عنوان: **راهنمای تغییر بخش احراز هویت و اعتبارسنجی کاربران سیستم مدیریت محتوای IRIS به ASP.NET Identity – بخش اول**
نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۴/۰۷/۱۷ ۱۷:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, Security, ASP.NET Identity, IrisCMS

سیستم مدیریت محتوای IRIS از سیستم‌های اعتبار سنجی و مدیریت کاربران رایج نظیر ASP.NET Membership و یا ASP.NET Simple Membership استفاده نمی‌کند و از یک سیستم احراز هویت سفارشی شده مبتنی بر FormsAuthentication بهره می‌برد.
زمانیکه در حال نوشتن پروژه‌ی IRIS بودم هنوز ASP.NET Identity معرفی نشده بود و به دلیل مشکلاتی که سیستم‌های قدیمی ذکر شده داشت، یک سیستم اعتبار سنجی کاربران سفارشی شده را در پروژه پیاده سازی کردم.
برای اینکه با معایب سیستم‌های مدیریت کاربران پیشین و مزایای ASP.NET Identity آشنا شوید، مقاله زیر می‌تواند شروع خیلی خوبی باشد:
معرفی ASP.NET Identity

به این نکته نیز اشاره کنم که هنوز هم می‌توان از FormsAuthentication مبتنی بر OWIN استفاده کرد؛ ولی چیزی که برای من بیشتر اهمیت دارد خود سیستم Identity هست، نه نحوه‌ی ورود و خروج به سایت و تولید کوکی اعتبارسنجی.
باید اعتراف کرد که سیستم جدید مدیریت کاربران مایکروسافت خیلی خوب طراحی شده است و اشکالات سیستم‌های پیشین خود را ندارد. به راحتی می‌توان آن را توسعه داد و یا قسمتی از آن را تغییر داد و به جرات می‌توان گفت که پایه و اساس هر سیستم اعتبار سنجی و مدیریت کاربری را که در نظر داشته باشید، به خوبی پیاده سازی کرده است.
در ادامه قصد دارم، چگونگی مهاجرت از سیستم فعلی به سیستم Identity را بدون از دست دادن اطلاعات فعلی شرح دهم.

**رفع باگ ثبت کاربرهای تکراری نسخه‌ی کنونی**

قبل از این که سراغ پیاده سازی Identity برویم، ابتدا باید یک باگ مهم را در نسخه‌ی قبلی، برطرف نماییم. نسخه‌ی کنونی مدیریت کاربران اجازه‌ی ثبت کاربر با ایمیل و یا نام کاربری تکراری را نمی‌دهد. جلوگیری از ثبت نام کاربر جدید با ایمیل یا نام کاربری تکراری از طریق کدهای زیر صورت گرفته است؛ اما در عمل، همیشه هم درست کار نمی‌کند.

```
public AddUserStatus Add(User user)
        {
            if (ExistsByEmail(user.Email))
                return AddUserStatus.EmailExist;
            if (ExistsByUserName(user.UserName))
                return AddUserStatus.UserNameExist;

            _users.Add(user);
            return AddUserStatus.AddingUserSuccessfully;
        }
```

شاید الان با خود بگویید که چرا برای فیلدهای UserName و Email ایندکس منحصر به فرد تعریف نشده است؟ دلیلش این بوده که در زمان نگارش پروژه، Entity Framework پشتیبانی پیش فرضی از تعریف ایندکس نداشت و نوشتن همین شرطها کافی به نظر می‌رسید. باز هم ممکن است بگویید که مسائل همزمانی چگونه مدیریت شده است و اگر دو کاربر مختلف در یک لحظه، نام کاربری یکسانی را انتخاب کنند، سیستم چگونه از ثبت دو کاربر مختلف با نام کاربری یکسان ممانعت می‌کند؟
جواب این است که ممانعتی نمی‌کند و دو کاربر با نام‌های کاربری یکسان ثبت می‌شوند؛ اما من برای وبسایت خودم که تعداد کاربرانش محدود بود این سناریو را محتمل نمی‌دانستم و کد خاصی برای جلوگیری از این اتفاق پیاده سازی نکرده بودم.
با این حال، در حال حاضر نزدیک به 20 کاربر تکراری در دیتابیسی که این سیستم استفاده می‌کند ثبت شده است. اما واقعا آیا دو کاربر مختلف اطلاعات یکسانی وارد کرده‌اند؟
دلیل رخ دادن این اتفاق این است که کاربری که در حال ثبت نام در سایت است، وقتی که بر روی دکمه‌ی ثبت نام کلیک می‌کند و اطلاعات به سرور ارسال می‌شوند، در سمت سرور بعد از رد شدن از شرطهای تکراری نبودن UserName و Email، قبل از رسیدن به متد SaveChanges برای ذخیره شدن اطلاعات کاربر جدید در دیتابیس، وقفه ای در ترد این درخواست به وجود می‌آید. کاربر که احساس می‌کند اتفاقی رخ نداده است، دوباره بر روی دکمه‌ی ثبت نام کلیک می‌کند و همان اطلاعات قبلی به سرور ارسال می‌شود و این درخواست نیز دوباره شرطهای تکراری نبودن اطلاعات را با موفقیت رد می‌کند( چون هنوز SaveChanges درخواست اول

