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

IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 1

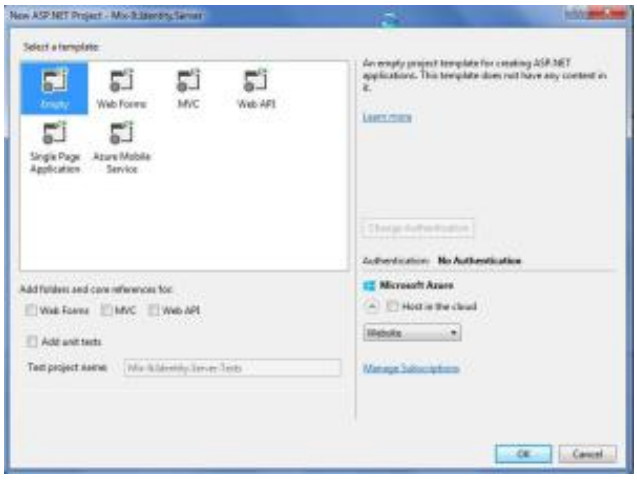
Home (http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/)

This article is part of a **series (http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/)**dealing with IdentityServer.v3 authentication and authorization.

This Part deals with IdentityServer.V3 installation and configuration using MemberShipReboot as the undelying account repository. We will also add BrockAllen's IdentityManager for admin purposes.

Step 1 : Create the web project

start visual studio and create an empty web project



(https://cedricdumont.files.wordpress.com/2014/12/1-1.jpg)

Step 2 : add the Dependencies

Add the following Nuget packages (some of them are in pre realease, so if you use Nuget UI tool, check that 'prerelease' is selected in the checkbox.)

For identity server

```
install-package Microsoft.Owin.Host.Systemweb
install-package Thinktecture.IdentityServer.v3 -pre
```

for use with membershipreboot

```
install-package Thinktecture.IdentityServer.v3.MembershipReboot -pre
```

for use of Identity Manager

```
install-package Thinktecture.IdentityManager.MembershipReboot - pre
Install-Package BrockAllen.MembershipReboot.Ef
```

Step 3: Create a self signed certificate.

Check **this post (http://www.jayway.com/2014/09/03/creating-self-signed-certificates-with-makecert-exe-for-development/)** for the long story

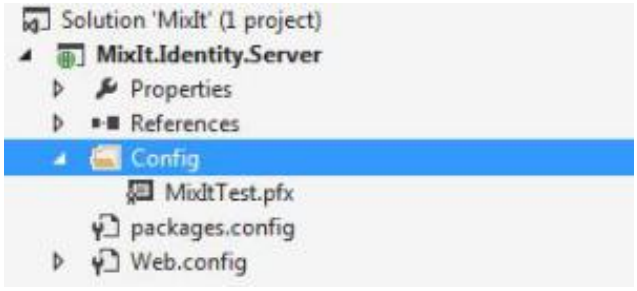
simply create a cmd file with the following content:

```
1  makecert.exe ^
2  -n "CN=MixItCARoot" ^
3  -r ^
4  -pe ^
5  -a sha512 ^
6  -len 4096 ^
7  -cy authority ^
8  -sv MixItTest.pvk ^
9  IdentitySrvCAROOT.cer
10
11 pvk2pfx.exe ^
12 -pvk MixItTest.pvk ^
13 -spc MixItTest.cer ^
14 -pfx MixItTest.pfx ^
15 -po IAmAPassword
```

on line 2 you have to specify a CA Root (choose whatever example : 'DevRoot') and line 15 a password (here : *IAmAPassword*).

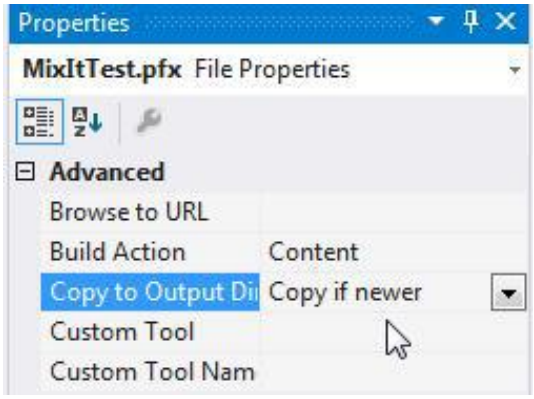
open a visual studio command prompt. (for me it was located here : C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools\Shortcuts , but really depends on your installation) and execute your command file. This will create 3 files. (a .cer for the certificate, a .pvk which is the private key and a .pfx which contains both)

Create a *Config* folder in your solution and copy the YourCertFileName**.pfx** over there. (here *MixItTest.pfx*)



(https://cedricdumont.files.wordpress.com/2014/12/1-2.jpg)

On the properties of the pfx file, set The Copy to Output Dir to '*Copy if newer*'



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

Step 4 : Create the Configuration classes.

