

[github](#) - [Stack Overflow](#) - [LinkedIn](#) - [MVA](#) - a developer braindump on .Net, c#, Java, Javascript ...

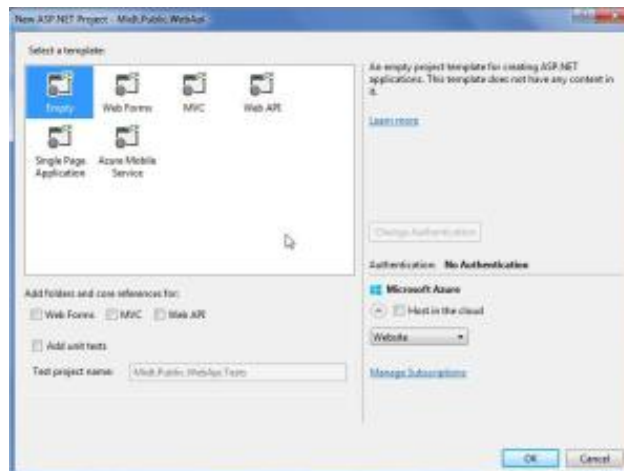
**{"@id":"cedric-dumont.com"}**

## IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 3

**Home (<http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/>)**

This Part will focus on creating a web api that could be called by our AngularJs application (in part 4) or by an Other client like an MVC application (we will create it in part 5).

### Step 1 : create an empty web project



**(<https://cedricdumont.files.wordpress.com/2014/12/3-1.jpg>)**

### Step 2 : add the required Nuget package

for Owin

- 1 | `install-package Microsoft.Owin.Host.SystemWeb`
- 2 | `install-package Microsoft.AspNet.WebApi.Owin`
- 3 | `install-package Microsoft.AspNet.WebApi.Cors`
- 4 | `install-package Thinktecture.IdentityServer.v3.AccessTokenValidation`

cors are added so ajax call can call this api from different origins and the AccessTokenValidation is used to validate the token we receive with Identityserver v3.

### Step 3 : Create the Controller

Here, we will create the Resource class that represents the Resources to be “protected” (for whom we need a consent of the owner to get access to...)

```
1 [RoutePrefix("resource")]
2 public class ResourceController : ApiController
3 {
4     [Route("{id}")]
5     public IHttpActionResult Get(long id)
6     {
7         var res = from element in Resource.GetDummyList()
8                   where element.Id == id select element;
9
10        if (res == null)
11        {
12            return NotFound();
13        }
14
15        return Json(res);
16    }
17 }
```

The controller has a simple Get Method that returns a resource based on its ID.

#### Step 4 : Create the Startup.cs class

here we just init the webapi.

```
1 public class Startup
2 {
3     public void Configuration(IApplicationBuilder app)
4     {
5         var config = new HttpConfiguration();
6         config.MapHttpAttributeRoutes();
7
8         app.UseWebApi(config);
9     }
10 }
```

#### Step 5 : Test it

Run the project and go to : <http://localhost:14117/resource/2> (<http://localhost:14117/resource/2>)

you should see something like this:



So far ... so good ....

## Step 6 : add Authorize

Now that our Api is working we will add some protection to it. Remember in [part 1 \(http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-part-1/\)](http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-part-1/), we've added a scope to our `Scopes.cs` class named `'publicApi'`.

This scope will be required by the caller to be allowed to call this api. so when asking for a token to the Identity server, we need to add the scope `publicapi`.

for this we need to add in our `Startup.cs` that we will use Bearer Token Authentication using our Identity server. We also need to add the `authorize` attribute to our controller

```

1  -- In Startup.cs
2
3      ...
4      app.UseIdentityServerBearerTokenAuthentication(
5          new IdentityServerBearerTokenAuthenticationOptions
6          {
7              Authority = "https://localhost:44305/identity (https://local
8              RequiredScopes = new[] { "publicApi" }
9          });
10     ...
11
12  -- In ResourceController.cs
13
14      [RoutePrefix("resource")]
15      [EnableCors(origins: "*", headers: "*", methods: "*")]
16      [Authorize]
17      public class ResourceController : ApiController
18      {
19      ...

```

N.B. : The url might be different in your case.

## Step 7 : Test with fiddler

recall in [Part 1 \(http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-part-1/\)](http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-part-1/) that we've tested the installation to get a Token, we will use the same to get a Bearer Token and use it to call our public web api. All that with fiddler.

