs: 'stateCtrl'
    }
});
```

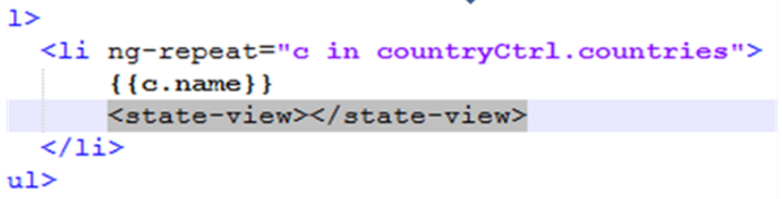
Fig: Directive with addStateTo function

From the diagram we can say that in the state-view directive we are making one functionality to add a state to our country. The Controller that we are referring is State Controller. The Functionality of adding the state remains the same.

In the above figure we can see that we have removed the part of the state Controller and named it as <state-view></state-view> which primarily means that we are including the state-view.html file inside that part. So whatever part that we have included in state-view.html will be displayed in that part.

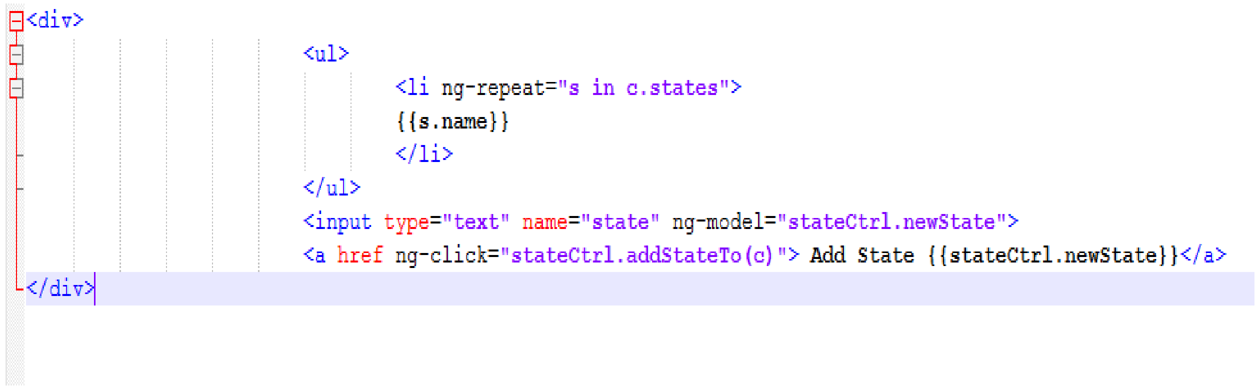


```
<li ng-repeat="c in countryCtrl.countries">
    {{c.name}}
    <div ng-controller="StateController as stateCtrl">
        <ul>
            <li ng-repeat="s in c.states">
            {{s.name}}
            </li>
        </ul>
        <input type="text" name="state" ng-model="stateCtrl.newState">
        <a href ng-click="stateCtrl.addStateTo(c)"> Add State {{stateCtrl.newState}}</a>
    </div>
</li>
>
```



```
l>
  <li ng-repeat="c in countryCtrl.countries">
      {{c.name}}
      <state-view></state-view>
  </li>
ul>
```

Fig: <state-view>

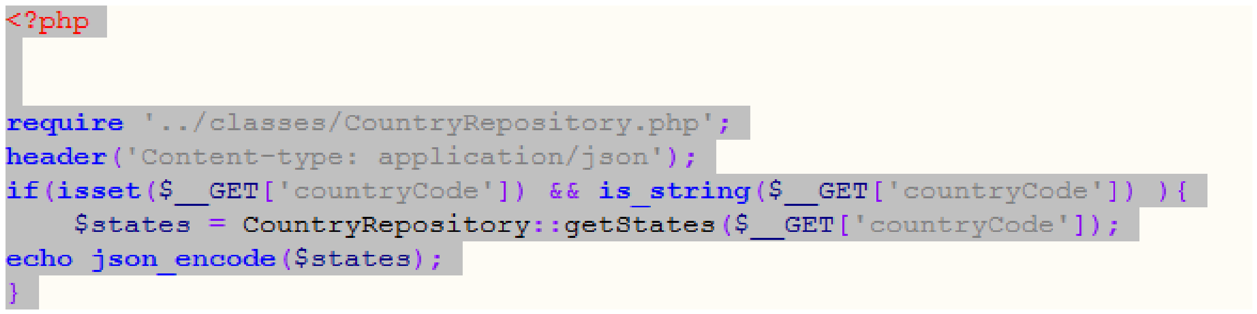