فراخوانی نشده است) و این بار SaveChanges درخواست دوم با موفقیت فراخوانی می‌شود و کاربر ثبت می‌شود. در نهایت هم ترد درخواست اول به ادامه‌ی کار خود می‌پردازد و SaveChanges درخواست اول نیز فراخوانی می‌شود و خیلی راحت دو کاربر با اطلاعات یکسان ثبت می‌شود. این سناریو را در ویژوال استادیو با قرار دادن یک break point قبل از فراخوانی متد SaveChanges می‌توانید شبیه سازی کنید.

احتمالا این سناریو با مباحث همزمانی در سیستم عامل و context switchهای بین تردها مرتبط است و این context switchها بین درخواست‌ها و atomic نبودن روند چک کردن اطلاعات و ثبت آن ها، سبب بروز چنین مشکلی میشود.

برای رفع این مشکل می‌توان از غیر فعال کردن یک دکمه در حین انجام پردازش‌های سمت سرور استفاده کرد تا کاربر بی حوصله، نتواند چندین بار بر روی یک دکمه کلیک کند و یا راه حل اصولی‌تر این است که ایندکس منحصر به فرد برای فیلدهای مورد نظر تعریف کنیم.

به طور پیش فرض در ASP.NET Identity برای فیلدهای UserName و Email ایندکس منحصر به فرد تعریف شده است. اما مشکل این است که به دلیل وجود کاربرانی با Email و UserName تکراری در دیتابیس کنونی، امکان تعریف Index منحصر به فرد وجود ندارد و پیش از انجام هر کاری باید این ناهنجاری را در دیتابیس برطرف نماییم.

به شخصه معمولا برای انجام کارهایی از این دست، یک کنترلر در برنامه خود تعریف می‌کنم و در آنجا کارهای لازم را انجام می‌دهم.

در اینجا من برای حذف کاربران با اطلاعات تکراری، یک کنترلر به نام Migration و اکشن متدی به نام RemoveDuplicateUsers تدارک دیدم.

