عنوان: **استفاده از MembershipProvider و RoleProvider در windows Application**
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۱۲/۱۷ ۱۸:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, MemberShip, RolProvider, SimpleMembership

برای استفاده از سیستم مدیریت کاربران و نقش‌های آنها به یک پیاده سازی از کلاس انتزاعی MembershipProvider نیاز داریم.
SQL Membership Provider تو کار دات نت، انتخاب پیش فرض ماست ولی به دلیل طراحی در دات نت 2 و نیاز سنجی قدیمی اون و همچنین گره زدن برنامه با sql server (استفاده از stored procedure و...) انتخاب مناسبی نیست. پیشنهاد خود مایکروسافت استفاده از SimpleMembership است که این پیاده سازی قابلیت‌های بیشتری از MembershipProvider پایه رو دارد. این قابلیت‌های بیشتر با استفاده از کلاس انتزاعی ExtendedMembershipProvider که خود از از MembershipProvider مشتق شده است میسر شده است.
برای این آموزش ما از SimpleMembership استفاده می‌کنیم اگر شما دوست ندارید از SimpleMembership استفاده کنید می‌تونید از Provider های دیگه ای استفاده کنید و حتی می‌تونید یک پروایدر سفارشی برای خودتون بنویسید.

Custom Membership Providers

Codefirstmembership

EFmembership

...

برای شروع یک پروژه ConsoleApplication تعریف کنید و رفرنس‌های زیر رو اضافه کنید.

```
System.Web.dll
System.Web.ApplicationServices.dll
```

خاصیت Copy Local دو کتابخانه زیر رو true ست کنید.

```
C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET Web Pages\v2.0\Assemblies\WebMatrix.Data.dll
C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET Web Pages\v2.0\Assemblies\WebMatrix.WebData.dll
```

- در صورتیکه یک پروژه Asp.Net MVC 4 به همراه تمپلت Internet Application بسازید بصورت خودکار SimpleMembership و رفرنس‌های آن به پروژه اضافه می‌شود.
یک فایل App.config با محتویات زیر به پروژه اضافه کنید و تنظیمات ConnectionString را مطابق با دیتابیس موجود در کامپیوتر خود تنظیم کنید:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=.;Initial Catalog=SimpleMembershipProviderDB;Integrated
Security=True;"
         providerName="System.Data.SqlClient"/>
  </connectionStrings>

  <system.web>
    <roleManager enabled="true" defaultProvider="SimpleRoleProvider">
      <providers>
        <clear/>
        <add name="SimpleRoleProvider" type="WebMatrix.WebData.SimpleRoleProvider, WebMatrix.WebData"/>
      </providers>
    </roleManager>
    <membership defaultProvider="SimpleMembershipProvider">
      <providers>
        <clear/>
        <add name="SimpleMembershipProvider" type="WebMatrix.WebData.SimpleMembershipProvider,
WebMatrix.WebData"/>
      </providers>
```

```
    </membership>

  </system.web>
</configuration>
```

محتویات فایل Program.cs :

```
using System;
using System.Security.Principal;
using System.Web.Security;
using WebMatrix.WebData;

namespace MemberShipConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            WebSecurity.InitializeDatabaseConnection("DefaultConnection",
                "UserProfile", "UserId", "UserName",
                autoCreateTables: true);

            AddUserAndRolSample();
            Login();
            if (System.Threading.Thread.CurrentPrincipal.Identity.IsAuthenticated)
                RunApp();

        }

        static void AddUserAndRolSample()
        {
            if (WebSecurity.UserExists("iman"))
                return;

            //  No implements in SimpleMembershipProvider :
            //     Membership.CreateUser("iman", "123");
            WebSecurity.CreateUserAndAccount("iman", "123");
            Roles.CreateRole("admin");
            Roles.CreateRole("User");
            Roles.AddUserToRole("iman", "admin");
        }

        static void Login()
        {
            for (int i = 0; i < 3; i++)
            {
                Console.Write("UserName: ");
                var userName = Console.ReadLine();
                Console.Write("Password: ");
                var password = Console.ReadLine();
                if (Membership.ValidateUser(userName, password))
                {

                    var user = Membership.GetUser(userName);
                    var identity = new GenericIdentity(user.UserName);
                    var principal = new RolePrincipal(identity);
                    System.Threading.Thread.CurrentPrincipal = principal;

                    Console.Clear();
                    return;
                }


                Console.WriteLine("User Name Or Password Not Valid.");
            }

        }
        static void RunApp()
        {
            Console.WriteLine("Welcome To MemberShip. User Name: {0}",
                System.Threading.Thread.CurrentPrincipal.Identity.Name);

            if (System.Threading.Thread.CurrentPrincipal.IsInRole("admin"))
                Console.WriteLine("Hello Admin User!");

            Console.Read();
        }
    }
```

```
}
```

در ابتدا با فراخوانی متد InitializeDatabaseConnection تنظیمات اولیه simpleMembership را مقدار دهی می‌کنیم. این متد حتما باید یکبار اجرا شود.

```
InitializeDatabaseConnection(string connectionStringName,
                             string userTableName,
                             string userIdColumn,
                             string userNameColumn,
                             bool autoCreateTables)
```

ملاحظه می‌کنید پارامتر آخر متد مربوط به ساخت جداول مورد نیاز این پروایدر است. در صورتی که بخواهیم در پروژه از EntityFramework استفاده کنیم می‌تونیم موجودیت‌های معادل جدول‌های مورد نیاز SimpleMembership رو در EF بسازیم و در این متد AutoCreateTables رو False مقدار دهی کنیم. برای بدست آوردن موجودیت‌های معادل جدول‌های این پروایدر با ابزار Entity Framework Power Tools و روش مهندسی معکوس ، تمام موجودیت‌های یک دیتابیس ساخته شده رو استخراج می‌کنیم.
- SimpleMembership از تمام خانواده microsoft sql پشتبانی می‌کنه (SQL Server, SQL Azure, SQL Server CE, SQL Server Express,LocalD) برای سایر دیتابیس‌ها به علت تفاوت در دستورات ساخت تیبل‌ها و ... مایکروسافت تضمینی نداده ولی اگر خودمون جدول‌های مورد نیاز SimpleMembership رو ساخته باشیم احتمالا در سایر دیتابیس‌ها هم قابل استفاده است.
در ادامه برنامه بالا یک کاربر و دو نقش تعریف کردیم و نقش admin رو به کاربر نسبت دادیم. در متد login در صورت معتبر بودن کاربر ، اون رو به ترد اصلی برنامه معرفی می‌کنیم. هر جا خواستیم می‌تونیم نقشهای کاربر رو چک کنیم و نکته آخر با اولین چک کردن نقش یک کاربر تمام نقش‌های اون در حافظه سیستم کش می‌شود و تنها مرتبه اول با دیتابیس ارتباط برقرار می‌کند.

عنوان: **سفارشی کردن ASP.NET Identity در MVC 5**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۱۹ ۱۹:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MemberShip, ASP.NET MVC 5, ASP.NET Identity, MVC, Security

یکی از نیازهای رایج توسعه دهندگان هنگام استفاده از سیستم عضویت ASP.NET سفارشی کردن الگوی داده‌ها است. مثلا ممکن است بخواهید یک پروفایل سفارشی برای کاربران در نظر بگیرید، که شامل اطلاعات شخصی، آدرس و تلفن تماس و غیره می‌شود. یا ممکن است بخواهید به خود فرم ثبت نام فیلدهای جدیدی اضافه کنید و آنها را در رکورد هر کاربر ذخیره کنید.
یکی از مزایای ASP.NET Identity این است که بر پایه EF Code First نوشته شده است. بنابراین سفارشی سازی الگوی دیتابیس و اطلاعات کاربران ساده است.
یک اپلیکیشن جدید ASP.NET MVC بسازید و نوع احراز هویت را Individual User Accounts انتخاب کنید. پس از آنکه پروژه جدید ایجاد شد فایل IdentityModels.cs را در پوشه Models باز کنید. کلاسی با نام *ApplicationUser* مشاهده می‌کنید که همتای *UserProfile* در فریم ورک SimpleMembership است. این کلاس خالی است و از کلاس *IdentityUser* ارث بری می‌کند و شامل خواص زیر است.

```
public class IdentityUser : IUser
    {
        public IdentityUser();
        public IdentityUser(string userName);

        public virtual ICollection<identityuserclaim> Claims { get; }
        public virtual string Id { get; set; }
        public virtual ICollection<identityuserlogin> Logins { get; }
        public virtual string PasswordHash { get; set; }
        public virtual ICollection<identityuserrole> Roles { get; }
        public virtual string SecurityStamp { get; set; }
        public virtual string UserName { get; set; }
    }
```

اگر دقت کنید خواهید دید که فیلد Id بر خلاف SimpleMembership یک عدد صحیح یا int نیست، بلکه بصورت یک رشته ذخیره می‌شود. پیاده سازی پیش فرض ASP.NET Identity مقدار این فیلد را با یک GUID پر می‌کند. در این پست تنها یک فیلد آدرس ایمیل به کلاس کاربر اضافه می‌کنیم. با استفاده از همین فیلد در پست‌های آتی خواهیم دید چگونه می‌توان ایمیل‌های تایید ثبت نام برای کاربران ارسال کرد. کلاس ApplicationUser بدین شکل خواهد بود.

```
public class ApplicationUser : IdentityUser
{
    public string Email { get; set; }
}
```

حال برای آنکه کاربر بتواند هنگام ثبت نام آدرس ایمیل خود را هم وارد کند، باید مدل فرم ثبت نام را بروز رسانی کنیم.

```
public class RegisterViewModel
{
    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength =
6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }

    [Required]
    [Display(Name = "Email address")]
```

```
    public string Email { get; set; }

}
```

سپس فایل View را هم بروز رسانی می‌کنیم تا یک برچسب و تکست باکس برای آدرس ایمیل نمایش دهد.

```
<div class="form-group">
     @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
     <div class="col-md-10">
         @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
     </div>
 </div>
```

برای تست این تغییرات، صفحه About را طوری تغییر می‌دهید تا آدرس ایمیل کاربر جاری را نمایش دهد. این قسمت همچنین نمونه ای از نحوه دسترسی به اطلاعات کاربران است.

```
public ActionResult About()
{
    ViewBag.Message = "Your application description page.";
    UserManager<ApplicationUser> UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(new ApplicationDbContext()));
    var user = UserManager.FindById(User.Identity.GetUserId());
    if (user != null)
        ViewBag.Email = user.Email;
    else
        ViewBag.Email = "User not found.";

    return View();
}
```

همین! تمام کاری که لازم بود انجام دهید همین بود. از آنجا که سیستم ASP.NET Identity توسط Entity Framework مدیریت می‌شود، روی الگوی دیتابیس سیستم عضویت کنترل کامل دارید. بنابراین به سادگی می‌توانید با استفاده از قابلیت Code First مدل‌های خود را سفارشی کنید.

در پست‌های آتی این مطلب را ادامه خواهیم داد تا ببینیم چگونه می‌توان ایمیل‌های تاییدیه برای کاربران ارسال کرد.

نظرات خوانندگان

نویسنده: رجایی
تاریخ: ۱۳۹۲/۱۱/۰۸ ۱۹:۲۰

سلام؛ از زحماتتون بسیار تشکر می‌کنم. من وب سایت را به روش چند لایه‌ی مرسوم می‌سازم:

data layer - domain models - website - service layer

با توجه به آن چگونه می‌توان از asp.net identity استفاده کرد؟ زیرا مثلا نیاز است از کلید جدول users در مدل‌های دیگه استفاده شود. آیا این کلید را باید در لایه دومین استفاده کرد؟

نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۰۸ ۲۳:۴۵

موجودیت‌های مربوط به ASP.NET Identity رو در لایه مدل‌ها قرار بدین و از یک DbContext استفاده کنید. یعنی DbSetهای مدل برنامه و Identity رو در یک کانتکست تعریف کنید.

نویسنده: رجایی
تاریخ: ۱۳۹۲/۱۱/۱۰ ۱۲:۴

سلام...ممنون از پاسختون.

با توجه به راهنمایی شما در قسمت context در لایه data layer بدین صورت درج کردم

```
public class Context : DbContext
    {
        public DbSet<IdentityUserClaim> AspNetUserClaims { set; get; }
        public DbSet<IdentityRole> AspNetRoles { set; get; }
        public DbSet<IdentityUserLogin> AspNetUserLogins { set; get; }
        public DbSet<IdentityUserRole> AspNetUserRoles { set; get; }
        public DbSet<IdentityUser> AspNetUsers { set; get; }
        //-------------------------------------
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<IdentityRole>().HasKey<string>(r => r.Id).ToTable("AspNetRoles");
            modelBuilder.Entity<IdentityUser>().ToTable("AspNetUsers");
            modelBuilder.Entity<IdentityUserLogin>().HasKey(l => new { l.UserId, l.LoginProvider,
l.ProviderKey }).ToTable("AspNetUserLogins");
            modelBuilder.Entity<IdentityUserRole>().HasKey(r => new { r.RoleId, r.UserId
}).ToTable("AspNetUserRoles");
            modelBuilder.Entity<IdentityUserClaim>().ToTable("AspNetUserClaims");
        }
    }
```

ولی وقتی سایت اجرا می‌شود ایراد زیر نمایش داده می‌شود

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۰ ۱۲:۱۴

خطا رو فراموش کردید ارسال کنید.

نویسنده: مهرداد راهی
تاریخ: ۱۳۹۲/۱۱/۱۱ ۰:۵۵

سلام من MVC کار نمیکنم توی ASP.Net وب‌فرمز استفاده میکنم از این امکان. فایل IdentityModels.cs رو نداره توی پروژه . مشکل کجاس؟
enable migrations- هم زدم خطا داد

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱:۵۳

از VS 2013 استفاده می‌کنی ؟

نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۱۱ ۲:۵۱

لازم نیست تمام این آبجکت‌ها رو به context نگاشت کنید. قالب پروژه‌های VS 2013 بصورت خودکار در پوشه Models کلاسی بنام IdentityModels میسازه. این کلاس شامل کلاسی بنام ApplicationDbContext میشه که تعریفی مانند لیست زیر داره:

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

این کلاس رو کلا حذف کنید، چون قراره از یک DbContext برای تمام موجودیت‌ها استفاده کنید.
کلاس ApplicationUser که معرف موجودیت کاربران هست رو در لایه دامنه‌ها تعریف کنید و دقت کنید که باید از IdentityUser ارث بری کنه، حال با نام پیش فرض یا با نام دلخواه. سپس باید کلاسی بسازید که از UserManager<T> مشتق میشه. با استفاده از این کلاس می‌تونید به موجودیت‌های کاربران دسترسی داشته باشید. بعنوان مثال:

```
public class AppUserManager : UserManager<AppUser>{
    public AppUserManager() : base(new UserStore<AppUser>(new ShirazBilitDbContext())) { }
}
```

همونطور که می‌بینید کلاس موجودیت کاربر در اینجا AppUser نام داره، پس هنگام استفاده از UserManager نوع داده رو بهش نگاشت می‌کنیم. کد کلاس AppUser هم مطابق لیست زیر خواهد بود.