But first let's replay Step 5 and call : <http://localhost:14117/resource/2> (<http://localhost:14117/resource/2>)

if everything is configured well, you should see:




<https://cedricdumont.files.wordpress.com/2014/12/3-3.jpg>

So, first let's get a token from identity server using fiddler like we did in part 1

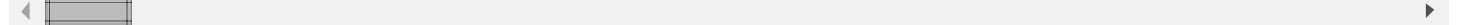
Issue the following Post

```
1  -----
2  Post :
3
4  https://localhost:44305/identity/connect/token (https://localhost:44305
5
6  Headers:
7  User-Agent: Fiddler
8  Content-Type: application/x-www-form-urlencoded
9  Authorization: Basic SWRlbnRpdHlXZWJVSTpzZWNYZXQ=
10
11  Body:
12  grant_type=password&username=test&password=test&scope=openid publicApi
13  -----
```



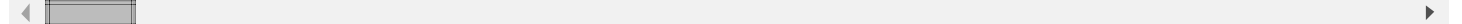
I got an access token:

```
1  {"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImRyRk1QZDg
```

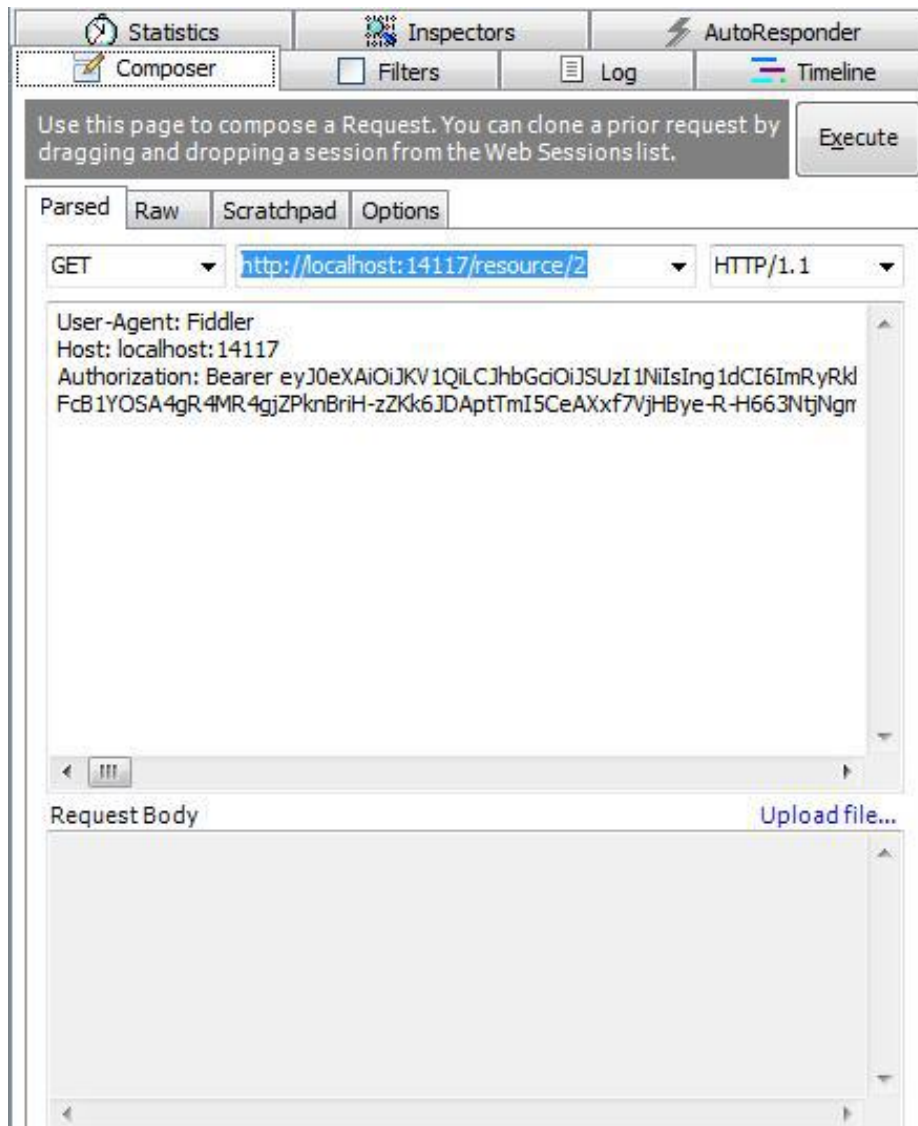


Now I will use this token to call my public api.

