



ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API - Part 4

March 11, 2015 By [Taiseer Joudeh](#) – 111 Comments

Be Sociable, Share!



This is the forth part of Building Simple Membership system using ASP.NET Identity 2.1, ASP.NET Web API 2.2 and AngularJS. The topics we'll cover are:

- [Configure ASP.NET Identity with ASP.NET Web API \(Accounts Management\) – Part 1.](#)
- [ASP.NET Identity 2.1 Accounts Confirmation, and Password/User Policy Configuration – Part 2.](#)
- [Implement JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3.](#)
- [ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – \(This Post\)](#)
- [ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5.](#)
- [AngularJS Authentication and Authorization with ASP.NET Web API and Identity 2.1 – Part 6](#)

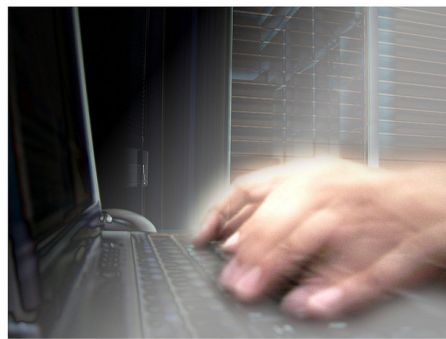
The [source code](#) for this tutorial is available on GitHub.

ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API

In the [previous post](#) we saw how we can authenticate individual users using the `[Authorize]` attribute in a very basic form, but there is some limitation with the previous approach where any authenticated user can perform sensitive actions such as deleting any user in the system, getting list of all users in the system, etc... where those actions should be executed only by subset of users with higher privileges (Admins only).

In this post we'll see how we can enhance the authorization mechanism to give finer grained control over how users can execute actions based on role membership, and how those roles will help us in differentiate between authenticated users.

The nice thing here that ASP.NET Identity 2.1 provides support for managing Roles (create, delete, update, assign users to a role, remove users from role, etc...) by using the `RoleManager<T>` class, so let's get started by adding support for roles management in our Web API.



Step 1: Add the Role Manager Class

The Role Manager class will be responsible to manage instances of the Roles class, the class will derive from "RoleManager<T>" where T will represent our "IdentityRole" class, once it derives from the "IdentityRole" class a set of methods will be available, those methods will facilitate managing roles in our Identity system, some of the exposed methods we'll use from the "RoleManager" during this tutorial are:

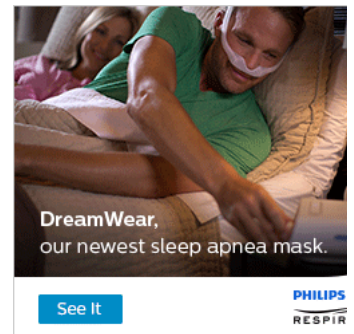
METHOD NAME	USAGE
<code>FindByIdAsync(id)</code>	Find role object based on its unique identifier

ABOUT TAISEER

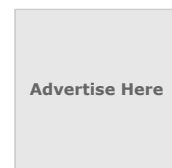


Father, MVP (ASP .NET/IIS), Scrum Master, Life Time Learner

CONNECT WITH ME



Advertise Here



Advertise Here

Advertise Here

Roles	Returns an enumeration of the roles in the system
FindByNameAsync(Rolename)	Find role based on its name
CreateAsync(IdentityRole)	Creates a new role
DeleteAsync(IdentityRole)	Delete role
RoleExistsAsync(RoleName)	Returns true if role already exists

Now to implement the “RoleManager” class, add new file named “ApplicationRoleManager” under folder “Infrastructure” and paste the code below:

```
1 public class ApplicationRoleManager : RoleManager<IdentityRole>
2 {
3     public ApplicationRoleManager(IRoleStore<IdentityRole> roleStore)
4         : base(roleStore)
5     {
6     }
7
8     public static ApplicationRoleManager Create(IdentityFactoryOptions<ApplicationRoleManager> options, IOwinContext context)
9     {
10         var appRoleManager = new ApplicationRoleManager(new RoleStore<IdentityRole>(context.Get<ApplicationDbContext>().RoleStore));
11         return appRoleManager;
12     }
13 }
14 }
```

Notice how the “Create” method will use the Owin middleware to create instances for each request where Identity data is accessed, this will help us to hide the details of how role data is stored throughout the application.

Step 2: Assign the Role Manager Class to Owin Context

Now we want to add a single instance of the Role Manager class to each request using the Owin context, to do so open file “Startup” and paste the code below inside method “ConfigureOAuthTokenGeneration”:

```
1 private void ConfigureOAuthTokenGeneration(IAppBuilder app)
2 {
3     // Configure the db context and user manager to use a single instance per request
4     // Rest of code is removed for brevity
5
6     app.CreatePerOwinContext<ApplicationRoleManager>(ApplicationRoleManager.Create);
7
8     // Rest of code is removed for brevity
9
10 }
```

Now a single instance of class “ApplicationRoleManager” will be available for each request, we’ll use this instance in different controllers, so it is better to create a helper property in class “BaseApiController” which all other controllers inherits from, so open file “BaseApiController” and add the following code:

```
1 public class BaseApiController : ApiController
2 {
3     // Code removed for brevity
4     private ApplicationRoleManager _AppRoleManager = null;
5
6     protected ApplicationRoleManager AppRoleManager
7     {
8         get
9         {
10             return _AppRoleManager ?? Request.GetOwinContext().GetUserManager<ApplicationRoleManager>();
11         }
12     }
13 }
```

Step 3: Add Roles Controller

Now we’ll add the controller which will be responsible to manage roles in the system (add new roles, delete existing ones, getting single role by id, etc...), but this controller should only be accessed by users in “Admin” role because it doesn’t make sense to allow any authenticated user to delete or create roles in the system, so



RECENT POSTS

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)

[Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)

[ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2](#)

[Interview with John about establishing a successful blog](#)

BLOG ARCHIVES

Blog Archives

Select Month

LEAVE YOUR EMAIL AND KEEP TUNED!

Sign up to receive email updates on every new post!

SUBSCRIBE

we will see how we will use the [Authorize] attribute along with the Roles to control this.