```
public class AppUser : IdentityUser
{
    public string Email { get; set; }
    public string ConfirmationToken { get; set; }
    public bool IsConfirmed { get; set; }
}
```

همونطور که مشخصه کلاس کاربران سفارشی سازی شده و سه فیلد به جدول کاربران اضافه کردیم. فیلدهای بیشتر یا موجودیت پروفایل کاربران هم باید به همین کلاس افزوده بشن. اگر پست‌ها رو بیاد بیارید گفته شد که ASP.NET Identity با مدل EF Code-First کار میکنه.

نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۱۱ ۲:۵۷

از VS 2013 استفاده کنید و .NET 4.5.
اگر این فایل برای شما ایجاد نمیشه پس در قالب پروژه‌های Web Forms وجود نداره. ارتباطی با مهاجرت‌ها هم نداره، کلاس موجودیت کاربر رو خودتون می‌تونید ایجاد کنید. اگر به نظرات بالا مراجعه کنید گفته شد که کلاس کاربران باید از

IdentityModels ارث بری کنه.

نویسنده: رجایی
تاریخ: ۱۳۹۲/۱۱/۱۲ ۹:۲۱

عذر خواهی می‌کنم فراموش کردم. ایراد بدین صورت است:

System.InvalidOperationException: The model backing the 'Context' context has changed since the database was created. Consider using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).

مشخص است که می‌گوید context تغییر می‌کند.ولی من از migration استفاده می‌کنم و codefirst ولی باز هم این ایراد رو در اتصال به دیتابیس نشان می‌دهد. من از add-migration هم استفاده می‌کنم تا تغییرات موجودیت‌ها رو کامل به من نشان دهد که چیزی را عنوان نمی‌کند.

نویسنده: افتاب
تاریخ: ۱۳۹۲/۱۲/۲۷ ۲۳:۳۸

سلام و متشکر ،
دنبال آن میگشتم که چه طوری میشه دو تا صفحه لوگین در پروژه داشت که ورودی کاربران از مدیران جدا باشد

نویسنده: افتاب
تاریخ: ۱۳۹۲/۱۲/۲۷ ۲۳:۴۴

میشه در مورد «این کلاس رو کلا حذف کنید، چون قراره از یک DbContext برای تمام موجودیت‌ها استفاده کنید....» بیشتر توضیح بفرمایید؟

نویسنده: سعید رضایی
تاریخ: ۱۳۹۳/۰۲/۰۱ ۱۶:۴۱

با عرض سلام.تو Vs2012+mvc4 نمیشه از identity استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۰۱ ۱۷:۲۶

خیر. با دات نت 4.5 کامپایل شده.

نویسنده: میثم سلیمانی
تاریخ: ۱۳۹۳/۰۵/۲۶ ۱۱:۷

با سلام و احترام
من دستور زیر رو تو asp.net mvc Identity sample 2 تو اکشن Login اضافه کردم

```
UserManager<ApplicationUser> UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(new ApplicationDbContext()));
    var user = UserManager.FindById(User.Identity.GetUserId());
```

اما user و null میده !
اکشن login

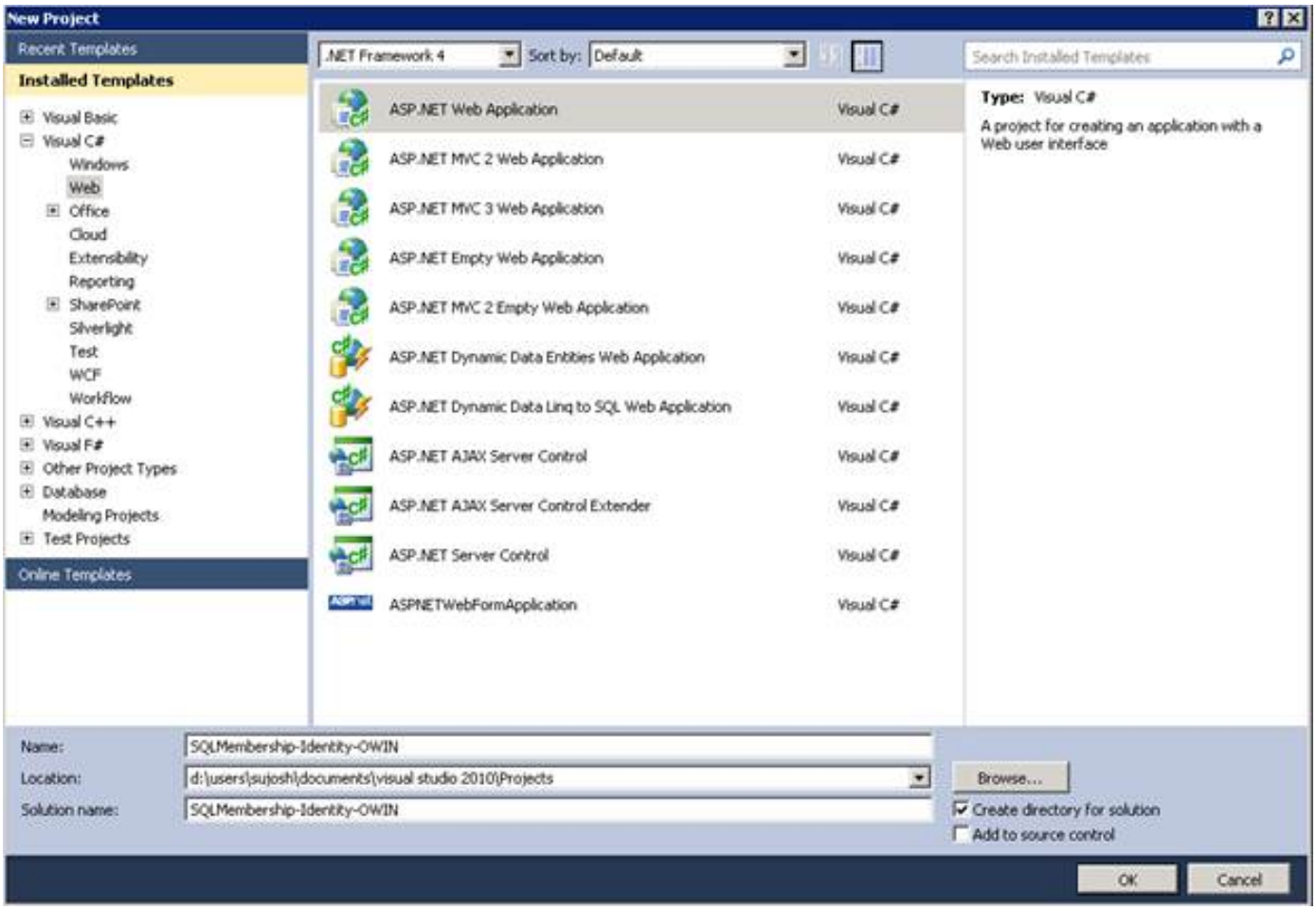
```
public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // This doen't count login failures towards lockout only two factor authentication
            // To enable password failures to trigger lockout, change to shouldLockout: true
            var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password,
model.RememberMe, shouldLockout: false);
            switch (result)
            {

                case SignInStatus.Success:
                    {
                        UserManager<ApplicationUser> UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(new ApplicationDbContext()));
                        var user = UserManager.FindById(User.Identity.GetUserId());
                        return RedirectToLocal(returnUrl);
                    }
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                    return RedirectToAction("SendCode", new { ReturnUrl = returnUrl });
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Invalid login attempt.");
                    return View(model);
            }
        }
```

عنوان: **مهاجرت از ASP.NET Identity به SQL Membership**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, Security, MemberShip, ASP.NET Identity

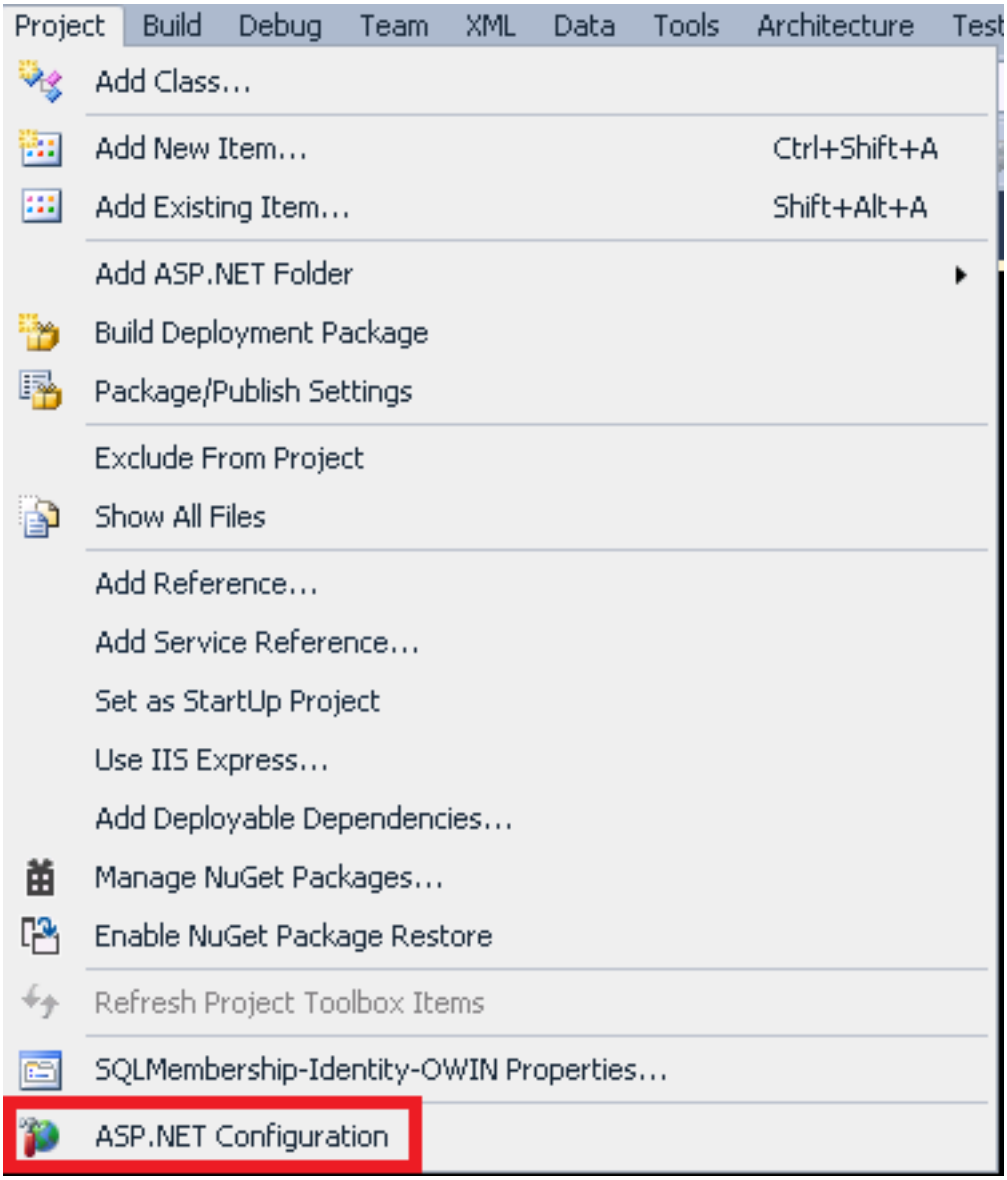
در این مقاله مهاجرت یک اپلیکیشن وب که توسط SQL Membership ساخته شده است را به سیستم جدید ASP.NET Identity بررسی می‌کنیم. برای این مقاله از یک قالب اپلیکیشن وب (Web Forms) که توسط Visual Studio 2010 ساخته شده است برای ساختن کاربران و نقش‌ها استفاده می‌کنیم. سپس با استفاده از یک SQL Script دیتابیس موجود را به دیتابیسی که ASP.NET Identity نیاز دارد تبدیل می‌کنیم. در قدم بعدی پکیج‌های مورد نیاز را به پروژه اضافه می‌کنیم و صفحات جدیدی برای مدیریت حساب‌های کاربری خواهیم ساخت. بعنوان یک تست، کاربران قدیمی که توسط SQL Membership ساخته شده بودند باید قادر باشند به سایت وارد شوند. همچنین کاربران جدید باید بتوانند بدون هیچ مشکلی در سیستم ثبت نام کنند. سورس کد کامل این مقاله را می‌توانید از این لینک دریافت کنید.

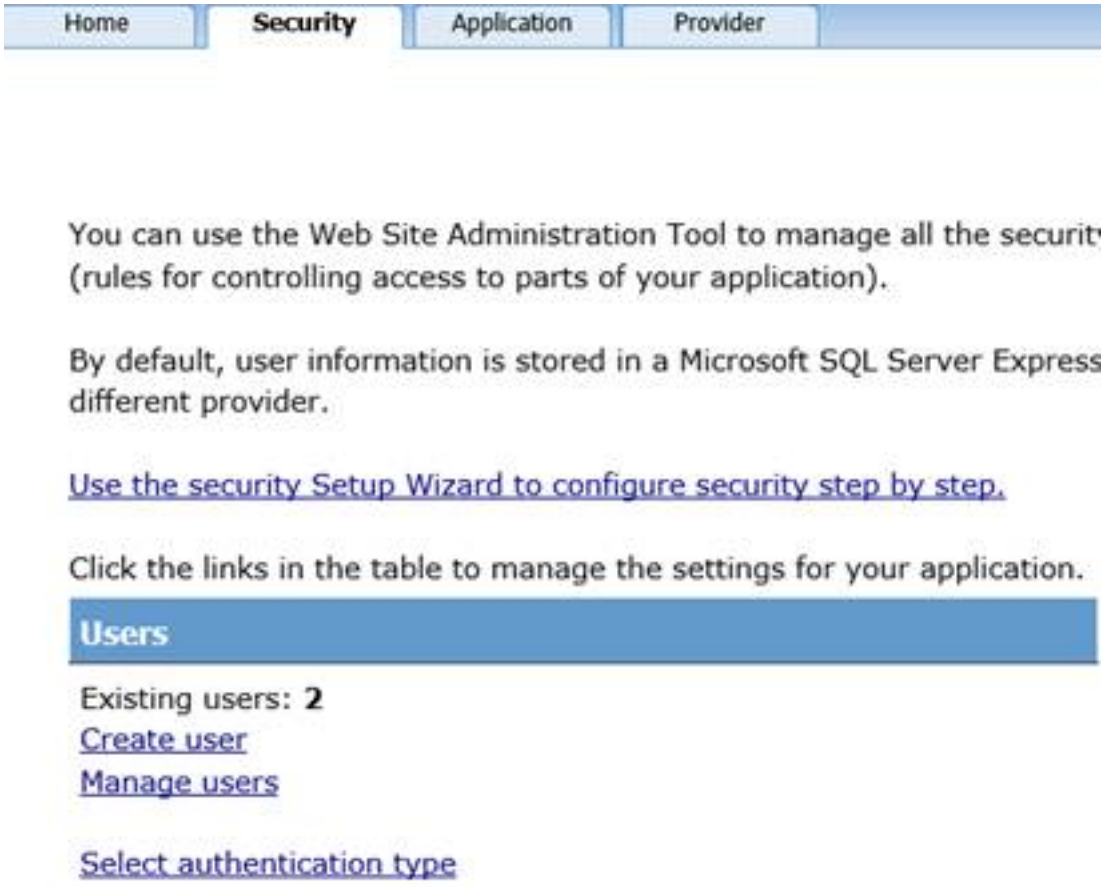
یک اپلیکیشن با SQL Membership بسازید

برای شروع به اپلیکیشنی نیاز داریم که از SQL Membership استفاده می‌کند و دارای داده هایی از کاربران و نقش‌ها است. برای این مقاله، بگذارید پروژه جدیدی توسط VS 2010 بسازیم.

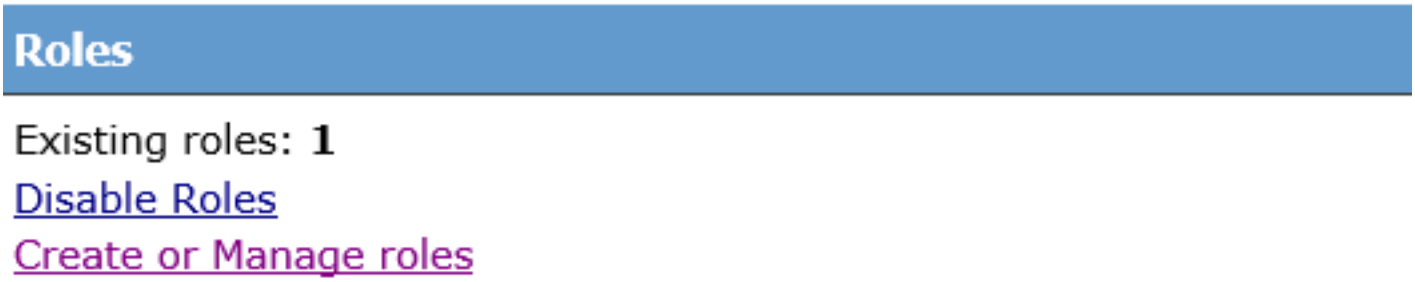


حال با استفاده از ابزار ASP.NET Configuration دو کاربر جدید بسازید: **oldAdminUser** و **oldUser** .

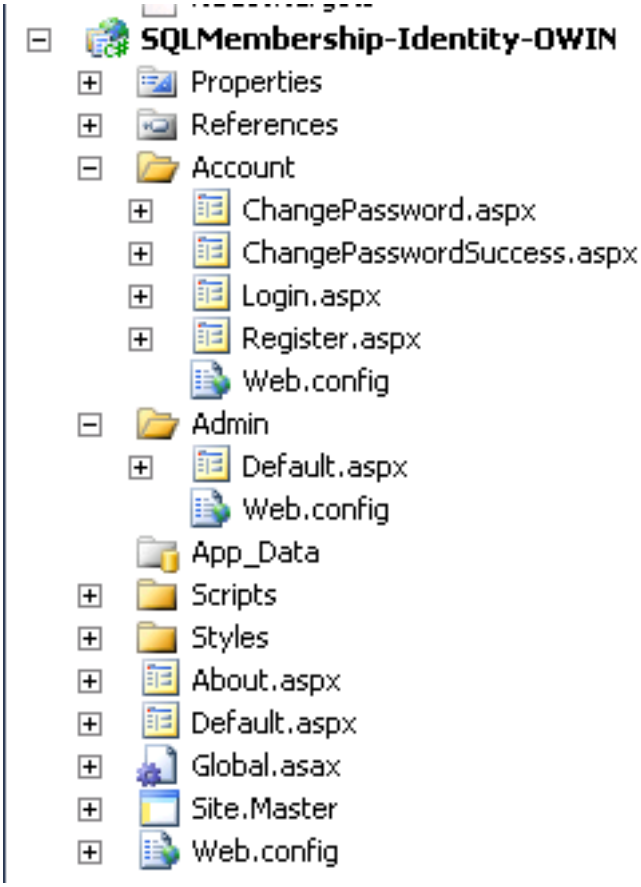

Project
Build
Debug
Team
XML
Data
Tools
Architecture
Test
Add Class...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Add ASP.NET Folder
Build Deployment Package
Package/Publish Settings
Exclude From Project
Show All Files
Add Reference...
Add Service Reference...
Set as StartUp Project
Use IIS Express...
Add Deployable Dependencies...
Manage NuGet Packages...
Enable NuGet Package Restore
Refresh Project Toolbox Items
SQLMembership-Identity-OWIN Properties...
ASP.NET Configuration



نقش جدیدی با نام Admin بسازید و کاربر oldAdminUser را به آن اضافه کنید.

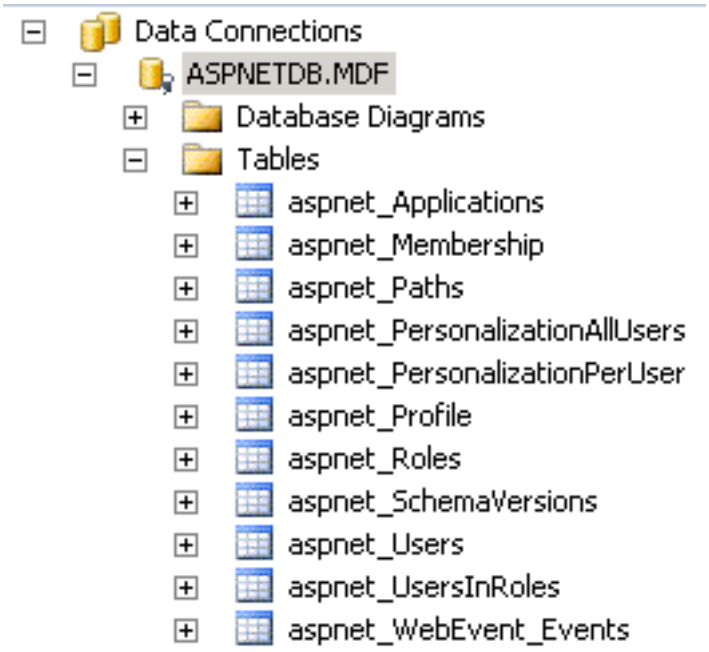


بخش جدیدی با نام Admin در سایت خود بسازید و فرمی بنام Default.aspx به آن اضافه کنید. همچنین فایل web.config این قسمت را طوری پیکربندی کنید تا تنها کاربرانی که در نقش Admin هستند به آن دسترسی داشته باشند. برای اطلاعات بیشتر به این لینک مراجعه کنید.



پنجره Server Explorer را باز کنید و جداول ساخته شده توسط SQL Membership را بررسی کنید. اطلاعات اصلی کاربران که برای ورود به سایت استفاده می‌شوند، در جداول **aspnet_Users** و **aspnet_Membership** ذخیره می‌شوند. داده‌های مربوط به نقش‌ها نیز در جدول **aspnet_Roles** ذخیره خواهند شد. رابطه بین کاربران و نقش‌ها نیز در جدول **aspnet_UsersInRoles** ذخیره می‌شود، یعنی اینکه هر کاربری به چه نقش هایی تعلق دارد.

برای مدیریت اساسی سیستم عضویت، مهاجرت جداول ذکر شده به سیستم جدید ASP.NET Identity کفایت می‌کند.



مهاجرت به Visual Studio 2013

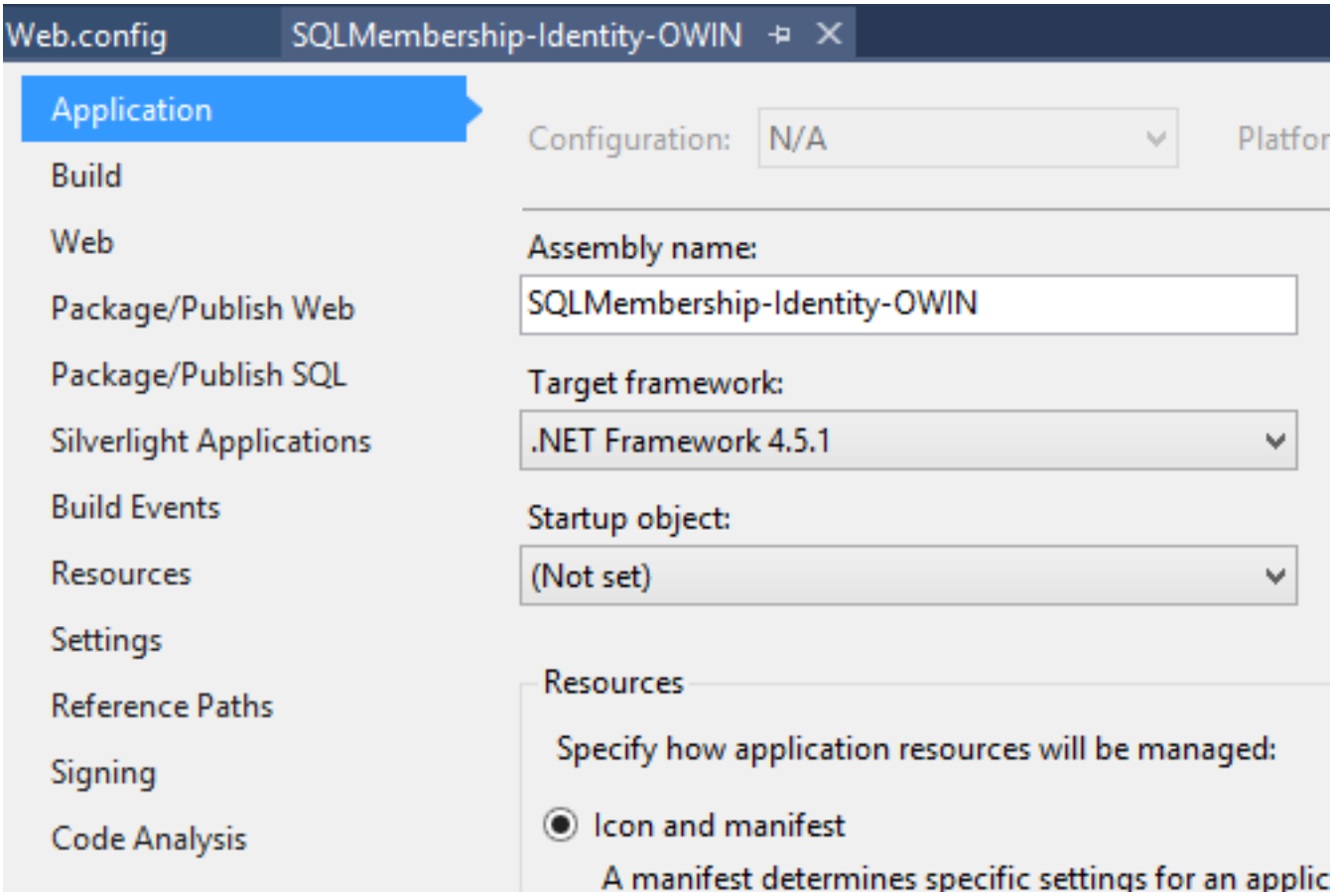
برای شروع ابتدا Visual Studio Express 2013 for Web یا Visual Studio 2013 را نصب کنید.

حال پروژه ایجاد شده را در نسخه جدید ویژوال استودیو باز کنید. اگر نسخه ای از SQL Server Express را روی سیستم خود نصب نکرده باشید، هنگام باز کردن پروژه پیغامی به شما نشان داده می‌شود. دلیل آن وجود رشته اتصالی است که از SQL Server Express استفاده می‌کند. برای رفع این مساله می‌توانید SQL Express را نصب کنید، و یا رشته اتصال را طوری تغییر دهید که از LocalDB استفاده کند.

فایل web.config را باز کرده و رشته اتصال را مانند تصویر زیر ویرایش کنید.

