Getting started with AngularJS and ASP.NET MVC - The long awaited Part Three

JMK-NI, 17 Nov 2014 CPOL

4.90 (53 votes)

How to hit the ground running with AngularJS on ASP.NET MVC: Part 3

Articles in series:

- Part One
- Part Two
- Part Three (this article)

Introduction

I wrote part's one and two of this series waay back in August, and have been meaning to write part three ever since but keep getting sidetracked by this and that (*squirrel!*), but here goes.

What we are going to do in part 3

- Build the basis of a RESTful API with ASP.Net Web API, with authentication, roles etc and then create a new Angular app which is wired up to this API

Source Code

The source code accompanying this article can be found here.

The RESTful API

Ok let's get started with our RESTful API. In Visual Studio select **New Project...** => **Web** => **ASP.Net Web Application** (named *Awesome Angular Web App 2.0)* => Web API. Leave Authentication at **Individual User Accounts** and click OK.

Now I would like you to install three applications if you haven't already:

- Postman
- Fiddler
- Node.js

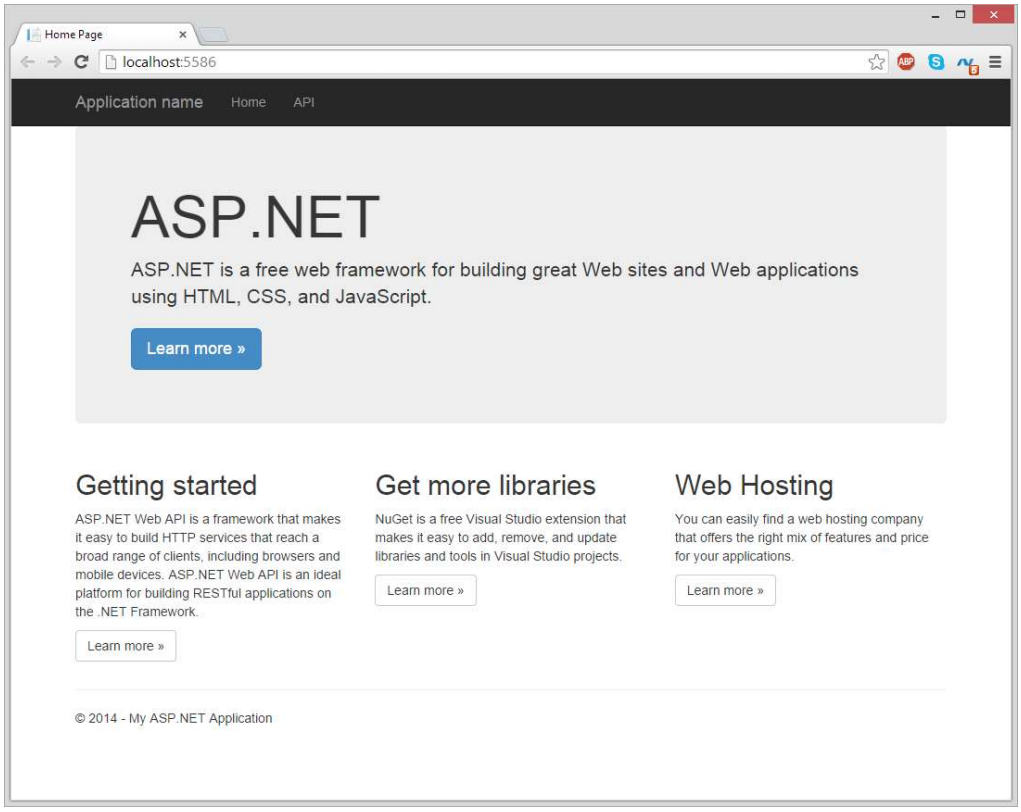
Once you have Postman installed, browse to the following URL to in Chrome:

chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm/index.html

If you are familiar with ASP.Net MVC, Web API will have a relatively small learning curve in some ways as it is based on the same technology and is also MVC based etc, but a larger learning curve in other ways (from the point of view of needing to learn about HTTP

and what RESTful API's are and how to architect them etc. This isn't that article, but I would recommend reading a good book on the topic.)

Press F5 (or whatever your debug key is) to fire up our Web API project. You should see something like this:



Click on API. As we build an API this page is automatically updated with information about all of our endpoints, and the associated HTTP request type. Clicking on a method here will give you details about that particular endpoint, including parameters etc. It is essentially self documenting to a degree.

With the website still running, run Postman. We are going to create a user and login. Web API uses token based authentication as opposed to cookie based. The way this works is quite simple, if you login with a valid username and password, a token will be returned by the API, along with information about the duration this token is valid for. You add this token as a header to all of your following HTTP requests, and the server then knows who you are.

Just like normal ASP.Net authentication, you can restrict access to a controller or method using the **[Authenticate]** attribute. You can also create roles and allow different users access to different parts of the API based on their role.

Let's create a user. Look at URL to determine the port your API is running on (it goes localhost, colon, port number) and then put the following into the URL bar in Postman, replacing 8888 with your port:

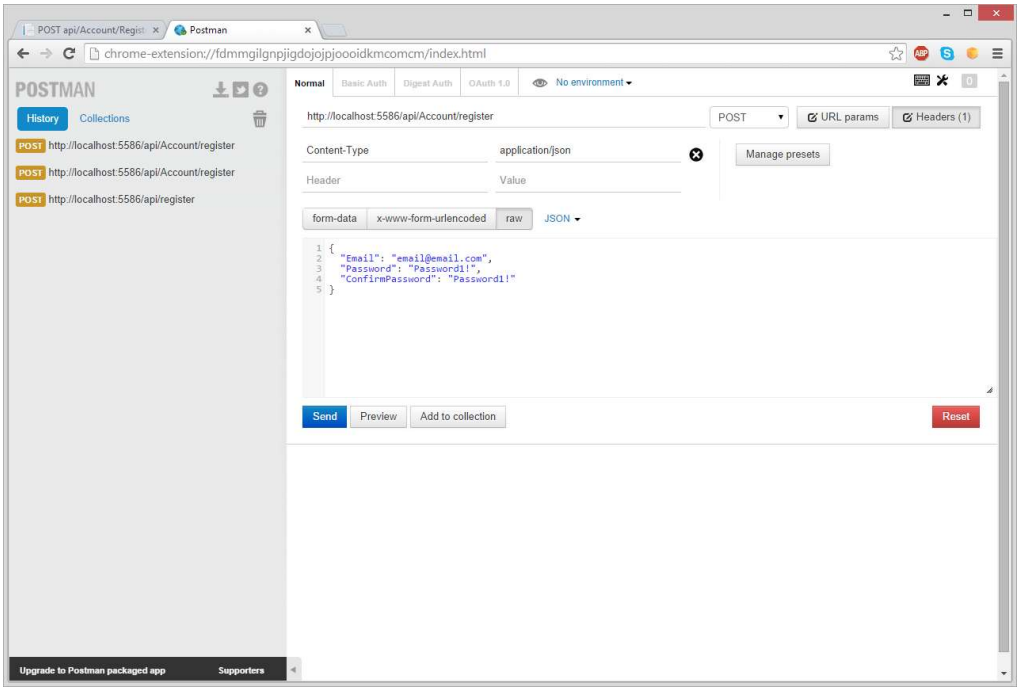
http://localhost:8888/api/Account/register