In this step we will create a ***Clients.cs*** class that will hold our Client and a ***Scopes.cs*** class that will hold our InMemory *Scope* Objects. (these could come from a database or another repository.)

For the **Users**, we will use a *MemberShipReboot* repository.

Below is the Clients.cs listing that contains a single Client for now. It uses the Resource Owner flow.

```
1  namespace  MixIt.Identity.Server.Config
2  {
3      public static class Clients
4      {
5          public static IEnumerable<Client> Get()
6          {
7              return new[]{
8              new Client
9                  {
10                     //Resource Owner Flow Client (our web UI)
11                     ClientName = "WebUI",
12                     Enabled = true,
13
14                     ClientId = "IdentityWebUI",
15                     ClientSecret = "secret",
16
17                     Flow = Flows.ResourceOwner,
18                     AccessTokenType = AccessTokenType.Jwt,
19                     AccessTokenLifetime = 3600
20
```

```
                    }
                };
            }
        }
    }
```

The **Scopes.cs** contains the scopes for our application. for now there are the standard scope (openid…) and a scope that we defined to access our public api (we will create it later).

```
namespace MixIt.Identity.Server.Config
{
    public static class Scopes
    {
        public static IEnumerable<Scope> Get()
        {
            var scopes = new List<Scope>
            {
                new Scope
                {
                    Enabled = true,
                    Name = "publicApi",
                    Description = "Access to our public API",
                    Type = ScopeType.Resource
                }
            };

            scopes.AddRange(StandardScopes.All);

            return scopes;
        }
    }
}
```

add a MemberShip Reboot User Service:

```
public class MembershipRebootUserServiceFactory
   {
       public static IUserService Factory(string connString)
       {
           var db = new DefaultMembershipRebootDatabase(connString);
           var repo = new DefaultUserAccountRepository(db);
           var userAccountService = new UserAccountService(config, repo
           var userSvc = new MembershipRebootUserService<UserAccount>(u
           return userSvc;
       }

       static MembershipRebootConfiguration config;
       static MembershipRebootUserServiceFactory()
       {
           System.Data.Entity.Database.SetInitializer(new System.Data.E

           config = new MembershipRebootConfiguration();
           config.PasswordHashingIterationCount = 50000;
           config.AllowLoginAfterAccountCreation = true;
           config.RequireAccountVerification = false;
       }
   }
```

and a Factory.cs class to configure all stores and services (here : scopes, clients and users)

```
public class Factory
  {
      public static IdentityServerServiceFactory Configure(string connS
      {
          var factory = new IdentityServerServiceFactory();

          factory.UserService =
              new Registration<IUserService>(resolver => MembershipRebo

          var scopeStore = new InMemoryScopeStore(Scopes.Get());
          factory.ScopeStore = new Registration<IScopeStore>(resolver =

          var clientStore = new InMemoryClientStore(Clients.Get());
          factory.ClientStore = new Registration<IClientStore>(resolver

          return factory;
      }
  }
```

Step 5 : Glue it with the Startup.cs

```
public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.Map("/identity", idsrvApp =>
            {
                idsrvApp.UseIdentityServer(new IdentityServerOptions
                {
                    SiteName = "Identity Server",
                    IssuerUri = "https://idsrv3/mixit (https://idsrv3/m
                    SigningCertificate = LoadCertificate(),

                    Factory = Factory.Configure("MyIdentityDb"),

                    CorsPolicy = CorsPolicy.AllowAll
                });
            });

        }

        X509Certificate2 LoadCertificate()
        {
            return new X509Certificate2(
                string.Format(@"{0}\bin\Config\MixItTest.pfx", AppDomai
        }
    }
```

Step 6 : add configuration in the *Web.config* file

- add the RAMMFAR

```
<system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
```