```
<configuration>
  <connectionStrings>
    <add name="ApplicationServices"
         connectionString="data source=(LocalDb)\v11.0;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\aspnetdb.mdf"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
```

پنجره Server Explorer را باز کنید و مطمئن شوید که الگوی جداول و داده‌ها قابل رویت هستند.
سیستم ASP.NET Identity با نسخه 4.5 دات نت فریم ورک و بالاتر سازگار است. پس نسخه فریم ورک پروژه را به آخرین نسخه (4.5.1) تغییر دهید.



پروژه را Build کنید تا مطمئن شوید هیچ خطایی وجود ندارد.

نصب پکیج‌های NuGet

در پنجره Solution Explorer روی نام پروژه خود کلیک راست کرده، و گزینه Manage NuGet Packages را انتخاب کنید. در قسمت جستجوی دیالوگ باز شده، عبارت "Microsoft.AspNet.Identity.EntityFramework" را وارد کنید. این پکیج را در لیست نتایج انتخاب کرده و آن را نصب کنید. نصب این بسته، نیازمندهای موجود را بصورت خودکار دانلود و نصب می‌کند:
EntityFramework و **ASP.NET Identity Core** . حال پکیج‌های زیر را هم نصب کنید (اگر نمی‌خواهید OAuth را فعال کنید، 4 پکیج آخر را نادیده بگیرید).

Microsoft.AspNet.Identity.Owin

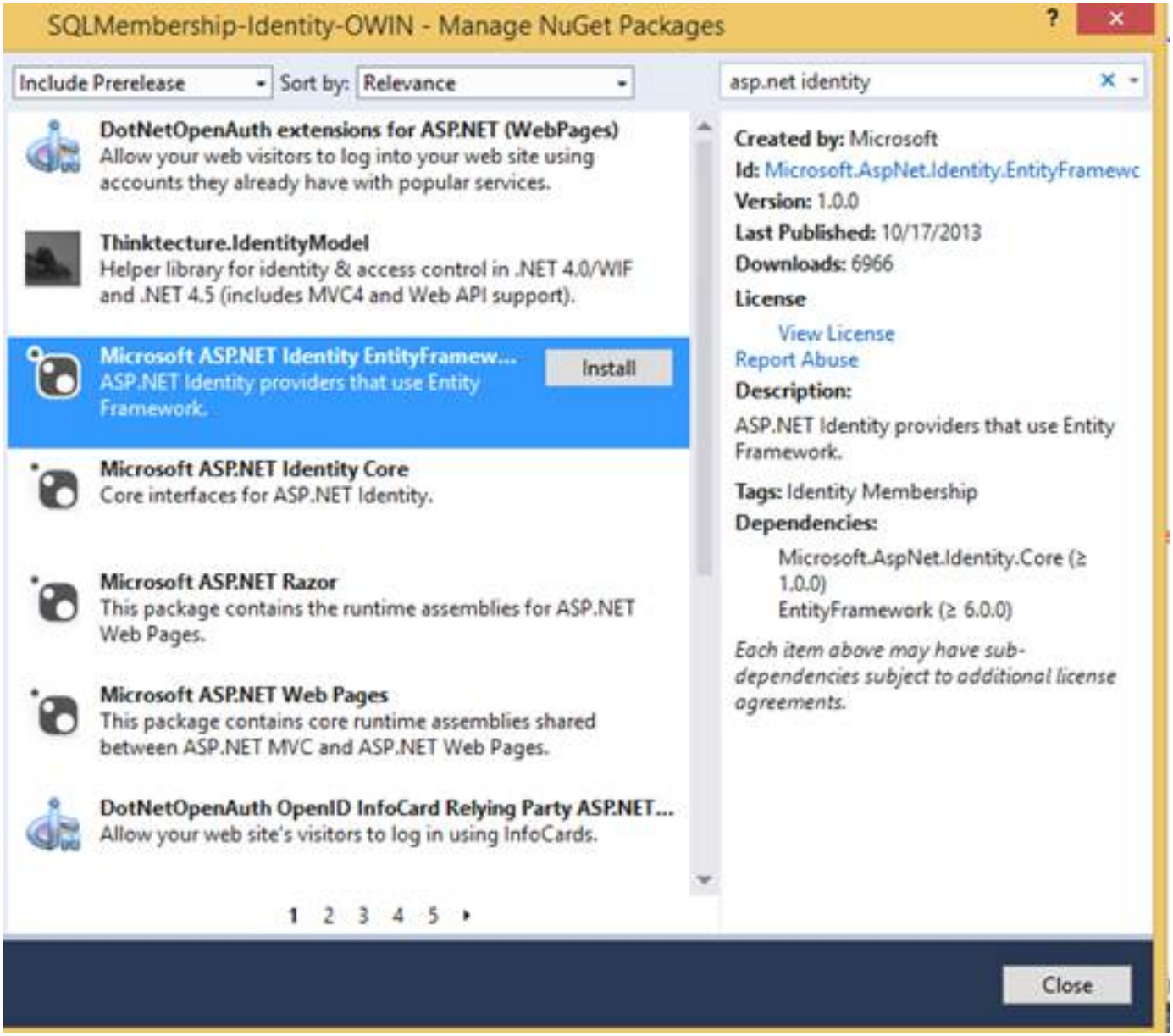
Microsoft.Owin.Host.SystemWeb

Microsoft.Owin.Security.Facebook

Microsoft.Owin.Security.Google

Microsoft.Owin.Security.MicrosoftAccount

Microsoft.Owin.Security.Twitter

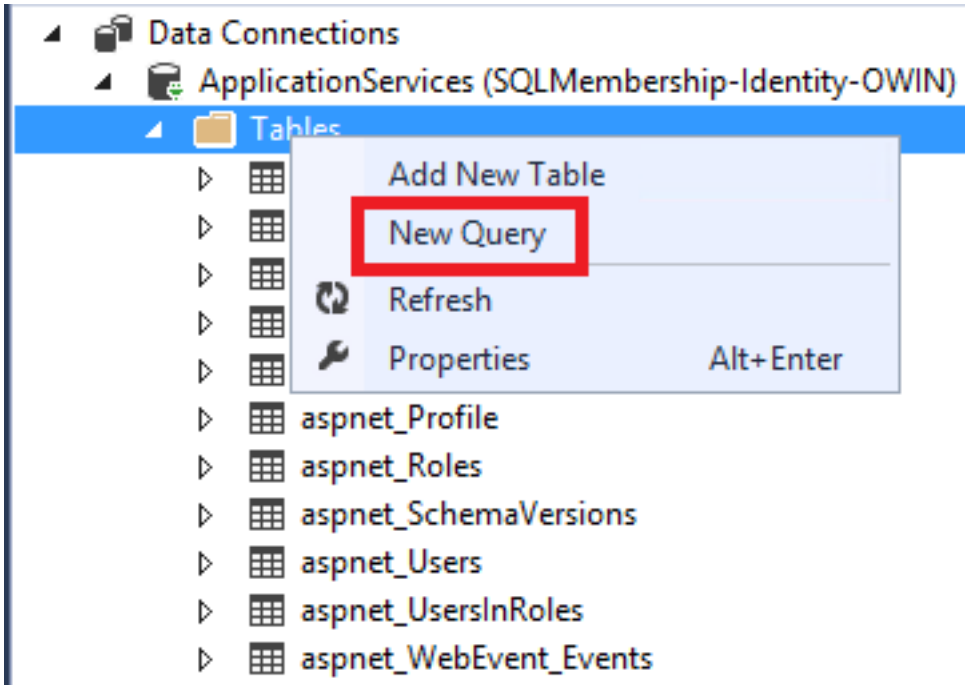


مهاجرت دیتابیس فعلی به سیستم ASP.NET Identity

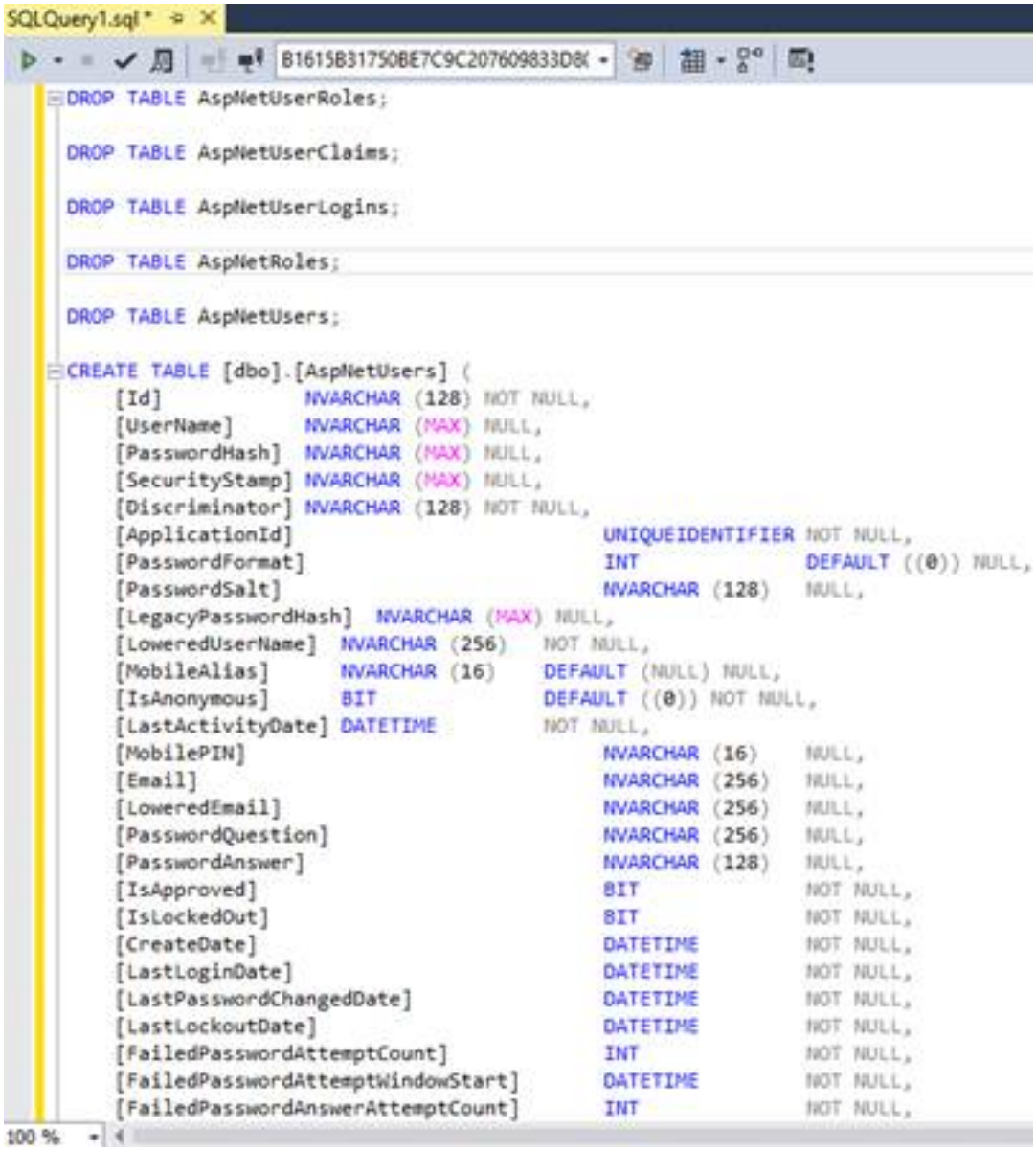
قدم بعدی مهاجرت دیتابیس فعلی به الگویی است، که سیستم ASP.NET Identity به آن نیاز دارد. بدین منظور ما یک اسکریپت SQL را اجرا می‌کنیم تا جداول جدیدی بسازد و اطلاعات کاربران را به آنها انتقال دهد. فایل این اسکریپت را می‌توانید از لینک https://github.com/suhasj/SQLMembership-Identity-OWIN دریافت کنید.

این اسکریپت مختص این مقاله است. اگر الگوی استفاده شده برای جداول سیستم عضویت شما ویرایش/سفارشی-سازی شده باید این اسکریپت را هم بر اساس این تغییرات بروز رسانی کنید.

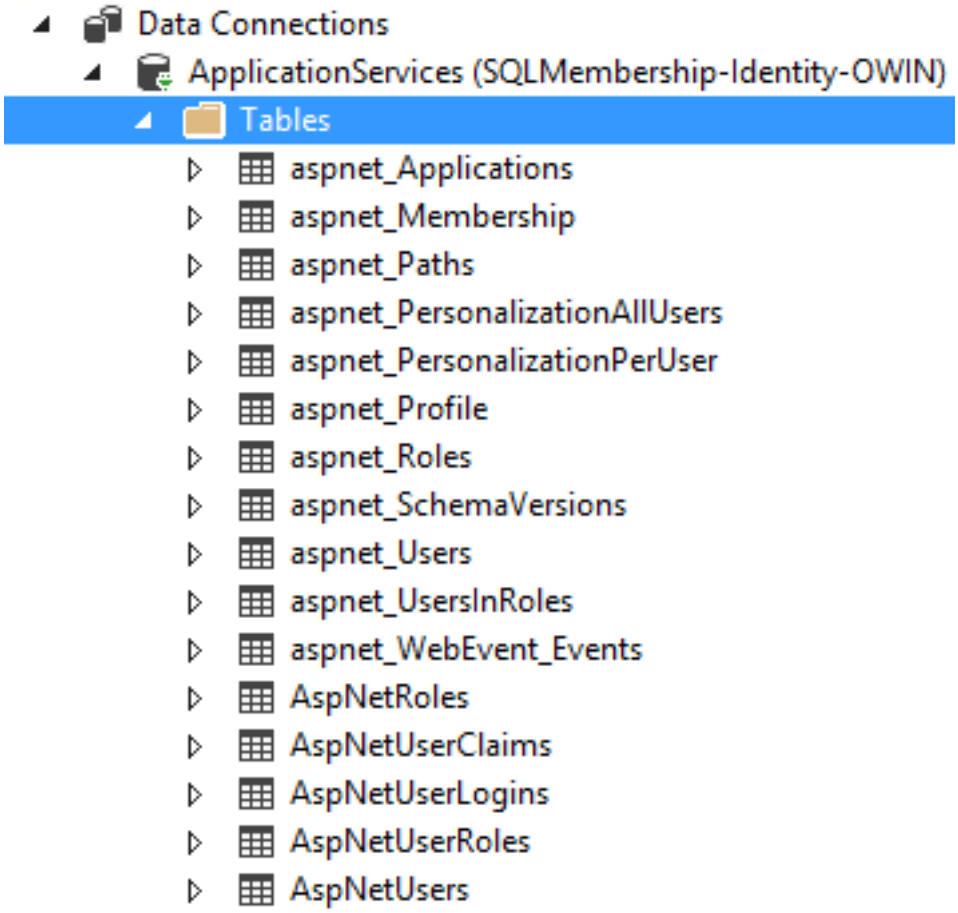
پنجره Server Explorer را باز کنید. گره اتصال ApplicationServices را باز کنید تا جداول را مشاهده کنید. روی گره Tables کلیک راست کرده و گزینه New Query را انتخاب کنید.



در پنجره کوئری باز شده، تمام محتویات فایل *Migrations.sql* را کپی کنید. سپس اسکریپت را با کلیک کردن دکمه Execute اجرا کنید.



ممکن است با اخطاری مواجه شوید مبنی بر آنکه امکان حذف (drop) بعضی از جداول وجود نداشت. دلیلش آن است که چهار عبارت اولیه در این اسکریپت، تمام جداول مربوط به Identity را در صورت وجود حذف می‌کنند. از آنجا که با اجرای اولیه این اسکریپت چنین جداولی وجود ندارند، می‌توانیم این خطاها را نادیده بگیریم. حال پنجره Server Explorer را تازه (refresh) کنید و خواهید دید که پنج جدول جدید ساخته شده اند.



لیست زیر نحوه Map کردن اطلاعات از جداول SQL Membership به سیستم Identity را نشان می‌دهد.

aspnet_Roles --> **AspNetRoles**

aspnet_Users, aspnet_Membership --> **AspNetUsers**

aspnet_UsersInRoles --> **AspNetUserRoles**

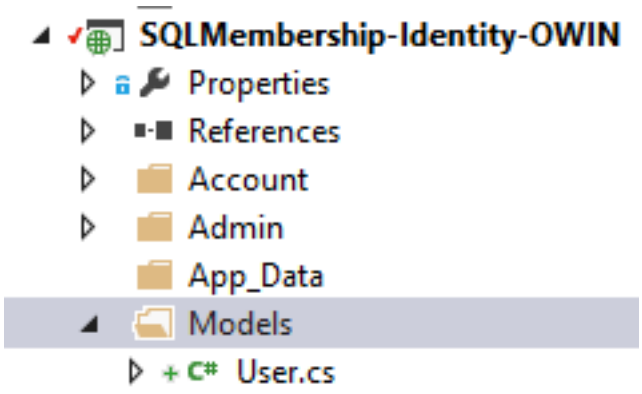
جداول AspNetUserClaims و AspNetUserLogins خالی هستند. فیلد تفکیک کننده (Discriminator) در جدول AspNetUsers باید مطابق نام کلاس مدل باشد، که در مرحله بعدی تعریف خواهد شد. همچنین ستون PasswordHash به فرم 'encrypted password|password salt|password format' می‌باشد. این شما را قادر می‌سازد تا از رمزنگاری برای ذخیره و بازیابی کلمه‌های عبور استفاده کنید. این مورد نیز در ادامه مقاله بررسی شده است.

ساختن مدل‌ها و صفحات عضویت

بصورت پیش فرض سیستم ASP.NET Identity برای دریافت و ذخیره اطلاعات در دیتابیس عضویت از Entity Framework استفاده می‌کند. برای آنکه بتوانیم با جداول موجود کار کنیم، می‌بایست ابتدا مدل هایی که الگوی دیتابیس را نمایندگی می‌کنند ایجاد کنیم. برای این کار مدل‌های ما یا باید اینترفیس‌های موجود در Identity.Core را پیاده سازی کنند، یا می‌توانند پیاده سازی‌های پیش فرض را توسعه دهند. پیاده سازی‌های پیش فرض در Microsoft.AspNet.Identity.EntityFramework وجود دارند.

در نمونه ما، جداول AspNetLogins, AspNetUserClaims, AspNetRoles و AspNetUserRole ستون هایی دارند که شباهت زیادی به پیاده سازی‌های پیش فرض سیستم Identity دارند. در نتیجه می‌توانیم از کلاس‌های موجود، برای Map کردن الگوی جدید استفاده کنیم. جدول AspNetUsers ستون‌های جدیدی نیز دارد. می‌توانیم کلاس جدیدی بسازیم که از **IdentityUser** ارث بری کند و آن را گسترش دهیم تا این فیلدهای جدید را پوشش دهد.

یوشه ای با نام Models بسازید (در صورتی که وجود ندارد) و کلاسی با نام User به آن اضافه کنید.



کلاس User باید کلاس IdentityUser را که در اسمبلی Microsoft.AspNet.Identity.EntityFramework وجود دارد گسترش دهد. خاصیت هایی را تعریف کنید که نماینده الگوی جدول AspNetUser هستند. خواص ID, Username, PasswordHash و SecurityStamp در کلاس IdentityUser تعریف شده اند، بنابراین این خواص را در لیست زیر نمی‌بینید.