Now add new file named "RolesController" under folder "Controllers" and paste the code below:

```
1  [Authorize(Roles="Admin")]
2  [RoutePrefix("api/roles")]
3  public class RolesController : BaseApiController
4  {
5
6      [Route("{id:guid}", Name = "GetRoleById")]
7      public async Task<IHttpActionResult> GetRole(string Id)
8      {
9          var role = await this.AppRoleManager.FindByIdAsync(Id);
10
11         if (role != null)
12         {
13             return Ok(TheModelFactory.Create(role));
14         }
15
16         return NotFound();
17     }
18
19     [Route("", Name = "GetAllRoles")]
20     public IHttpActionResult GetAllRoles()
21     {
22         var roles = this.AppRoleManager.Roles;
23
24         return Ok(roles);
25     }
26
27     [Route("create")]
28     public async Task<IHttpActionResult> Create(CreateRoleBindingModel model)
29     {
30         if (!ModelState.IsValid)
31         {
32             return BadRequest(ModelState);
33         }
34
35         var role = new IdentityRole { Name = model.Name };
36
37         var result = await this.AppRoleManager.CreateAsync(role);
38
39         if (!result.Succeeded)
40         {
41             return GetErrorResult(result);
42         }
43
44         Uri locationHeader = new Uri(Url.Link("GetRoleById", new { id = role.Id }));
45
46         return Created(locationHeader, TheModelFactory.Create(role));
47     }
48
49     [Route("{id:guid}")]
50     public async Task<IHttpActionResult> DeleteRole(string Id)
51     {
52         var role = await this.AppRoleManager.FindByIdAsync(Id);
53
54         if (role != null)
55         {
56             IdentityResult result = await this.AppRoleManager.DeleteAsync(role);
57
58             if (!result.Succeeded)
59             {
60                 return GetErrorResult(result);
61             }
62
63             return Ok();
64         }
65
66         return NotFound();
67     }
68
69     [Route("ManageUsersInRole")]
70     public async Task<IHttpActionResult> ManageUsersInRole(UsersInRoleModel model)
71     {
72         var role = await this.AppRoleManager.FindByIdAsync(model.Id);
73
74         if (role == null)
75         {
76             ModelState.AddModelError("", "Role does not exist");
77             return BadRequest(ModelState);
78         }
79
80         foreach (string user in model.EnrolledUsers)
81         {
82             var appUser = await this.AppUserManager.FindByIdAsync(user);
83
84             if (appUser == null)
85             {
86
87             }
88         }
89     }
```

```

90         ModelState.AddModelError("", String.Format("User: {0} does not exists", user));
91         continue;
92     }
93
94     if (!this.AppUserManager.IsInRole(user, role.Name))
95     {
96         IdentityResult result = await this.AppUserManager.AddToRoleAsync(user, role.Name);
97
98         if (!result.Succeeded)
99         {
100             ModelState.AddModelError("", String.Format("User: {0} could not be added to role", user));
101         }
102     }
103
104 }
105
106 foreach (string user in model.RemovedUsers)
107 {
108     var appUser = await this.AppUserManager.FindByIdAsync(user);
109
110     if (appUser == null)
111     {
112         ModelState.AddModelError("", String.Format("User: {0} does not exists", user));
113         continue;
114     }
115
116     IdentityResult result = await this.AppUserManager.RemoveFromRoleAsync(user, role.Name);
117
118     if (!result.Succeeded)
119     {
120         ModelState.AddModelError("", String.Format("User: {0} could not be removed from role", user));
121     }
122 }
123
124 if (!ModelState.IsValid)
125 {
126     return BadRequest(ModelState);
127 }
128
129 return Ok();
130 }
131 }

```

What we have implemented in this lengthy controller code is the following:

- We have attribute the controller with **[Authorize(Roles="Admin")]** which allows only authenticated users who belong to "Admin" role only to execute actions in this controller, the "Roles" property accepts comma separated values so you can add multiple roles if needed. In other words the user who will have an access to this controller should have valid JSON Web Token which contains claim of type "Role" and value of "Admin".
- The method "GetRole(Id)" will return a single role based on it is identifier, this will happen when we call the method "FindByIdAsync", this method returns object of type "RoleReturnModel" which we'll create in the next step.
- The method "GetAllRoles()" returns all the roles defined in the system.
- The method "Create(CreateRoleBindingModel model)" will be responsible of creating new roles in the system, it will accept model of type "CreateRoleBindingModel" where we'll create it the next step. This method will call "CreateAsync" and will return response of type "RoleReturnModel".
- The method "DeleteRole(string Id)" will delete existing role by passing the unique id of the role then calling the method "DeleteAsync".
- Lastly the method "ManageUsersInRole" is proprietary for the AngularJS app which we'll build in the coming posts, this method will accept a request body containing an object of type "UsersInRoleModel" where the application will add or remove users from a specified role.

Step 4: Add Role Binding Models

Now we'll add the models used in the previous step, the first class to add will be named "RoleBindingModels" under folder "Models", so add this file and paste the code below:

```

1  public class CreateRoleBindingModel
2  {
3      [Required]
4      [StringLength(256, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 2)]
5      [Display(Name = "Role Name")]
6      public string Name { get; set; }
7
8  }
9
10 public class UsersInRoleModel {
11

```

```
12     public string Id { get; set; }
13     public List<string> EnrolledUsers { get; set; }
14     public List<string> RemovedUsers { get; set; }
15 }
```

Now we'll adjust the "ModelFactory" class to include the method which returns the response of type "RoleReturnModel", so open file "ModelFactory" and paste the code below:

```
1 public class ModelFactory
2 {
3     //Code removed for brevity
4
5     public RoleReturnModel Create(IdentityRole appRole) {
6
7         return new RoleReturnModel
8         {
9             Url = _UrlHelper.Link("GetRoleById", new { id = appRole.Id }),
10             Id = appRole.Id,
11             Name = appRole.Name
12         };
13     }
14 }
15
16 public class RoleReturnModel
17 {
18     public string Url { get; set; }
19     public string Id { get; set; }
20     public string Name { get; set; }
21 }
```

Step 5: Allow Admin to Manage Single User Roles

Until now the system doesn't have an endpoint which allow users in Admin role to manage the roles for a selected user, this endpoint will be needed in the AngularJS app, in order to add it open "AccountsController" class and paste the code below:

```
1 [Authorize(Roles="Admin")]
2 [Route("User/{id:guid}/roles")]
3 [HttpPut]
4 public async Task<IHttpActionResult> AssignRolesToUser([FromUri] string id, [FromBody] string[] rolesToAssign)
5 {
6
7     var appUser = await this.AppUserManager.FindByIdAsync(id);
8
9     if (appUser == null)
10     {
11         return NotFound();
12     }
13
14     var currentRoles = await this.AppUserManager.GetRolesAsync(appUser.Id);
15
16     var rolesNotExists = rolesToAssign.Except(this.AppRoleManager.Roles.Select(x => x.Name)).ToArray();
17
18     if (rolesNotExists.Count() > 0) {
19         ModelState.AddModelError("", string.Format("Roles '{0}' does not exists in the system", string.Join(",", rolesNotExists)));
20         return BadRequest(ModelState);
21     }
22
23     IdentityResult removeResult = await this.AppUserManager.RemoveFromRolesAsync(appUser.Id, currentRoles.ToArray());
24
25     if (!removeResult.Succeeded)
26     {
27         ModelState.AddModelError("", "Failed to remove user roles");
28         return BadRequest(ModelState);
29     }
30
31     IdentityResult addResult = await this.AppUserManager.AddToRolesAsync(appUser.Id, rolesToAssign);
32
33     if (!addResult.Succeeded)
34     {
35         ModelState.AddModelError("", "Failed to add user roles");
36         return BadRequest(ModelState);
37     }
38
39     return Ok();
40 }
41 }
```

What we have implemented in this method is the following:

- This method can be accessed only by authenticated users who belongs to "Admin" role, that's why we have added the attribute **[Authorize(Roles="Admin")]**
- The method accepts the UserId in its URI and array of the roles this user Id should be enrolled in.
- The method will validates that this array of roles exists in the system, if not, HTTP Bad response will be

sent indicating which roles doesn't exist.

