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

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

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

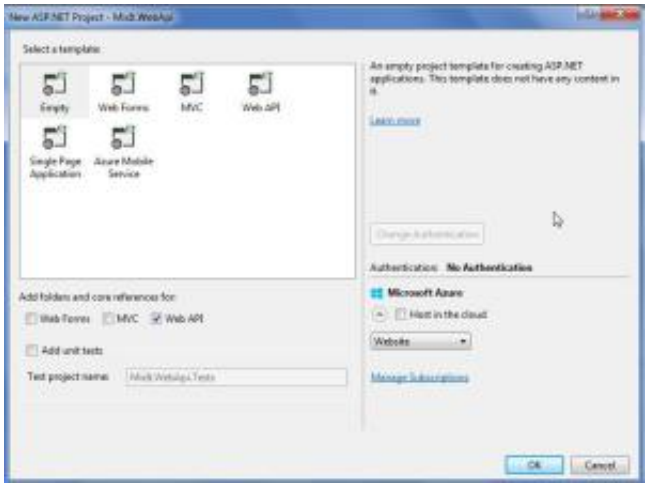
Introduction

In this part we will create and configure the web api that will create User accounts in the database. This api will be used by our angularJs based Web UI application to create users. Authentication is left to the Identity server. In fact, we use the facilities of membershipReboot to create User accounts.

full ongoing code (https://github.com/CedricDumont/Mix-It)

Step 1 : Create the project

In visual studi create an empty web project and select webapi.



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

Step 2 : Add the nuget packages

For membershipreboot

```
1 Install-Package BrockAllen.MembershipReboot.WebHost
```

```
Install-Package BrockAllen.MembershipReboot.Ef
```

This will also install BrockAllen.MembershipReboot

For Ninject

```
Install-Package ninject.web.common.webhost
```

This will also install ninject.web.common

Step 3: Create the Models

In our example, we will add some custom properties to the user account. these will be the First and Last Name. Therefor, we need to create our own custom UserAccount class and we'll add these properties to our Model.
Create a UserAccount class and add properties (These added properties MUST be virtual)

```
public class UserAccount : RelationalUserAccount
{
    public virtual string FirstName { get; set; }
    public virtual string LastName { get; set; }
}
```

create the RegisterInputModel class that will be used to transport the data from the UI to the web api.

```
public class RegisterInputModel
{
    public string Username { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public string ConfirmPassword { get; set; }
}
```

Step 4 : configure MR Repository

in the Web.config file, add the connection string to your membershipReboot DB. In our case it is "MyIdentityDb". If the database is not created, it well be created at runtime.

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

add the MR DbContext classes

```
namespace MixIt.WebApi.Customization
{

    public class CustomDb : MembershipRebootDbContext<UserAccount>
    {
        public CustomDb()
            : base("MyIdentityDb")
        {
```

```
        }
    }

    public class CustomUserAccountRepository : DbContextUserAccountRep
    {
        public CustomUserAccountRepository(CustomDb db)
            : base(db)
        {
        }
    }
}
```

also add a class to help init MembershipReboot in Ninject

```
public class MembershipRebootConfig
  {
      public static MembershipRebootConfiguration<CustomUserAccount> C
      {
          var settings = SecuritySettings.Instance;

          var config = new MembershipRebootConfiguration<CustomUserAcc

          return config;
      }
  }
```

In Web.config, you must also add a section : membershipreboot if you want to configure it there.

```
<section name="membershipReboot" type="BrockAllen.MembershipReboot.Secu
...
<membershipReboot requireAccountVerification="false"
                  emailIsUsername="false"
                  multiTenant="false"
                  passwordHashingIterationCount="0"
                  accountLockoutDuration="00:01:00"
                  passwordResetFrequency="0" />
```

in NinjectWebCommon#RegisterService() method add the following to DI MemberShipReboot:

```
var config = MembershipRebootConfig.Create();
kernel.Bind<UserAccountService<CustomUserAccount>>().ToSelf();
kernel.Bind<MembershipRebootConfiguration<CustomUserAccount>>().ToConst
kernel.Bind<IUserAccountRepository<CustomUserAccount>>().To<CustomUserA
kernel.Bind<CustomDb>().ToSelf().InRequestScope();
```

