

با هر بار عرضه‌ی نسخه‌های جدید ویژوال استادیو، علاوه بر اضافه شدن امکانات جدید، برخی از امکانات هم به دلایل نامعلومی از این نرم افزار حذف می‌شوند. در Visual Studio 2012 امکان بسیار کارآمد Setup and Deployment حذف گردید و این بار برخلاف انتظار در Visual Studio 2013 با عدم پشتیبانی از Sql Server Compact مواجه شدیم و هنوز دلایل این کار از سوی تیم ویژوال استادیو توضیح داده نشده است. شاید مایکروسافت در حال توسعه نسخه NoSql جدیدی برای جایگزینی باشد.

می توانید از ابزار [SQL Server Compact Toolbox](#) استفاده نمایید که کارایی خوبی ندارد و بیشتر یک مکمل است. اما راهی برای بازگشت این ابزار به Visual Studio 2013 وجود دارد؟

قابلیت Data Designer Extensibility

در نگارش‌های مختلف ویژوال استادیو امکانی به نام DDEX Provider وجود دارد که توسط آن می‌توانید یک Data Designer جدید را به ویژوال استادیو اضافه نمایید. در واقع اگر از پنجره Server Explorer بر روی Data Connections راست کلیک و یک کانکشن جدید بسازید، لیست Data Source های پیش فرض ویژوال استادیو به شما نشان داده می‌شود که به کمک همین قابلیت DDEX به ویژوال استادیو اضافه شده است. با این قابلیت، امکان اضافه نمودن یک Data Designer برای یک پایگاه داده نیز وجود دارد. از آدرس [Data Designer Extensibility \(DDEX\) SDK](#) می‌توانید نحوه تولید و رجیستر کردن یک DDEX Provider را بیاموزید. برای مثال رجیستری زیر IBM DB2 Data Provider را به ویژوال استادیو اضافه می‌نماید

```
HKLM
{
    %REGROOTBEGIN%

    'DataProviders'
    {
        '{6085DDE2-2EE1-4768-82C3-5425D9B98DAD}' = s 'IBM DB2 Provider'
        {
            val 'DisplayName' = s 'Provider_DisplayName, IBM.DB2.Resources'
            val 'ShortDisplayName' = s 'Provider_ShortDisplayName, IBM.DB2.Resources'
            val 'Description' = s 'Provider_Description, IBM.DB2.Resources'
            val 'FactoryService' = s '{45E1413D-896C-4a2a-A75C-1CBA51C80CB}'
            val 'Technology' = s '{6565551F-A496-45f3-AFFB-D1AECA082824}'
            val 'InvariantName' = s 'IBM.DB2'
            val 'PlatformVersion' = s '2.0'

            'SupportedObjects'
            {
                'IVsDataViewSupport'
                'IVsDataObjectSupport'
                'IVsDataConnectionUIControl'
                'IVsDataConnectionProperties'
                'IVsDataConnectionSupport'
            }
        }
    }

    'Services'
    {
        '{45E1413D-896C-4a2a-A75C-1CBA51C80CB}' = s '{7B7F1923-D8F9-430f-9FA7-7919677E5EAC}'
        {
            val 'Name' = 'IBM DB2 Provider Object Factory'
        }
    }

    'Packages'
    {
        '{7B7F1923-D8F9-430f-9FA7-7919677E5EAC}' = 'DB2 Package'
        {
            val 'InProcServer32' = s 'mscoree.dll'
            val 'Class' = s 'IBM.DB2.DB2Package'
            val 'Codebase' = s '%MODULE%'

            'SatelliteDll'
            {
                val 'Path' = s '%PATH%'
            }
        }
    }
}
```

```

        val 'DllName' = s 'IBM.DB2UI.DLL'
    }
}
%REGROOTEND%
}

```

ابزار SSCEVSTools for Visual Studio 2013

برای اضافه نمودن Sql Server Compact Data Provider به Visual Studio 2013 از نسخه قبلی SSCEVSTools که برای Visual Studio 2012 عرضه شده است استفاده می‌کنیم. در واقع این ابزار یک DDEX Provider را به ویژوال استادیو برای Sql Server Compact اضافه می‌کند. اما این نصب کننده، برای نسخه‌ی قبل، تهیه شده است و امکان نصب آن بر روی Visual Studio 2013 نمی‌باشد. یک راهکار عملی، دسترسی به فایل‌ها و رجیستری‌های موجود در این نصب کننده و تولید نصب کننده جدیدی می‌باشد.

