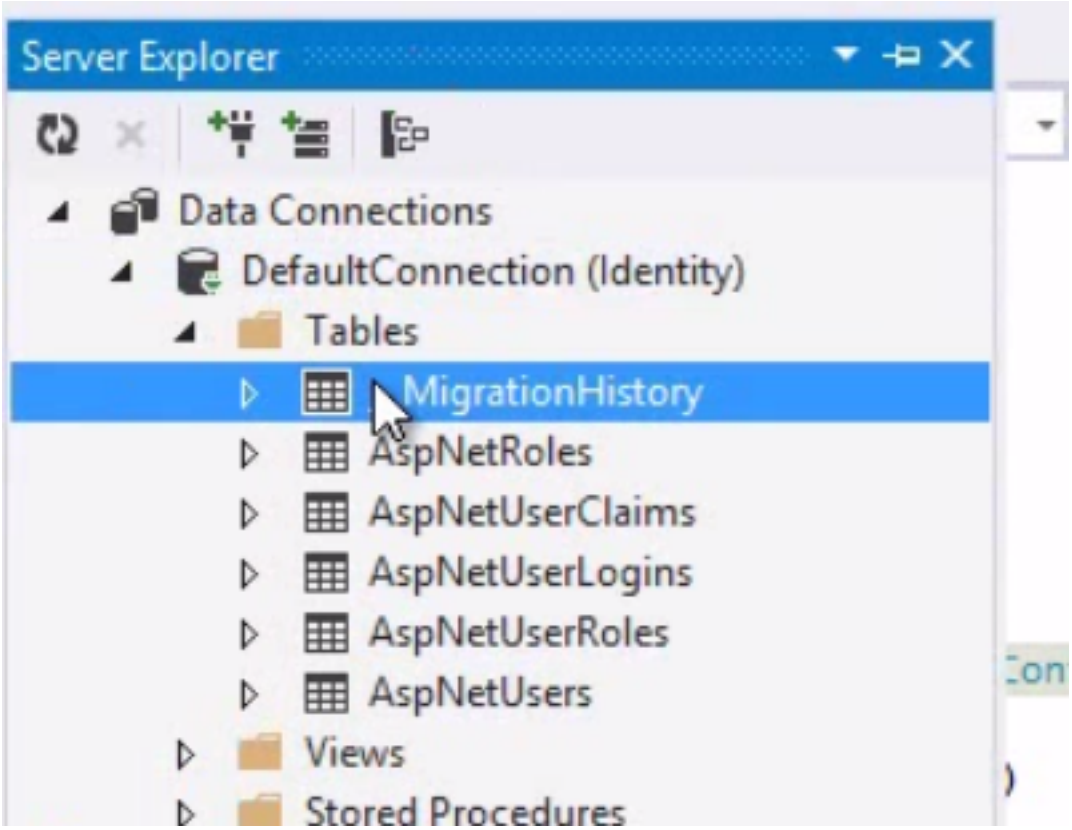
عنوان: **نگاهی به هویت سنجی کاربران در ASP.NET MVC 5**
نویسنده: مرتضی دلیل
تاریخ: ۱۳۹۳/۰۳/۲۲ ۱۱:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, Security, ASP.NET Identity

در مقاله پیش رو، سعی شده‌است به شکلی تقریبا عملی، کلیاتی در مورد Authentication در MVC5 توضیح داده شود. هدف روشن شدن ابهامات اولیه در هویت سنجی MVC5 و حل شدن مشکلات اولیه برای ایجاد یک پروژه است. در MVC 4 برای دسترسی به جداول مرتبط با اعتبار سنجی (مثلا لیست کاربران) مجبور به استفاده از متدهای از پیش تعریف شده‌ی رفرنس‌هایی که برای آن نوع اعتبار سنجی وجود داشت، بودیم. راه حلی نیز برای دسترسی بهتر وجود داشت و آن هم ساختن مدل‌های مشابه آن جدول‌ها و اضافه کردن چند خط کد به برنامه بود. با اینکار دسترسی ساده به Roles و Users برای تغییر و اضافه کردن محتوای آنها ممکن می‌شد. در لینک زیر توضیحاتی در مورد روش اینکار وجود دارد.
[ اینجا ]

در MVC5 داستان کمی فرق کرده است. برای درک موضوع پروژه ای بسازید و حالت پیش فرض آن را تغییر ندهید و آن را اجرا کنید و ثبت نام را انجام دهید، بلافاصله تصویر زیر در دیتابیس نمایان خواهد شد.