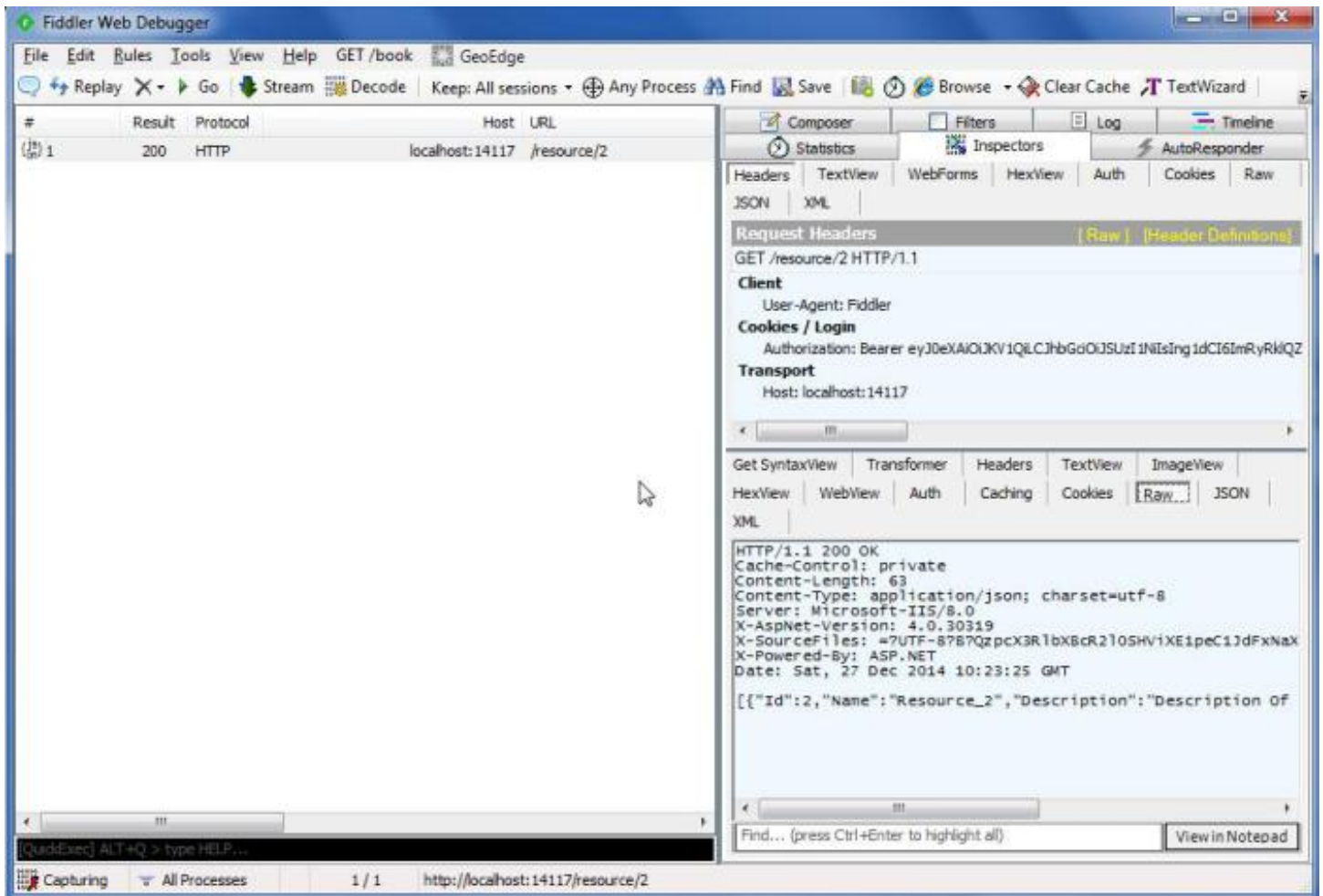
```
1  Request
2  -----
3  Get : http://localhost:14117/resource/2 (http://localhost:14117/resource
4
5  Headers:
6  User-Agent: Fiddler
7  Host: localhost:14117
8  Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImRyR
9
10  Response
11  -----
12
13  [{"Id":2,"Name":"Resource_2","Description":"Description Of 2"}]
```



<https://cedricdumont.files.wordpress.com/2014/12/3-4.jpg>



**(<https://cedricdumont.files.wordpress.com/2014/12/3-4.jpg>)Result :**



(<https://cedricdumont.files.wordpress.com/2014/12/3-5.jpg>)

## 4 thoughts on “IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 3”

1. Pingback: [IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! | {"@id":"cedric-dumont.com"}](https://cedricdumont.files.wordpress.com/2014/12/3-5.jpg)

2.

**ericswann** says:

**February 6, 2015 at 18:25**

Cedric, what do you see as the advantage in separating the resource service and the account service into separate physical API's? Is there a reason other than basic separation of concerns?

**REPLY**

3.

**cedricdumontc** says:

**February 6, 2015 at 19:07**

in my own case, the account service is kind of private to my app where there will not only be an api to manage accounts but also private api calls for my spa that will be hosted on one server or one webrole on azure. the resource api is more a public api that will be more exposed and will be called more often by customers so deployed on another web role and even another domain. this could be all set up in one app, one web role, but i found it easier to manage and to track api usage. the identity server will perhaps be hosted in another domain and not on azure but another server on another provider. hope it makes sense

### REPLY

4.

**Joshua** says:

**February 18, 2015 at 19:27**

I'm not sure if I'm doing something wrong, or if I'm expecting more than what is covered in the tutorial. I have a server up and running IdentityServer, for now I've used the IdentityManager admin UI to create a user and add a couple claims to that user. I can use fiddler to call out to the IdentityServer, and get a token. If the token is valid my WebApi will server a resource, if it's invalid it returns a 401. So far so good!

My problem starts when I want at any of the claims (I'm assuming should have come over) I can't see any of the role claims I've assigned via the Admin UI, or even a name claim which I think should be present. Any suggestions?

### REPLY

[Blog at WordPress.com.](#) — [The Sequential Theme.](#)

© Follow

Follow {"@id":"cedric-dumont.com"}

**Build a website with WordPress.com**