```
public class User : IdentityUser
    {
        public User()
        {
            CreateDate = DateTime.Now;
            IsApproved = false;
            LastLoginDate = DateTime.Now;
            LastActivityDate = DateTime.Now;
            LastPasswordChangedDate = DateTime.Now;
            LastLockoutDate = DateTime.Parse("1/1/1754");
            FailedPasswordAnswerAttemptWindowStart = DateTime.Parse("1/1/1754");
            FailedPasswordAttemptWindowStart = DateTime.Parse("1/1/1754");
        }

        public System.Guid ApplicationId { get; set; }
        public string MobileAlias { get; set; }
        public bool IsAnonymous { get; set; }
        public System.DateTime LastActivityDate { get; set; }
        public string MobilePIN { get; set; }
        public string Email { get; set; }
        public string LoweredEmail { get; set; }
        public string LoweredUserName { get; set; }
        public string PasswordQuestion { get; set; }
        public string PasswordAnswer { get; set; }
        public bool IsApproved { get; set; }
        public bool IsLockedOut { get; set; }
        public System.DateTime CreateDate { get; set; }
        public System.DateTime LastLoginDate { get; set; }
        public System.DateTime LastPasswordChangedDate { get; set; }
        public System.DateTime LastLockoutDate { get; set; }
        public int FailedPasswordAttemptCount { get; set; }
        public System.DateTime FailedPasswordAttemptWindowStart { get; set; }
        public int FailedPasswordAnswerAttemptCount { get; set; }
        public System.DateTime FailedPasswordAnswerAttemptWindowStart { get; set; }
        public string Comment { get; set; }
    }
```

حال برای دسترسی به دیتابیس مورد نظر، نیاز به یک DbContext داریم. اسمبلی Microsoft.AspNet.Identity.EntityFramework کلاسی با نام **IdentityDbContext** دارد که پیاده سازی پیش فرض برای دسترسی به دیتابیس ASP.NET Identity است. نکته قابل توجه این است که IdentityDbContext آبجکتی از نوع **TUser** را می‌پذیرد. TUser می‌تواند هر کلاسی باشد که از IdentityUser ارث بری کرده و آن را گسترش می‌دهد.

در پوشه Models کلاس جدیدی با نام **ApplicationDbContext** بسازید که از IdentityDbContext ارث بری کرده و از کلاس

User استفاده می‌کند.

```
public class ApplicationDbContext : IdentityDbContext<User>
{

}
```

مدیریت کاربران در ASP.NET Identity توسط کلاسی با نام UserManager انجام می‌شود که در اسمبلی Microsoft.AspNet.Identity.EntityFramework قرار دارد. چیزی که ما در این مرحله نیاز داریم، کلاسی است که از UserManager ارث بری می‌کند و آن را طوری توسعه می‌دهد که از کلاس User استفاده کند.

در پوشه Models کلاس جدیدی با نام UserManager بسازید.

```
public class UserManager : UserManager<User>
{

}
```

کلمه عبور کاربران بصورت رمز نگاری شده در دیتابیس ذخیره می‌شوند. الگوریتم رمز نگاری SQL Membership با سیستم ASP.NET Identity تفاوت دارد. هنگامی که کاربران قدیمی به سایت وارد می‌شوند، کلمه عبورشان را توسط الگوریتم‌های قدیمی SQL Membership رمزگشایی می‌کنیم، اما کاربران جدید از الگوریتم‌های ASP.NET Identity استفاده خواهند کرد.

کلاس UserManager خاصیتی با نام **PasswordHasher** دارد. این خاصیت نمونه ای از یک کلاس را ذخیره می‌کند، که اینترفیس IPasswordHasher را پیاده سازی کرده است. این کلاس هنگام تراکنش‌های احراز هویت کاربران استفاده می‌شود تا کلمه‌های عبور را رمزنگاری/رمزگشایی شوند. در کلاس UserManager کلاس جدیدی بنام SQLPasswordHasher بسازید. کد کامل را در لیست زیر مشاهده می‌کنید.