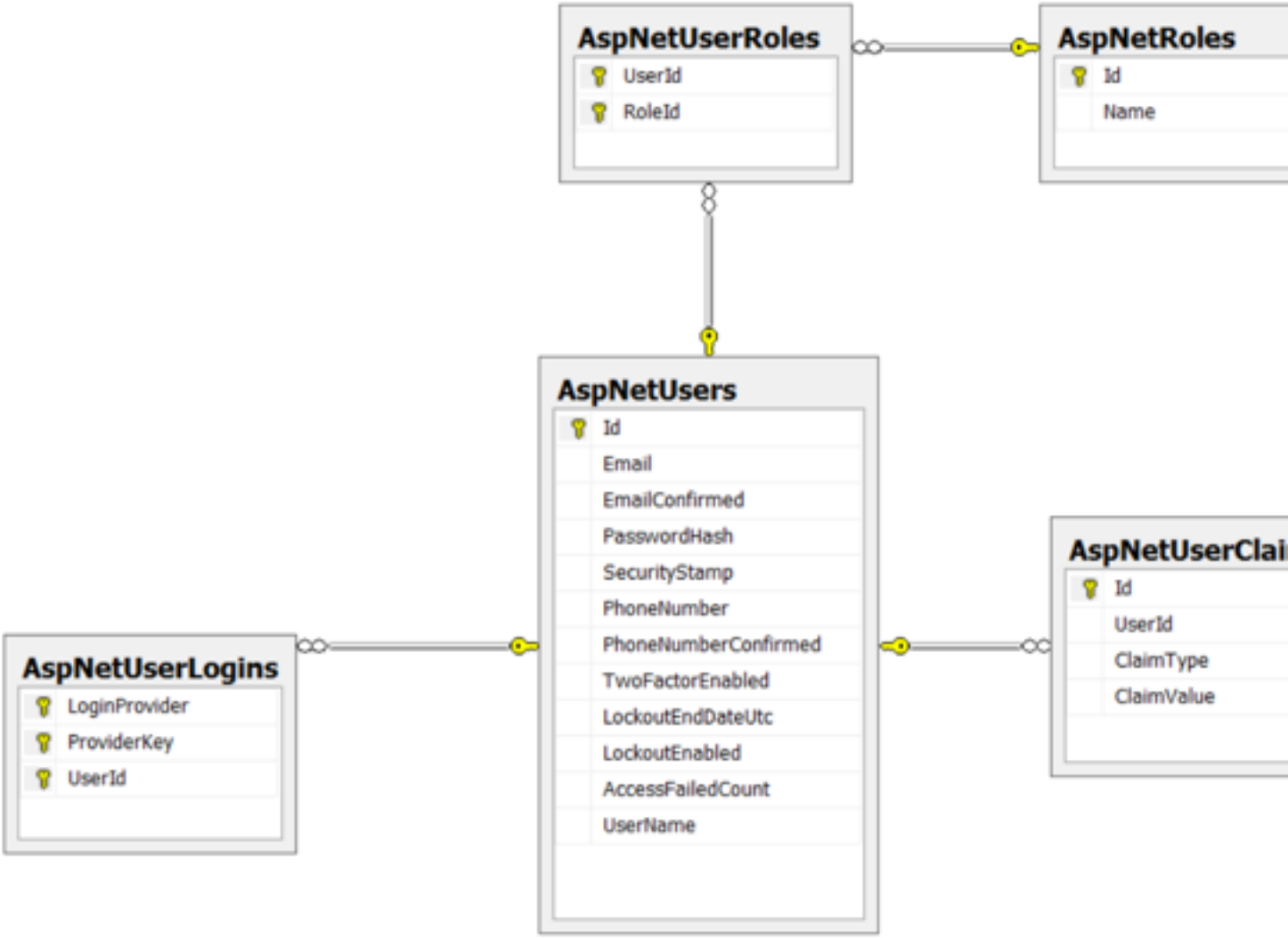
```
using System.Linq;
using System.Web.Mvc;
using Iris.Datalayer.Context;

namespace Iris.Web.Controllers
{
    public class MigrationController : Controller
    {
        public ActionResult RemoveDuplicateUsers()
        {
            var db = new IrisDbContext();

            var lstDuplicateUserGroup = db.Users
                                          .GroupBy(u => u.UserName)
                                          .Where(g => g.Count() > 1)
                                          .ToList();

            foreach (var duplicateUserGroup in lstDuplicateUserGroup)
            {
                foreach (var user in duplicateUserGroup.Skip(1).Where(user => user.UserMetaData !=
null))
                {
                    db.UserMetaDatas.Remove(user.UserMetaData);
                }

                db.Users.RemoveRange(duplicateUserGroup.Skip(1));
            }

            db.SaveChanges();

            return new EmptyResult();
        }
    }
}
```

در اینجا کاربران بر اساس نام کاربری گروه بندی می‌شوند و گروه‌هایی که بیش از یک عضو داشته باشند، یعنی کاربران آن گروه دارای نام کاربری یکسان هستند و به غیر از کاربر اول گروه، بقیه باید حذف شوند. البته این را متذکر شوم که منطق وبسایت من به این شکل بوده است و اگر منطق کدهای شما فرق می‌کند، مطابق با منطق خودتان این کدها را تغییر دهید.