```
<div>
                    <ul>
                        <li ng-repeat="s in c.states">
                        {{s.name}}
                        </li>
                    </ul>
                    <input type="text" name="state" ng-model="stateCtrl.newState">
                    <a href ng-click="stateCtrl.addStateTo(c)"> Add State {{stateCtrl.newState}}</a>
</div>
```

Fig: state-view.html

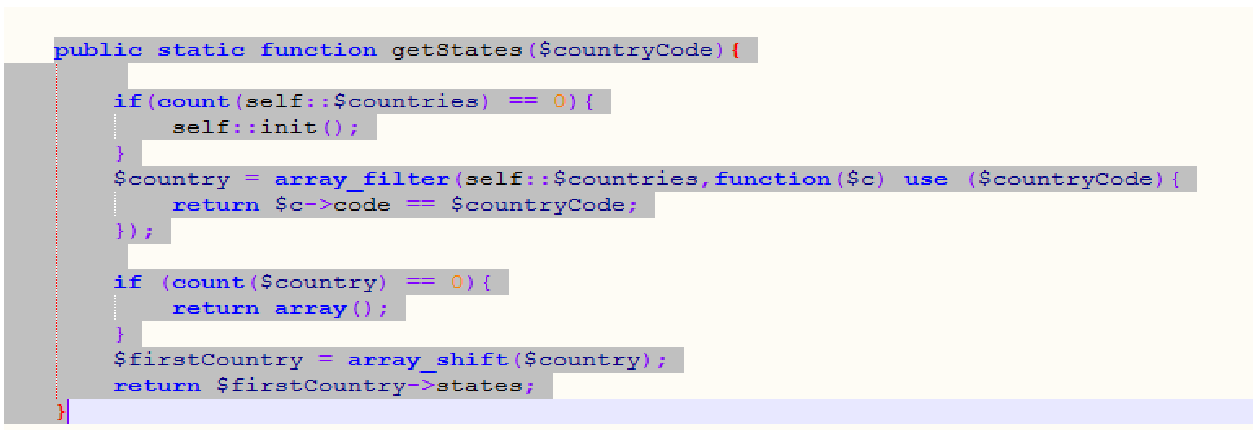
6. Creating getStates() function

As we know that we have now made a state controller for handling various activities related to state. So now we will make getStates( ) file to retrieve the list of the states specific to particular country code. As we can see that we are passing the parameters to get the countrycode.



```php
<?php


require '../classes/CountryRepository.php';
header('Content-type: application/json');
if(isset($__GET['countryCode']) && is_string($__GET['countryCode']) ){
    $states = CountryRepository::getStates($__GET['countryCode']);
echo json_encode($states);
}
```

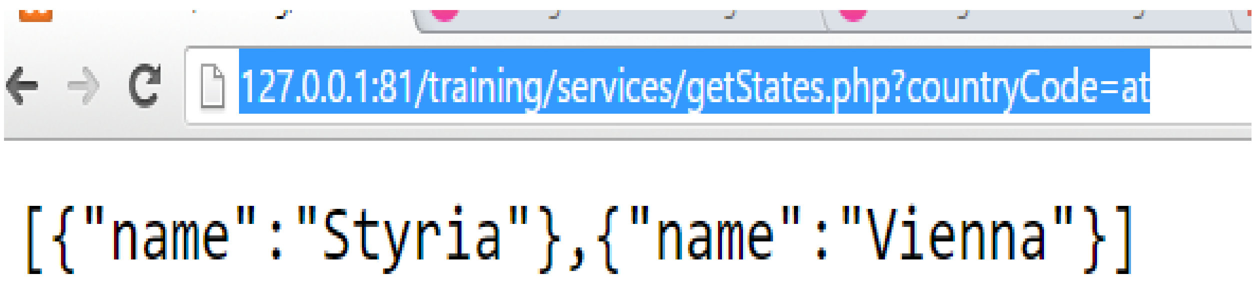
Fig: getStates.php