Step 5 : make Ninject work

For Ninject to work, I had to aadd a NinjectDependencyResolver : I followed more or less this article **http://www.peterprovost.org/blog/2012/06/19/adding-ninject-to-web-api/ (http://www.peterprovost.org/blog/2012/06/19/adding-ninject-to-web-api/)**

So I added a **NinjectDependencyResolver**.cs and **NinjectDependencyScope**.cs and configured them in **ninjectWebCommon** by adding the following line:

```
GlobalConfiguration.Configuration.DependencyResolver = new NinjectDepen
```

Step 6 : create the controller

```
public class RegisterController : ApiController
   {
       UserAccountService<CustomUserAccount> userAccountService;

       public RegisterController(UserAccountService<CustomUserAccount>
       {
           this.userAccountService = userAccountService;
       }

       [HttpPost]
       [Route("api/account/create")]
       public IHttpActionResult RegisterAccount(RegisterInputModel mod
       {
           try
           {
               var account = this.userAccountService.CreateAccount(mod
           }
           catch (ValidationException vex)
           {
               throw vex;
           }

           return Ok();
       }
   }
```

The controller has one single method for now that registers an account in the repository. We keep things simple here. Here, more checks should be added and better exception handling should be coded. We will improve it later on.

Step 7 : Test it with fiddler

Start the project, and using Fiddler, issue the following Post request :

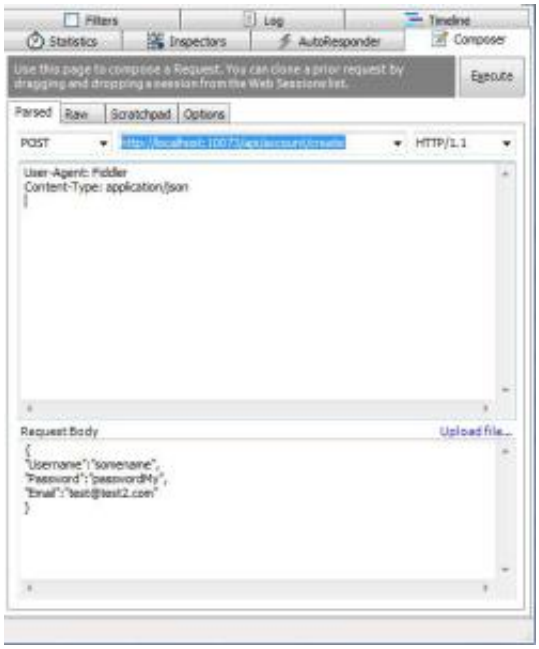
```
Post:
http://localhost:10073/api/account/create (http://localhost:10073/api/

Headers:
User-Agent: Fiddler
Content-Type: application/json

Body:
{
"Username":"somename",
"Password":"passwordMy",
```

```
12  "Email":"test@test2.com"
13  }
```

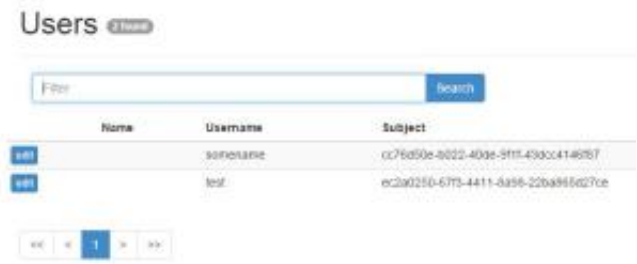
Illustration:



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

If everything went well, you should receive a "200 OK" response.

You can also start the project we created in **Part 1 (IdentityServer config) (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/)** to check if the account is created.



(https://cedricdumont.files.wordpress.com/2014/12/2-3.jpg)

One thought on "IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 2"

1. Pingback: **IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! | {"@id":"cedric-dumont.com"}**