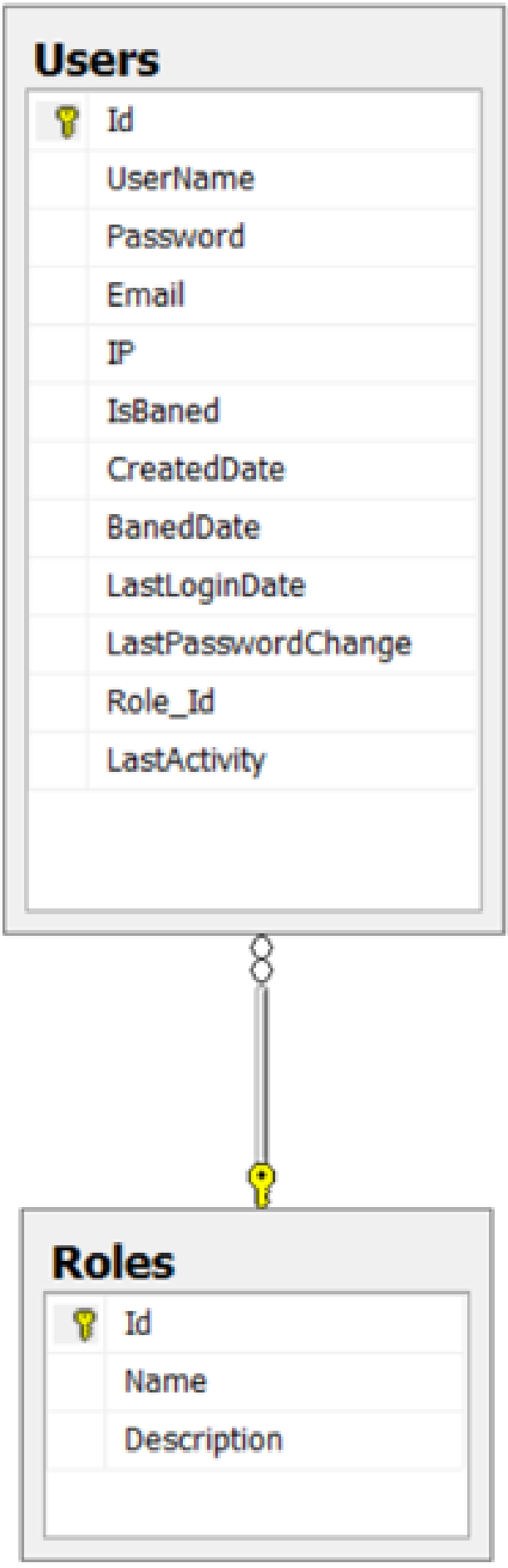
**تذکر:** اینجا شاید بگویید که چرا cascade delete برای UserMetaData فعال نیست و بخواهید که آن را اصلاح کنید. پیشنهاد من این است که اکنون از هدف اصلی منحرف نشوید و تمام تمرکز خود را بر روی انتقال به ASP Identity با حداقل تغییرات بگذارید. این گونه نباشد که در اواسط کار با خود بگویید که بد نیست حالا فلان کتابخانه را آپدیت کنم یا این تغییر را بدهم بهتر می‌شود. سعی کنید با حداقل تغییرات رو به جلو حرکت کنید؛ آپدیت کردن کتابخانه‌ها را هم بعدا می‌شود انجام داد.

**مقایسه ساختار جداول دیتابیس کاربران IRIS با ASP.NET Identity**

ساختار جداول ASP.NET Identity به شکل زیر است:




ساختار جداول سیستم کنونی هم بدین شکل است:


Users
Id
UserName
Password
Email
IP
IsBaned
CreatedDate
BanedDate
LastLoginDate
LastPasswordChange
Role_Id
LastActivity
Roles
Id
Name
Description

همان طور که مشخص است در هر دو سیستم، بین ساختار جداول و رابطه‌ی بین آن‌ها شباهت‌ها و تفاوت هایی وجود دارد. سیستم Identity دو جدول بیشتر از IRIS دارد و برای جداولی که در سیستم کنونی وجود ندارند نیاز به انجام کاری نیست و به هنگام پیاده سازی Identity، این جداول به صورت خودکار به دیتابیس اضافه خواهند شد.

دو جدول مشترک در این دو سیستم، جداول Users و Roles هستندکه نحوه‌ی ارتباطشان با یکدیگر متفاوت است. در Iris بین User و Role رابطه‌ی یک به چند وجود دارد ولی در Identity، رابطه‌ی بین این دو جدول چند به چند است و جدول واسط بین آن‌ها نیز UserRoles نام دارد.