- The system will delete all the roles assigned for the user then will assign only the roles sent in the request.

Step 6: Protect the existing end points with [Authorize(Roles="Admin")] Attribute

Now we'll visit all the end points we have created earlier in the previous posts and mainly in "AccountsController" class. We'll add "Roles=Admin" to the **Authorize** attribute for all the end points the should be accessed only by users in Admin role.

The end points are:

– **GetUsers, GetUser, GetUserByName, and DeleteUser** should be accessed by users enrolled in "Admin" role. The code change will be as simple as the below:

```
1 [Authorize(Roles="Admin")]
2 [Route("users")]
3 public IHttpActionResult GetUsers()
4 {}
5
6 [Authorize(Roles="Admin")]
7 [Route("user/{id:guid}", Name = "GetUserById")]
8 public async Task<IHttpActionResult> GetUser(string id)
9 {}
10
11
12 [Authorize(Roles="Admin")]
13 [Route("user/{username}")]
14 public async Task<IHttpActionResult> GetUserByName(string username)
15 {}
16
17 [Authorize(Roles="Admin")]
18 [Route("user/{id:guid}")]
19 public async Task<IHttpActionResult> DeleteUser(string id)
20 {}
```

Step 7: Update the DB Migration File

Last change we need to do here before testing the changes, is to create a default user and assign it to Admin role when the application runs for the first time. To implement this we need to introduce a change to the file "Configuration" under folder "Infrastructure", so open the files and paste the code below:

```
1 internal sealed class Configuration : DbMigrationsConfiguration<AspNetIdentity.WebApi.Infrastructure.ApplicationDb
2 {
3     public Configuration()
```

BIT OF TECHNOLOGY

[ARCHIVE](#)[ABOUT ME](#)[SPEAKING](#)[CONTACT](#)

```
13 var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(new ApplicationDbContext()));
14
15 var user = new ApplicationUser()
16 {
17     UserName = "SuperPowerUser",
18     Email = "taiseer.joudeh@gmail.com",
19     EmailConfirmed = true,
20     FirstName = "Taiseer",
21     LastName = "Joudeh",
22     Level = 1,
23     JoinDate = DateTime.Now.AddYears(-3)
24 };
25
26
27 manager.Create(user, "MySuperP@ss!");
28
29 if (roleManager.Roles.Count() == 0)
30 {
31     roleManager.Create(new IdentityRole { Name = "SuperAdmin" });
32     roleManager.Create(new IdentityRole { Name = "Admin" });
33     roleManager.Create(new IdentityRole { Name = "User" });
34 }
35
36 var adminUser = manager.FindByName("SuperPowerUser");
37
38 manager.AddToRoles(adminUser.Id, new string[] { "SuperAdmin", "Admin" });
```

```
39 }  
40 }
```

What we have implemented here is simple, we created a default user named “SuperPowerUser”, then created there roles in the system (SuperAdmin, Admin, and User), then we assigned this user to two roles (SuperAdmin, Admin).

In order to fire the “Seed()” method, we have to drop the exiting database, then from package manager console you type `update-database` which will create the database on our SQL server based on the connection string we specified earlier and runs the code inside the seed method and creates the user and roles in the system.

Step 7: Test the Role Authorization

Now the code is ready to be tested, first thing to do is to obtain a JWT token for the user “SuperPowerUser”, after you obtain this JWT and if you try to decode it using [JWT.io](#) you will notice that this token contains claim of type “Role” as the below:

```
1 {  
2   "nameid": "29e21f3d-08e0-49b5-b523-3d68cf623fd5",  
3   "unique_name": "SuperPowerUser",  
4   "http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider": "ASP.NET Identity",  
5   "AspNet.Identity.SecurityStamp": "832d5f6b-e71c-4c31-9fde-07fe92f5ddfd",  
6   "role": [  
7     "Admin",  
8     "SuperAdmin"  
9   ],  
10  "Phone": "123456782",  
11  "Gender": "Male",  
12  "iss": "http://localhost:59822",  
13  "aud": "414e1927a3884f68abc79f7283837fd1",  
14  "exp": 1426115380,  
15  "nbf": 1426028980  
16 }
```

Those claims will allow this user to access any endpoint attribute with [Authorize] attribute and locked for users in Roles (Admin or SuperAdmin).

To test this out we’ll create new role named “Supervisor” by issuing HTTP Post to the endpoint (/api/roles/create), and as we stated before this endpoint should be accessed by users in “Admin” role, so we will pass the JWT token in the Authorization header using Bearer scheme as usual, the request will be as the image below:

Create Role	
http://localhost:59822/api/roles/create	
Content-Type	application/json
Accept	application/json
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbG
Header	Value
form-data x-www-form-urlencoded raw binary JSON (application/json)	
1	{
2	"Name": "Supervisor"
3	}

If all is valid we’ll revive HTTP status 201 Created.

In the [next post](#) we’ll see how we’ll implement Authorization access using Claims.

The **source code** for this tutorial is available on [GitHub](#).

Follow me on Twitter [@tjoudeh](#)

Be Sociable, Share!



Like this:



Be the first to like this.

Related Posts

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)




[AngularJS Authentication Using Azure Active Directory Authentication Library \(ADAL\)](#)

[JSON Web Token in ASP.NET Web API 2 using Owin](#)

[Decouple OWIN Authorization Server from Resource Server](#)

Filed Under: [ASP.NET](#), [ASP.NET Identity](#), [ASP.NET Web API](#), [Web API Security](#), [Web API Tutorial](#)

Tagged With: [Authentication](#), [Authorization Server](#), [OAuth](#), [Roles](#)



Question
1/10

Your score
0


Can You Uncover These Android OS Easter Eggs?

Android 5.0 Lollipop OS featured a version of the popular game "Flappy Bird" starring the Android Bot

TRUE ☒ FALSE ☐

Powered by [Carambola](#)

1 2 3 4 5 6 7 8 9 10



Comments



Miguel Delgado says
March 11, 2015 at 3:31 am

Hey Taiseer, kudos on this terrific project... I can't wait for the other two posts...

[Reply](#)



Taiseer Joudeh says
March 11, 2015 at 4:40 pm

Thank you Miguel, glad you liked the series 😊

[Reply](#)



terencevs says

March 11, 2015 at 7:26 am

Thanks, got so excited for the next part when I checked this morning!

[Reply](#)



Taiseer Joudah says

March 11, 2015 at 4:35 pm

You are welcome, glad that you liked the post.

[Reply](#)



Akinsanya Olanrewaju says

March 11, 2015 at 10:11 am

Not all good programmer are good teachers, Not all teachers can make good documentations, Believe me Taiseer, You are a good teacher, goods programmers, good mentors. Thank you so much.

[Reply](#)



Taiseer Joudah says

March 11, 2015 at 4:35 pm

Thanks for your nice words Askinsanya, really happy to hear that my posts are useful to learn from 😊

[Reply](#)



terencevs says

March 11, 2015 at 8:20 pm

I agree with you 100%

[Reply](#)



Reno Natallino says

October 12, 2015 at 8:56 pm

Same here.. i really appreciate your contribution specially for beginner like me.

