عنوان: **راهنمای تغییر بخش احراز هویت و اعتبارسنجی کاربران سیستم مدیریت محتوای IRIS به ASP.NET Identity – بخش دوم**
نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۴/۰۷/۲۷ ۱۰:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, Security, ASP.NET Identity, IrisCMS

در بخش اول ، کارهایی که انجام دادیم به طور خلاصه عبارت بودند از:
-1 حذف کاربرانی که نام کاربری و ایمیل تکراری داشتند
-2 تغییر نام فیلد Password به PasswordHash در جدول User
سیستم مدیریت محتوای IRIS، برای استفاده از Entity Framework، از الگوی واحد کار (Unit Of Work) و تزریق وابستگی استفاده کرده است و اگر با نحوه‌ی پیاده سازی این الگوها آشنا نیستید، خواندن مقاله EF Code First #12 را به شما توصیه می‌کنم.
برای استفاده از ASP.NET Identity نیز باید از الگوی واحد کار استفاده کرد و برای این کار، ما از مقاله اعمال تزریق وابستگی‌ها به مثال رسمی ASP.NET Identity استفاده خواهیم کرد. **نکته مهم:** در ادامه اساس کار ما بر پایه‌ی مقاله اعمال تزریق وابستگی‌ها به مثال رسمی ASP.NET Identity است و چیزی که بیشتر برای ما اهمیت دارد کدهای نهایی آن هست؛ پس حتما به مخزن کد آن مراجعه کرده و کدهای آن را دریافت کنید. **تغییر نام کلاس User به ApplicationUser**
اگر به کدهای مثال رسمی ASP.NET Identity نگاهی بیندازید، می‌بینید که کلاس مربوط به جدول کاربران نام ApplicationUser دارد، ولی در سیستم IRIS نام آن User است. بهتر است که ما هم نام کلاس خود را از User به ApplicationUser تغییر دهیم چرا که مزایای زیر را به دنبال دارد:
-1 به راحتی می‌توان کدهای مورد نیاز را از مثال Identity کپی کرد.
-2 در سیستم Iris، بین کلاس User متعلق به پروژه خودمان و User مربوط به HttpContext تداخل رخ می‌داد که با تغییر نام کلاس User دیگر این مشکل را نخواهیم داشت.
برای این کار وارد پروژه Iris.DomainClasses شده و نام کلاس User را به ApplicationUser تغییر دهید. دقت کنید که این تغییر نام را از طریق Solution Explorer انجام دهید و نه از طریق کدهای آن. پس از این تغییر ویژوال استودیو می‌پرسد که آیا نام این کلاس را هم در کل پروژه تغییر دهد که شما آن را تایید کنید.
برای آن که نام جدول Users در دیتابیس تغییری نکند، وارد پوشه‌ی Entity Configuration شده و کلاس UserConfig را گشوده و در سازنده‌ی آن کد زیر را اضافه کنید:

```
ToTable("Users");
```

نصب ASP.NET Identity

برای نصب ASP.NET Identity دستور زیر را در کنسول Nuget وارد کنید:

```
Get-Project Iris.DomainClasses, Iris.Datalayer, Iris.Servicelayer, Iris.Web | Install-Package
Microsoft.AspNet.Identity.EntityFramework
```

از پروژه AspNetIdentityDependencyInjectionSample.DomainClasses کلاس‌های CustomUserRole، CustomUserLogin، CustomUserClaim و CustomRole را به پروژه Iris.DomainClasses منتقل کنید. تنها تغییری که در این کلاس‌ها باید انجام دهید، اصلاح namespace آنهاست.
همچنین بهتر است که به کلاس CustomRole، یک property به نام Description اضافه کنید تا توضیحات فارسی نقش مورد نظر را هم بتوان ذخیره کرد:

```
public class CustomRole : IdentityRole<int, CustomUserRole>
    {
        public CustomRole() { }
        public CustomRole(string name) { Name = name; }

        public string Description { get; set; }

    }
```

نکته: پیشنهاد می‌کنم که اگر می‌خواهید مثلا نام CustomRole را به IrisRole تغییر دهید، این کار را از طریق find and replace انجام ندهید. با همین نام‌های پیش فرض کار را تکمیل کنید و سپس از طریق خود ویژوال استودیو نام کلاس را تغییر دهید تا ویژوال استودیو به نحو بهتری این نام‌ها را در سرتاسر پروژه تغییر دهد.

سپس کلاس ApplicationUser پروژه IRIS را باز کرده و تعریف آن را به شکل زیر تغییر دهید:

```
public class ApplicationUser : IdentityUser<int, CustomUserLogin, CustomUserRole, CustomUserClaim>
```

اکنون می‌توانید propertyهای Id، UserName، PasswordHash و Email را حذف کنید؛ چرا که در کلاس پایه IdentityUser تعریف شده اند.