از آن جایی که من قصد دارم در سیستم جدید هم رابطه‌ی بین کاربر و نقش چند به چند باشد، به پیش فرض‌های Identity کاری ندارم. به رابطه‌ی کنونی یک به چند کاربر و نقشش نیز دست نمی‌گذارم تا در انتها با یک کوئری از دیتابیس، اطلاعات نقش‌های کاربران را به جدول جدیدش منتقل کنم.

جدولی که در هر دو سیستم مشترک است و هسته‌ی آن‌ها را تشکیل می‌دهد، جدول Users است. اگر دقت کنید می‌بینید که این جدول در هر دو سیستم، دارای یک سری فیلد مشترک است که دقیقا هم نام هستند مثل Id، UserName و Email؛ پس این فیلدها از نظر کاربرد در هر دو سیستم یکسان هستند و مشکلی ایجاد نمی‌کنند.

یک سری فیلد هم در جدول User در سیستم IRIS هست که در Identity نیست و بلعکس. با این فیلدها نیز کاری نداریم چون در هر دو سیستم کار مخصوص به خود را انجام می‌دهند و تداخلی در کار یکدیگر ایجاد نمی‌کنند.

اما فیلدی که برای ذخیره سازی پسورد در هر دو سیستم استفاده می‌شود دارای نام‌های متفاوتی است. در Iris این فیلد Password نام دارد و در Identity نامش PasswordHash است.

برای اینکه در سیستم کنونی، نام فیلد Password جدول User را به PasswordHash تغییر دهیم قدم‌های زیر را بر می‌داریم:

وارد پروژه‌ی DomainClasses شده و کلاس User را باز کنید. سپس نام خاصیت Password را به PasswordHash تغییر دهید. پس از این تغییر بلافاصله یک گزینه زیر آن نمایان می‌شود که می‌خواهد در تمام جاهایی که از این نام استفاده شده است را به نام جدید تغییر دهد؛ آن را انتخاب کرده تا همه جا Password به PasswordHash تغییر کند.

برای این که این تغییر نام بر روی دیتابیس نیز اعمال شود باید از Migration استفاده کرد. در اینجا من از مهاجرت دستی که بر اساس کد هست استفاده می‌کنم تا هم بتوانم کدهای مهاجرت را پیش از اعمال بررسی و هم تاریخچه‌ای از تغییرات را ثبت کنم.

برای این کار، Package Manager Console را باز کرده و از نوار بالایی آن، پروژه پیش فرض را بر روی DataLayer قرار دهید. سپس در کنسول، دستور زیر را وارد کنید:

```
Add-Migration Rename_PasswordToPasswordHash_User
```

اگر وارد پوشه Migrations پروژه DataLayer خود شوید، باید کلاسی با نامی شبیه به
201510090808056_Rename_PasswordToPasswordHash_User بببینید. اگر آن را باز کنید کدهای زیر را خواهید دید:

```
public partial class Rename_PasswordToPasswordHash_User : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Users", "PasswordHash", c => c.String(nullable: false, maxLength: 200));
            DropColumn("dbo.Users", "Password");
        }

        public override void Down()
```

```
        {
            AddColumn("dbo.Users", "Password", c => c.String(nullable: false, maxLength: 200));
            DropColumn("dbo.Users", "PasswordHash");
        }
    }
```

بدیهی هست که این کدها عمل حذف ستون Password را انجام میدهند که سبب از دست رفتن اطلاعات می‌شود. کدهای فوق را به شکل زیر ویرایش کنید تا تنها سبب تغییر نام ستون Password به PasswordHash شود.

```
public partial class Rename_PasswordToPasswordHash_User : DbMigration
    {
        public override void Up()
        {
            RenameColumn("dbo.Users", "Password", "PasswordHash");
        }

        public override void Down()
        {
            RenameColumn("dbo.Users", "PasswordHash", "Password");
        }
    }
```

سپس باز در کنسول دستور Update-Database را وارد کنید تا تغییرات بر روی دیتابیس اعمال شود.

دلیل اینکه این قسمت را مفصل بیان کردم این بود که می‌خواستم در مهاجرت از سیستم اعتبارسنجی خودتان به ASP.NET Identity دید بهتری داشته باشید.

تا به این جای کار فقط پایگاه داده سیستم کنونی را برای مهاجرت آماده کردیم و هنوز ASP.NET Identity را وارد پروژه نکردیم. در بخش‌های بعدی Identity را نصب کرده و تغییرات لازم را هم انجام می‌دهیم.

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.