[Reply](#)

Bilal™ (@akaMBS) says



March 11, 2015 at 4:21 pm

Thank you very much, I was eagerly waiting for this, and the one for claims based is eagerly awaited.
Once again thank you very much.

Reply



Taiseer Joudeh says

March 11, 2015 at 4:24 pm

You are welcome Bilal, glad you find it useful.

Reply



Владимир Землянушкин says

March 11, 2015 at 10:26 pm

You're awesome! Thanks for the great work!

Reply



Taiseer Joudeh says

March 11, 2015 at 10:59 pm

You are welcome, glad you liked it 😊

Reply



Luis Alexander Aldazabal Gil says

March 12, 2015 at 3:33 pm

Great Post, i'm always learning something new reading your posts.

Reply



Taiseer Joudeh says

March 16, 2015 at 5:16 pm

You are welcome, glad you liked them Luis 😊

Reply



yazanrawashdeh says
March 15, 2015 at 2:34 pm

Awesome Articles , I really love them and they're really helpful , and I'm proud of such a Jordanian talent , I was just wondering if the coming posts are coming anytime soon according to a schedule or you'll continue writing in spare time?

[Reply](#)



Taiseer Joudeh says
March 16, 2015 at 9:25 am

You are welcome Yazan, glad you liked it, part 5 should be next Monday, so keep tuned 😊

[Reply](#)



E. Timothy Uy says
March 17, 2015 at 4:54 pm

I'm so excited for the next installment!

[Reply](#)



agc93 says
March 16, 2015 at 5:03 am

Just for anyone else caught out by this: The IdentityRole class and a couple of the other Role-based objects are actually EF-specific. If you are using your own UserStore or using Identity 2.1 without EntityFramework, most of this tutorial won't work as advertised..

[Reply](#)



watea69 says
March 16, 2015 at 5:05 pm

Awesome Articles , I'm new with Angular, have you also a Front-End Angular Sample ?

[Reply](#)



Taiseer Joudeh says
March 16, 2015 at 5:24 pm

Glad it was helpful, the last post is about AngularJS front-end app.

[Reply](#)



Simba says

March 17, 2015 at 2:54 am

Great post and very informative. I'm impatiently waiting for the remaining posts. Good job.

Reply



Taiseer Joudeh says

March 17, 2015 at 9:15 am

Thanks for your comment, next post coming on Monday.

Reply



Joost says

March 25, 2015 at 10:38 pm

Monday past, no new post. Me sad 😞

Reply



Taiseer Joudeh says

March 25, 2015 at 10:40 pm

Sorry about this, it was crazy week 😞 will do my best to publish it ASAP and thanks for your message, you are one of my loyal readers 😊

Reply



Joost says

March 26, 2015 at 12:54 am

Tnx. Of course we understand that but I also wanted to let you know how eagerly we're awaiting your new posts! I'm writing 'we' because I'm sure this goes for all of your readers.



Victor says

March 26, 2015 at 9:14 pm

Can't wait like Joost said 😊



E. Timothy Uy says

March 17, 2015 at 3:31 pm

This sentence was a little bit odd “the first class to add will be named “RoleBindingModels” under folder “Models”, so add this file and paste the code below.” Because the file is called RoleBindingModels.cs, but you have the class RoleBindingModel in it. I’m not sure how but from the last tutorial, I also ended up with AccountBindingModel instead of AccountBindingModels. I made the change and put my ChangePasswordBindingModel into it. Not a big deal.

[Reply](#)



staphill says

March 17, 2015 at 9:47 pm

Hi Taiseer, thanks for a great series of post. I am having an issue when testing the application after this post, I get an “Unable to connect to SQL Server database.” but only when I hit the roles api, i can authenticate, change password, but get user and all the roles call gives me that db error. any ideas what I could be doing wrong?

[Reply](#)



Taiseer Joudeh says

March 18, 2015 at 7:15 pm

Hi,

If you are not able to connect to the database as the message states then you should not be able to authenticate or change password, that’s strange, I can’t think of reason for this issue, if you solved please share it here so I update the post if there is something missing.

[Reply](#)



staphill says

March 18, 2015 at 7:28 pm

Hi,

I am looked at it again today and it seen like the issue happens only on functions or classes that has “[Authorize(Roles = “Admin”)]”. After searching around I found this:

“<http://stackoverflow.com/questions/14521003/mvc4-userisinrole-unable-to-connect-to-sql-server-database>”

I added and now it works. I am not exactly sure why though.

[Reply](#)



Bilal™ (@akaMBS) says

March 27, 2015 at 1:36 pm

Eagerly awaiting part 5 and 6 – One request, you can add decouple the resource server and Authorisation server in part 7 to make this ultimate tutorial.

[Reply](#)



Taiseer Joudeh says

March 27, 2015 at 4:06 pm

Hi Bilal, Part 5 is almost ready, regarding decoupling them it should be easy process and I have [blogged about it earlier here](#), you maybe be interested in forking the repo and implement it and will be happy to refresh your repo too.

[Reply](#)



Bilal™ (@akaMBS) says

March 29, 2015 at 4:34 pm

ohh, I followed the old version of your post and got the api. I managed move the orders controller to resource server. Api acces was successful but could not access Orders via Angular client, it was still looking for ordersController in authorisation server and returned 404.

I need to to learn Github as well 😊 – I am an accountant by the way.

[Reply](#)



Raul says

April 13, 2015 at 9:25 pm

Hi Taiseer, thank you by share with us your knowledge.

I have some questions:

- a) Can I add social logins and refresh token a this approach?, like you explained in ASP.NET Web API 2 external logins with Facebook and Google in AngularJS app.
- b) Can I implement dependency injection using Ninject?, like you explained in Building ASP.Net Web API RESTful Service – Part 4, or it is not necessary?, because I want to have a single instance of database context object shared by all objects created via the kernel for that HTTP request. Until now I'm following ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – from Part 1 to Part 4.

Thank you very much for this excellent tutorial.

Kind regards,

Raul

[Reply](#)



Taiseer Joudeh says

April 15, 2015 at 11:28 am

Hi Raul, sure you can use any IoC container, but I prefer to `app.CreatePerOwinContext()` to create DB instance per owin request. Regarding the social logins I guess it should work even with JWT tokens, I didn't do it before but it should work.

[Reply](#)



Tshaiman says

April 19, 2015 at 11:15 pm

Found Some strange security behavior after implementing this article.

after running your last command that created the "Supervisor" I've done the following :

1.Calling the "assignRolesToUser" endpoint (<http://localhost:60672/api/Account/user/id/roles>) with the string array

["Supervisor"] as the body -> this should remove the Admin Role and assign a Supervisor role according the the logic implemented in that endpoint (removing all and then adding).

2. now , a second call to this endpoint should not succeed since my user is no longer in Admin Role and the endpoint has [Authorize(Roles = "Admin")] attribute.

3. But it does -> Very strange ! anyhow , just before the call I've opened the DB to make sure the user has only 1 role ("Supervisor") and it has.

so why am I given access to the endpoint if the user is no longer the admin ?
is it spooky Owin /Context magic ?

please verify it yourself by following my scenario.