تغییرات DataLayer

وارد Iris.DataLayer شده و کلاس IrisDbContext را به شکل زیر ویرایش کنید:

```
public class IrisDbContext : IdentityDbContext<ApplicationUser, CustomRole, int, CustomUserLogin,
CustomUserRole, CustomUserClaim>,
        IUnitOfWork
```

اکنون می‌توانید property زیر را نیز حذف کنید چرا که در کلاس پایه تعریف شده است:

```
public DbSet<ApplicationUser> Users { get; set; }
```

نکته مهم: حتما برای کلاس IrisDbContext سازنده ای تعریف کنید که صراحتا نام رشته اتصالی را ذکر کرده باشد، اگر این کار را انجام ندهید با خطاهای عجیب غریبی روبرو می‌شوید.

```
public IrisDbContext()
            : base("IrisDbContext")
        {
        }
```

همچنین درون متد OnModelCreating کدهای زیر را پس از فراخوانی متد base.OnModelCreating(modelBuilder) جهت تعیین نام جداول دیتابیس بنویسید:

```
modelBuilder.Entity<CustomRole>().ToTable("AspRoles");
            modelBuilder.Entity<CustomUserClaim>().ToTable("UserClaims");
            modelBuilder.Entity<CustomUserRole>().ToTable("UserRoles");
            modelBuilder.Entity<CustomUserLogin>().ToTable("UserLogins");
```

از این جهت نام جدول CustomRole را در دیتابیس AspRoles انتخاب کردم تا با نام جدول Roles نقش‌های کنونی سیستم Iris تداخلی پیش نیاید. اکنون دستور زیر را در کنسول Nuget وارد کنید تا کدهای مورد نیاز برای مهاجرت تولید شوند:

```
Add-Migration UpdateDatabaseToAspIdentity
```

```
public partial class UpdateDatabaseToAspIdentity : DbMigration
{
        public override void Up()
        {
            CreateTable(
                "dbo.UserClaims",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        UserId = c.Int(nullable: false),
                        ClaimType = c.String(),
                        ClaimValue = c.String(),
                        ApplicationUser_Id = c.Int(),
                    })
                .PrimaryKey(t => t.Id)
                .ForeignKey("dbo.Users", t => t.ApplicationUser_Id)
```

```
                .Index(t => t.ApplicationUser_Id);

            CreateTable(
                "dbo.UserLogins",
                c => new
                    {
                        LoginProvider = c.String(nullable: false, maxLength: 128),
                        ProviderKey = c.String(nullable: false, maxLength: 128),
                        UserId = c.Int(nullable: false),
                        ApplicationUser_Id = c.Int(),
                    })
                .PrimaryKey(t => new { t.LoginProvider, t.ProviderKey, t.UserId })
                .ForeignKey("dbo.Users", t => t.ApplicationUser_Id)
                .Index(t => t.ApplicationUser_Id);

            CreateTable(
                "dbo.UserRoles",
                c => new
                    {
                        UserId = c.Int(nullable: false),
                        RoleId = c.Int(nullable: false),
                        ApplicationUser_Id = c.Int(),
                    })
                .PrimaryKey(t => new { t.UserId, t.RoleId })
                .ForeignKey("dbo.Users", t => t.ApplicationUser_Id)
                .ForeignKey("dbo.AspRoles", t => t.RoleId, cascadeDelete: true)
                .Index(t => t.RoleId)
                .Index(t => t.ApplicationUser_Id);

            CreateTable(
                "dbo.AspRoles",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Description = c.String(),
                        Name = c.String(nullable: false, maxLength: 256),
                    })
                .PrimaryKey(t => t.Id)
                .Index(t => t.Name, unique: true, name: "RoleNameIndex");

            AddColumn("dbo.Users", "EmailConfirmed", c => c.Boolean(nullable: false));
            AddColumn("dbo.Users", "SecurityStamp", c => c.String());
            AddColumn("dbo.Users", "PhoneNumber", c => c.String());
            AddColumn("dbo.Users", "PhoneNumberConfirmed", c => c.Boolean(nullable: false));
            AddColumn("dbo.Users", "TwoFactorEnabled", c => c.Boolean(nullable: false));
            AddColumn("dbo.Users", "LockoutEndDateUtc", c => c.DateTime());
            AddColumn("dbo.Users", "LockoutEnabled", c => c.Boolean(nullable: false));
            AddColumn("dbo.Users", "AccessFailedCount", c => c.Int(nullable: false));
        }

        public override void Down()
        {
            DropForeignKey("dbo.UserRoles", "RoleId", "dbo.AspRoles");
            DropForeignKey("dbo.UserRoles", "ApplicationUser_Id", "dbo.Users");
            DropForeignKey("dbo.UserLogins", "ApplicationUser_Id", "dbo.Users");
            DropForeignKey("dbo.UserClaims", "ApplicationUser_Id", "dbo.Users");
            DropIndex("dbo.AspRoles", "RoleNameIndex");
            DropIndex("dbo.UserRoles", new[] { "ApplicationUser_Id" });
            DropIndex("dbo.UserRoles", new[] { "RoleId" });
            DropIndex("dbo.UserLogins", new[] { "ApplicationUser_Id" });
            DropIndex("dbo.UserClaims", new[] { "ApplicationUser_Id" });
            DropColumn("dbo.Users", "AccessFailedCount");
            DropColumn("dbo.Users", "LockoutEnabled");
            DropColumn("dbo.Users", "LockoutEndDateUtc");
            DropColumn("dbo.Users", "TwoFactorEnabled");
            DropColumn("dbo.Users", "PhoneNumberConfirmed");
            DropColumn("dbo.Users", "PhoneNumber");
            DropColumn("dbo.Users", "SecurityStamp");
            DropColumn("dbo.Users", "EmailConfirmed");
            DropTable("dbo.AspRoles");
            DropTable("dbo.UserRoles");
            DropTable("dbo.UserLogins");
            DropTable("dbo.UserClaims");
        }
}
```