```
public class SQLPasswordHasher : PasswordHasher
{
        public override string HashPassword(string password)
        {
            return base.HashPassword(password);
        }

        public override PasswordVerificationResult VerifyHashedPassword(string hashedPassword, string
providedPassword)
        {
            string[] passwordProperties = hashedPassword.Split('|');
            if (passwordProperties.Length != 3)
            {
                return base.VerifyHashedPassword(hashedPassword, providedPassword);
            }
            else
            {
                string passwordHash = passwordProperties[0];
                int passwordformat = 1;
                string salt = passwordProperties[2];
                if (String.Equals(EncryptPassword(providedPassword, passwordformat, salt),
passwordHash, StringComparison.CurrentCultureIgnoreCase))
                {
                    return PasswordVerificationResult.SuccessRehashNeeded;
                }
                else
                {
                    return PasswordVerificationResult.Failed;
                }
            }
        }

        //This is copied from the existing SQL providers and is provided only for back-compat.
        private string EncryptPassword(string pass, int passwordFormat, string salt)
        {
            if (passwordFormat == 0) // MembershipPasswordFormat.Clear
                return pass;

            byte[] bIn = Encoding.Unicode.GetBytes(pass);
```

```
            byte[] bSalt = Convert.FromBase64String(salt);
            byte[] bRet = null;

            if (passwordFormat == 1)
            { // MembershipPasswordFormat.Hashed
                HashAlgorithm hm = HashAlgorithm.Create("SHA1");
                if (hm is KeyedHashAlgorithm)
                {
                    KeyedHashAlgorithm kha = (KeyedHashAlgorithm)hm;
                    if (kha.Key.Length == bSalt.Length)
                    {
                        kha.Key = bSalt;
                    }
                    else if (kha.Key.Length < bSalt.Length)
                    {
                        byte[] bKey = new byte[kha.Key.Length];
                        Buffer.BlockCopy(bSalt, 0, bKey, 0, bKey.Length);
                        kha.Key = bKey;
                    }
                    else
                    {
                        byte[] bKey = new byte[kha.Key.Length];
                        for (int iter = 0; iter < bKey.Length; )
                        {
                            int len = Math.Min(bSalt.Length, bKey.Length - iter);
                            Buffer.BlockCopy(bSalt, 0, bKey, iter, len);
                            iter += len;
                        }
                        kha.Key = bKey;
                    }
                    bRet = kha.ComputeHash(bIn);
                }
                else
                {
                    byte[] bAll = new byte[bSalt.Length + bIn.Length];
                    Buffer.BlockCopy(bSalt, 0, bAll, 0, bSalt.Length);
                    Buffer.BlockCopy(bIn, 0, bAll, bSalt.Length, bIn.Length);
                    bRet = hm.ComputeHash(bAll);
                }
            }

            return Convert.ToBase64String(bRet);
    }
}
```

دقت کنید تا فضاهای نام System.Text و System.Security.Cryptography را وارد کرده باشید.

متد EncodePassword کلمه عبور را بر اساس پیاده سازی پیش فرض SQL Membership رمزنگاری می‌کند. این الگوریتم از System.Web گرفته می‌شود. اگر اپلیکیشن قدیمی شما از الگوریتم خاصی استفاده می‌کرده است، همینجا باید آن را منعکس کنید. دو متد دیگر نیز بنام‌های *HashPassword* و *VerifyHashedPassword* نیاز داریم. این متدها از EncodePassword برای رمزنگاری کلمه‌های عبور و تایید آنها در دیتابیس استفاده می‌کنند.

سیستم SQL Membership برای رمزنگاری (Hash) کلمه‌های عبور هنگام ثبت نام و تغییر آنها توسط کاربران، از PasswordHash, PasswordSalt و PasswordFormat استفاده می‌کرد. در روند مهاجرت، این سه فیلد در ستون PasswordHash جدول AspNetUsers ذخیره شده و با کاراکتر '|' جدا شده اند. هنگام ورود کاربری به سایت، اگر کله عبور شامل این فیلدها باشد از الگوریتم SQL Membership برای بررسی آن استفاده می‌کنیم. در غیر اینصورت از پیاده سازی پیش فرض ASP.NET Identity استفاده خواهد شد. با این روش، کاربران قدیمی لازم نیست کلمه‌های عبور خود را صرفا بدلیل مهاجرت اپلیکیشن ما تغییر دهند.

کلاس UserManager را مانند قطعه کد زیر بروز رسانی کنید.

```
public UserManager()
            : base(new UserStore<User>(new ApplicationDbContext()))
        {
            this.PasswordHasher = new SQLPasswordHasher();
 }
```

ایجاد صفحات جدید مدیریت کاربران

قدم بعدی ایجاد صفحاتی است که به کاربران اجازه ثبت نام و ورود را می‌دهند. صفحات قدیمی SQL Membership از کنترل هایی استفاده می‌کنند که با ASP.NET Identity سازگار نیستند. برای ساختن این صفحات جدید به این مقاله مراجعه کنید. از آنجا که در این مقاله پروژه جدید را ساخته ایم و پکیج‌های لازم را هم نصب کرده ایم، می‌توانید مستقیما به قسمت Adding Web Forms for registering users to your application بروید.

چند تغییر که باید اعمال شوند:

فایل‌های Register.aspx.cs و Login.aspx.cs از کلاس UserManager استفاده می‌کنند. این ارجاعات را با کلاس UserManager جدیدی که در پوشه Models ساختید جایگزین کنید.

همچنین ارجاعات استفاده از کلاس IdentityUser را به کلاس User که در پوشه Models ساختید تغییر دهید.

لازم است توسعه دهنده مقدار ApplicationId را برای کاربران جدید طوری تنظیم کند که با شناسه اپلیکیشن جاری تطابق داشته باشد. برای این کار می‌توانید پیش از ساختن حساب‌های کاربری جدید در فایل Register.aspx.cs ابتدا شناسه اپلیکیشن را بدست آورید و اطلاعات کاربر را بدرستی تنظیم کنید.

مثال: در فایل Register.aspx.cs متد جدیدی تعریف کنید که جدول aspnet_Applications را بررسی میکند و شناسه اپلیکیشن را بر اساس نام اپلیکیشن بدست می‌آورد.

```
private Guid GetApplicationID()
{
    using (SqlConnection connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["ApplicationServices"].ConnectionString))
            {
                string queryString = "SELECT ApplicationId from aspnet_Applications WHERE
ApplicationName = '/'"; //Set application name as in database

                SqlCommand command = new SqlCommand(queryString, connection);
                command.Connection.Open();

                var reader = command.ExecuteReader();
                while (reader.Read())
                {
                    return reader.GetGuid(0);
                }

            return Guid.NewGuid();
        }
}
```

حال می‌توانید این مقدار را برای آبجکت کاربر تنظیم کنید.

```
var currentApplicationId = GetApplicationID();

User user = new User() { UserName = Username.Text,
ApplicationId=currentApplicationId, …};
```

در این مرحله می‌توانید با استفاده از اطلاعات پیشین وارد سایت شوید، یا کاربران جدیدی ثبت کنید. همچنین اطمینان حاصل کنید که کاربران پیشین در نقش‌های مورد نظر وجود دارند.

مهاجرت به ASP.NET Identity مزایا و قابلیت‌های جدیدی را به شما ارائه می‌کند. مثلا کاربران می‌توانند با استفاده از تامین کنندگان ثالثی مثل Facebook, Google, Microsoft, Twitter و غیره به سایت وارد شوند. اگر به سورس کد این مقاله مراجعه کنید خواهید دید که امکانات OAuth نیز فعال شده اند.

در این مقاله انتقال داده‌های پروفایل کاربران بررسی نشد. اما با استفاده از نکات ذکر شده می‌توانید پروفایل کاربران را هم بسادگی منتقل کنید. کافی است مدل‌های لازم را در پروژه خود تعریف کرده و با استفاده از اسکریپت‌های SQL داده‌ها را انتقال دهید.

نظرات خوانندگان

نویسنده: سمیرا قادری
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۳:۳۳

سلام ممنون از مطالب خوبتون .
زمانیکه در اپلکیشن خودم asp.net Configuration و تب Security را انتخاب می‌نمایم با پیغام زیر مواجه می‌شوم Unable to connect to SQL Server database.
دستور زیر را هم اجرا کردم
C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_regsql.exe -S . -A all -d tt -U sa_tt -P asd123
در صورتیکه از login و password آن مطمئن هستم ولی مشکل حل نشد
در صورتیکه مشکل از permission هست چگونه باید آن را حل کنم
webconfig برنامه به شرح زیر است
Data Source=.;Initial Catalog=tt;User ID=sa_tt;Password=asd123
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۴:۴۵

تنظیمات وب کانفیگ خودتون رو با نمونه MSDN برای تعریف membership مطابقت بدید. جزئیات قسمت‌های مختلف آن

نویسنده: حمید
تاریخ: ۱۳۹۲/۱۰/۲۳ ۲۱:۱۰

خیلی ممنون آموزش‌های مفیدتون!
فرض کنید ما یک سیستم Authentication مثل Membership داریم با 2 تا نقش. حالا می‌خواهیم نقش A به فرم F1 و بعضی از متدهاش دسترسی داشته و در فرم F2، نقش B به آن دسترسی داشته باشه. خوب تا اینجا فکر کنم پیاده سازیش راحت باشه. ولی مساله از جایی پیچیده میشه که ما بخواهیم در زمان اجرا این تنظیمات رو عوض کنیم مثلا: در فرم F1، نقش A دسترسیش به متدها و حتی به کل فرم تغییر کنه و یا اجازه دسترسی به فرم B بهش داده بشه و این شرایط رو برای چندین نقش و یا در سطح کاربر داشت توصیه چیه؟ از چه تکنولوژی و یا راهکاری میشه به این مقصود رسید.
خیلی ممنون

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۲۳ ۲۱:۳۷

پایه‌اش همین مسایل هست. فقط قسمتی که کار Authorization انجام میشه رو می‌توان سفارشی کرد. مثلا:

Dynamic Controller/Action Authorization in ASP.NET MVC
MVC Dynamic Authorization
ASP.NET MVC Custom Authorize Attribute with Roles Parser

نویسنده: کامران سادین
تاریخ: ۱۳۹۲/۱۱/۰۸ ۲:۱۳

ممنون از مطلب خوبتون.
من جداول خودم رو برای احراز هویت دارم آیا میشه همین جداول رو با ASP.NET Identity کار کنم؟
اگر نمیشه، این بانک ASP.NET Identity که توی SQL Server نمیسازه بانک Database.mdf رو میسازه، میشه توی SQL بسازیم از

همون ابتدای پروژه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۸ ۹:۲۰

سیستم کارش EF Code first هست. این سیستم کدهاش گره خورده به بانک اطلاعاتی خاصی نیست. الان در این مثال رشته اتصالی به یک localdb اشاره می‌کنه. شما می‌تونید کلا این رشته و نحوه‌ی تعریف اون رو برای کار با SQL Server یا SQL CE یا هر بانک اطلاعاتی دیگری که پروایدر code first داره، تغییر بدید و استفاده کنید. (و اگر با ef code first آشنایی ندارید، کم کم در آینده نمی‌تونید با کتابخانه‌های کمکی و جانبی دات نت کار کنید)

نویسنده: کامران سادین
تاریخ: ۱۳۹۲/۱۱/۰۸ ۱۰:۲۷

مقاله ای نیز راجع به تغییر فیلدهاش هم بنویسید بد نیست دوستان علاقه دارند. با تشکر.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۸ ۱۰:۵۸

« سفارشی کردن ASP.NET Identity در MVC 5 » بیشتر در اینجا

نویسنده: مهدی
تاریخ: ۱۳۹۳/۰۱/۱۹ ۱۲:۳۴

دوست عزیز برای حل این مشکل شما باید پوشه سورس برنامه رو تغییر بدین. این مشکل به دلیل وجود خط فاصله تو آدرس (دایرکتوری) محل قرار گیری پروژه هست. اگه محل پروژه رو در جایی قرار دهید که در آدرس آن از فاصله استفاده نشده باشد به خوبی کار خواهد کرد.

عنوان: **مهاجرت داده عضویت و پروفایل از Universal Providers به ASP.NET Identity**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۲۲ ۶:۵۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MemberShip, Identity, ASP.NET Identity, Universal Providers

در این مقاله مهاجرت داده‌های سیستم عضویت، نقش‌ها و پروفایل‌های کاربران که توسط Universal Providers ساخته شده اند به مدل ASP.NET Identity را بررسی می‌کنیم. رویکردی که در این مقاله استفاده شده و قدم‌های لازمی که توضیح داده شده اند، برای اپلیکیشنی که با SQL Membership کار می‌کند هم می‌توانند کارساز باشند.
با انتشار Visual Studio 2013، تیم ASP.NET سیستم جدیدی با نام ASP.NET Identity معرفی کردند. می‌توانید در این لینک بیشتر درباره این انتشار بخوانید.
در ادامه مقاله قبلی تحت عنوان مهاجرت از SQL Membership به ASP.NET Identity ، در این پست به مهاجرت داده‌های یک اپلیکیشن که از مدل Providers برای مدیریت اطلاعات کاربران، نقش‌ها و پروفایل‌ها استفاده می‌کند به مدل جدید ASP.NET Identity می‌پردازیم. تمرکز این مقاله اساسا روی مهاجرت داده‌های پروفایل کاربران خواهد بود، تا بتوان به سرعت از آنها در اپلیکیشن استفاده کرد. مهاجرت داده‌های عضویت و نقش ها، شبیه پروسه مهاجرت SQL Membership است. رویکردی که در ادامه برای مهاجرت داده پروفایل‌ها دنبال شده است، می‌تواند برای اپلیکیشنی با SQL Membership نیز استفاده شود.
بعنوان یک مثال، با اپلیکیشن وبی شروع می‌کنیم که توسط Visual Studio 2012 ساخته شده و از مدل Providers استفاده می‌کند. پس از آن یک سری کد برای مدیریت پروفایل ها، ثبت نام کاربران، افزودن اطلاعات پروفایل به کاربران و مهاجرت الگوی دیتابیس می‌نویسیم و نهایتا اپلیکیشن را بروز رسانی می‌کنیم تا برای استفاده از سیستم Identity برای مدیریت کاربران و نقش‌ها آماده باشد. و بعنوان یک تست، کاربرانی که قبلا توسط Universal Providers ساخته شده اند باید بتوانند به سایت وارد شوند، و کاربران جدید هم باید قادر به ثبت نام در سایت باشند.
سورس کد کامل این مثال را می‌توانید از این لینک دریافت کنید.

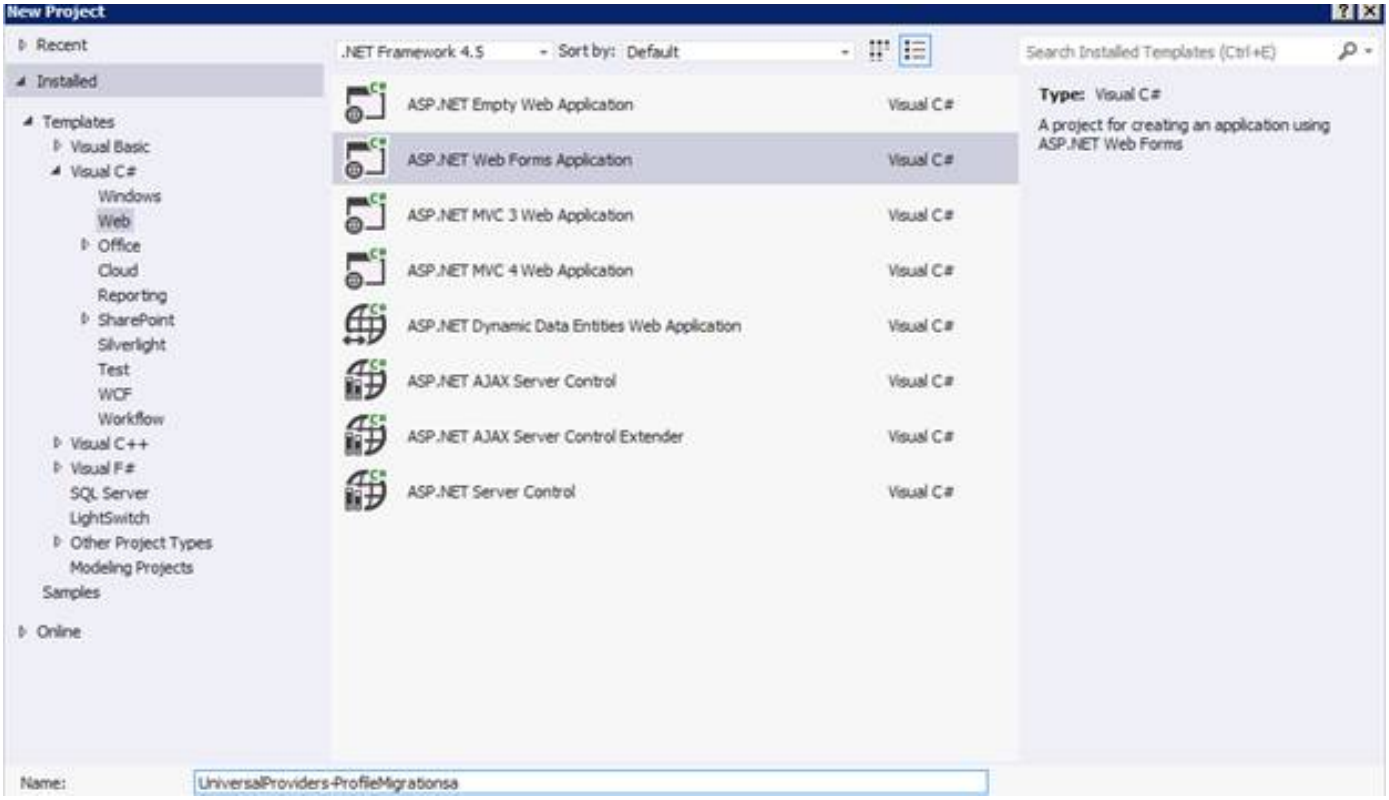
خلاصه مهاجرت داده پروفایل ها

قبل از آنکه با مهاجرت‌ها شروع کنیم، بگذارید تا نگاهی به تجربه مان از ذخیره اطلاعات پروفایل‌ها در مدل Providers بیاندازیم.
اطلاعات پروفایل کاربران یک اپلیکیشن به طرق مختلفی می‌تواند ذخیره شود. یکی از رایج‌ترین این راه ها، استفاده از تامین کننده‌های پیش فرضی است که بهمراه Universal Providers منتشر شدند. بدین منظور انجام مراحل زیر لازم است
کلاس جدیدی بسازید که دارای خواصی برای ذخیره اطلاعات پروفایل است.
کلاس جدیدی بسازید که از 'ProfileBase' ارث بری می‌کند و متدهای لازم برای دریافت پروفایل کاربران را پیاده سازی می‌کند.
استفاده از تامین کننده‌های پیش فرض را، در فایل web.config فعال کنید. و کلاسی که در مرحله 2 ساختید را بعنوان کلاس پیش فرض برای خواندن اطلاعات پروفایل معرفی کنید.

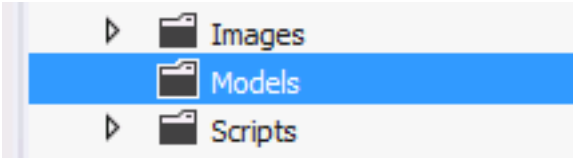
اطلاعات پروفایل‌ها بصورت xml serialized و binary در جدول 'Profiles' ذخیره می‌شوند.

پس از آنکه به سیستم ASP.NET Identity مهاجرت کردیم، اطلاعات پروفایل deserialized شده و در قالب خواص کلاس User ذخیره می‌شوند. هر خاصیت، بعدا می‌تواند به یک ستون در دیتابیس متصل شود. مزیت بدست آمده این است که مستقیما از کلاس User به اطلاعات پروفایل دسترسی داریم. ناگفته نماند که دیگر داده‌ها serialize/deserialize هم نمی‌شوند.

شروع به کار در Visual Studio 2012 پروژه جدیدی از نوع ASP.NET 4.5 Web Forms application بسازید. مثال جاری از یک قالب Web Forms استفاده می‌کند، اما می‌توانید از یک قالب MVC هم استفاده کنید.



پوشه جدیدی با نام 'Models' بسازید تا اطلاعات پروفایل را در آن قرار دهیم.



بعنوان یک مثال، بگذارید تا تاریخ تولد کاربر، شهر سکونت، قد و وزن او را در پروفایلش ذخیره کنیم. قد و وزن بصورت یک کلاس سفارشی (custom class) بنام 'PersonalStats' ذخیره می‌شوند. برای ذخیره و بازیابی پروفایل ها، به کلاسی احتیاج داریم که 'ProfileBase' را ارث بری می‌کند. پس کلاس جدیدی با نام 'AppProfile' بسازید.