```php
public static function getStates($countryCode){

        if(count(self::$countries) == 0){
            self::init();
        }
        $country = array_filter(self::$countries,function($c) use ($countryCode){
            return $c->code == $countryCode;
        });

        if (count($country) == 0){
            return array();
        }
        $firstCountry = array_shift($country);
        return $firstCountry->states;
}
```

Fig: getStates() function in CountryRepository.php

As we can see from the getStates.php file, that we are referring country repository file so we are including the functions of getStates with parameter countryCode as we want the results with respect to countryCode.



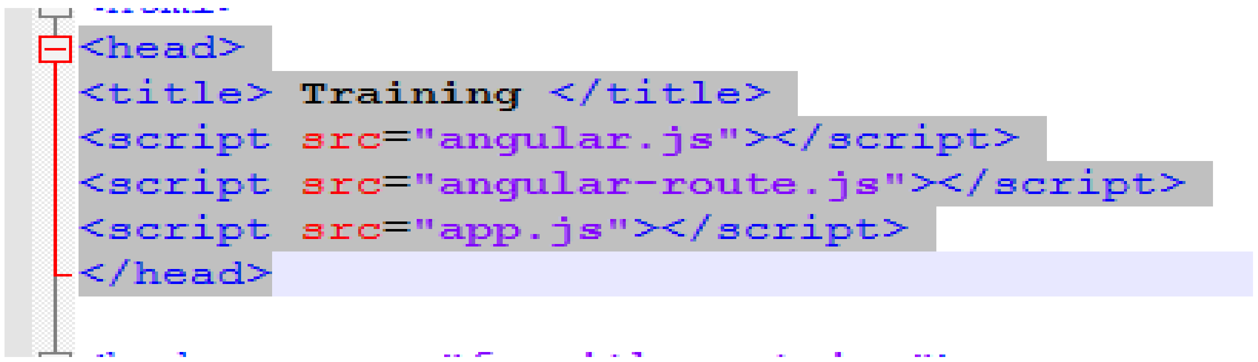
127.0.0.1:81/training/services/getStates.php?countryCode=at

```
[{"name":"Styria"},{"name":"Vienna"}]
```

Fig: Output of getStates.php

7. Routing

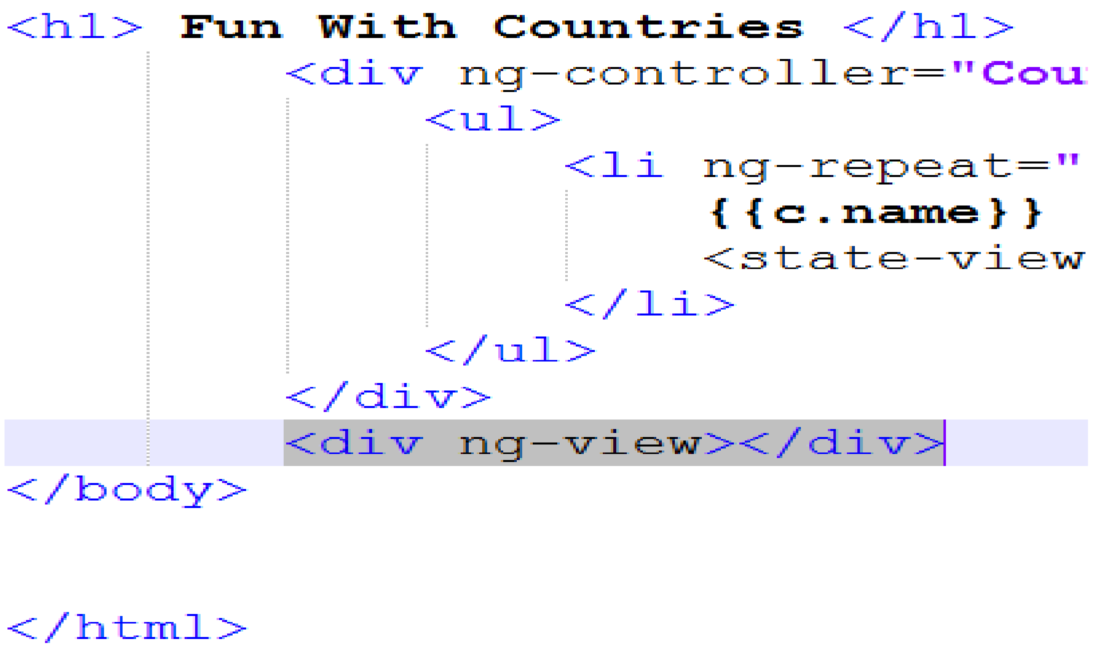
The ngRoute module provides routing and deep linking services and directives for angular applications. You have to download the file (angular-route.js) containing the ngRoute module from AngularJS official website to use the routing feature.