بهتر است که در کدهای تولیدی فوق، اندکی متد Up را با کد زیر تغییر دهید:

```
AddColumn("dbo.Users", "EmailConfirmed", c => c.Boolean(nullable: false, defaultValue:true));
```

چون در سیستم جدید احتیاج به تایید ایمیل به هنگام ثبت نام است، بهتر است که ایمیل‌های قبلی موجود در سیستم نیز به طور پیش فرض تایید شده باشند.
در نهایت برای اعمال تغییرات بر روی دیتابیس دستور زیر را در کنسول Nuget وارد کنید:

```
Update-Database
```

تغییرات ServiceLayer

ابتدا دستور زیر را در کنسول Nuget وارد کنید:

```
Get-Project Iris.Servicelayer, Iris.Web | Install-Package Microsoft.AspNet.Identity.Owin
```

سپس از فولدر Contracts پروژه AspNetIdentityDependencyInjectionSample.ServiceLayer فایل‌های
IApplicationRoleManager، IApplicationSignInManager، IApplicationUserManager، ICustomRoleStore و
ICustomUserStore را در فولدر Interfaces پروژه Iris.ServiceLayer کپی کنید. تنها کاری هم که نیاز هست انجام بدهید اصلاح namespace هاست.
باز از پروژه AspNetIdentityDependencyInjectionSample.ServiceLayer کلاس‌های ،ApplicationRoleManager
SmsService و ApplicationSignInManager، ApplicationUserManager، CustomRoleStore، CustomUserStore، EmailService
را به پوشه EFServcies پروژه‌ی Iris.ServiceLayer کپی کنید.
نکته: پیشنهاد می‌کنم که EmailService را به IdentityEmailService تغییر نام دهید چرا که در حال حاضر سیستم Iris دارای کلاسی به نامی EmailService هست.