```
public class ProfileInfo
{
    public ProfileInfo()
    {
        UserStats = new PersonalStats();
    }
    public DateTime? DateOfBirth { get; set; }
    public PersonalStats UserStats { get; set; }
    public string City { get; set; }
}

public class PersonalStats
{
    public int? Weight { get; set; }
    public int? Height { get; set; }
}

public class AppProfile : ProfileBase
{
    public ProfileInfo ProfileInfo
```

```
    {
        get { return (ProfileInfo)GetPropertyValue("ProfileInfo"); }
    }
    public static AppProfile GetProfile()
    {
        return (AppProfile)HttpContext.Current.Profile;
    }
    public static AppProfile GetProfile(string userName)
    {
        return (AppProfile)Create(userName);
    }
}
```

پروفایل را در فایل web.config خود فعال کنید. نام کلاسی را که در مرحله قبل ساختید، بعنوان کلاس پیش فرض برای ذخیره و بازیابی پروفایل‌ها معرفی کنید.

```
<profile defaultProvider="DefaultProfileProvider" enabled="true"
    inherits="UniversalProviders_ProfileMigrations.Models.AppProfile">
  <providers>
    .....
  </providers>
</profile>
```

برای دریافت اطلاعات پروفایل از کاربر، فرم وب جدیدی در پوشه Account بسازید و آنرا 'AddProfileData.aspx' نامگذاری کنید.

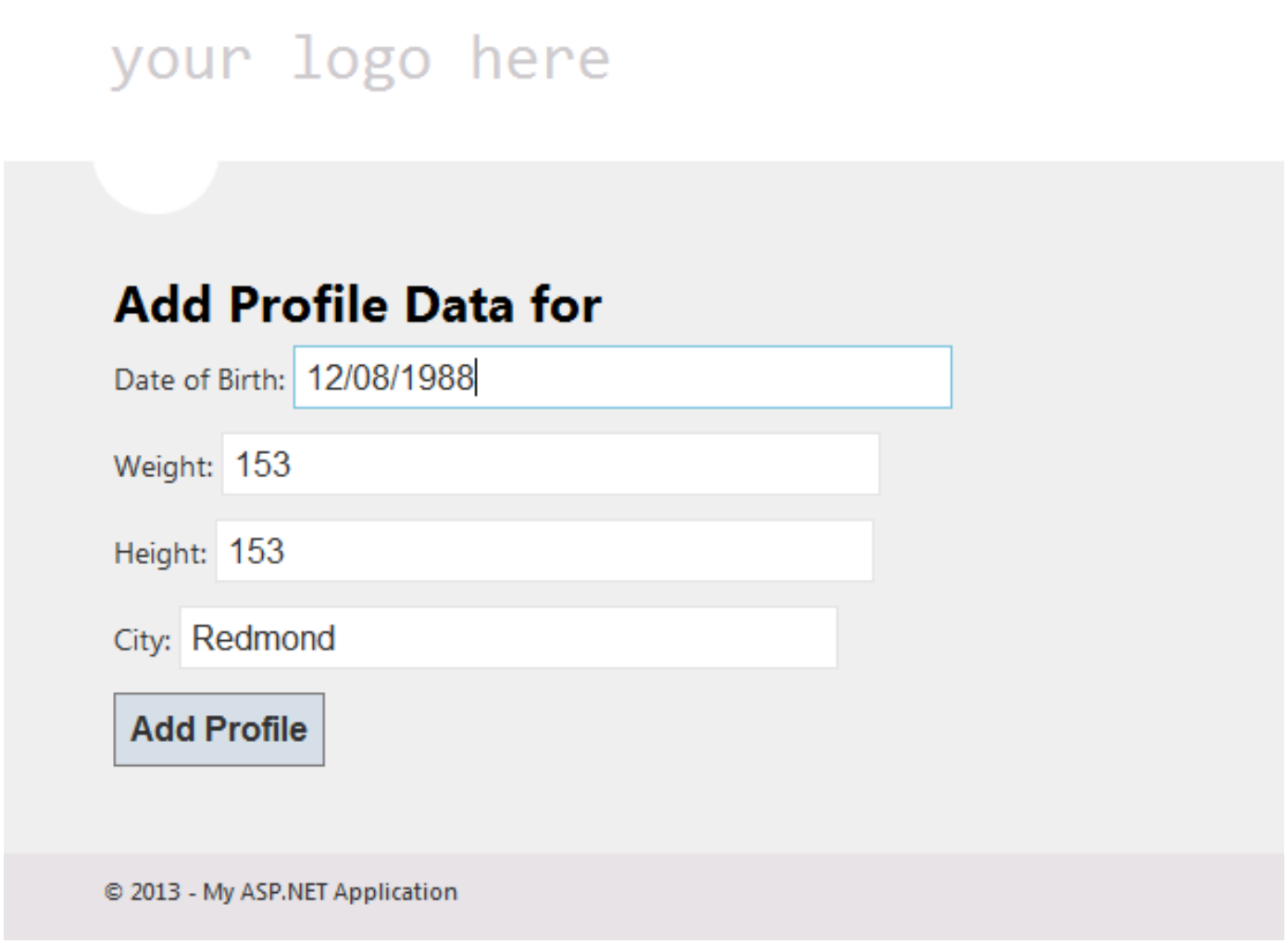
```
<h2> Add Profile Data for <%# User.Identity.Name %></h2>
<asp:Label Text="" ID="Result" runat="server" />
<div>
    Date of Birth:
    <asp:TextBox runat="server" ID="DateOfBirth"/>
</div>
<div>
    Weight:
    <asp:TextBox runat="server" ID="Weight"/>
</div>
<div>
    Height:
    <asp:TextBox runat="server" ID="Height"/>
</div>
<div>
    City:
    <asp:TextBox runat="server" ID="City"/>
</div>
<div>
    <asp:Button Text="Add Profile" ID="Add" OnClick="Add_Click" runat="server" />
</div>
```

کد زیر را هم به فایل code-behind اضافه کنید.

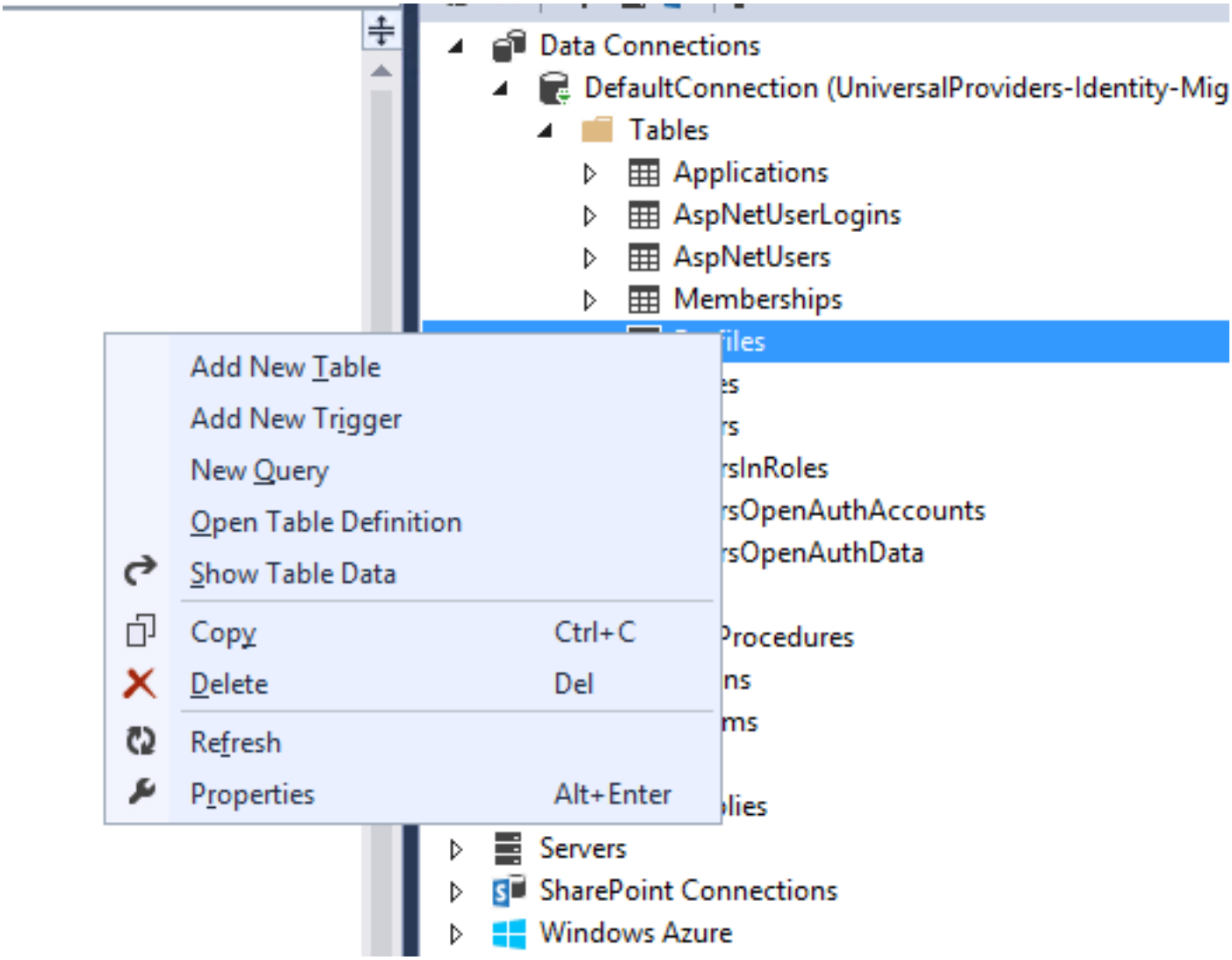
```
protected void Add_Click(object sender, EventArgs e)
{
    AppProfile profile = AppProfile.GetProfile(User.Identity.Name);
    profile.ProfileInfo.DateOfBirth = DateTime.Parse(DateOfBirth.Text);
    profile.ProfileInfo.UserStats.Weight = Int32.Parse(Weight.Text);
    profile.ProfileInfo.UserStats.Height = Int32.Parse(Height.Text);
    profile.ProfileInfo.City = City.Text;
    profile.Save();
}
```

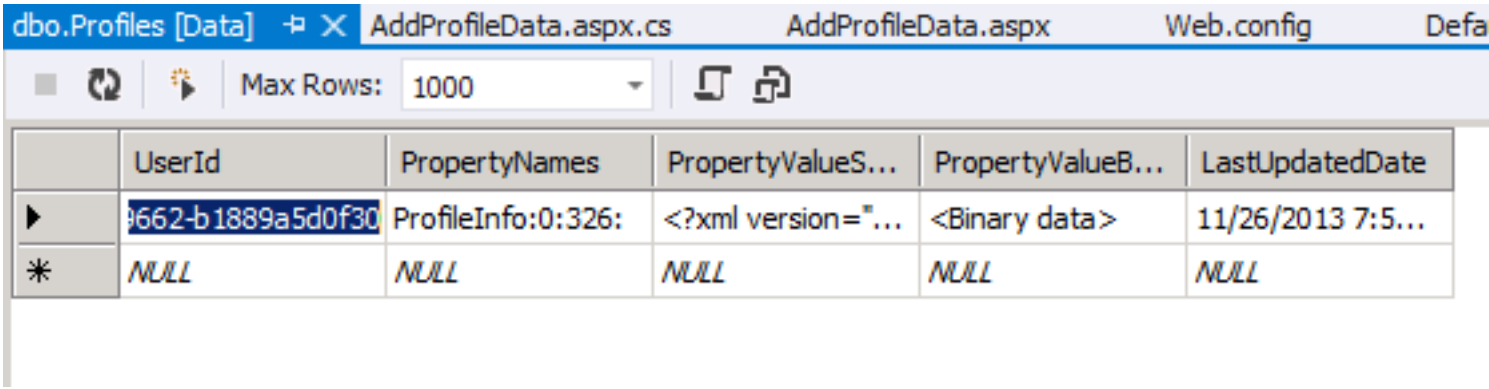
دقت کنید که فضای نامی که کلاس AppProfile در آن قرار دارد را وارد کرده باشید.

اپلیکیشن را اجرا کنید و کاربر جدیدی با نام ' **olduser** ' بسازید. به صفحه جدید 'AddProfileData' بروید و اطلاعات پروفایل کاربر را وارد کنید.



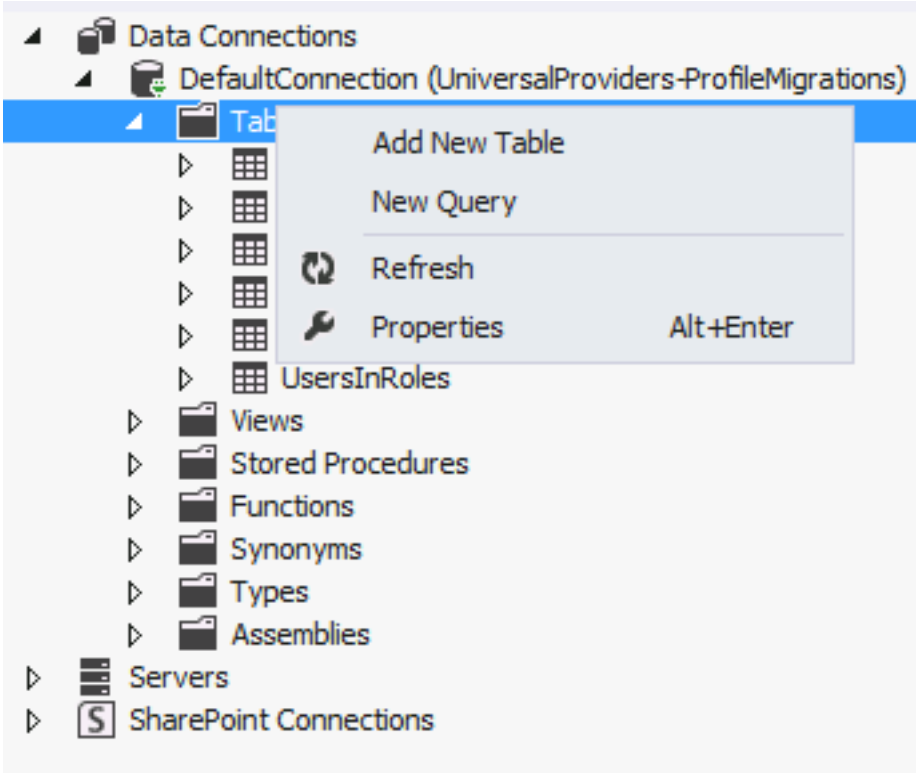
با استفاده از پنجره Server Explorer می‌توانید تایید کنید که اطلاعات پروفایل با فرمت xml در جدول 'Profiles' ذخیره می‌شوند.



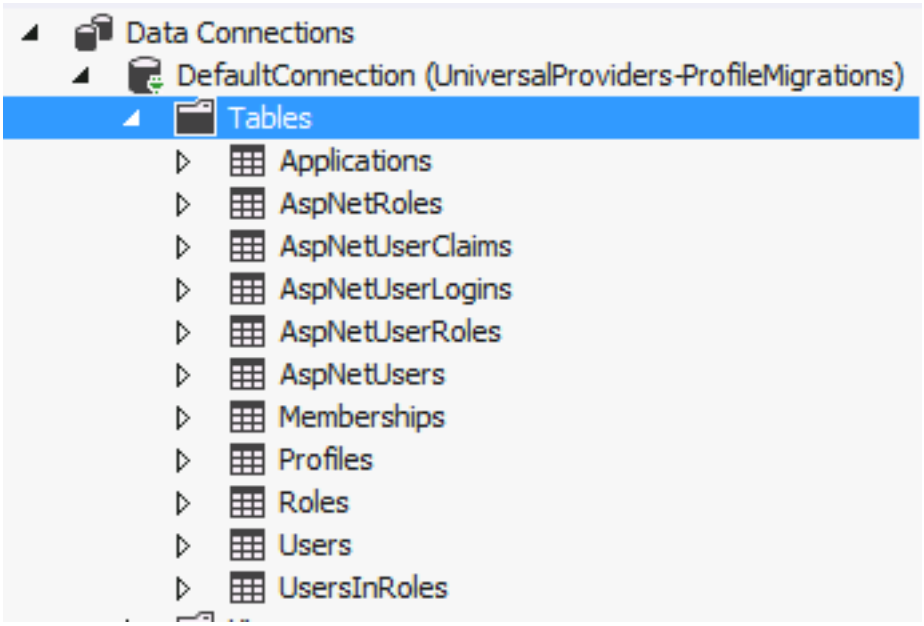


مهاجرت الگوی دیتابیس

برای اینکه دیتابیس فعلی بتواند با سیستم ASP.NET Identity کار کند، باید الگوی ASP.NET Identity را بروز رسانی کنیم تا فیلدهای جدیدی که اضافه کردیم را هم در نظر بگیرد. این کار می‌تواند توسط اسکریپت‌های SQL انجام شود، باید جداول جدیدی بسازیم و اطلاعات موجود را به آنها انتقال دهیم. در پنجره 'Server Explorer' گره 'DefaultConnection' را باز کنید تا جداول لیست شوند. روی Tables کلیک راست کنید و 'New Query' را انتخاب کنید.



اسکریپت مورد نیاز را از آدرس https://raw.github.com/suhasj/UniversalProviders-Identity-Migrations/master/Migration.txt دریافت کرده و آن را اجرا کنید. اگر اتصال خود به دیتابیس را تازه کنید خواهید دید که جداول جدیدی اضافه شده اند. می‌توانید داده‌های این جداول را بررسی کنید تا ببینید چگونه اطلاعات منتقل شده اند.



مهاجرت اپلیکیشن برای استفاده از ASP.NET Identity

پکیج‌های مورد نیاز برای ASP.NET Identity را نصب کنید:

Microsoft.AspNet.Identity.EntityFramework
Microsoft.AspNet.Identity.Owin
Microsoft.Owin.Host.SystemWeb
Microsoft.Owin.Security.Facebook
Microsoft.Owin.Security.Google
Microsoft.Owin.Security.MicrosoftAccount
Microsoft.Owin.Security.Twitter

اطلاعات بیشتری درباره مدیریت پکیج‌های NuGet از اینجا قابل دسترسی هستند.