```
<head>
<title> Training </title>
<script src="angular.js"></script>
<script src="angular-route.js"></script>
<script src="app.js"></script>
</head>
```

Fig: General Scenario of Loading Framework

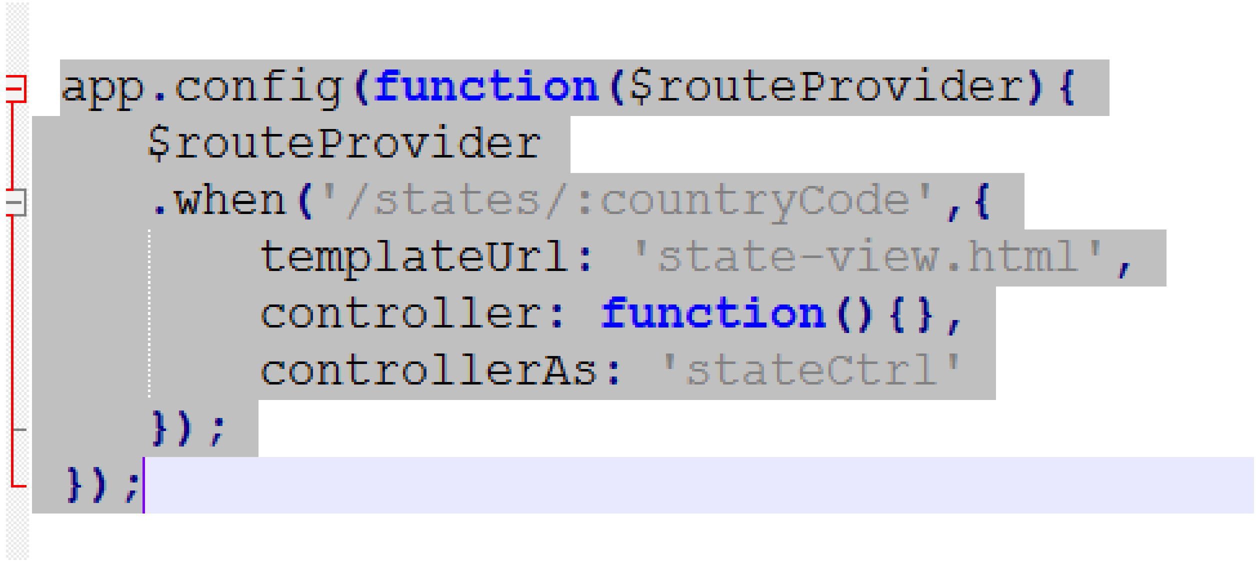
Routing generally works on the basis of angular-route.js. So angular-route.js file should be mandatorily included in between angular.js and app.js.



```
<h1> Fun With Countries </h1>
        <div ng-controller="Cou
            <ul>
                <li ng-repeat="
                    {{c.name}}
                    <state-view
                </li>
            </ul>
        </div>
        <div ng-view></div>
</body>


</html>
```

Fig: Usage of ng-view

The ngView directive is used to display the HTML templates or views in the specified routes. Every time the current route changes, the included view changes with it according to the configuration of the $route service.