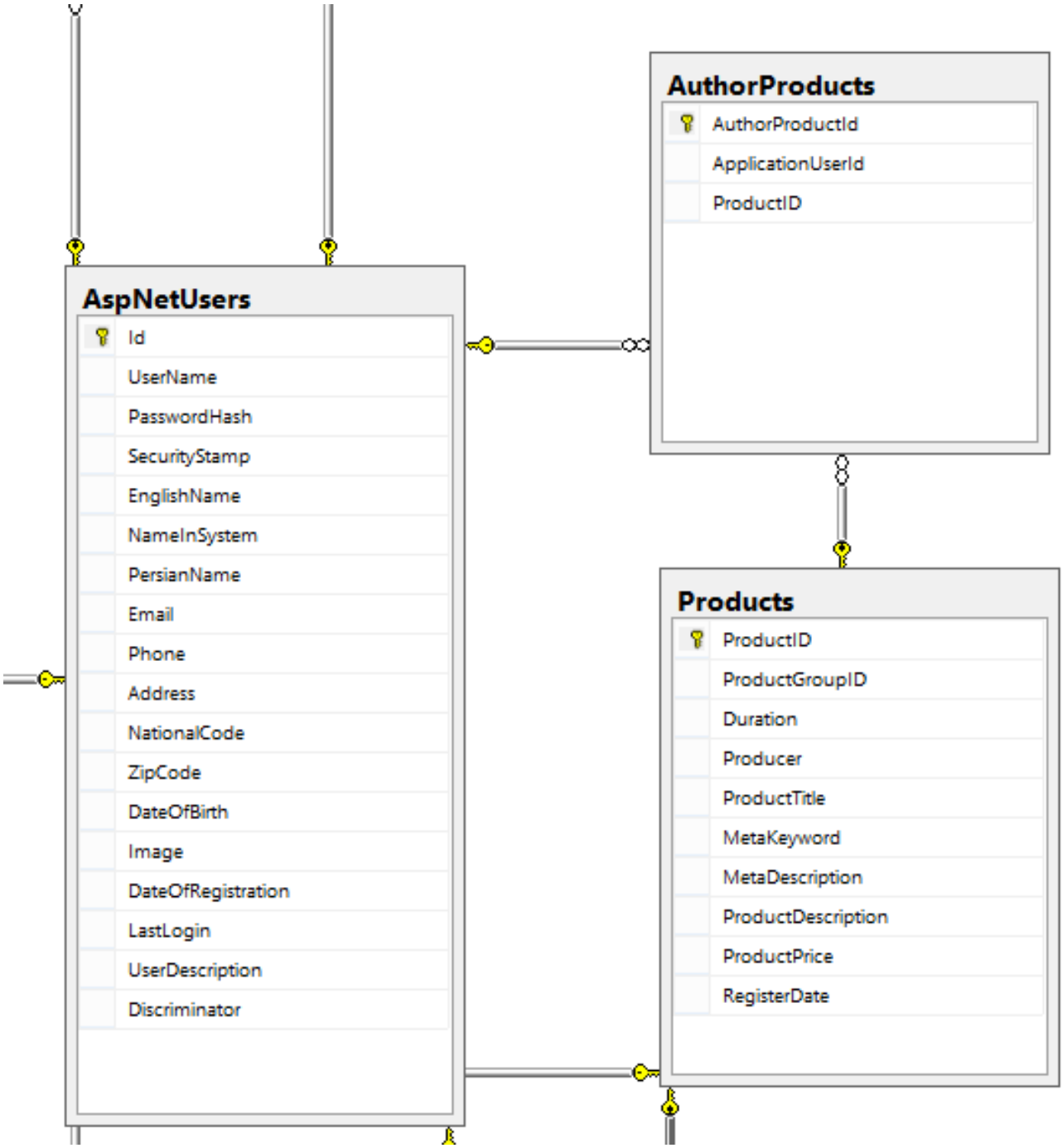



دقت کنید بعد از ایجاد پروژه در MVC5 دو پکیج بصورت اتوماتیک از طریق Nuget به پروژه شما اضافه میشود:

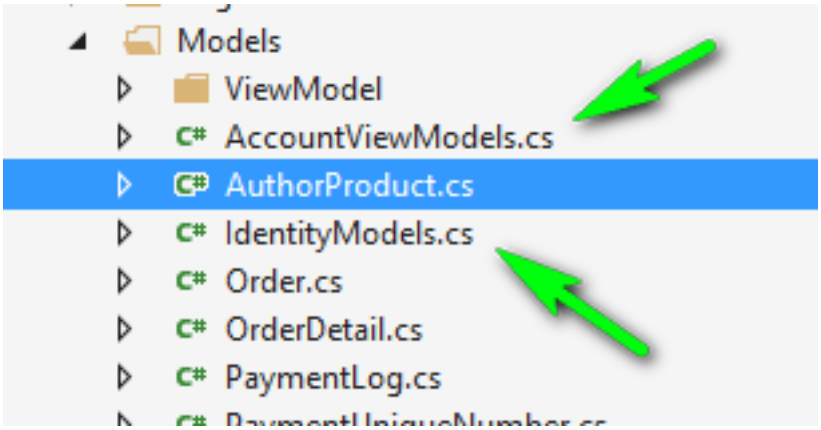
```
Microsoft.AspNet.Identity.Core
Microsoft.AspNet.Identity.EntityFrameWork
```

عامل اصلی تغییرات جدید، همین دو پکیج فوق است.
اولین پکیج شامل اینترفیس‌های IUser و IRole است که شامل فیلدهای مرتبط با این دو می‌باشد. همچنین اینترفیسی به نام IUserStore وجود دارد که چندین متد داشته و وظیفه اصلی هر نوع اضافه و حذف کردن یا تغییر در کاربران، بر دوش آن است.

دومین پکیج هم وظیفه پیاده سازی آن‌چیزی را دارد که در پکیج اول معرفی شده است. کلاس‌های موجود در این پکیج ابزارهایی برای ارتباط EntityFramework با دیتابیس هستند.
اما از مقدمات فوق که بگذریم برای درک بهتر رفتار با دیتابیس یک مثال را پیاده سازی خواهیم کرد.




فرض کنید میخواهیم چنین ارتباطی را بین سه جدول در دیتابیس برقرار کنیم، فقط به منظور یادآوری، توجه کنید که جدول ASPNetUsers جدولی است که به شکل اتوماتیک پیش از این تولید شد و ما قرار است به کمک یک جدول واسط (AuthorProduct) آن را به جدول Product مرتبط سازیم تا مشخص شود هر کتاب (به عنوان محصول) به کدام کاربر (به عنوان نویسنده) مرتبط است.
بعد از اینکه مدل‌های مربوط به برنامه خود را ساختیم، اولا نیاز به ساخت کلاس کانتکست نداریم چون خود MVC5 کلاس کانتکست را دارد؛ ثانیا نیاز به ایجاد مدل برای جداول اعتبارسنجی نیست، چون کلاسی برای فیلدهای اضافی ما که علاقمندیم به جدول Users اضافه شود، از پیش تعیین گردیده است.




دو کلاسی که با فلش علامت گذاری شده اند، تنها فایل‌های موجود در پوشه مدل، بعد از ایجاد یک پروژه هستند. فایل IdentityModel را به عنوان فایل کانتکست خواهیم شناخت (چون یکی از کلاسهایش Context است). همانطور که پیش از این گفتیم با وجود این فایل نیازی به ایجاد یک کلاس مشتق شده از DbContext نیست. همانطور که در کد زیر میبینید این فایل دارای دو کلاس است:

```
namespace MyShop.Models
{
    // You can add profile data for the user by adding more properties to your ApplicationUser class,
please visit http://go.microsoft.com/fwlink/?LinkID=317594 to learn more.
    public class ApplicationUser : IdentityUser
    {
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection")
        {
        }

}
```