GREAT POST !

[Reply](#)



Taiseer Joudeh says

April 21, 2015 at 11:52 am

In point 2, did you obtain a new token for the new created supervisor user? I doubt that you kept using the same admin access token.

[Reply](#)



Tshaiman says

April 21, 2015 at 12:41 pm

correct. it is the old token.

but still... it means that for the period of 30 minutes (or the time the access token was provisioned), no changes in the role affect the resource server access policy.

I found it to be problematic.

I would have expect that this will re-create the SecurityStamp field in the user table and revoke the access token

Which is not whats happening since the SecurityStamp field only changes for user/password change.

[Reply](#)



Taiseer Joudeh says

April 21, 2015 at 12:45 pm

This is how access tokens work, any change on claims will be reflected on your next access to the system.

[Reply](#)



groovycoder says
May 11, 2015 at 8:14 am

Hi Taiseer,

I am having a problem with routing here, did you adjust the default routing in WebApiConfig?

When testing the create role api call, I get the following error:

```
{"Message": "An error has occurred.", "ExceptionMessage": "Multiple actions were found that match the request: \r\nCreate ...
```

[Reply](#)



Taiseer Joudeh says
May 12, 2015 at 12:07 am

Hi,

I'm using attributes routing here, double check how you attribute your controller using RoutePrefix attribute as well how you attribute you action methods using Route attribute.

[Reply](#)



groovycoder says
May 12, 2015 at 4:40 am

Thanks for your reply, I managed to fix the issue, I had "using System.Web.Mvc" included which was conflicting with "System.Web.Http" classes for routing. Removing System.Web.Mvc fixed the problem.

[Reply](#)



Fred Gassmann says
May 26, 2015 at 10:40 pm

Great post Taiseer! How would I get a list of users in a role? If I get a role from AppRoleManager.FindByIdAsync(guid_id) it returns the role but the users collection inside the role is empty.

[Reply](#)



Taiseer Joudeh says
May 29, 2015 at 8:23 am

Hi Fred,

You need to query this and do a modification on the "ModelFactory" class, I do not have access to the source code now but as far as I remember there is method that you can provide the role id for it and it gets a the users in this role.

[Reply](#)



Hugo says

May 27, 2015 at 7:17 pm

Hello Taiseer ,

Thanks for this amazing tutorials!!

My question is about getting resources only created by his owner.

Imagine that we have a table with several "orders" (created by millions of users) and we only should see our "orders".

How I solve this problem based on your JWT authentication?

Best

[Reply](#)



Taiseer Joudeh says

May 29, 2015 at 8:06 am

Hi Hugo, you can get the user name from the provided token by reading the claim of type "sub", then based on this you can use it to filter your orders records.

[Reply](#)



Troels Larsen says

May 28, 2015 at 10:37 am

Just a minor correction: The null-coalescing operator (??) will not assign any value to _AppRoleManager, leaving it null at all times. The getter for AppUserManager/AppRoleManager should be either:

```
//Same behaviour as now  
get { return Request.GetOwinContext().GetUserManager(); }
```

```
//What you probably want  
get { _AppRoleManager ?? (_AppRoleManager = Request.GetOwinContext().GetUserManager()); }
```

Great series, though I wish you would have separated the Authorization from the Resource server from the start. It is easier to put them in one project than it is to separate them afterwards.

Thanks!

[Reply](#)



Taiseer Joudeh says

May 29, 2015 at 8:35 am

Hi Troels,
Thanks for your comment, I agree with you but it will add more complexity for learning purposes.

[Reply](#)



Tshaiman says
May 29, 2015 at 2:33 pm

Hello once again.

Microsoft has another API product in Preview called Azure APP Service API (www.)

I wonder if the Identity 2.0 approach presented in this series will be valid for this new Service.

the official documentation says you can use Azure Active Directory for Authentication ,and that it supports OAuth 2.0

but the Visual Studio template comes rather empty and I wonder if all customization we are talking about will be valid/necessary in the new service.

Thanks !

Reply



Tshaiman says
May 29, 2015 at 2:36 pm

sorry forgot the link , its <https://azure.microsoft.com/en-gb/documentation/articles/app-service-api-dotnet-add-authentication/>

Reply



hugo says
May 30, 2015 at 2:31 am

I'm using jwt.io to debug the generated jwt token and I can't get the green sign of "Signature Verified". What i'm doing wrong? The "secret" is the AudienceSecret, right? (with secret base64 encoded checked)

Reply



hugo says
May 30, 2015 at 3:21 am

Solved!

Reply



Mike Sullivan says
July 23, 2015 at 4:33 am

Hugo, I have the same issue, but cannot solve it. Is it because my provider is localhost? Can you shed some light on what you found?

Thanks

Reply

**vidriduch** says

June 3, 2015 at 10:36 pm

Excellent article! Just a one question. If a user with lower privileges tries to access restricted action, the response comes as 401 which is Unauthorized and not as 403 Forbidden. This might be a bit pain for the frontend when distinguishing why the resource cannot be accessed ...

[Reply](#)**Taiseer Joudeh** says

June 5, 2015 at 2:09 am

Hmm I agree with you here, and this is the default implementation for the [Authorize attribute], you can override this attribute and return the desired status code.

[Reply](#)**vidriduch** says

June 5, 2015 at 4:21 pm

just in case someone will have a same problem ... to override your Authorize attribute

```
protected override void HandleUnauthorizedRequest(System.Web.Http.Controllers.HttpActionContext
actionContext)
{
    HttpResponseMessage response;

    if (actionContext.RequestContext.Principal.Identity.IsAuthenticated)
    {
        response = actionContext.Request.CreateErrorResponse(HttpStatusCode.Forbidden,
        "insufficient_scope");
        response.Headers.Add("WWW-Authenticate", "Bearer error=\"insufficient_scope\"");
    }
    else
    {
        response = actionContext.Request.CreateErrorResponse(HttpStatusCode.Unauthorized,
        "Unauthorized");
    }

    actionContext.Response = response;
}
```

as per

<http://leastprivilege.com/2014/10/02/401-vs-403/>

and

<https://github.com/IdentityModel/Thinkecture.IdentityModel/blob/master/source/WebApi.ScopeAuthorization/ScopeAuthorizeAttribute.cs>

[Reply](#)



mathues says

June 21, 2015 at 5:42 pm

My application returns null on property protected ApplicationRoleManager AppRoleManager (return _AppRoleManager ?? Request.GetOwinContext().GetUserManager());

[Reply](#)



Ivan Bohannon says

July 10, 2015 at 6:16 am

I'm also getting the crash when trying to add the supervisor role.

[Reply](#)



Mike Sullivan says

July 23, 2015 at 4:36 am

I'm having an issue trying to assign roles via the API. The security works fine, but I cannot get the body of the PUT to bind to the rolesToAssign string array in the controller. It comes in empty, which wipes all roles from the user.

here is my request body (raw) JSON:

```
[  
  { "rolesToAssign": ["Admin", "SuperAdmin"] }  
]
```

What am I doing wrong.?