```
app.config(function($routeProvider){
    $routeProvider
    .when('/states/:countryCode',{
        templateUrl: 'state-view.html',
        controller: function(){},
        controllerAs: 'stateCtrl'
    });
});
```

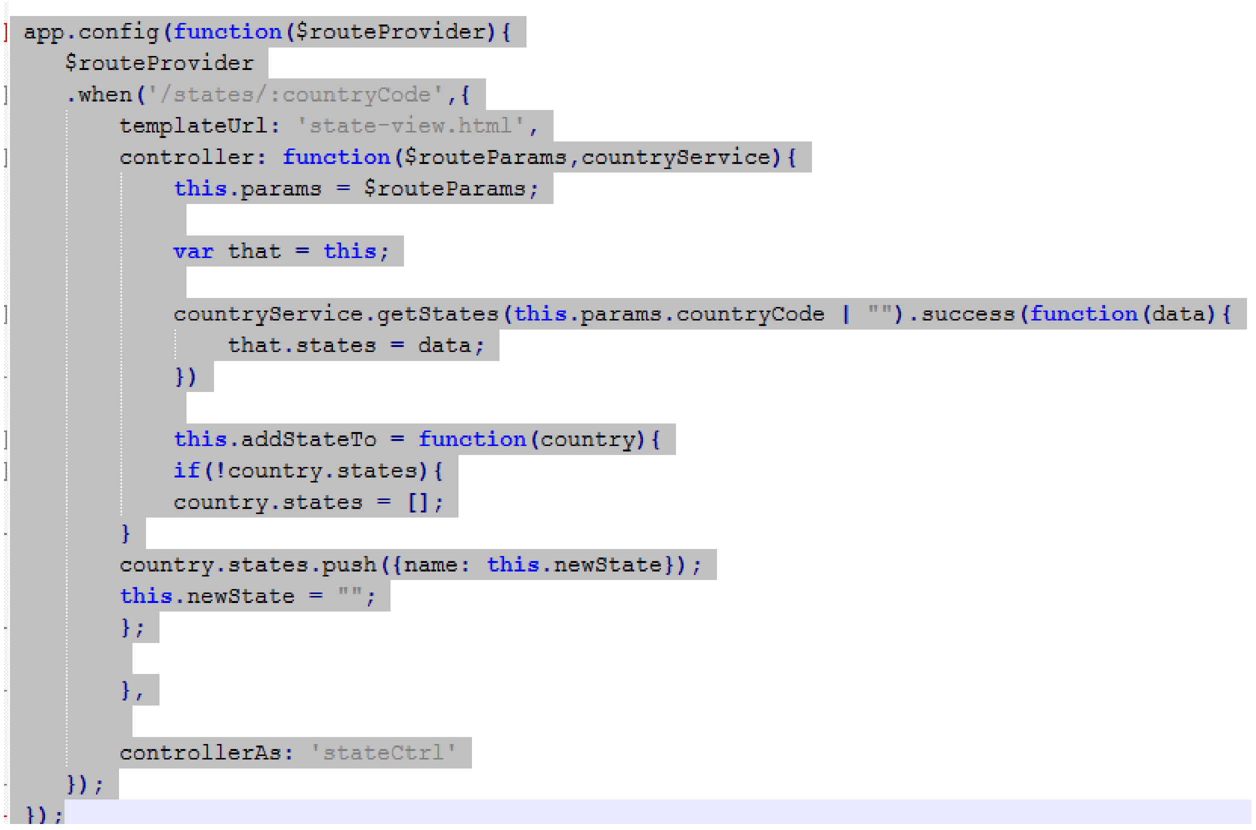
Fig: General Syntax of Routing

$routeProvider : The $routeProvider (provider in ngRoute Module) is used to configure the routes. We use the module's config() to configure the $routeProvider. The config() takes a function which takes the $routeProvider as parameter and the routing configuration goes inside the function.

$routeProvider has a simple API, accepting either the when() or otherwise() method. The when() method takes a path and a route as parameters. The path is a part of the URL after the # symbol. The route contains two properties named templateUrl and controller. The templateUrl property tells which HTML template AngularJS should load and display inside the div with the ngView directive. The controller property tells which controllers should be used with the HTML template.

When the application is loaded, the path is matched against the part of the URL after the # symbol. If no route paths matches the given URL the browser will be redirected to the path specified in the otherwise() function.

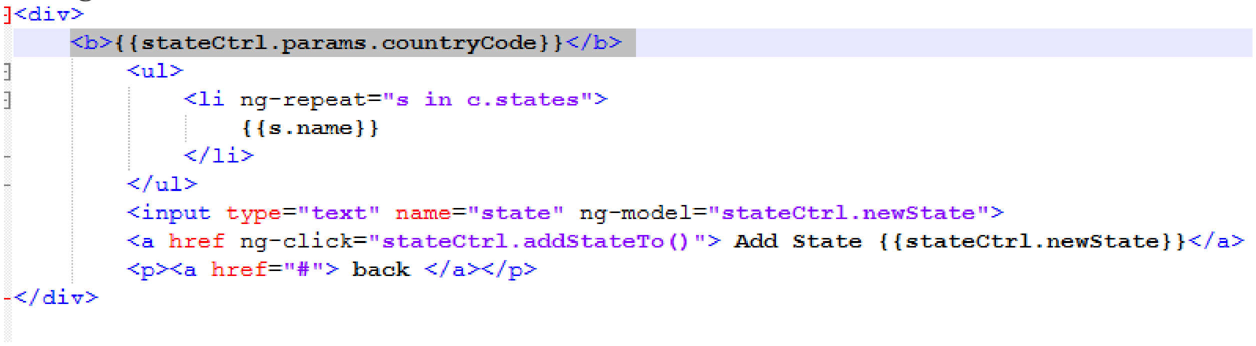
After taking a long consideration on the above sections we have seen the various ways of using AngularJS. So the main part that after routing takes place in our application is as shown below.