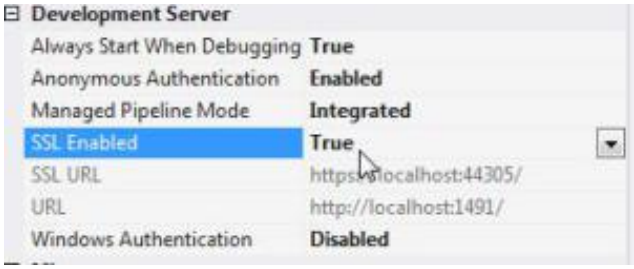
```
3     </system.webServer>
```

- add the connection string (for the membershipreboot repository)

```
1  <connectionStrings>
2     <add name="MyIdentityDb" connectionString="Data Source=(LocalDb)\v11.
3   </connectionStrings>
```

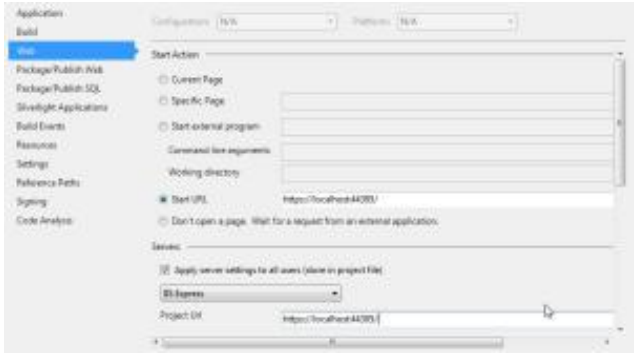
Step 7 : enable SSL

easy to do with visual studio in dev mode : set the property SSL Enabled to true:



(https://cedricdumont.files.wordpress.com/2014/12/1-4.jpg)

then copy the SSL URL to the clipboard and right click on the project to open the properties. Click the 'Web' Tab and copy the ssl url to Start Url and Project URL .



(https://cedricdumont.files.wordpress.com/2014/12/1-5.jpg)

Step 8 : Test it

At this point, you should have a running IdentityServer. Just run the project and check the following address (your port might be different than 44305:

https://localhost:44305/identity/.well-known/openid-configuration
(https://localhost:44305/identity/.well-known/openid-configuration)

the browser should display something like this:

(https://cedricdumont.files.wordpress.com/2014/12/1-6.jpg)

Step 9 : configure IdentityManager.

Identity Manager is a tool created by **BrockAllen (http://brockallen.com/)**to manage Identity (can be through MembershipReboot or Asp.net Identity). Here we deal with MembershipReboot. We will create a class to configure the IdentityManager (*MembershipRebootIdentityManagerFactory.cs*) then map it in the **Startup.cs** file. here is the class taken from the MembershipReboot samples:

```
namespace MixIt.Identity.Server.Config
{
    public class MembershipRebootIdentityManagerFactory
    {
        static MembershipRebootConfiguration<RelationalUserAccount> con
        static MembershipRebootIdentityManagerFactory()
        {
            System.Data.Entity.Database.SetInitializer(new System.Data.

            config = new MembershipRebootConfiguration<RelationalUserAc
            config.PasswordHashingIterationCount = 5000;
            config.RequireAccountVerification = false;
        }

        string connString;
        public MembershipRebootIdentityManagerFactory(string connString
        {
            this.connString = connString;
        }

        public IIdentityManagerService Create()
        {
            var db = new DefaultMembershipRebootDatabase(this.connStrin
            var userrepo = new DefaultUserAccountRepository(db);
            var usersvc = new UserAccountService<RelationalUserAccount>

            var grprepo = new DefaultGroupRepository(db);
            var grpsvc = new GroupService<RelationalGroup>(config.Defau

            var svc = new MembershipRebootIdentityManagerService<Relati
            return new DisposableIdentityManagerService(svc, db);
        }
    }
}
```

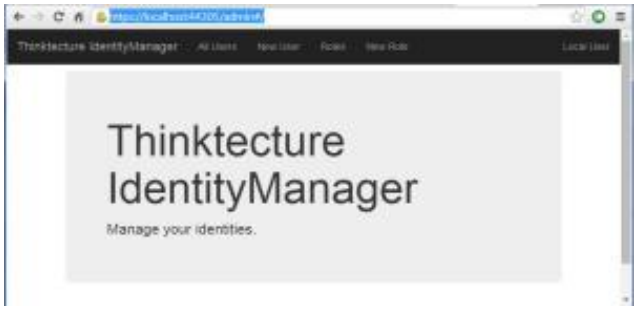
In the Startup.cs class add the following mapping

```
...
  app.Map("/admin", adminApp =>
            {
                var factory = new MembershipRebootIdentityManagerFactor
                adminApp.UseIdentityManager(new Thinktecture.IdentityMa
                {
                    IdentityManagerFactory = factory.Create
                });
            });

...
```

"MyIdentityDb" is the connection string that you configured in the *Web.config*.