کلاس اول همان کلاسی است که اگر به آن پراپرتی اضافه کنیم، بطور اتوماتیک آن پراپرتی به جدول ASPNetUsers در دیتابیس اضافه می‌شود و دیگر نگران فیلدهای نداشته‌ی جدول کاربران ASP.NET نخواهیم بود. مثلا در کد زیر چند عنوان به این جدول اضافه کرده ایم.

```
namespace MyShop.Models
{
    // You can add profile data for the user by adding more properties to your ApplicationUser class,
please visit http://go.microsoft.com/fwlink/?LinkID=317594 to learn more.
    public class ApplicationUser : IdentityUser
    {
        [Display(Name = "نام انگلیسی")]
        public string EnglishName { get; set; }

        [Display(Name = "نام سیستمی")]
        public string NameInSystem { get; set; }

        [Display(Name = "نام فارسی")]
        public string PersianName { get; set; }

        [Required]
        [DataType(DataType.EmailAddress)]
        [Display(Name = "آدرس ایمیل")]
        public string Email { get; set; }
     }
}
```

کلاس دوم نیز محل معرفی مدلها به منظور ایجاد در دیتابیس است. به ازای هر مدل یک جدول در دیتابیس خواهیم داشت. مثلا در

شکل فوق سه پراپرتی به جدول کاربران اضافه میشود. دقت داشته باشید با اینکه هیچ مدلی برای جدول کاربران نساخته ایم اما کلاس ApplicatioUsers کلاسی است که به ما امکان دسترسی به مقادیر این جدول را می‌دهد(دسترسی به معنای اضافه و حذف وتغییر مقادیر این جدول است) (در MVC4 به کمک کلاس membership کارهای مشابهی انجام میدادیم)
در ساختن مدل هایمان نیز اگر نیاز به ارتباط با جدول کاربران باشد، از همین کلاس فوق استفاده میکنیم. کلاس واسط(مدل واسط) بین AspNetUsers و Product در کد زیر زیر نشان داده شده است :

```
namespace MyShop.Models
{
    public class AuthorProduct
    {
        [Key]
        public int AuthorProductId { get; set; }
       /* public int UserId { get; set; }*/

        [Display(Name = "User")]
        public string ApplicationUserId { get; set; }

        public int ProductID { get; set; }

        public virtual Product Product { get; set; }

        public virtual ApplicationUser ApplicationUser { get; set; }
    }
}
```

همانطور که مشاهده میکنید، به راحتی ارتباط را برقرار کردیم و برای برقراری این ارتباط از کلاس ApplicationUser استفاده کردیم. پراپرتی ApplicationUserId نیز فیلد ارتباطی ما با جدول کاربران است. جدول product هم نکته خاصی ندارد و به شکل زیر مدل خواهد شد.

```
namespace MyShop.Models
{
    [DisplayName("محصول")]
    [DisplayPluralName("محصولات")]
    public class Product
    {
        [Key]
        public int ProductID { get; set; }

        [Display(Name = "گروه محصول")]
        [Required(ErrorMessage = "لطفا {0} را وارد کنید")]
        public int ProductGroupID { get; set; }

        [Display(Name = "مدت زمان")]
        public string Duration { get; set; }


        [Display(Name = "نام تهیه کننده")]
        public string Producer { get; set; }

        [Display(Name = "عنوان محصول")]
        [Required(ErrorMessage = "لطفا {0} را وارد کنید")]
        public string ProductTitle { get; set; }

        [StringLength(200)]
        [Display(Name = "کلید واژه")]
        public string MetaKeyword { get; set; }

        [StringLength(200)]
        [Display(Name = "توضیح")]
        public string MetaDescription { get; set; }

        [Display(Name = "شرح محصول")]
        [UIHint("RichText")]
        [AllowHtml]
        public string ProductDescription { get; set; }

        [Display(Name = "قیمت محصول")]
        [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:#,0 ریال}")]
        [UIHint("Integer")]
        [Required(ErrorMessage = "لطفا {0} را وارد کنید")]
        public int ProductPrice { get; set; }
        [Display(Name = "تاریخ ثبت محصول")]
```

```
        public DateTime? RegisterDate { get; set; }

    }
}
```

به این ترتیب هم ارتباطات را برقرار کرده‌ایم و هم از ساختن یک UserProfile اضافی خلاص شدیم.