In the bar below, click on **raw** and paste the following JSON into the textbox underneath (don't worry, the AngularJS stuff is coming, but we need to get through this first):

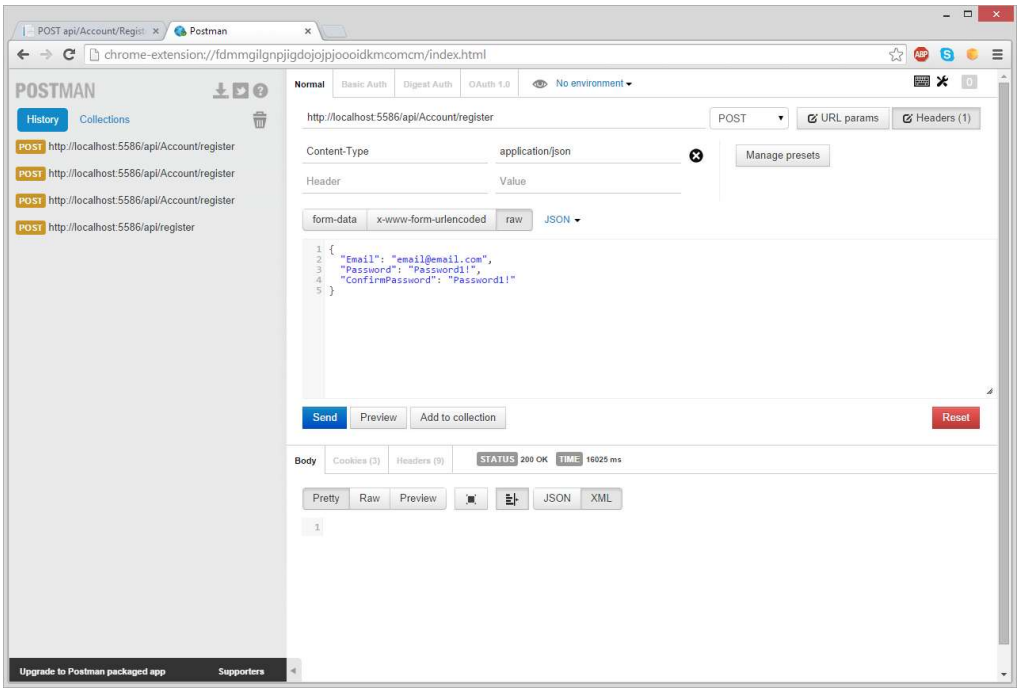
```
{
  "Email": "email@email.com",
  "Password": "Password1!",
  "ConfirmPassword": "Password1!"
}
```

We also need to set the content type of this request to json, so click on **Headers** and add a new header called **Content-Type** which has the value **application/json**. Finally, change the request type to POST in the dropdown list.

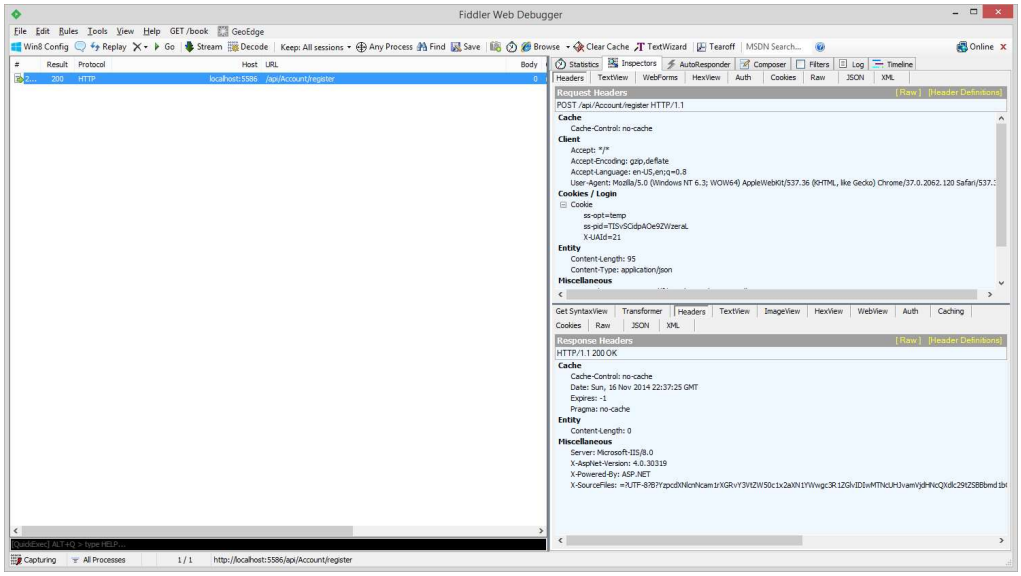
Postman should look something like this:



Hit send. This will take a couple of seconds as ASP.Net is creating the database in the background, but once it has done that, you should get a response of 200 OK, like so:



If you look in fiddler you should be able to find this request also. Check out the different tabs/buttons here to explore the requests details:



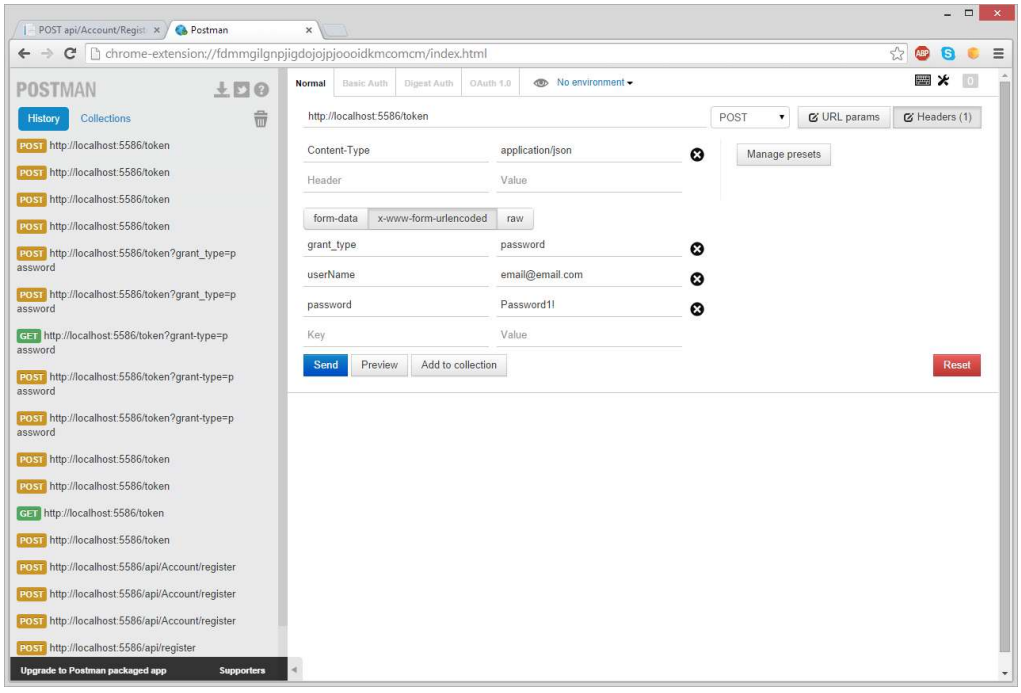
Now that we have created a user, let's go ahead and login. Change the URL to:

http://localhost:8888/token

This is still a post, but now we want to change the data mode to **x-www-form-urlencoded**, so click on this button. Add the following fields:

- **grant_type**: *password*
- **userName**: *email@email.com* (or whatever you registered with)
- **password**: *Password1!* (or whatever password you set when registering)

Postman should look like this now:



Hit send and something similar to the following will be returned:

```
{
    "access_token": "R-AejC88wImTKUulwlZBRsR620zXuHcrjV26UGObVjl5s9aqJIhs2hzt60CdLhL0hXNR-
kyLTgrTfiMDV4JZJsmC1jV3MQHKcScsW6lYAMz1kegSyQiSfRHVj8W1E76x9uiHYJVIWhwA_RH7GkTn3K_Z0ugV_0qsSd1cWZ5qpqRedrS1
vbHNIr7PR-FvAcKGA5c0S7ffadD8TP6N8OX8AyEg2t5rxppAeT2AlqlY3G5HdJqDkPgXQx5pL_xXRWkQCuOhIgUCm-6TDAksNf-
EJ7HzPKD7nl7KU8Pd66rQO56p_vtq6eOO9OtgAmN8FviR-gNKGHCsz4udPrAKTExF_Ht4hBpbLoiGIXIbVUpzTeB-
RMZUMMcRgByo4tCELjd41pV0mjaXHS6s7mTuwlgGmxiAU5AoYgNTXVOe9YegZMvjW_lAIUw0YlZ0m7RAiPOTTDlRzmV1ntm3YGvAN9h9_m0
27twqfGz5YsHsbh3RYW8",
```

```
    "token_type": "bearer",
    "expires_in": 1209599,
    "userName": "email@email.com",
    ".issued": "Sun, 16 Nov 2014 22:55:45 GMT",
    ".expires": "Sun, 30 Nov 2014 22:55:45 GMT"
}
```

RESTful API's and Web API, a very limited primer

When we created our Web API project, it created a controller for us called **ValuesController** containing five methods:

- Get (overloaded with a version which takes a parameter)
- Post
- Put
- Delete

The name of the controller corresponds to the URL (as in ASP.Net MVC), and the names of the methods correspond to the HTTP verb of the request. So if you want to write an endpoint for an HTTP Get request, you create a method called Get for example.

How to architect a RESTful API is outside the scope of this article, but generally you use a:

- Get request to get something
- Post to do something (carry out an action)
- Put to put something somewhere (like in a database)
- Delete to delete

You always respond to an HTTP request with a status code indicating the result .When we registered and logged in we got a 200 response each time. Feel free to play around with those requests in Postman (everything is saved in the history) providing different values, removing/adding values etc to see different status codes returned in different scenarios. For example if you provide an incorrect password when logging in, you get a response of **400 Bad Request**.

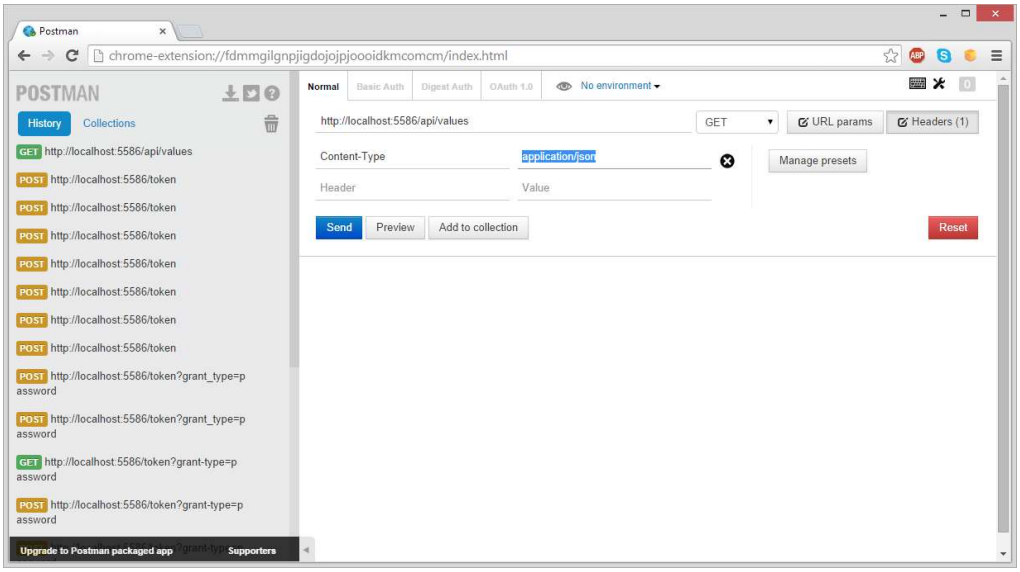
The wikipedia page on this does a good job of breaking down the various status codes. Once you have read this, I would recommend heading over here to read about the 7xx range for a laugh. You will need it after the Wikipedia article.