برای اینکه بتوانیم از الگوی جاری دیتابیس استفاده کنیم، ابتدا باید مدل‌های لازم ASP.NET Identity را تعریف کنیم تا موجودیت‌های دیتابیس را Map کنیم. طبق قرارداد سیستم Identity کلاس‌های مدل یا باید اینترفیس‌های تعریف شده در Identity.Core dll را پیاده سازی کنند، یا می‌توانند پیاده سازی‌های پیش فرضی را که در Microsoft.AspNet.Identity.EntityFramework وجود دارند گسترش دهند. ما برای نقش ها، اطلاعات ورود کاربران و claimها از پیاده سازی‌های پیش فرض استفاده خواهیم کرد. نیاز به استفاده از یک کلاس سفارشی User داریم. پوشه جدیدی در پروژه با نام 'IdentityModels' بسازید. کلاسی با نام 'User' در این پوشه بسازید و کد آن را با لیست زیر تطابق دهید.

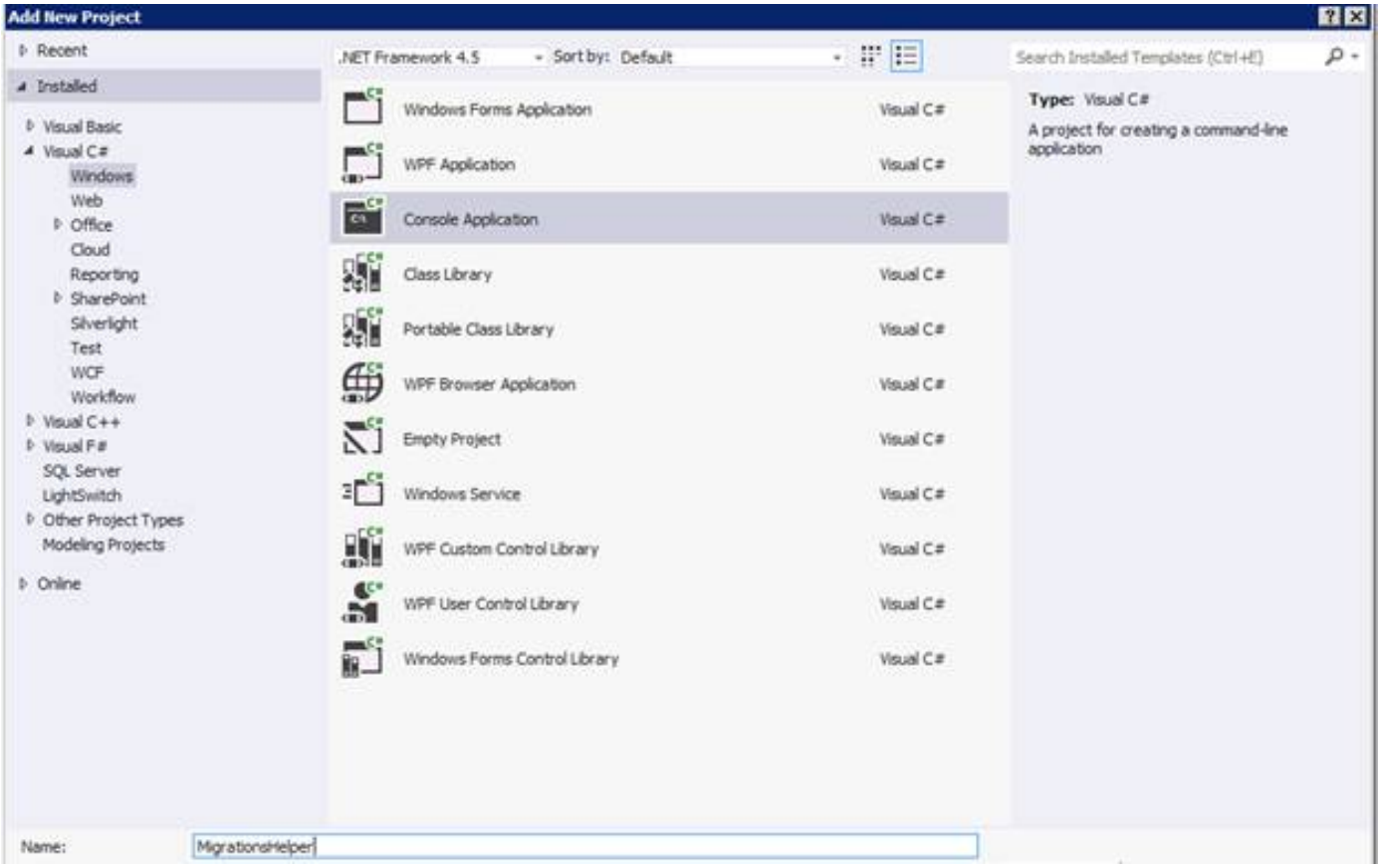
```
using Microsoft.AspNet.Identity.EntityFramework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using UniversalProviders_ProfileMigrations.Models;

namespace UniversalProviders_Identity_Migrations
{
    public class User : IdentityUser
    {
        public User()
        {
            CreateDate = DateTime.UtcNow;
            IsApproved = false;
            LastLoginDate = DateTime.UtcNow;
            LastActivityDate = DateTime.UtcNow;
            LastPasswordChangedDate = DateTime.UtcNow;
            Profile = new ProfileInfo();
        }

        public System.Guid ApplicationId { get; set; }
        public bool IsAnonymous { get; set; }
        public System.DateTime? LastActivityDate { get; set; }
        public string Email { get; set; }
        public string PasswordQuestion { get; set; }
        public string PasswordAnswer { get; set; }
        public bool IsApproved { get; set; }
        public bool IsLockedOut { get; set; }
        public System.DateTime? CreateDate { get; set; }
        public System.DateTime? LastLoginDate { get; set; }
        public System.DateTime? LastPasswordChangedDate { get; set; }
        public System.DateTime? LastLockoutDate { get; set; }
        public int FailedPasswordAttemptCount { get; set; }
        public System.DateTime? FailedPasswordAttemptWindowStart { get; set; }
        public int FailedPasswordAnswerAttemptCount { get; set; }
        public System.DateTime? FailedPasswordAnswerAttemptWindowStart { get; set; }
        public string Comment { get; set; }
        public ProfileInfo Profile { get; set; }
    }
}
```

دقت کنید که 'ProfileInfo' حالا بعنوان یک خاصیت روی کلاس User تعریف شده است. بنابراین می‌توانیم مستقیما از کلاس کاربر با اطلاعات پروفایل کار کنیم.
محتویات پوشه‌های **IdentityModels** و **IdentityAccount** را از آدرس https://github.com/suhasj/UniversalProviders-Identity-Migrations/tree/master/UniversalProviders-Identity-Migrations دریافت و کپی کنید. این فایل‌ها مابقی مدل ها، و صفحاتی برای مدیریت کاربران و نقش‌ها در سیستم جدید ASP.NET Identity هستند.

انتقال داده پروفایل‌ها به جداول جدید

همانطور که گفته شد ابتدا باید داده‌های پروفایل را deserialize کرده و از فرمت xml خارج کنیم، سپس آنها را در ستون‌های جدول AspNetUsers ذخیره کنیم. ستون‌های جدید در مرحله قبل به دیتابیس اضافه شدند، پس تنها کاری که باقی مانده پر کردن این ستون‌ها با داده‌های ضروری است. بدین منظور ما از یک اپلیکیشن کنسول استفاده می‌کنیم که تنها یک بار اجرا خواهد شد، و ستون‌های جدید را با داده‌های لازم پر می‌کند.

در solution جاری یک پروژه اپلیکیشن کنسول بسازید.



آخرین نسخه پکیج Entity Framework را نصب کنید. همچنین یک رفرنس به اپلیکیشن وب پروژه بدهید (کلیک راست روی پروژه و گزینه 'Add Reference').

کد زیر را در کلاس Program.cs وارد کنید. این قطعه کد پروفایل تک تک کاربران را می‌خواند و در قالب 'ProfileInfo' آنها را serialize می‌کند و در دیتابیس ذخیره می‌کند.

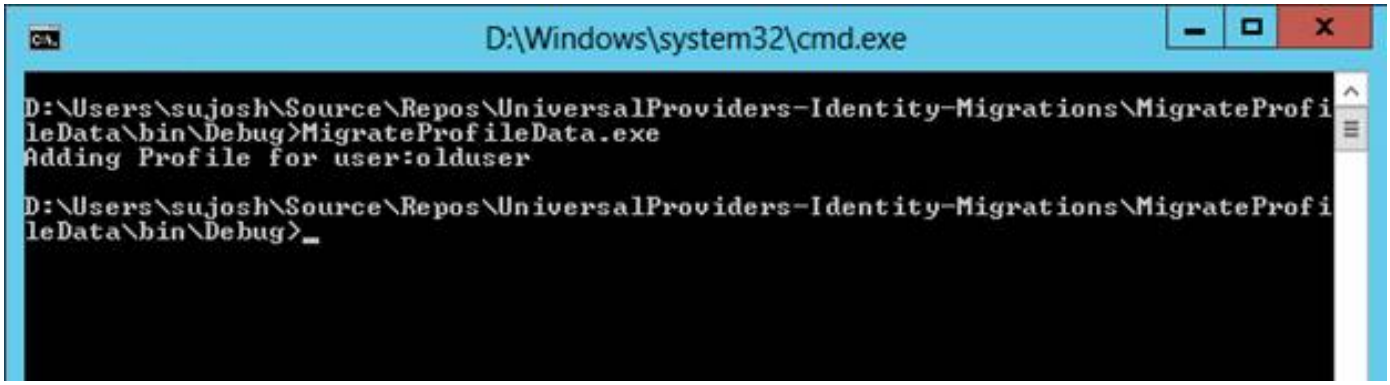
```
public class Program
{
    var dbContext = new ApplicationDbContext();
    foreach (var profile in dbContext.Profiles)
    {
        var stringId = profile.UserId.ToString();
        var user = dbContext.Users.Where(x => x.Id == stringId).FirstOrDefault();
        Console.WriteLine("Adding Profile for user:" + user.UserName);
        var serializer = new XmlSerializer(typeof(ProfileInfo));
        var stringReader = new StringReader(profile.PropertyValueStrings);
        var profileData = serializer.Deserialize(stringReader) as ProfileInfo;
        if (profileData == null)
        {
            Console.WriteLine("Profile data deserialization error for user:" + user.UserName);
        }
        else
```