[Reply](#)



Ishwor says

November 5, 2015 at 12:51 am

I am having the same issue. Has anyone succeeded in assigning role(s) to user?

[Reply](#)



Taiseer Joudeh says

November 6, 2015 at 3:43 am

Strange, please download the repo as it is working at my end with no problems at all.

The request body should be as the below:

```
["Dataentry","SuperSupervisor"]
```

[Reply](#)



Manoj Bisht says
July 26, 2015 at 11:10 am

Very nice post. Kudos on the great job.

Small correction if you can make..in Step 7: Update the DB Migration File..configuration.cs is in migration folder and not infrastructure

[Reply](#)



Taiseer Joudeh says
July 27, 2015 at 4:18 am

Thanks Manoj for the clarification, will check it and update the post.

[Reply](#)



Michael Hodgson says
August 6, 2015 at 4:50 pm

Useful series. I notice that you use Postman as a client. Any tips on how we can use the Authorization section in Postman with this template rather than having to append the Authorize header to request headers by hand?

[Reply](#)



Taiseer Joudeh says
August 8, 2015 at 7:33 am

Hi Michael,
Good question that it is an item in my long TODO list 😊 I want to try it out and see how I can use it, once I try it I will post the outcome here. If you already did it please share the results.

[Reply](#)



Paul Wade says
September 3, 2015 at 10:11 pm

Hi Taiseer,
I have finished reading Part 5 of your blog and have got the download working from GitHub.
Just wanted to say THANK YOU.
Any idea when you will have Part 6 ready re. AngularJS application which connects all those posts together.

Kind Regards,
Paul

[Reply](#)



Taiseer Joudeh says

September 5, 2015 at 10:20 pm

You are most welcome, it's hard post which I need to finish soon, thanks for your message 😊

[Reply](#)



krishh86 says

September 9, 2015 at 12:54 am

Hi Taiseer,

Kudos for this awesome series. Very eagerly waiting on your next post. Thank you very much!..

[Reply](#)



Taiseer Joudeh says

September 10, 2015 at 12:47 pm

You are welcome, thanks for your message.

[Reply](#)



Apollo says

September 19, 2015 at 6:55 pm

Hats of to you sir, thanks again!

[Reply](#)



Vikas says

October 7, 2015 at 9:16 am

Hi Taiseer,

First of all Thanks for this great article. I have implemented same in my environment.

I have few queries regarding security.

1. What about session hijacking?
2. If I need to return new refresh token on each subsequent request what changes I need to make?

[Reply](#)



Taiseer Joudeh says

October 12, 2015 at 10:31 am

Hi Vikas,

There is no session established with REST Api, everything is stateless and you are sending token and validating it with each request.

Why you want to generate refresh token with each request? You should not do this at all, you need to get refresh token only once when you get an access token and you store it at the client for future use when the access token expires.

[Reply](#)



Bálint Arszenovits says

October 24, 2015 at 11:34 pm

Dear Taiser,

I completed your tutorial, but every Time I try to use the controllers with Role authorization, I get "Authorization has been denied for this request.". I checked the JWT token, and it contained the required "Admin" role. I

downloaded your solution from GitHub, but the result is the same.

Thanks for the articles, they are great, I learned much from them!

[Reply](#)



Taiseer Joudeh says

October 26, 2015 at 10:22 am

Hi Bailnt,

That is strange, I have just tried it and it works fine, If you removed the Authorized attribute with a role and replace it just with [Authorized] this will work? I'm afraid that your JWT token is not accepted by the Api.

[Reply](#)



Shan Sundaram says

January 20, 2016 at 12:58 am

The authorize attribute works with roles. I think everyone is making the same mistake which I did. First try just with Authorize Attribute. Then use the same jwt token and change the code to add roles to authorize attribute and try.

When attribute changes, need to get new token. Then with new token, it works perfectly

[Reply](#)



Taiseer Joudeh says

January 27, 2016 at 8:19 am

Yup, claims are stored in the token, so if you changed a role for a user who already owns the token, then the user needs to obtain new one in order to have those claims encoded within the token.

[Reply](#)



prashantchalise says

December 14, 2015 at 9:33 am

Hi Taiseer,

First, thank you so much for all these great articles. You are the best.

While implementing roles, I got a strange issue where the [Authorize] attribute works just perfectly fine but got error while adding roles. The issue is a connection time out issue while connecting to SQL server. Fixed this by adding

on web.config file. Don't know how this works but is working. 😊

Thank you so much once again.

Reply



Taiseer Joudeh says

December 16, 2015 at 10:59 am

Hi,

I cant see what you added to the web.config, seems wordpress filter removed it. You should not add anything extra to web.config in order to make roles working.

Reply



Leonardo Culjak says

December 19, 2015 at 1:25 am

Excelent Article ! I have tried angular application pointing to the web api and everything works perfect. The only issue that I have is about [Authorize] attribute. It is working fine adding users, but it is not working (for me) with Roles.

I have used the Seed method in order to add an admin user with three roles, Admin, SuperAdmin, User.

if I login with admin user, I can not access to [Authorize(Roles = "Admin")] methods. Do I have to assign manually the roles to Context.User. I think that it is done on GrantResourceOwnerCredentials method. But just double checking.

See reference:

<http://stackoverflow.com/questions/18807806/authorize-attribute-not-working-with-roles>

Reply



Taiseer Joudeh says

December 24, 2015 at 3:11 am

Hi Leo, thaks for your message.

You do not need to add them manually, those will come from ASP.NET identity helper method as the [code here](#) Can you check the jwt token using jwt.io and make sure you have a claim of type "Role" set to the value "Admin"?

Reply



Leo says

December 24, 2015 at 12:01 pm

Thanks for your quick answer ! Got it, I think that I found the issue. I am using your previous version called AngularJSAuthentication.API and there you have a method GrantResourceOwnerCredentials and there there is this line that allays assign user Role. "identity.AddClaim(new Claim(ClaimTypes.Role, "user"));"

I have updated that line using " foreach (var role in user.Roles)

```
{
identity.AddClaim(new Claim(ClaimTypes.Role, role));
}" and now is working fine. Excellent article !!
```

[Reply](#)


alexstrakh says

January 11, 2016 at 2:48 pm

I tried to implement the role based solution and it provides proper grant, claims, roles:

I can access resources protected with Authorize attribute only when provide the bearer token.

Unfortunately once I specify role on the Authorize attribute it stops working with some strange "unable to open database error".

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

And here my claims when grant is created:

```
[
{
"subject": "Admin",
"type": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier",
"value": "cf55a34e-ba04-4a12-bb6a-ee9bce5e903f"
},
{
"subject": "Admin",
"type": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name",
"value": "Admin"
},
{
"subject": "Admin",
"type": "http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider",
"value": "ASP.NET Identity"
},
{
"subject": "Admin",
"type": "AspNet.Identity.SecurityStamp",
"value": "18fd3ecc-40e2-4613-be63-4bae20f9c93b"
},
{
"subject": "Admin",
"type": "http://schemas.microsoft.com/ws/2008/06/identity/claims/role",
"value": "Admin"
},
{
"subject": "Admin",
"type": "iss",
"value": "http://localhost:1684"
},
{
"subject": "Admin",
```

```
"type": "aud",  
"value": "cca73ab6c73d401ca28b4d306d342647"  
},  
{  
"subject": "Admin",  
"type": "exp",  
"value": "1452597814"  
},  
{  
"subject": "Admin",  
"type": "nbf",  
"value": "1452511414"  
}  
}
```

<https://www.dropbox.com/s/kb208ssgmbjebup/Screenshot%202016-01-11%2006.48.22.png?dl=0>

What could be wrong here?