All HTTP requests can include headers regardless of their HTTP verb (Get, Post etc) and some can also include a body, whereas others cannot. For example a Post can include a body, but a Get can't. In Web API, this body is built up using stronly typed C# objects and returned from the method.

Let's try and call the Get method of this values controller. In Postman, change the HTTP action type to GET in the dropdown and the URL as follows:

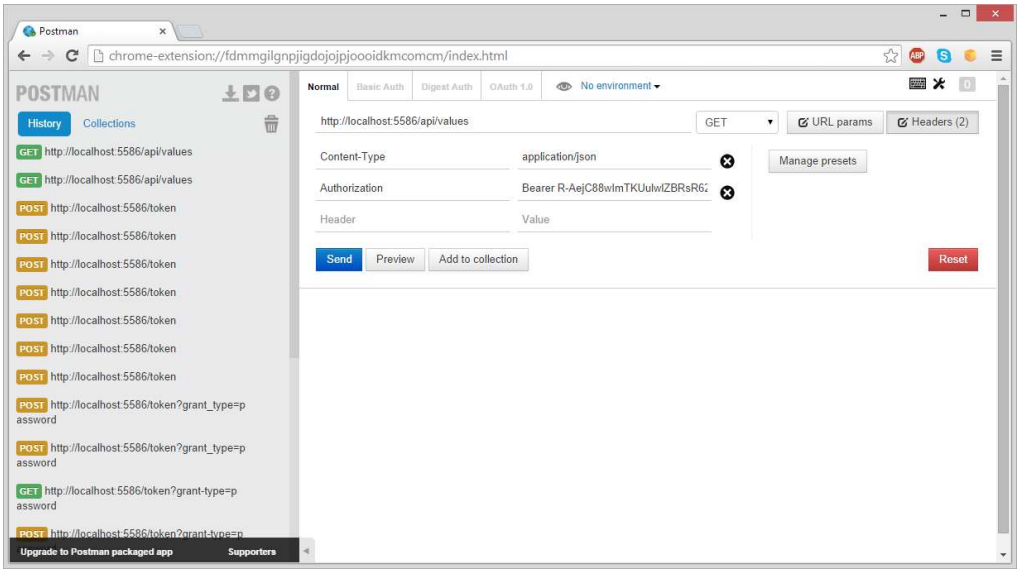
http://localhost:8888/api/values

Leave the Content-Type as **application/json** for now. Postman should look like this:



Hit Send, and we get a 401 Unauthorized response from the API. This is because the Values controller is protected with the **[Authorize]** attribute! Doh!

In Postman, add a new header called **Authorization** with the value of **Bearer**, followed by a space, followed by the token we got back earlier on. Postman should now look like this:



Hit Send again, and success! We get the following JSON back from the API:

```
[
    "value1",
    "value2"
]
```

Play around with Postman calling each of the methods in the values controller, and alter the methods to return different types of objects. You will be able to see all of this action in Postman too, along with detailed information about each request. Pretty cool, eh?

This was supposed to be an article about AngularJS, but we haven't done any AngularJS yet! Get to the AngularJS already!

Alright, alright. This time, we aren't going to create an ASP.Net MVC application to house our website, we will create everything using pure old CSS, HTML and Javascript and a little bit of Node.JS.

I asked you earlier to install Node.JS. Because of a bug in the current version of Node.JS, you will also need to create a directory called **npm** in the following location:

C:\Users\{{your user name}}\AppData\Roaming

This may be fixed by the time you are reading this article, if so, disregard (and maybe leave a comment).

Create a directory for your website, and create an **index.html** file containing the following using your favourite text editor (I'm using Atom):

```
<!DOCTYPE html>
<html>

<head>
<title>Awesome Angular Web App 2.0</title>
</head>

<body>
Hello Again!
</body>
```

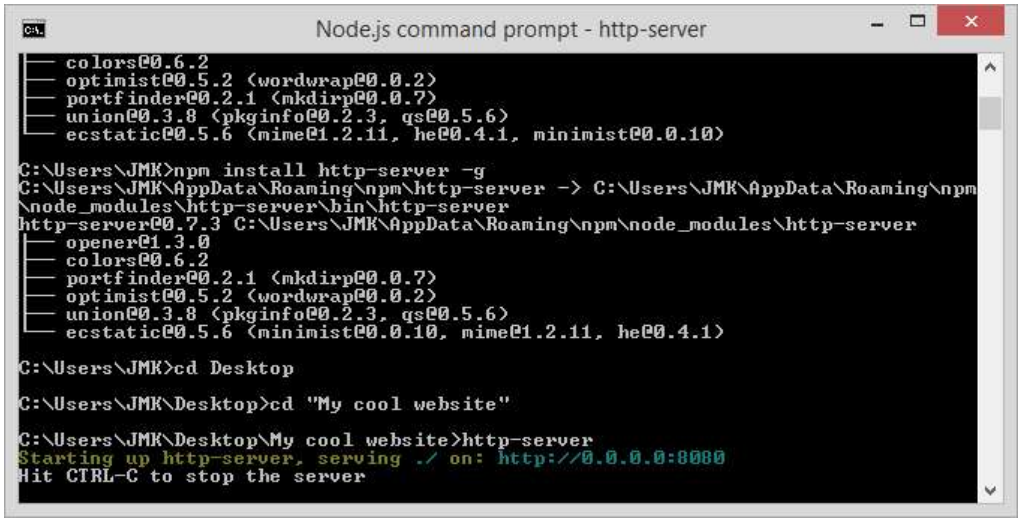
```
</html>
```

Now open **Node.js command prompt**, and run the following command:

npm install http-server -g

This will install a light-weight web server built in Javascript called **http-server** that we can run our Angular app on. The **-g** flag stands for global. This means the server is installed onto your PC and can be run everywhere. If you ommited this, the server would instead be installed into the directory you are currently in.

Now navigate to the directory housing your website, and run http-server. Your window should look like this:



In your browser, browse to **http://localhost:8080/**.

Is that not awesome?