```
        {
            user.Profile = profileData;
        }
    }
    dbContext.SaveChanges();
}
```

برخی از مدل‌های استفاده شده در پوشه 'IdentityModels' تعریف شده اند که در پروژه اپلیکیشن وبمان قرار دارند، بنابراین افزودن فضاهای نام مورد نیاز فراموش نشود.

کد بالا روی دیتابیسی که در پوشه App_Data وجود دارد کار می‌کند، این دیتابیس در مراحل قبلی در اپلیکیشن وب پروژه ایجاد شد. برای اینکه این دیتابیس را رفرنس کنیم باید رشته اتصال فایل app.config اپلیکیشن کنسول را بروز رسانی کنید. از همان رشته اتصال web.config در اپلیکیشن وب پروژه استفاده کنید. همچنین آدرس فیزیکی کامل را در خاصیت 'AttachDbFilename' وارد کنید.

یک Command Prompt باز کنید و به پوشه bin اپلیکیشن کنسول بالا بروید. فایل اجرایی را اجرا کنید و نتیجه را مانند تصویر زیر بررسی کنید.



در پنجره Server Explorer جدول 'AspNetUsers' را باز کنید. حال ستون‌های این جدول باید خواص کلاس مدل را منعکس کنند.

کارایی سیستم را تایید کنید

با استفاده از صفحات جدیدی که برای کار با ASP.NET Identity پیاده سازی شده اند سیستم را تست کنید. با کاربران قدیمی که در دیتابیس قبلی وجود دارند وارد شوید. کاربران باید با همان اطلاعات پیشین بتوانند وارد سیستم شوند. مابقی قابلیت‌ها را هم بررسی کنید. مثلا افزودن OAuth، ثبت کاربر جدید، تغییر کلمه عبور، افزودن نقش ها، تخصیص کاربران به نقش‌ها و غیره.
داده‌های پروفایل کاربران قدیمی و جدید همگی باید در جدول کاربران ذخیره شده و بازیابی شوند. جدول قبلی دیگر نباید رفرنس شود.

عنوان: **انتشار پیش نمایش ASP.NET Identity 2.0.0-alpha1**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۶:۳۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MemberShip, Visual Studio 2013, ASP.NET Identity

مایکروسافت در تاریخ 20 دسامبر 2013 پیش نمایش نسخه جدید ASP.NET Identity را معرفی کرد. تمرکز اصلی در این انتشار، رفع مشکلات نسخه 1.0 بود. امکانات جدیدی هم مانند Account Confirmation و Password Reset اضافه شده اند.

دانلود این انتشار

ASP.NET Identity را می‌توانید در قالب یک پکیج NuGet دریافت کنید. در پنجره Manage NuGet Packages می‌توانید پکیج‌های Preview را لیست کرده و گزینه مورد نظر را

نصب کنید. برای نصب پکیج‌های pre-release توسط Package Manager Console از فرامین زیر استفاده کنید.

```
Install-Package Microsoft.AspNet.Identity.EntityFramework -Version 2.0.0-alpha1 -Pre
Install-Package Microsoft.AspNet.Identity.Core -Version 2.0.0-alpha1 -Pre
Install-Package Microsoft.AspNet.Identity.OWIN -Version 2.0.0-alpha1 -Pre
```

دقت کنید که حتما از گزینه "Include Prerelease" استفاده می‌کنید. برای اطلاعات بیشتر درباره نصب پکیج‌های Pre-release لطفا به این لینک و یا این لینک مراجعه کنید.

در ادامه لیست امکانات جدید و مشکلات رفع شده را می‌خوانید.

Account Confirmation

سیستم ASP.NET Identity حالا از Account Confirmation پشتیبانی می‌کند. این یک سناریوی بسیار رایج است. در اکثر وب سایت‌های امروزی پس از ثبت نام، حتما باید ایمیل خود را تایید کنید. پیش از تایید ثبت نام قادر به انجام هیچ کاری در وب سایت نخواهید بود، یعنی نمی‌توانید Login کنید. این روش مفید است، چرا که از ایجاد حساب‌های کاربری نامعتبر (bogus) جلوگیری می‌کند. همچنین این روش برای برقراری ارتباط با کاربران هم بسیار کارآمد است. از آدرس‌های ایمیل کاربران می‌توانید در وب سایت‌های فروم، شبکه‌های اجتماعی، تجارت آنلاین و بانکداری برای اطلاع رسانی و دیگر موارد استفاده کنید.

نکته: برای ارسال ایمیل باید تنظیمات SMTP را پیکربندی کنید. مثلا می‌توانید از سرویس‌های ایمیل محبوبی مانند SendGrid استفاده کنید، که با Windows Azure براحتی یکپارچه می‌شود و از طرف توسعه دهنده اپلیکیشن هم نیاز به پیکربندی ندارد.

در مثال زیر نیاز دارید تا یک سرویس ایمیل برای ارسال ایمیل‌ها پیکربندی کنید. همچنین کاربران پیش از تایید ایمیل شان قادر به بازنشانی کلمه عبور نیستند.

Password Reset

این هم یک سناریوی رایج و استاندارد است. کاربران در صورتی که کلمه عبورشان را فراموش کنند، می‌توانند از این قابلیت برای بازنشانی آن استفاده کنند. کلمه عبور جدیدی بصورت خودکار تولید شده و برای آنها ارسال می‌شود. کاربران با استفاده از این رمز عبور جدید می‌توانند وارد سایت شوند و سپس آن را تغییر دهند.

Security Token Provider

هنگامی که کاربران کلمه عبورشان را تغییر می‌دهند، یا اطلاعات امنیتی خود را بروز رسانی می‌کنند (مثلا حذف کردن لاگین‌های خارجی مثل فیسبوک، گوگل و غیره) باید شناسه امنیتی (security token) کاربر را بازتولید کنیم و مقدار قبلی را Invalidate یا بی اعتبار سازیم. این کار بمنظور حصول اطمینان از بی اعتبار بودن تمام شناسه‌های قبلی است که توسط کلمه عبور پیشین تولید شده بودند. این قابلیت، یک لایه امنیتی بیشتر برای اپلیکیشن شما فراهم می‌کند. چرا که وقتی کاربری کلمه عبورش را تغییر بدهد از همه جا logged-out می‌شود. یعنی از تمام مرورگرهایی که برای استفاده از اپلیکیشن استفاده کرده خارج خواهد شد.

برای پیکربندی تنظیمات این قابلیت می‌توانید از فایل Startup.Auth.cs استفاده کنید. می‌توانید مشخص کنید که میان افزار OWIN cookie هر چند وقت یکبار باید شناسه امنیتی کاربران را بررسی کند. به لیست زیر دقت کنید.

```
// 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(newCookieAuthenticationOptions {
    AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
    LoginPath = newPathString("/Account/Login"),
    Provider = newCookieAuthenticationProvider {
        OnValidateIdentity = SecurityStampValidator.OnValidateIdentity<ApplicationUserManager,
ApplicationUser>(
```

```
            validateInterval: TimeSpan.FromSeconds(5),
            regenerateIdentity: (manager, user) => user.GenerateUserIdentityAsync(manager))
    }
});
```

امکان سفارشی کردن کلیدهای اصلی Users و Roles

در نسخه 1.0 نوع فیلدهای کلید اصلی در جداول Users و Roles از نوع رشته (string) بود. این بدین معنا است که وقتی از Entity Framework و Sql Server برای ذخیره داده‌های ASP.NET Identity استفاده می‌کنیم داده‌های این فیلدها بعنوان nvarchar ذخیره می‌شوند. درباره این پیاده سازی پیش فرض در فروم هایی مانند سایت StackOverflow بسیار بحث شده است. و در آخر با در نظر گرفتن تمام بازخورد ها، تصمیم گرفته شد یک نقطه توسعه پذیری (extensibility) اضافه شود که توسط آن بتوان نوع فیلدهای اصلی را مشخص کرد. مثلا شاید بخواهید کلیدهای اصلی جداول Users و Roles از نوع int باشند. این نقطه توسعه پذیری مخصوصا هنگام مهاجرت داده‌های قبلی بسیار مفید است، مثلا ممکن است دیتابیس قبلی فیلدهای UserId را با فرمت GUID ذخیره کرده باشد.

اگر نوع فیلدهای کلید اصلی را تغییر دهید، باید کلاس‌های مورد نیاز برای Claims و Logins را هم اضافه کنید تا کلید اصلی معتبری دریافت کنند. قطعه کد زیر نمونه ای از نحوه استفاده این قابلیت برای تعریف کلیدهای int را نشان می‌دهد.

```
de Snippet
publicclassApplicationUser : IdentityUser<int, CustomUserLogin, CustomUserRole, CustomUserClaim>
{
}
publicclassCustomRole : IdentityRole<int, CustomUserRole>
{
    public CustomRole() { }
    public CustomRole(string name) { Name = name; }
}
publicclassCustomUserRole : IdentityUserRole<int> { }
publicclassCustomUserClaim : IdentityUserClaim<int> { }
publicclassCustomUserLogin : IdentityUserLogin<int> { }

publicclassApplicationDbContext : IdentityDbContext<ApplicationUser, CustomRole, int, CustomUserLogin,
CustomUserRole, CustomUserClaim>
{
}
```

پشتیبانی از IQueryable روی Users و Roles

کلاس‌های UserStore و RoleStore حالا از IQueryable پشتیبانی می‌کنند، بنابراین می‌توانید براحتی لیست کاربران و نقش‌ها را کوئری کنید.

بعنوان مثال قطعه کد زیر دریافت لیست کاربران را نشان می‌دهد. از همین روش برای دریافت لیست نقش‌ها از RoleManager می‌توانید استفاده کنید.

```
//
// GET: /Users/
public async Task<ActionResult> Index()
{
    return View(await UserManager.Users.ToListAsync());
}
```

پشتیبانی از عملیات Delete از طریق UserManager

در نسخه 1.0 اگر قصد حذف یک کاربر را داشتید، نمی‌توانستید این کار را از طریق UserManager انجام دهید. اما حالا می‌توانید مانند قطعه کد زیر عمل کنید.

```
var user = await UserManager.FindByIdAsync(id);

if (user == null)
{
    return HttpNotFound();
```

```
}
var result = await UserManager.DeleteAsync(user);
```

میان افزار UserManagerFactory

شما می‌توانید با استفاده از یک پیاده سازی Factory، وهله ای از UserManager را از OWIN context دریافت کنید. این الگو مشابه چیزی است که برای گرفتن AuthenticationManager در OWIN context استفاده می‌کنیم. این الگو همچنین روش توصیه شده برای گرفتن یک نمونه از UserManager به ازای هر درخواست در اپلیکیشن است.

قطعه کد زیر نحوه پیکربندی این میان افزار در فایل StartupAuth.cs را نشان می‌دهد.

```
// Configure the UserManager
app.UseUserManagerFactory(newUserManagerOptions<ApplicationUserManager>()
{
    DataProtectionProvider = app.GetDataProtectionProvider(),
    Provider = newUserManagerProvider<ApplicationUserManager>()
    {
        OnCreate = ApplicationUserManager.Create
    }
});
```

و برای گرفتن یک وهله از UserManager:

```
HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
```

میان افزار DbContextFactory

سیستم ASP.NET Identity از Entity Framework برای ذخیره داده هایش در Sql Server استفاده می‌کند. بدین منظور، ASP.NET Identity کلاس ApplicationDbContext را رفرنس می‌کند. میان افزار DbContextFactory به ازای هر درخواست در اپلیکیشن یک وهله از ApplicationDbContext را به شما تحویل می‌دهد.

می توانید پیکربندی لازم را در StartupAuth.cs انجام دهید.

```
app.UseDbContextFactory(ApplicationDbContext.Create);
```

Samples

امکانات جدید را می‌توانید در پروژه https://aspnet.codeplex.com پیدا کنید. لطفا به پوشه Identity در سورس کد مراجعه کنید. برای اطلاعاتی درباره نحوه اجرای پروژه هم فایل readme را بخوانید.

برای مستندات ASP.NET Identity 1.0 هم به http://www.asp.net/identity سر بزنید. هنوز مستنداتی برای نسخه 2.0 منتشر نشده، اما بزودی با انتشار نسخه نهایی مستندات و مثال‌های جدیدی به سایت اضافه خواهند شد.

Known Issues

در کنار قابلیت‌های جدیدی مانند Account Confirmation و Password Reset، دو خاصیت جدید به کلاس IdentityUser اضافه شده اند: 'Email' و 'IsConfirmed'. این تغییرات الگوی دیتابیسی که توسط ASP.NET Identity 1.0 ساخته شده است را تغییر می‌دهد. بروز رسانی پکیج‌ها از نسخه 1.0 به 2.0 باعث می‌شود که اپلیکیشن شما دیگر قادر به دسترسی به دیتابیس عضویت نباشد، چرا که مدل دیتابیس تغییر کرده. برای بروز رسانی الگوی دیتابیس می‌توانید از Code First Migrations استفاده کنید.

نکته: نسخه جدید به EntityFramework 6.1.0-alpha1 وابستگی دارد، که در همین تاریخ (20 دسامبر 2013) پیش نمایش شد. http://blogs.msdn.com/b/adonet/archive/2013/12/20/ef-6-1-alpha-1-available.aspx

EntityFramework 6.1.0-alpha1 بروز رسانی هایی دارد که سناریوی مهاجرت در ASP.NET Identity را تسهیل می‌کند، به همین دلیل از نسخه جدید EF استفاده شده. تیم ASP.NET هنوز باگ‌های زیادی را باید رفع کند و قابلیت‌های جدیدی را هم باید پیاده سازی کند. بنابراین پیش از نسخه نهایی RTM شاهد پیش نمایش‌های دیگری هم خواهیم بود که در ماه‌های آتی منتشر می‌شوند.

برای اطلاعات بیشتر درباره آینده ASP.NET Identity به لینک زیر سری بزنید.

https://aspnetidentity.codeplex.com/wikipage?title=Roadmap&version=1

نظرات خوانندگان

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۲/۱۰/۳۰ ۲۰:۴۴

سلام

پیاده سازی Single Sign on در این سیستم کجا کار قرار داره؟ در واقع چطور میشه پیاده سازی ش کرد؟
در سیستم membership قبلی، اگر کلید اپلیکیشن رو در وب کانفیگ برنامه هاتون که دامین هاشون مشترک بود (در واقع ساب دامین ها)، یکسان وارد میکردی، برنامه‌ها بصورت SSO کار می‌کرد و احتیاجی به هیچ کاری نداشت. حالا در سیستم جدید همون روش جواب میده؟ برای برنامه‌های با دامین‌های متفاوت چطور؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۳۰ ۲۳:۷

CookieDomain رو باید تنظیم کنید.