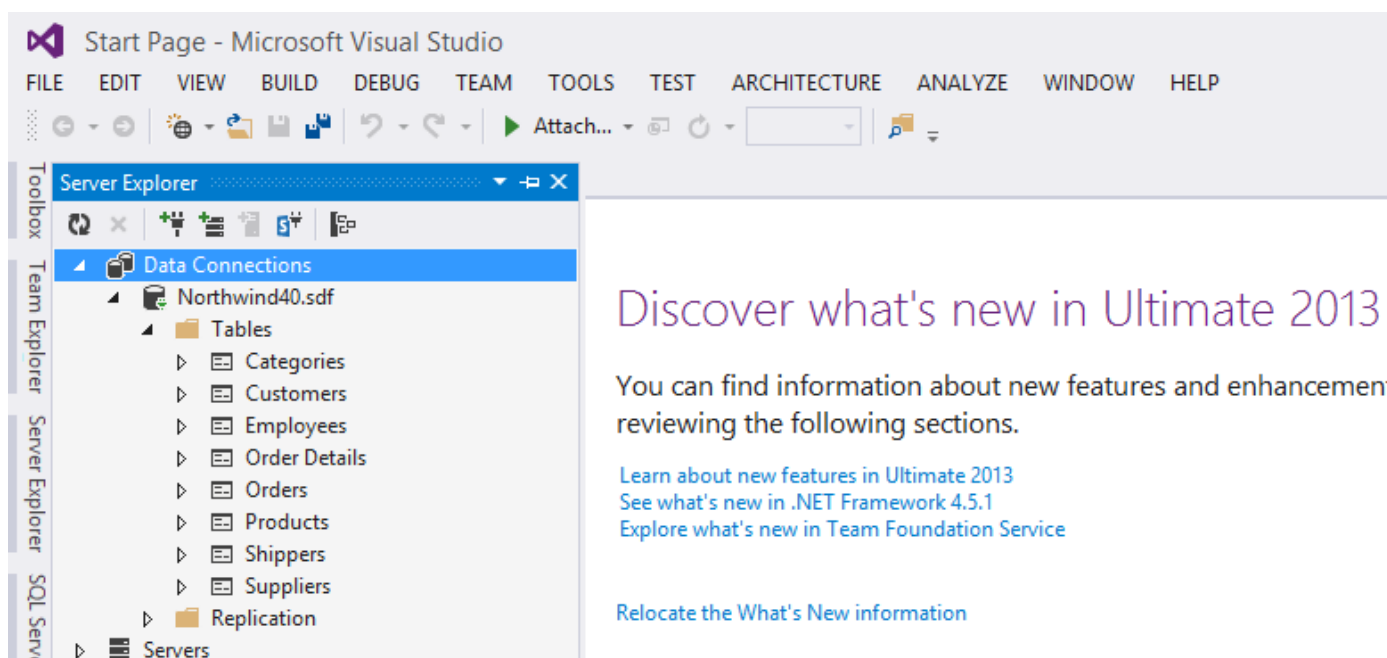
دسترسی به محتوی فایل‌های Setup

ابزار [Orca](#) در Windows SDK برای ویرایش فایل‌های نصب کننده توسط مایکروسافت تولید شده است که امکان مشاهده تمامی جزئیات آن را فراهم می‌نماید. ابزار قبلی، شامل فایل‌های dll و رجیستری است و امکان اتصال به Sql Server Compact را به ویژوال استادیو اضافه می‌نمود.

حال با یک برنامه Setup ساز، فایل‌ها و رجیستری را برای Visual Studio 2013 تنظیم نموده و با نصب ابزار جدید، دوباره امکان استفاده از Sql Server Compact در Visual Studio 2013 میسر می‌شود.

برای نصب این ابزار، آن را از گالری ویژوال استادیو به نام [SSCEVSTools for Visual Studio 2013](#) دانلود نمایید.

البته چون این ابزار بصورت غیر رسمی تولید و عرضه شده است گاهی اوقات به صورت خودکار از لیست Data Source ها حذف شده که لازم است آن را حذف و مجدداً نصب نمایید.



اگر مایل به بازگشت و کار بر روی نسخه جدید 5 Sql Server Compact هستید اینجا در [Visual Studio UserVoice](#) رای دهید.

ابزار ASP.NET برای Windows Azure Active Directory فعال کردن احراز هویت در وب اپلیکیشن هایی که روی [Windows Azure Web Sites](#) میزبانی شده اند را ساده تر می کند. می توانید از Windows Azure Authentication برای احراز هویت کاربران Office 365 استفاده کنید، حساب های کاربری را از On-Premise Active Directory خود همگام سازی (Sync) کنید و یا از یک دامنه سفارشی Windows Azure Active Directory بهره ببرید. فعال سازی Windows Azure Authentication، اپلیکیشن شما را طوری پیکربندی می کند تا تمامی کاربران را با استفاده از یک [Windows Azure Active Directory tenant](#) احراز هویت کند. این مقاله ساختن یک اپلیکیشن ASP.NET را که بر اساس [organizational accounts](#) پیکربندی شده و بر روی [Windows Azure Active Directory](#) میزبانی می شود، مرور می کند.