```
app.config(function($routeProvider){
    $routeProvider
    .when('/states/:countryCode',{
        templateUrl: 'state-view.html',
        controller: function($routeParams,countryService){
            this.params = $routeParams;

            var that = this;

            countryService.getStates(this.params.countryCode | "").success(function(data){
                that.states = data;
            })

            this.addStateTo = function(country){
            if(!country.states){
            country.states = [];
        }
        country.states.push({name: this.newState});
        this.newState = "";
        };

        },

        controllerAs: 'stateCtrl'
    });
});
```

Fig: Routing in "funwithcountries"

If we look at this diagram, we are using the parameter named "$routeProvider", so our main focus will be to view different pages on the basis of this parameter. Here we are using one important part that is **".when"** which means that when the URL will find the path that we are passing in it then it will migrate the path to the templateUrl thatwe have given to it.

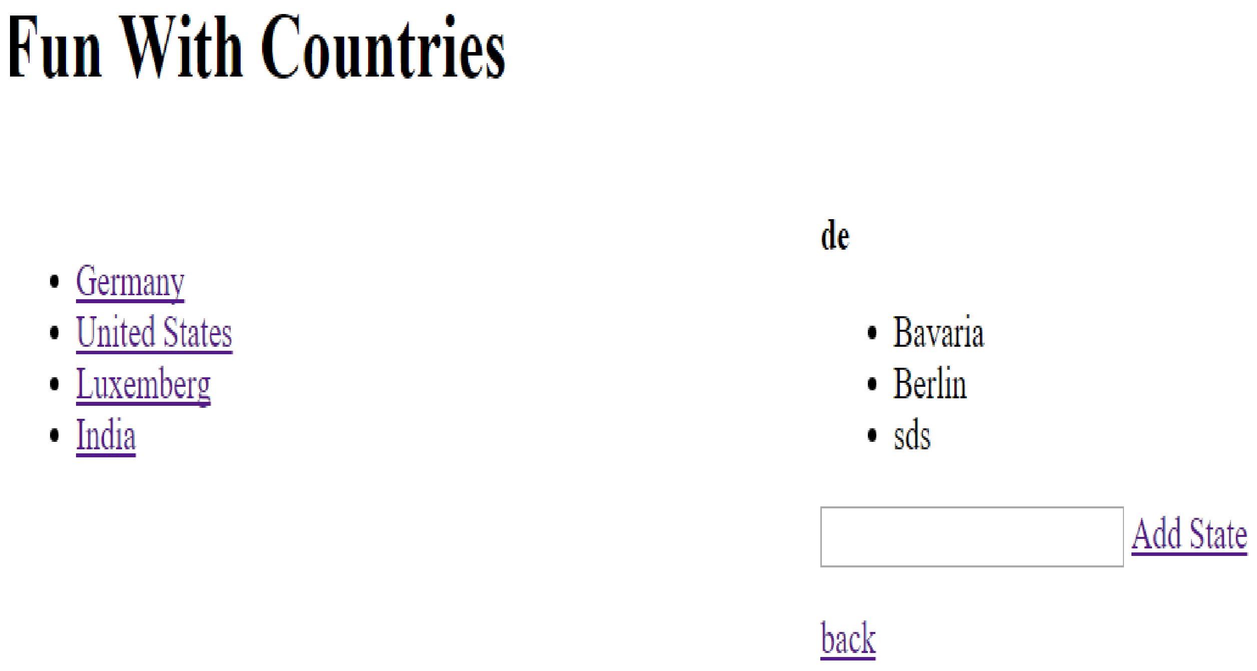


```
<div>
    <b>{{stateCtrl.params.countryCode}}</b>
        <ul>
            <li ng-repeat="s in c.states">
                {{s.name}}
            </li>
        </ul>
        <input type="text" name="state" ng-model="stateCtrl.newState">
        <a href ng-click="stateCtrl.addStateTo()"> Add State {{stateCtrl.newState}}</a>
        <p><a href="#"> back </a></p>
</div>
```

Fig: state-view.html

After moving to the desired path the controller part will come into action and the functionality that have been provided will be performed. In our application this part is getting the name of the states as per the countryCode.

In addition to it we are also adding the states into particular country. So routing part plays a vital role in any of the AngularJS application.



Fun With Countries

- Germany
- United States
- Luxemberg
- India

de

- Bavaria
- Berlin
- sds

Fig: Desired Output