برای پر کردن مقادیر اولیه نیز به راحتی از seed موجود در Configuration.cs مربوط به migration استفاده میکنیم. نمونه‌ی اینکار در کد زیر موجود است:

```
protected override void Seed(MyShop.Models.ApplicationDbContext context)
        {
            context.Users.AddOrUpdate(u => u.Id,
                        new ApplicationUser() {  Id = "1",EnglishName = "MortezaDalil", PersianName =
"مرتضی دلیل", UserDescription = "توضیح در مورد مرتضی", Email = "mm@mm.com", Phone = "2323", Address =
"test", NationalCode = "2222222222", ZipCode = "2222222222" },
                                new ApplicationUser() { Id = "2", EnglishName = "MarhamatZeinali",
PersianName = "محسن احمدی", UserDescription = "توضیح در مورد محسن", Email = "mm@mm.com", Phone =
"2323", Address = "test", NationalCode = "2222222222", ZipCode = "2222222222" },
                                new ApplicationUser() { Id = "3", EnglishName = "MahdiMilani", PersianName
= "مهدی محمدی", UserDescription = "توضیح در مورد مهدی", Email = "mm@mm.com", Phone = "2323", Address
= "test", NationalCode = "2222222222", ZipCode = "2222222222" },
                                new ApplicationUser() { Id = "4", EnglishName = "Babak", PersianName =
"بابک", UserDescription = "کاربر معمولی بدون توضیح", Email = "mm@mm.com", Phone = "2323", Address =
"test", NationalCode = "2222222222", ZipCode = "2222222222" }

                            );


            context.AuthorProducts.AddOrUpdate(u => u.AuthorProductId,
              new AuthorProduct() { AuthorProductId = 1, ProductID = 1, ApplicationUserId = "2" },
              new AuthorProduct() { AuthorProductId = 2, ProductID = 2, ApplicationUserId = "1" },
              new AuthorProduct() { AuthorProductId = 3, ProductID = 3, ApplicationUserId = "3" }

          );
```

می‌توانیم از کلاس‌های خود Identity برای انجام روش فوق استفاده کنیم؛ فرض کنید بخواهیم یک کاربر به نام admin و با نقش admin به سیستم اضافه کنیم.

```
if (!context.Users.Where(u => u.UserName == "Admin").Any())
            {
                var roleStore = new RoleStore<IdentityRole>(context);
                var rolemanager = new RoleManager<IdentityRole>(roleStore);

                var userstore = new UserStore<ApplicationUser>(context);
                var usermanager = new UserManager<ApplicationUser>(userstore);

                var user = new ApplicationUser {UserName = "Admin"};

                usermanager.Create(user, "121212");
                rolemanager.Create(new IdentityRole {Name = "admin"});

                usermanager.AddToRole(user.Id, "admin");
            }
```

در عبارت شرطی موجود کد فوق، ابتدا چک کردیم که چنین یوزری در دیتابیس نباشد، سپس از کلاس RoleStore که پیاده سازی شده‌ی اینترفیس IRoleStore است استفاده کردیم. سازنده این کلاس به کانتکست نیاز دارد؛ پس به آن context را به عنوان ورودی می‌دهیم. در خط بعد، کلاس rolemanager را داریم که بخشی از پکیج Core است و پیش از این درباره اش توضیح دادیم ( یکی از دو رفرنسی که خوبخود به پروژه اضافه میشوند) و از ویژگی‌های Identity است. به آن آبجکتی که از RoleStore ساختیم را پاس میدهیم و خود کلاس میداند چه چیز را کجا ذخیره کند.

برای ایجاد کاربر نیز همین روند را انجام می‌دهیم. سپس یک آبجکت به نام user را از روی کلاس ApplicationUser میسازیم. برای آن پسورد 121212 سِت میکنیم و نقش ادمین را به آن نسبت میدهیم. این روش قابل تسری به تمامی بخش‌های برنامه شماست. میتوانید عملیات کنترل و مدیریت اکانت را نیز به همین شکل انجام دهید. ساخت کاربر و لاگین کردن یا مدیریت پسورد

نیز به همین شکل قابل انجام است.
بعد از آپدیت دیتابیس تغییرات را مشاهده خواهیم کرد.