تنظیمات StructureMap برای تزریق وابستگی ها

پروژه Iris.Web را باز کرده، به فولدر DependencyResolution بروید و به کلاس IoC کدهای زیر را اضافه کنید:

```
x.For<IIdentity>().Use(() => (HttpContext.Current != null && HttpContext.Current.User != null) ?
HttpContext.Current.User.Identity : null);

                x.For<IUnitOfWork>()
                    .HybridHttpOrThreadLocalScoped()
                    .Use<IrisDbContext>();

                x.For<IrisDbContext>().HybridHttpOrThreadLocalScoped()
                   .Use(context => (IrisDbContext)context.GetInstance<IUnitOfWork>());
                x.For<DbContext>().HybridHttpOrThreadLocalScoped()
                   .Use(context => (IrisDbContext)context.GetInstance<IUnitOfWork>());

                x.For<IUserStore<ApplicationUser, int>>()
                    .HybridHttpOrThreadLocalScoped()
                    .Use<CustomUserStore>();

                x.For<IRoleStore<CustomRole, int>>()
                    .HybridHttpOrThreadLocalScoped()
                    .Use<RoleStore<CustomRole, int, CustomUserRole>>();

                x.For<IAuthenticationManager>()
                      .Use(() => HttpContext.Current.GetOwinContext().Authentication);

                x.For<IApplicationSignInManager>()
                      .HybridHttpOrThreadLocalScoped()
                      .Use<ApplicationSignInManager>();

                x.For<IApplicationRoleManager>()
                      .HybridHttpOrThreadLocalScoped()
                      .Use<ApplicationRoleManager>();

                // map same interface to different concrete classes
                x.For<IIdentityMessageService>().Use<SmsService>();
                x.For<IIdentityMessageService>().Use<IdentityEmailService>();

                x.For<IApplicationUserManager>().HybridHttpOrThreadLocalScoped()
                   .Use<ApplicationUserManager>()
                   .Ctor<IIdentityMessageService>("smsService").Is<SmsService>()
```

```
                    .Ctor<IIdentityMessageService>("emailService").Is<IdentityEmailService>()
                    .Setter<IIdentityMessageService>(userManager =>
userManager.SmsService).Is<SmsService>()
                    .Setter<IIdentityMessageService>(userManager =>
userManager.EmailService).Is<IdentityEmailService>();

                x.For<ApplicationUserManager>().HybridHttpOrThreadLocalScoped()
                    .Use(context =>
(ApplicationUserManager)context.GetInstance<IApplicationUserManager>());

                x.For<ICustomRoleStore>()
                      .HybridHttpOrThreadLocalScoped()
                      .Use<CustomRoleStore>();

                x.For<ICustomUserStore>()
                      .HybridHttpOrThreadLocalScoped()
                      .Use<CustomUserStore>();
```

اگر HttpContext.Current.GetOwinContext() شناسایی نمی‌شود دلیلش این است که متد GetOwinContext یک متد الحاقی است که برای استفاده از آن باید پکیج نیوگت زیر را نصب کنید:

```
Install-Package Microsoft.Owin.Host.SystemWeb
```

تغییرات Iris.Web

در ریشه پروژه‌ی Iris.Web یک کلاس به نام Startup بسازید و کدهای زیر را در آن بنویسید:

```
using System;
using Iris.Servicelayer.Interfaces;
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Microsoft.Owin.Security.DataProtection;
using Owin;
using StructureMap;

namespace Iris.Web
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            configureAuth(app);
        }

        private static void configureAuth(IAppBuilder app)
        {
            ObjectFactory.Container.Configure(config =>
            {
                config.For<IDataProtectionProvider>()
                      .HybridHttpOrThreadLocalScoped()
                      .Use(() => app.GetDataProtectionProvider());
            });

            //ObjectFactory.Container.GetInstance<IApplicationUserManager>().SeedDatabase();

            // Enable the application to use a cookie to store information for the signed in user
            // and to use a cookie to temporarily store information about a user logging in with a
third party login provider
            // Configure the sign in cookie
            app.UseCookieAuthentication(new CookieAuthenticationOptions
            {
                AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                LoginPath = new PathString("/Account/Login"),
                Provider = new CookieAuthenticationProvider
                {
                    // Enables the application to validate the security stamp when the user logs in.
                    // This is a security feature which is used when you change a password or add an
external login to your account.
                    OnValidateIdentity =
ObjectFactory.Container.GetInstance<IApplicationUserManager>().OnValidateIdentity()
                }
            });
            app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);
```

```
            // Enables the application to temporarily store user information when they are verifying
the second factor in the two-factor authentication process.
            app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie,
TimeSpan.FromMinutes(5));

            // Enables the application to remember the second login verification factor such as phone
or email.
            // Once you check this option, your second step of verification during the login process
will be remembered on the device where you logged in from.
            // This is similar to the RememberMe option when you log in.

app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes.TwoFactorRememberBrowserCookie);

            app.CreatePerOwinContext(
               () => ObjectFactory.Container.GetInstance<IApplicationUserManager>());

            // Uncomment the following lines to enable logging in with third party login providers
            //app.UseMicrosoftAccountAuthentication(
            //    clientId: "",
            //    clientSecret: "");

            //app.UseTwitterAuthentication(
            //   consumerKey: "",
            //   consumerSecret: "");

            //app.UseFacebookAuthentication(
            //   appId: "",
            //   appSecret: "");

            //app.UseGoogleAuthentication(
            //    clientId: "",
            //    clientSecret: "");

        }
    }
}
```

تا به این جای کار اگر پروژه را اجرا کنید نباید هیچ مشکلی مشاهده کنید. در بخش بعدی کدهای مربوط به کنترلرهای ورود، ثبت نام، فراموشی کلمه عبور و ... را با سیستم Identity پیاده سازی می‌کنیم.