Add a sub-directory called **Scripts**, and one called **Styles**. We have already been over this stuff in parts one and two so I'm not going to explain too much here, just copy and paste the contents of each file I create:

index.html

```
<!DOCTYPE html>
<html>

<head>
<title>Awesome Angular Web App 2.0</title>
</head>
```

```
<body ng-app="AwesomeAngularWebApp" ng-controller="BaseController">
{{helloAgain}}

<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular.min.js"></script>

<script src="/Scripts/Controllers/BaseController.js"></script>
<script src="/Scripts/AwesomeAngularWebApp.js"></script>
</body>

</html>
```

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', []);
AwesomeAngularWebApp.controller('BaseController', BaseController);
```

Scripts/Controllers/BaseController.js

```
var BaseController = function($scope){
    $scope.helloAgain = 'Hello Angular 1.3!';
}
BaseController.$inject = ['$scope'];
```

http://localhost:8080/ should now look like so:

We are going to need somewhere to store the authentication token once it comes back from the API. We could do this in a service. Services in AngularJS are singletons, one instance of them is created when the app bootstraps, and you can pass this instance around via dependency injection. They are a good place to store values needed in multiple parts of your app.

Scripts/Services/SessionService.js

```
var AuthenticationService = function(){
    this.token = undefined;
}
```

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', []);
AwesomeAngularWebApp.controller('BaseController', BaseController);
AwesomeAngularWebApp.service('SessionService', SessionService);
```

Let's create factories to login and register:

Scripts/Factories/LoginFactory.js

```
var LoginFactory = function($http, $q, SessionService){
    return function(username, password){
        var result = $q.defer();

        var params = {grant_type: "password", userName: username, password: password};

        $http({
            method: 'POST',
            url: SessionService.apiUrl + '/token',
            transformRequest: function(obj) {
                var str = [];
                for(var p in obj)
                str.push(encodeURIComponent(p) + "=" + encodeURIComponent(obj[p]));
                return str.join("&");
            },
            data: params,
            headers: {'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8;'}
        })
        .success(function(response){
            result.resolve(response);
        })
        .error(function(response){
            result.reject(response);
        });

        return result.promise;
    }
}

LoginFactory.$inject = ['$http', '$q', 'SessionService'];
```

Scripts/Factories/RegisterFactory.js

```
var RegisterFactory = function($http, $q, SessionService){
    return function(email, password, confirmPassword){
        var result = $q.defer();

        $http({
            method: 'POST',
            url: SessionService.apiUrl + '/api/Account/register',
            data: {Email: email, Password: password, ConfirmPassword: confirmPassword},
            headers: {'Content-Type': 'application/json'}
        })
        .success(function(response){
            result.resolve(response);
        })
        .error(function(response){
            result.reject(response);
        });

        return result.promise;
    }

}

RegisterFactory.$inject = ['$http', '$q', 'SessionService'];
```

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', []);
AwesomeAngularWebApp.controller('BaseController', BaseController);
AwesomeAngularWebApp.service('SessionService', SessionService);
AwesomeAngularWebApp.factory('LoginFactory', LoginFactory);
AwesomeAngularWebApp.factory('RegisterFactory', RegisterFactory);
```

Now I'm going to create a controller for both the login and register views (which we will make in a minute):

Scripts/Controllers/LoginController.js

```
var LoginController = function($scope, $location, LoginFactory, SessionService){
    $scope.loginForm = {
        username: undefined,
        password: undefined,
        errorMessage: undefined
    };

    $scope.login = function(){
        LoginFactory($scope.loginForm.username, $scope.loginForm.password)
        .then(function(response){
            SessionService.token = response.access_token;
            $location.path('/');
        }, function(response){
            $scope.loginForm.errorMessage = response.error_description;
        });
    }
}
LoginController.$inject = ['$scope', '$location', 'LoginFactory', 'SessionService'];
```

Scripts/Controllers/RegisterController.js

```
var RegisterController = function($scope, LoginFactory, RegisterFactory, SessionService){
    $scope.registerForm = {
        username: undefined,
        password: undefined,
        confirmPassword: undefined,
        errorMessage: undefined
    };

    $scope.register = function(){
        RegisterFactory($scope.registerForm.username, $scope.registerForm.password,
$scope.registerForm.confirmPassword)
        .then(function(){
            LoginFactory($scope.registerForm.username, $scope.registerForm.password)
            .then(function(response){
                SessionService.token = response.access_token;
            }, function(response){
                $scope.registerForm.errorMessage = response;
            });
        }, function(response){
            $scope.registerForm.errorMessage = response;
        });
    }
}
RegisterController.$inject = ['$scope', 'LoginFactory', 'RegisterFactory', 'SessionService'];
```

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', []);
AwesomeAngularWebApp.controller('BaseController', BaseController);
AwesomeAngularWebApp.controller('LoginController', LoginController);
AwesomeAngularWebApp.controller('RegisterController', RegisterController);
AwesomeAngularWebApp.service('SessionService', SessionService);
AwesomeAngularWebApp.factory('LoginFactory', LoginFactory);
AwesomeAngularWebApp.factory('RegisterFactory', RegisterFactory);
```

And the views for these controllers, with updated routing code:

Views/Login.html

```
<form role="form" ng-submit="login()">
  <div class="form-group">
```

```
    <label for="emailAddress">Email address</label>
    <input type="email" class="form-control" id="emailAddress" placeholder="Enter email" ng-
model="loginForm.username">
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password" placeholder="Password" ng-
model="loginForm.password">
  </div>
    <p class="help-block" ng-if="loginForm.errorMessage">{{loginForm.errorMessage}}</p>
  </div>
  <button type="submit" class="btn btn-default">Login</button>
</form>
```

Views/Register.html

```
<form role="form" ng-submit="register()">
  <div class="form-group">
    <label for="emailAddress">Email address</label>
    <input type="email" class="form-control" id="emailAddress" placeholder="Enter email" ng-
model="registerForm.username">
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password" placeholder="Password" ng-
model="registerForm.password">
  </div>
  <div class="form-group">
    <label for="password">Confirm Password</label>
    <input type="password" class="form-control" id="password" placeholder="Confirm Password" ng-
model="registerForm.confirmPassword">
  </div>
    <p class="help-block" ng-if="registerForm.errorMessage">{{registerForm.errorMessage}}</p>
  </div>
  <button type="submit" class="btn btn-default">Register</button>
</form>
```

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', ['ngRoute']);
AwesomeAngularWebApp.controller('BaseController', BaseController);
AwesomeAngularWebApp.controller('LoginController', LoginController);
AwesomeAngularWebApp.controller('RegisterController', RegisterController);
AwesomeAngularWebApp.service('SessionService', SessionService);
AwesomeAngularWebApp.factory('LoginFactory', LoginFactory);
AwesomeAngularWebApp.factory('RegisterFactory', RegisterFactory);

var ConfigFunction = function($routeProvider) {
  $routeProvider
   .when('/login', {
    templateUrl: 'views/login.html',
    controller: 'LoginController'
  })
  .when('/register', {
    templateUrl: 'views/register.html',
    controller: 'RegisterController'
  });
};
ConfigFunction.$inject = ['$routeProvider'];
AwesomeAngularWebApp.config(ConfigFunction);
```

index.html