To test it go to the following address: **https://localhost:44305/admin#/ (https://localhost:44305/admin#/)**
you should see something like this:



(https://cedricdumont.files.wordpress.com/2014/12/1-7.jpg)

Try it and create a user 'test' for example. we will use 'Fiddler' to get a token using post requests with the Client that we have configured.

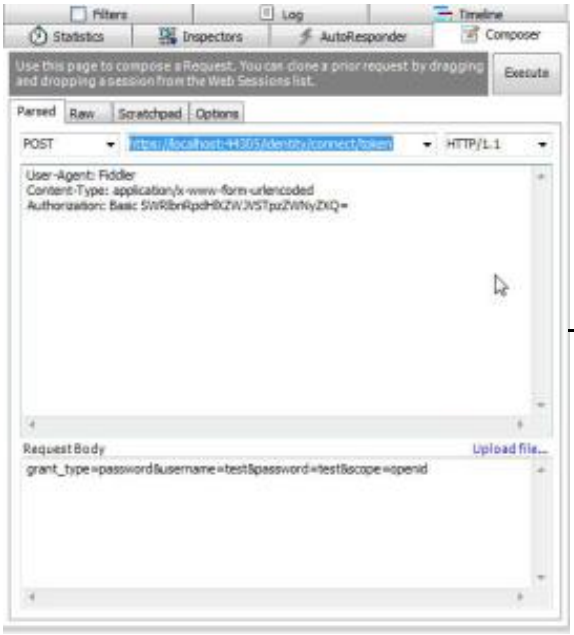
NB : the menus (All Users, Create new…) may take a moment to load.

Step 10 : Test IdentityServer with Fiddler

When you check the document received from **https://localhost:44305/identity/.well-known/openid-configuration (https://localhost:44305/identity/.well-known/openid-configuration)**, you can see that the property *token_endpoint* is set to : **https://localhost:44305/identity/connect/token (https://localhost:44305/identity/connect/token)**. We will use that endpoint to get a token. Therefor, just issue the following HTTP POST request to that endpoint:

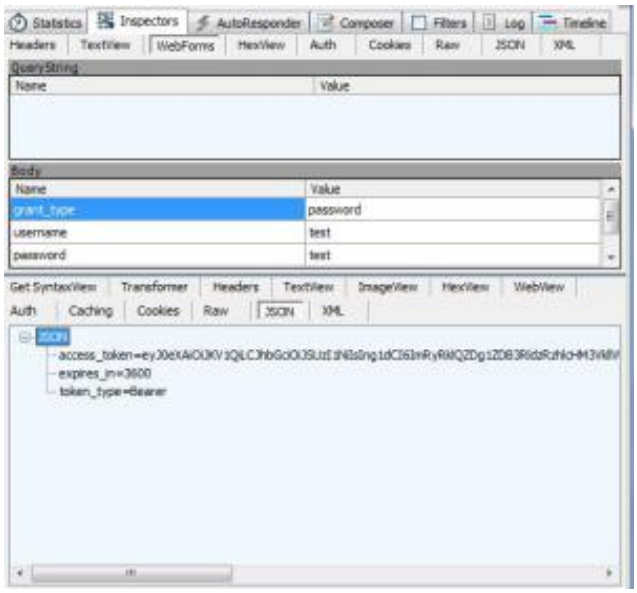
```
1  ------------------
2  Post :
3  https://localhost:44305/identity/connect/token (https://localhost:44305
4
5  Headers:
6  User-Agent: Fiddler
7  Content-Type: application/x-www-form-urlencoded
8  Authorization: Basic SWRlbnRpdHlXZWJVSTpzZWNyZXQ=
9
10 Body:
11 grant_type=password&username=test&password=tes5&scope=openid
12 ------------------
```

(the string SWRlbnRpdHlXZWJVSTpzZWNyZXQ= comes from the Base64 encoding of IdentityWebUI:secret which is our clientId and client secret located in our Clients.cs file. I use the Fiddler text wizard to do that.)