Reply



alexstrakh says

January 11, 2016 at 3:23 pm

I just found the answer:

<http://stackoverflow.com/questions/23941027/asp-net-identity-authorize-attribute-with-roles-doesnt-work-on-azure>

All you need is to disabled default RoleManager:

Reply



Jothi Kiruthika says

January 20, 2016 at 11:12 am

This is such an amazing series! Helped a lot! Thanks for all the nice effort Taiseer!

Reply



Taiseer Joudeh says

January 27, 2016 at 8:18 am

You are welcome jothi, happy to help!

Reply



AssetFENCE says

January 23, 2016 at 8:45 pm

Can you please update, some of these methods are deprecated in 2.2

Reply



Taiseer Joudeh says

January 27, 2016 at 8:12 am

I do not think there is any deprecated methods I'm using in 2.1 version, what did you face?

[Reply](#)



Tobias Koller says

January 27, 2016 at 11:40 pm

thanks for these articles, they are really great!!

I have just one question:

Wouldn't it be much simpler to just get all user-roles when the user logs in the system and assign them right to the identity?

with this I wouldn't need all the AppRoleManager, etc...

Here is my example (most of the code comes from you 😊)

- I'm using userManager.ClaimsIdentityFactory.CreateAsync
- and assign the role like identity.AddClaim(new Claim(identity.RoleClaimType, "admin"));

this should be enough to handle roles in [Authorize(Roles="admin")]

```
public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
```

```
{
    context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });
```

```
    using (var userManager = CreateUserManager(context.UserName))
    {
        var user = await userManager.FindAsync(context.UserName, context.Password);
```

```
        if (user == null)
        {
            context.SetError("invalid_grant", "The user name or password is incorrect.");
            return;
        }
```

```
        var identity = await userManager.ClaimsIdentityFactory.CreateAsync(userManager, user,
            context.Options.AuthenticationType);
```

```
        foreach (var role in user.Roles.Select(r => r.RoleId).Distinct().ToList())
        {
            identity.AddClaim(new Claim(identity.RoleClaimType, role));
        }
```

```
        context.Validated(identity);
```

```
        //return new GenericPrincipal(null, new string[0])
    }
}
```

[Reply](#)



Ashy says

February 17, 2016 at 2:33 am

Can I use this if I am using Odata. authenticate against a database custom user table and custom role table. We dont use identity tables we have our custom tables for user and roles ?

[Reply](#)

Taiseer Joudeh says

February 17, 2016 at 3:10 pm

Hi Ashy,

Sure thing you can use this with OData with your own custom tables. Let me know if you need more details.

[Reply](#)

Ashy says

February 17, 2016 at 6:05 pm

Hi Taiseer,

I posted a question in stackoverflow

<http://stackoverflow.com/questions/35447730/authentication-and-authorization-asp-net-web-api-and-angularjs-windows-and-for>

I am directed back to your blog. Your tutorials have so much respect and credibility in the developers world. Great job and keep up the good work.

Can you please help with me with the question on stackoverflow ?

Thank you

[Reply](#)

Ashy says

February 18, 2016 at 3:32 am

Hi Taisser,

I know this question is asked before and I could get any definitive answer.

how to decide odatacontroller vs webapi controller ?

Thank you,

Ashy

[Reply](#)

Nitija says

March 11, 2016 at 11:45 am

Hi ,

I am trying to implement Identity Management in my Application. I am getting below error.
"The entity type IdentityRole is not part of the model for the current context."

I have below code for RoleManager.

```
public class ApplicationRoleManager : RoleManager
{
    public ApplicationRoleManager(
        IRoleStore roleStore)
        : base(roleStore)
    {
    }
    public static ApplicationRoleManager Create(
        IdentityFactoryOptions options, IOwinContext context)
    {
        return new ApplicationRoleManager(
            new RoleStore(context.Get()));
    }
}
```

And below code for ApplicationUserManager.

```
public class ApplicationDbContext : IdentityDbContext
{

    public ApplicationDbContext(string connString)
        : base(connString)
    {

    }

    public DbSet UserRoles { get; set; }
    public DbSet PreviousPasswords { get; set; }

    public static ApplicationDbContext Create()
    {
        return new ApplicationDbContext("RPMS");
    }

    // public DbSet Users { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Conventions.Remove();
    }

}
```

Can you please advice on as what mistake am I making.
Thanks in advance.

[Reply](#)



Taiseer Joudeh says
March 15, 2016 at 1:10 pm

Hi,

I'm not sure what is your issue but [this might help](#).

[Reply](#)



Amol Kulkarni says

March 22, 2016 at 1:36 pm

Hi

Instead using the Identity framework to validate the user credentials I want to use a webservice to validate this user and then generate a token. How can do the same ? The webservice is essentially a layer to access membership users. Please note in my case this webservice to access the users is already present. Do you have any example of this anywhere ?

[Reply](#)



Taiseer Joudeh says

March 24, 2016 at 12:18 pm

Hi Amol,

You need to replace the [highlighted section here](#), with a call to your remote web service where you provide credentials and it returns if those credentials are valid or not, and you continue the logic as is.

Hope this help.

[Reply](#)



Meriem says

March 23, 2016 at 4:50 pm

Hello Taiseer , i'm sorry about the rush i need an urgent help please , i followed your serie " Token Based Authentication using ASP.NET Web API 2, Owin, and Identity " with angular and now i need to add role management so i tried the easiest thing, i added the role on the database and then added [Authorize(Roles="Superusers")] in the controller methodes but it's throwing me 401 unauthorized i don't know what i am missing.

[Reply](#)



Taiseer Joudeh says

March 23, 2016 at 6:53 pm

Hello Meriem,

When you try to decode the JWT using jwt.io do you see a claims of type "Roles"? If not then your JWT doesn't contain those roles and the Api will never understand them. If you removed the "Roles" from "Authorize" attribute it is working, right?

[Reply](#)



Meriem says

March 23, 2016 at 7:59 pm

i am trying to decode but getting an error like invalid signature, this is the json i have

```
{
  "access_token": "Q7RrQnV0Az_AHpBXGYWEjP6BZBD0NW49e9x-MfD3zlaaglx16DyOALEQSqcrupEVBAD7aO04Z5LrDoNpcD8eMzlErtWblpfsNpR4NfNVlUuncXAmiXG66fIRKrDNDkm7DgAB70uobdQ9_OB0x5o9BiNZryrQx_I"
```

```

zHEp2BfuWi0BQJFd8wVFzR6YsmLg5Aba7pyMFjpFSIYbR4tyg8RgkTaITgwejLw09_rQaPdQQmVQELa3t8TEzqCu8hBsZ3HkZxb8Y1uJugAGTP9Nw3Ar-
B9dbUA",
"token_type": "bearer",
"expires_in": 1799,
"as:client_id": "",
"user_name": "admin9",
"issued": "Wed, 23 Mar 2016 16:50:28 GMT",
"expires": "Wed, 23 Mar 2016 17:20:28 GMT"
}

```

Reply



Shoaib says

March 24, 2016 at 10:07 am

Hi... Meriem,

Please make following changes to your desired goal,

Prerequisite :

Your database contains one Users Table, one Roles Table, and one UserRoles table

- 1) Users Table contain your application users
- 2) Roles table contain your application Roles to be assigned to Users
- 3) UserRoles table contains mapping of Roles to Users means Role 'Admin' to be assigned to 'ABC' user.

