

عنوان: راهنمای تغییر بخش احراز هویت و اعتبارسنجی کاربران سیستم مدیریت محتوای IRIS به ASP.NET Identity - بخش اول

نویسنده: مهدی سعیدی فر

تاریخ: ۱۷:۳۰ ۱۳۹۴/۰۷/۱۷

آدرس: www.dotnettips.info

گروه‌ها: Entity framework, MVC, Security, ASP.NET Identity, IrisCMS

سیستم مدیریت محتوای IRIS از سیستم‌های اعتبارسنجی 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 اعتبارسنجی 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 میکروسافت 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 اعتبارسنجی 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 Email و UserName ایندکس منحصر به فرد تعریف نشده است؟ دلایل 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 دیتابیس 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، وقفه‌ای 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 switch های بین تردها مرتبط است و این context switch ها بین درخواست‌ها و atomic نبودن روند چک کردن اطلاعات و ثبت آن‌ها، 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();

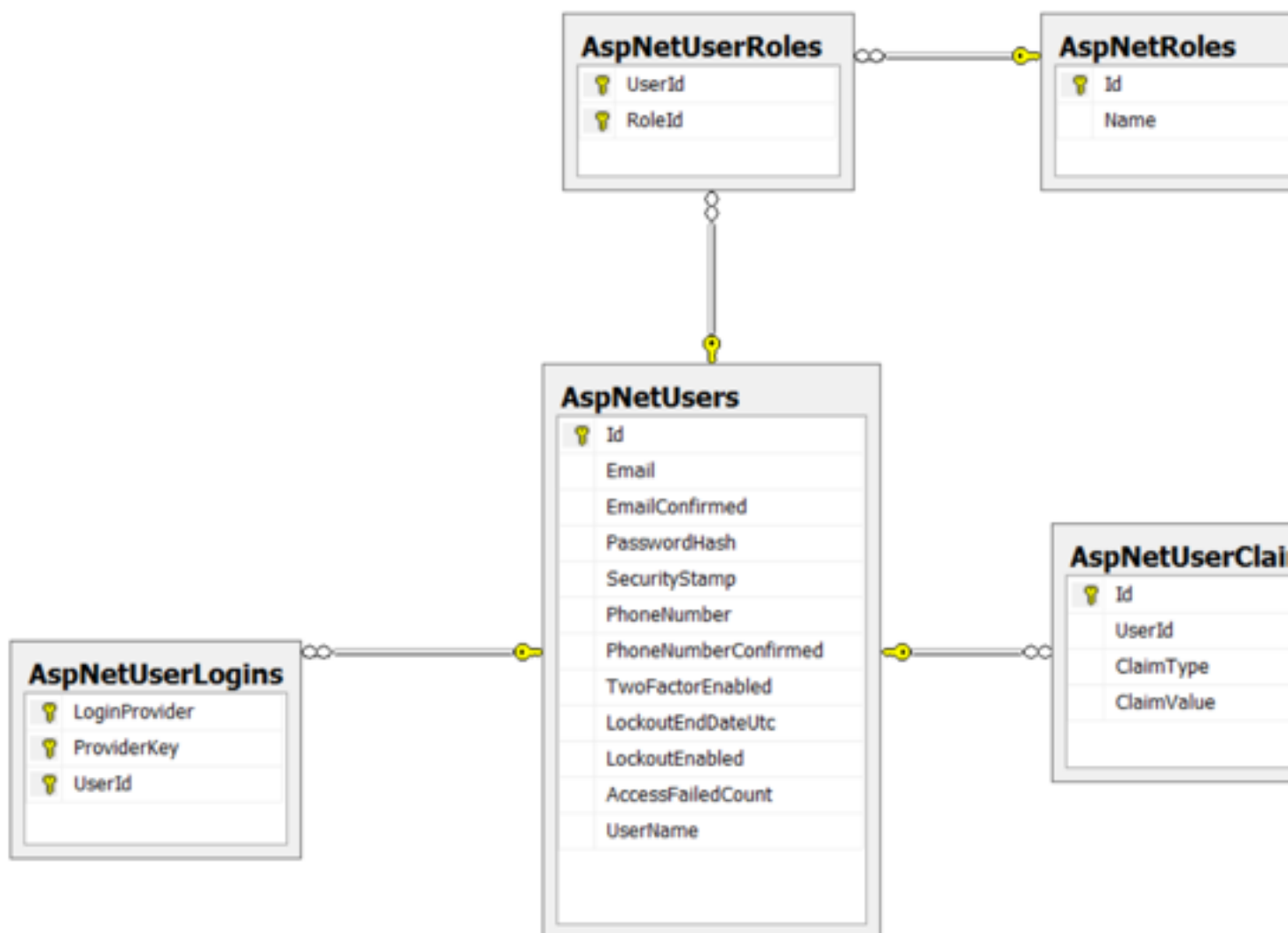
            return new EmptyResult();
        }
    }
}
```

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.

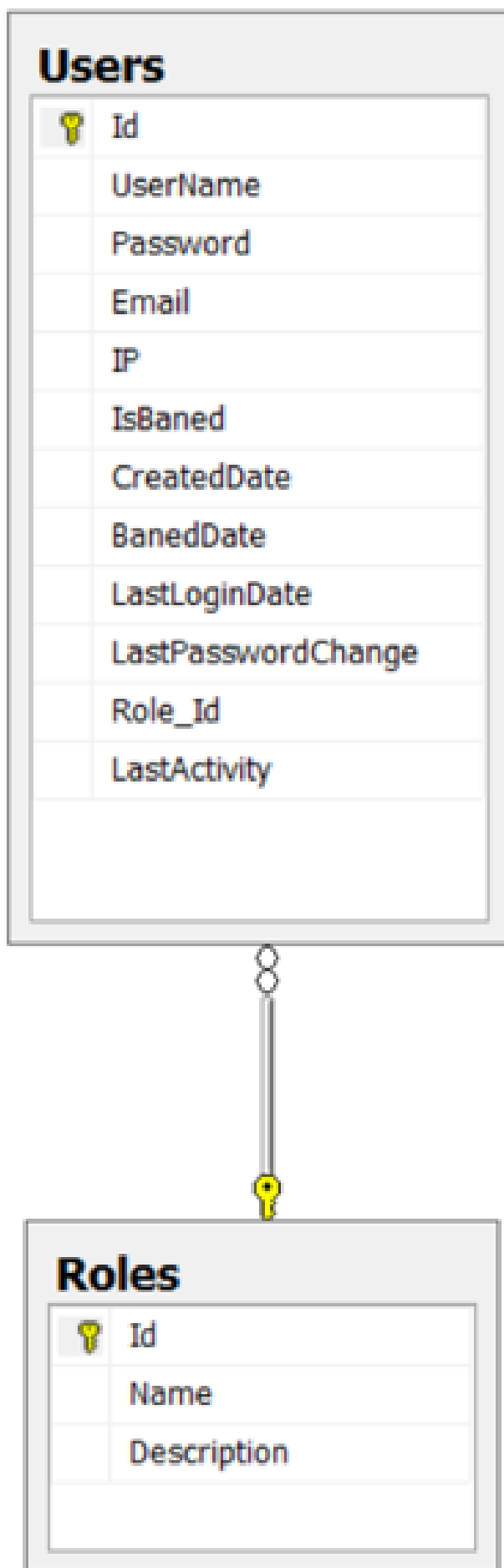
تذکر: اینجا شاید بگویید که چرا cascade delete برای UserMetaData فعال نیست و بخواهید که آن را اصلاح کنید. پیشنهاد من این است که اکنون از هدف اصلی منحرف نشوید و تمام تمرکز خود را بر روی انتقال به ASP Identity با حداقل تغییرات بگذارید. این گونه نباشد که در اواسط کار با خود بگویید که بد نیست حالا فلان کتابخانه 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:



همان طور که مشخص است در هر دو سیستم، بین ساختار جداول و رابطه‌ی بین آن‌ها شباهت‌ها و تفاوت‌هایی 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 هستند که 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 دقیقاً 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 Rename_PasswordToPasswordHash_User_201510090808056 ببینید. 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.