```
<!DOCTYPE html>
<html>

<head>
```

```
<title>Awesome Angular Web App 2.0</title>
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/3.3.1/css/bootstrap.min.css">
</head>

<body ng-app="AwesomeAngularWebApp" ng-controller="BaseController">
{{helloAgain}}

<a href="/#/login" ng-if="!loggedIn()">Login</a>
<a href="/#/register" ng-if="!loggedIn()">Register</a>

<span class="label label-success" ng-if="loggedIn()">Logged In</span>

<div ng-view></div>

<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular-route.min.js"></script>

<script src="/Scripts/Controllers/BaseController.js"></script>
<script src="/Scripts/Controllers/LoginController.js"></script>
<script src="/Scripts/Controllers/RegisterController.js"></script>
<script src="/Scripts/Services/SessionService.js"></script>
<script src="/Scripts/Factories/LoginFactory.js"></script>
<script src="/Scripts/Factories/RegisterFactory.js"></script>
<script src="/Scripts/AwesomeAngularWebApp.js"></script>
</body>

</html>
```

Scripts/BaseController.js

```
var BaseController = function($scope, SessionService){
    $scope.helloAgain = 'Hello Angular 1.3!';

    $scope.loggedIn = function(){
        return SessionService.token !== undefined;
    }
}
BaseController.$inject = ['$scope', 'SessionService'];
```

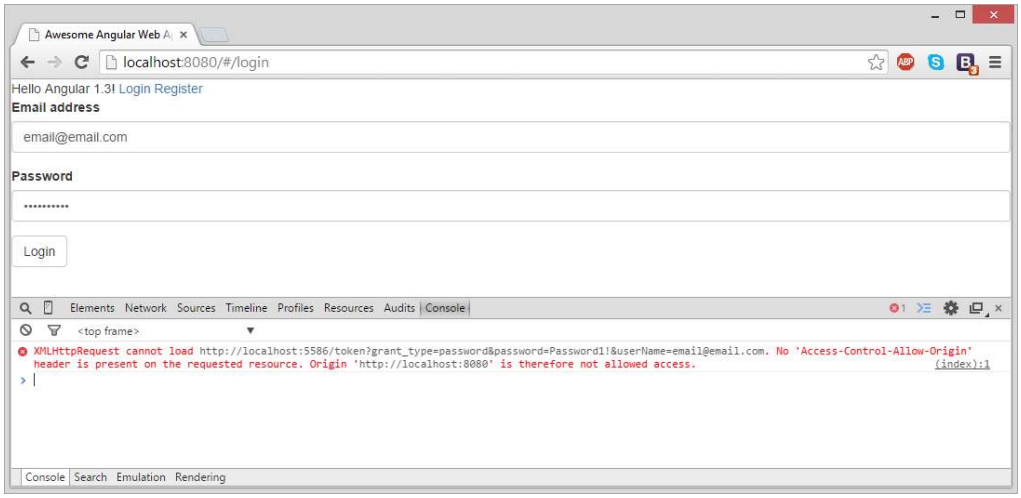
Ok let's navigate back to **http://localhost:8080/** and make sure everything is wired up ok. You should be able to browse to both the Login and Register views. Check your browsers console, if there are errors you have probably missed a step (or I've messed up somehow, if so let me know in the comments)

Let's try and login

To save you from having to scroll up, here is the username and password of the user we created earlier (unless you changed them, if so scrolling up won't help):

- **userName**: *email@email.com*
- **password**: *Password1!*

Let's browse to the login form, and try and login. Oh noes:



Our API is on one server, and our website on another one so the login request is being blocked by the browser for security reasons. We can fix this with CORS.

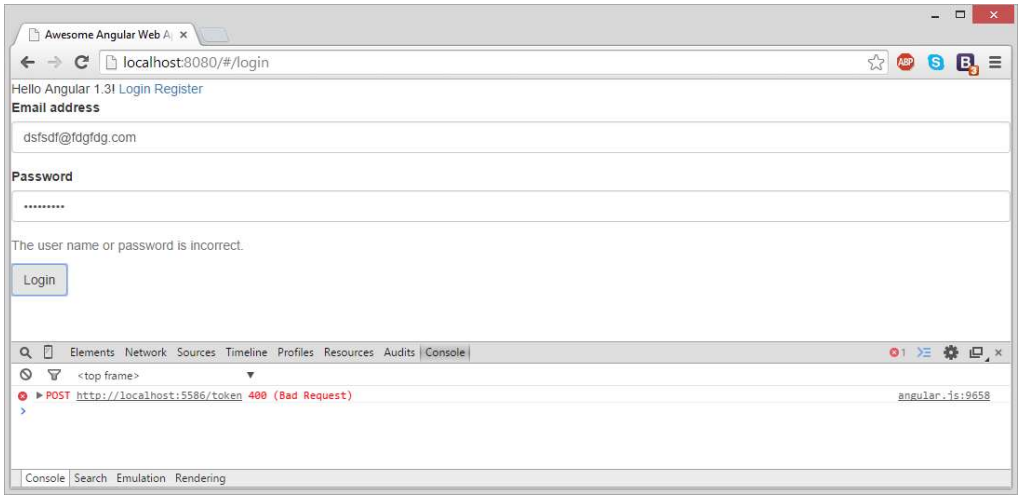
Go back into Visual Studio and open the nuget package manager console from **View** => **Other Windows** => **Package Manager Console**. Run the following command:

```
Install-Package Microsoft.Owin.Cors
```

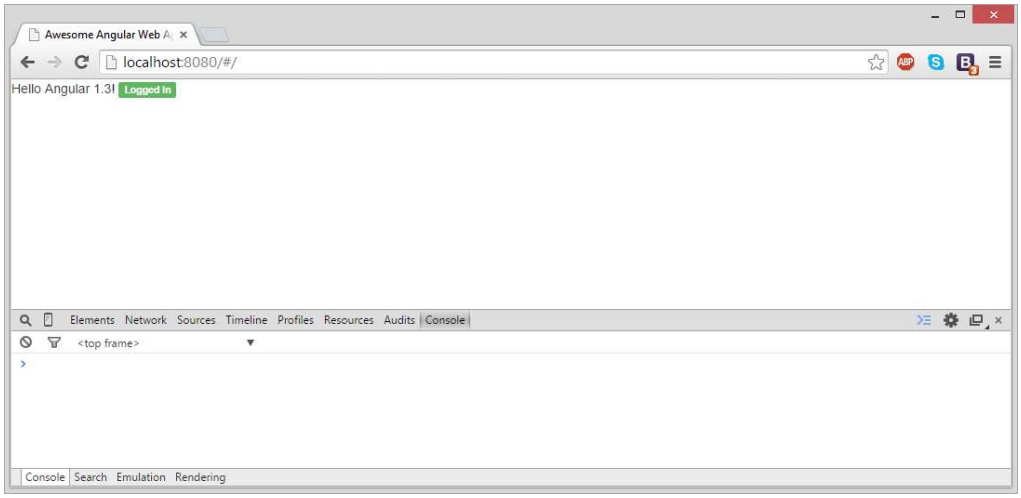
In **Startup.cs**, amend **Configuration** so it contains the following:

```
        public void Configuration(IAppBuilder app)
        {
            app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
            ConfigureAuth(app);
        }
```

Let's try and login again, first with an intentionally incorrect username/password combo:



Cool, we get a 400 back from the server, and display the error message on the screen. Now let's login properly:



Epic! We are back at the homepage, our token is stored in the service and the login and register links are gone.

Now we are going to write a factory to **Get** the **Values** from the ValuesController class in our API:

Scripts/Factories/GetValuesFactory.js

```
var GetValuesFactory = function($http, $q, SessionService){
    return function(){
        var result = $q.defer();

        $http({
            method: 'GET',
            url: SessionService.apiUrl + '/api/Values',
            headers: {'Content-Type': 'application/json', 'Authorization': 'Bearer ' +
SessionService.token}
        })
        .success(function(response){
            result.resolve(response);
        })
        .error(function(response){
            result.reject(response);
        });

        return result.promise;
    }
}

GetValuesFactory.$inject = ['$http', '$q', 'SessionService'];
```

A controller for the view we are going to create in a second:

Scripts/Controllers/ValuesController.js

```
var ValuesController = function($scope, GetValuesFactory, SessionService){
    $scope.values = [];
    $scope.error = {
        message: undefined
    };

    $scope.getValues = function(){
        GetValuesFactory()
        .then(function(response){
            $scope.values = response;
        }, function(response){
            $scope.error.message = response.Message;
        });
    }
}
ValuesController.$inject = ['$scope', 'GetValuesFactory', 'SessionService'];
```

A view:

Views/values.html

```
<button ng-click="getValues()">Get Values</button>

<ul class="nav nav-pills" ng-if="values.length > 0">
    <li ng-repeat="value in values">{{value}}<li>
</ul>

<p class="help-block" ng-if="error.message">{{error.message}}</p>
```

An updated route and registration of the new jazz:

Scripts/AwesomeAngularWebApp.js

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', ['ngRoute']);
AwesomeAngularWebApp.controller('BaseController', BaseController);
AwesomeAngularWebApp.controller('LoginController', LoginController);
AwesomeAngularWebApp.controller('RegisterController', RegisterController);
AwesomeAngularWebApp.controller('ValuesController', ValuesController);
AwesomeAngularWebApp.service('SessionService', SessionService);
AwesomeAngularWebApp.factory('LoginFactory', LoginFactory);
AwesomeAngularWebApp.factory('RegisterFactory', RegisterFactory);
AwesomeAngularWebApp.factory('GetValuesFactory', GetValuesFactory);

var ConfigFunction = function($routeProvider) {
  $routeProvider
   .when('/login', {
    templateUrl: 'views/login.html',
    controller: 'LoginController'
  })
  .when('/register', {
    templateUrl: 'views/register.html',
    controller: 'RegisterController'
  })
  .when('/values', {
    templateUrl: 'views/values.html',
    controller: 'ValuesController'
  });
};
ConfigFunction.$inject = ['$routeProvider'];
AwesomeAngularWebApp.config(ConfigFunction);
```

A link, and script tags for the new controller/factory:

```
<!DOCTYPE html>
<html>

<head>
<title>Awesome Angular Web App 2.0</title>
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/3.3.1/css/bootstrap.min.css">
</head>

<body ng-app="AwesomeAngularWebApp" ng-controller="BaseController">
{{helloAgain}}

<a href="/#/login" ng-if="!loggedIn()">Login</a>
<a href="/#/register" ng-if="!loggedIn()">Register</a>
<a href="/#/values">Values</a>

<span class="label label-success" ng-if="loggedIn()">Logged In</span>

<div ng-view></div>

<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular.min.js"></script>
```

```
<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular-route.min.js"></script>

<script src="/Scripts/Controllers/BaseController.js"></script>
<script src="/Scripts/Controllers/LoginController.js"></script>
<script src="/Scripts/Controllers/RegisterController.js"></script>
<script src="/Scripts/Controllers/ValuesController.js"></script>
<script src="/Scripts/Services/SessionService.js"></script>
<script src="/Scripts/Factories/LoginFactory.js"></script>
<script src="/Scripts/Factories/RegisterFactory.js"></script>
<script src="/Scripts/Factories/GetValuesFactory.js"></script>
<script src="/Scripts/AwesomeAngularWebApp.js"></script>
</body>

</html>
```

Ok let's fire everything up. First of all, navigate to our new values view and hit the button. Web API returns us a 401 (unauthorized) http error which is displayed on the screen. Login and hit the button again and everything works as it should.

Don't forget me

At the moment, if you login, and then refresh the page, you are kicked back out and need to re-authenticate. We can fix that by storing our token in a cookie, using AngularJS' **ng-cookies** module. Update index.html to pull this down from Cloudflare like so (add it after the **angular-route** tag):

```
<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.0/angular-cookies.min.js"></script>
```

Update AwesomeAngularWebApp.js to register this module with our app:

```
var AwesomeAngularWebApp = angular.module('AwesomeAngularWebApp', ['ngRoute', 'ngCookies']);
```

Update SessionService to take the token and store it in a cookie:

```
var SessionService = function($cookies){
    this.token = undefined;

    this.getToken = function(){
        if(!$cookies.awesomeAngularWebAppToken){
            if(!this.token){
                return undefined;
            }
            this.setToken(this.token);
        }
        return $cookies.awesomeAngularWebAppToken;
    }

    this.setToken = function(token){
        this.token = token;
        $cookies.awesomeAngularWebAppToken = token;
    }

    this.apiUrl = 'http://localhost:5586';
}

SessionService.$inject = ['$cookies'];
```

LoginController and RegisterController, BaseController and GetValuesFactory to call the new getter and setter methods:

Scripts/Controllers/LoginController.js

```
var LoginController = function($scope, $location, LoginFactory, SessionService){
    $scope.loginForm = {
        username: undefined,
        password: undefined,
        errorMessage: undefined
```

```
    };

    $scope.login = function(){
        LoginFactory($scope.loginForm.username, $scope.loginForm.password)
        .then(function(response){
            SessionService.setToken(response.access_token);
            $location.path('/');
        }, function(response){
            $scope.loginForm.errorMessage = response.error_description;
        });
    }
}
LoginController.$inject = ['$scope', '$location', 'LoginFactory', 'SessionService'];
```

Scripts/Controllers/RegisterController.js

```
var RegisterController = function($scope, LoginFactory, RegisterFactory, SessionService){
    $scope.registerForm = {
        username: undefined,
        password: undefined,
        confirmPassword: undefined,
        errorMessage: undefined
    };

    $scope.register = function(){
        RegisterFactory($scope.registerForm.username, $scope.registerForm.password,
$scope.registerForm.confirmPassword)
        .then(function(){
            LoginFactory($scope.registerForm.username, $scope.registerForm.password)
            .then(function(response){
                SessionService.setToken(response.access_token);
            }, function(response){
                $scope.loginForm.errorMessage = response;
            });
        }, function(response){
            $scope.registerForm.errorMessage = response;
        });
    }
}
RegisterController.$inject = ['$scope', 'LoginFactory', 'RegisterFactory', 'SessionService'];
```

Scripts/Controllers/BaseController.js

```
var BaseController = function($scope, SessionService){
    $scope.helloAgain = 'Hello Angular 1.3!';

    $scope.loggedIn = function(){
        return SessionService.getToken() !== undefined;
    }
}
BaseController.$inject = ['$scope', 'SessionService'];
```

Scripts/Factories/GetValuesFactory.js

```
var GetValuesFactory = function($http, $q, SessionService){
    return function(){
        var result = $q.defer();

        $http({
            method: 'GET',
            url: SessionService.apiUrl + '/api/Values',
            headers: {'Content-Type': 'application/json', 'Authorization': 'Bearer ' +
SessionService.getToken()}
        })
        .success(function(response){
            result.resolve(response);
```

```
        })
        .error(function(response){
            result.reject(response);
        });

        return result.promise;
    }
}

GetValuesFactory.$inject = ['$http', '$q', 'SessionService'];
```

Now when you login and hit refresh, you are still logged in! Wohoo!

Roles

As I mentioned earlier, different users can have different roles. I find it easiest to setup roles just using SQL. Let's do this now.

Open SQL Server Management Studio and type the **Server name** in as **(localdb)\v11.0** like so and hit connect:

You should see a database named something like **aspnet-Awesome Angular Web App 2.0-20141116095710**, the numbers on the end are just the timestamp of when this database was created.

Let's create a role called SuperAdmin and assign the user we created this role. Update the AspNetRoles table using the following SQL:

```
INSERT INTO [aspnet-Awesome Angular Web App 2.0-20141116095710].[dbo].[AspNetRoles] VALUES('8138aad6-dfdb-422d-862d-8d5acab5c8a1', 'SuperAdmin')
```

Now let's update our ValuesController to only allow Super admins to **Get** values. Amend the [Authorize] attribute on the controller like so:

```
[Authorize(Roles = "SuperAdmin")]
```

Important: When we stop and start the API, none of our tokens will work anymore as these are stored in memory so you will need to clear your browsers cache at the same time.

Now when we go to our site and login (with the following credentials for the lazy):

- **userName**: *email@email.com*
- **password**: *Password1!*

We get a 401 unauthorized back. This is because we are not a member of the SuperAdmin group. Let's fix that, first run the following SQL against the [AspNetUsers] to see what our id is:

```
SELECT [Id] FROM [aspnet-Awesome Angular Web App 2.0-20141116095710].[dbo].[AspNetUsers] WHERE [Email] = 'email@email.com'
```

For me the id was **e79c2336-265e-42d0-a62c-df44a3dfa0f4**, yours will be different so replace this id in the following SQL with yours. The following SQL adds the user to the role:

```
INSERT INTO [aspnet-Awesome Angular Web App 2.0-20141116095710].[dbo].[AspNetUserRoles] VALUES('e79c2336-265e-42d0-a62c-df44a3dfa0f4', '8138aad6-dfdb-422d-862d-8d5acab5c8a1')
```

You are now a super admin, but you weren't when you logged in so clear your browser cache, login again and try now. If all is right with the universe you will be able to successfully complete HTTP requests against the Values controller, but other users not in the SuperAdmin role will not.

Recap

Today we created a RESTful API with ASP.Net Web API, an AngularJS application running on a node.js server, and successfully configured the application to authenticate against the API, storing the token in a in a cookie. We also created a role and restricted an area of the API to only users in this role.

Coming in Part Four

What would you like to see? Leave a comment.

Security

Troy Hunt from Microsoft used the code presented in this article as the basis of a client side framework security course on Pluralsight available here. I highly recommend checking it out.

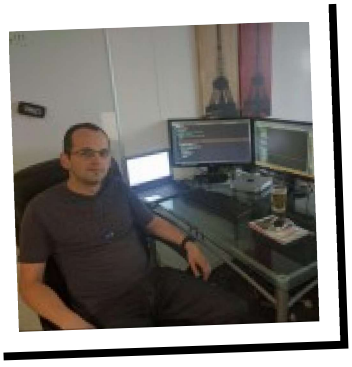
Comments/Criticism etc

Feel free to comment on the article with any comments/criticisms/questions etc and I will always reply. Thanks for reading Smile | :)

License

Share

About the Author



JMK-NI

Software Developer

United Kingdom

26 years old, from Belfast, lunatic, love what I do

Follow me on Twitter @chancymajormusk

You may also be interested in...

- Getting started with AngularJS and ASP.NET MVC - Part One
- IoT JumpWay Intel® Edison Basic LED Example
- Getting started with AngularJS and ASP.NET MVC - Part Two
- Visual COBOL New Release: Small point. Big deal
- Flashing the Zephyr Application Using a JTAG Adapter on the Arduino 101 (branded Genuino 101 outside the U.S.) on Ubuntu under VMware
- SAPrefs - Netscape-like Preferences Dialog

Comments and Discussions

40 messages have been posted for this article Visit **https://www.codeproject.com/Articles/843044/Getting-started-with-AngularJS-and-ASP-NET-MVC-The** to post and view comments on this article, or click **here** to get a print view with messages.