After that

Open method 'GrantResourceOwnerCredentials' that is present in class

'SimpleAuthorizationServerProvider' which is inherited from 'OAuthAuthorizationServerProvider'

Now You see the code like code snippet 1:

code snippet 1:

```

var identity = new ClaimsIdentity(context.Options.AuthenticationType);
identity.AddClaim(new Claim(ClaimTypes.Name, context.UserName));
identity.AddClaim(new Claim(ClaimTypes.Role, "user"));
identity.AddClaim(new Claim("sub", context.UserName));

```

Just make following changes:

code snippet 2:

```

var userRoles = await _authService.GetUserRolesByUserID(UserID);
foreach (var role in userRoles)
{
    identity.AddClaim(new Claim(ClaimTypes.Role, role.RoleName.ToString()));
}

```

This above code snippet 2 is use for retrieving UserRoles from database. for your side you just write a code to retrieve UserRoles from database

What you have to performed here is:

Just comment the 3rd line of code snippet 1, and copy the code in snippet 2 and paste it instead of 3rd line of code snippet 1 that is look like,

```
identity.AddClaim(new Claim(ClaimTypes.Role, "user"));
```

What it actually means:

Here in 3rd line of code snippet 1 contains static role named as 'user' so all methods in your controller are accessible for only for the role 'user'

So by the code snippet 2 you just take all the roles for your desired User's from your database and assigning these roles in foreach loop...

Have a nice day...

Reply



Taiseer Joudeh says

March 24, 2016 at 11:57 am

Thanks Sohaib for posting this detailed answer, you are correct, I guess you are missing **this highlighted LOC**, you can get the roles from the database and you can add many roles for the user. Hope this help.



meriem says

March 24, 2016 at 8:37 pm

thanks a lot i can't be more gratefull since i'm beginner and working on my graduation project. Shoaib may i ask what's authService for you and UserID has to be extracted right ? but from where because here is all the code before sinippet 1

```
{
var allowedOrigin = context.OwinContext.Get("as:clientAllowedOrigin");

if (allowedOrigin == null) allowedOrigin = "*";

context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] {
allowedOrigin });

using (AuthRepository _repo = new AuthRepository())
{
IdentityUser user = await _repo.FindUser(context.UserName, context.Password);

if (user == null)
{
context.SetError("invalid_grant", "The user name or password is incorrect.");
return;
}
}
...
}
```



meriem says

March 24, 2016 at 9:12 pm

ops sorry i missedd the part saying i have to write it



meriem says

March 24, 2016 at 9:59 pm

i am kind of struggling can you help me with the method to extract all roles from the db



Taiseer Joudeh says
March 24, 2016 at 12:00 pm

Hi Meriem,

This id not a JWT token, this is the default opeque access token, if you would like to know more about JWT, please check this [post](#).

[Reply](#)



Shoaib says
March 24, 2016 at 2:43 pm

Hi... Taiseer,
As you said in previous post that I'd missed to write the highlighted LOC such as,

```
identity.AddClaim(new Claim(ClaimTypes.Role, "user"));
```

but this LOC is actually included in foreach loop of code snippet 2.
the LOC in foreach loop of code snippet 2 is look like:

```
identity.AddClaim(new Claim(ClaimTypes.Role, role.RoleName.ToString()));
```

Thanks for your valuable response...



Shoaib says
March 25, 2016 at 3:51 pm

Hi... Meriem,
just find the below snippet in 'SimpleAuthorizationServerProvider' class

```
using (AuthRepository _repo = new AuthRepository())  
{  
    IdentityUser user = await _repo.FindUser(context.UserName, context.Password);  
  
    if (user == null)  
    {  
        context.SetError("invalid_grant", "The user name or password is incorrect.");  
        return;  
    }  
}
```

In above code when you just type a object name of IdentityUser that is 'user' then just type dot(.) then you get the list of properties of IdentityUser class and in that class you find the 'Id' property like.

user.Id,
and after that you just pass this user.Id to your method that you get all the UserRoles.

[Reply](#)



Shoaib says
March 26, 2016 at 9:30 am

And for simplicity the methos for getting all the roles from database is like..

```
public List GetRolesByUserBasicID(long UserID)
{
    List UserRoleList = new List();
    try
    {
        using (SqlConnection con = new SqlConnection("Your database connection string that is present
in web.config file"))
        {
            SqlCommand cmd = new SqlCommand("select RoleName from UserRoles where UserID =
"+UserID+"", con);
            cmd.CommandType = System.Data.CommandType.Text;
            SqlDataReader dr = null;
            dr = cmd.ExecuteReader();
            if (!dr.IsClosed && dr.HasRows)
            {
                while (dr.Read())
                {
                    UserRoleList.Add(new UserRoles
                    {
                        RoleID = Convert.ToInt32(dr["RoleName"])
                    });
                }
            }
            return UserRoleList;
        }
    }
    catch (Exception ex)
    {
        //error writing code is here
        return null;
    }
}
```

[Reply](#)



Meriem says

March 26, 2016 at 2:54 pm

thank you again for the help and sorry for the mess, if i understood even if i added claims like identity.AddClaim(new Claim(ClaimTypes.Role, "admin")); (instead of snippet2) and then added admin as a role (in roles table) and added its role id to a user it will work and only admins will be able to user the method ?



Meriem says

March 26, 2016 at 3:10 pm

After trying all the users are able to use the method i'm really confused



Meriem says

March 28, 2016 at 2:18 pm

Thank you very much shoaib for your generous help, now if i want to do this with angular (like allow one html page to admin and others to users) do you have an idea



Leave a Reply

Enter your comment here...

RECENT POSTS

ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5

ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4

Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3

ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2

Interview with John about establishing a successful blog

TAGS

AJAX AngularJS API API Versioning

ASP.NET Attribute Routing Authentication

Autherization Server basic authentication C# CacheCow

Client Side Templating Code First Dependency Injection

Entity Framework ETag Foursquare API

HTTP Caching HTTP Verbs IMDB API IoC Javascript jQuery JSON

JSON Web Tokens JWT Model Factory Ninject OAuth

OData Pagination Resources Association Resource Server

REST RESTful Single Page

Applications SPA Token Authentication

Tutorial Web API Web API 2 Web

API Security Web Service wordpress.com

wordpress.org

CONNECT WITH ME

[f](#) [@](#) [g+](#) [i](#) [in](#) [RSS](#) [t](#)

SEARCH

Search this website...