(https://cedricdumont.files.wordpress.com/2014/12/1-8.jpg)

If you have created a user with username = 'test' and password ='test', then the result will give you an access token:



(https://cedricdumont.files.wordpress.com/2014/12/1-9.jpg)

If you try with an incorrect password for example, you should receive the following:

{"error":"invalid_grant"}

That's it ! in the next article, we will create the webapi that will be used to create a user using membershipreboot

As Usual, comments are welcome and even remarks to update this little documentation.

10 thoughts on "IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 1"

1. Pingback: **IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! | {"@id":"cedric-dumont.com"}**
2. **ericswann** says:
   February 6, 2015 at 15:23
   Thanks for the demo, seems like this has fallen behind the MembershipReboot setup with IdentityServer. I used the following sample project to get things working:
   https://github.com/IdentityServer/Thinktecture.IdentityServer3.Samples/tree/master/source/MembershipReboot/SelfHost

   REPLY
3. **cedricdumontc** says:
   February 6, 2015 at 19:11
   yes i have to update my code to use the new released version. i ve been following thinktecture identity server for month now and wanted to check if i could use it in my own use case. response is yes of course, but i have not much time to update

   REPLY
   1. **cbmdk** says:
      March 24, 2015 at 10:18
      Hey Cedric,
      Do you have any plans on updating your solution to match the recent MembershipReboot changes.

      Awesome articles, helped me a lot on getting things setup!!!

      REPLY
      1. **cedricdumontc** says:
         March 25, 2015 at 10:16
         Hi,
         yes I will update it, but I am currently working on the angular part to have a service that will ease using openid (some patterns). =>
         https://github.com/CedricDumont/angular-toolkit
4. **Srinivas** says:
   April 16, 2015 at 21:00
   Hello Cedric,

   Your artcile helping several people. Yours is the best among all other whatever I came accross.

   Could you please update your solution on GitHub with new packages?

There are two errors with new packages.
1) Not finding IdentityManagerConfiguration in Startup.cs
public void Configuration(IAppBuilder app)
{
app.Map("/admin", adminApp =>
{
var factory = new MembershipRebootIdentityManagerFactory("MyIdentityDb");
adminApp.UseIdentityManager(new IdentityManagerConfiguration()
{
IdentityManagerFactory = factory.Create
});
});
2) Not finding DisposableIdentityManagerService in
MembershipRebootIdentityManagerFactory
return new DisposableIdentityManagerService(svc, db);

I tried to remove these errors but could not succeeded.

Thanks for all your hardwork.

REPLY

5.

skorlipara says:
April 16, 2015 at 21:00
Hello Cedric,

Your artcile helping several people. Yours is the best among all other whatever I came accross.

Could you please update your solution on GitHub with new packages?

There are two errors with new packages.
1) Not finding IdentityManagerConfiguration in Startup.cs
public void Configuration(IAppBuilder app)
{
app.Map("/admin", adminApp =>
{
var factory = new MembershipRebootIdentityManagerFactory("MyIdentityDb");
adminApp.UseIdentityManager(new IdentityManagerConfiguration()
{
IdentityManagerFactory = factory.Create
});
});
2) Not finding DisposableIdentityManagerService in
MembershipRebootIdentityManagerFactory
return new DisposableIdentityManagerService(svc, db);

I tried to remove these errors but could not succeeded.

Thanks for all your hardwork.

REPLY

6.

cedricdumontc says:
April 17, 2015 at 06:14
Hi, in fact I will update it with some changes : Owin to asp.net 5 (or vnext in Vs 2015) and angular to aurelia…. but I have a huge heap of work and I started a new own project where I will use Idsrv 3 and aurelia (aurelia.io) … so it's taking me a lot of time. But perhaps it would be better to just update the code for now (with a new angular-toolkit.js file I've created to get openid (idsrv3, salesforce, google…) support and add changes later on… I will try to do it this weekend

REPLY

7.

Michal says:
April 17, 2015 at 21:33
Now it's required for the secret to be hashed. You can simply change "secret" to "secret".Sha256()

REPLY

8.

cedricdumontc says:
April 18, 2015 at 09:28
FYI : hi, here there is a repo with idsrv on asp.net 5 (vnext) : **https://github.com/CedricDumont/vnext-playground/tree/master/idsrv3-vnext/idsrv3**.

REPLY