پیش نیاز ها

[Visual Studio Express 2013 RC for Web](#) یا [Visual Studio 2013 RC](#)

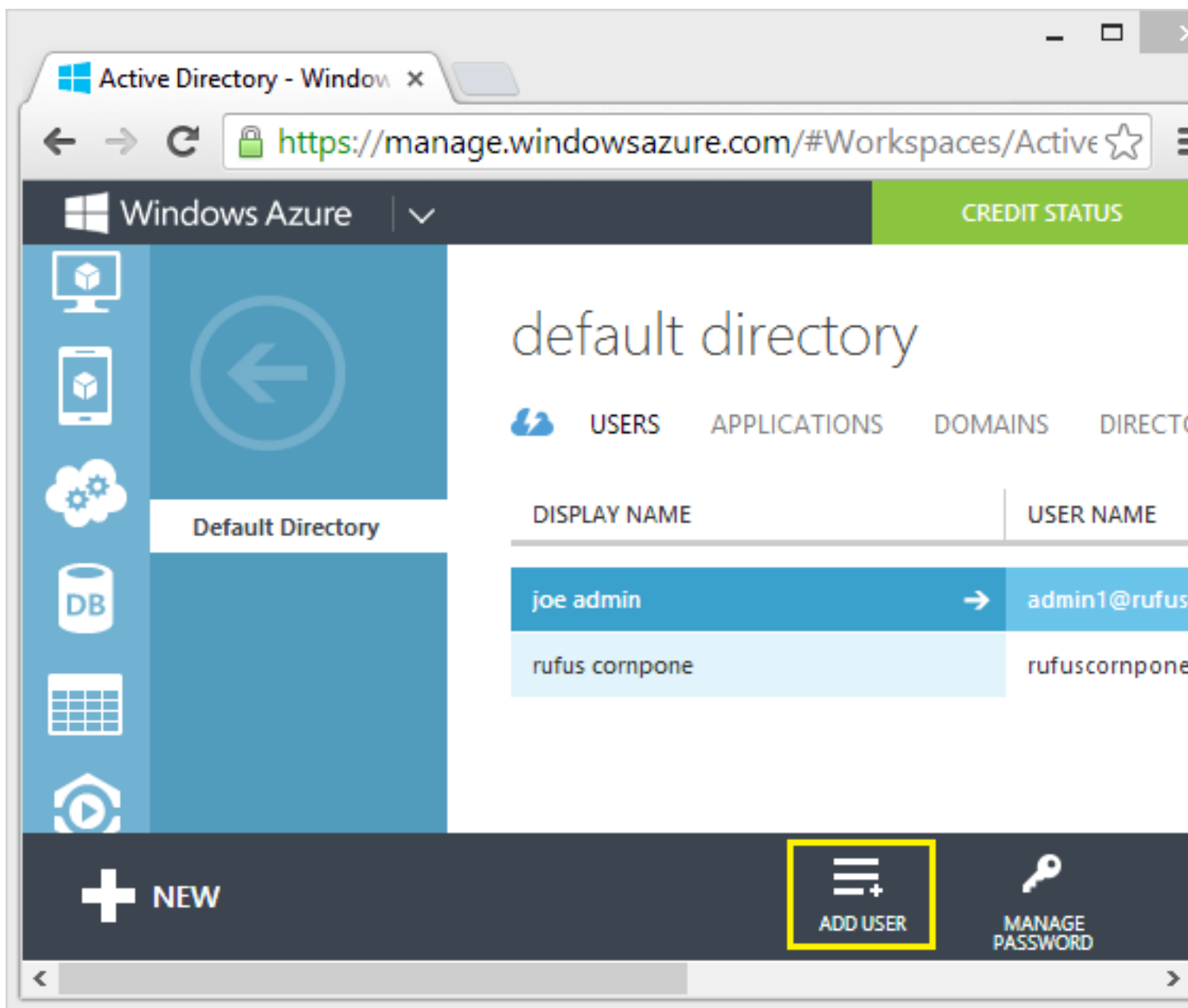
یک حساب کاربری در Windows Azure. می توانید یک [حساب رایگان بسازید](#).

یک مدیر کلی به حساب کاربری **Active Directory** خود اضافه کنید

وارد [Windows Azure Portal](#) شوید.

یک حساب کاربری (AD) Windows Azure Active Directory (AD) انتخاب یا ایجاد کنید. اگر قبلاً حساب کاربری ساخته اید از همان استفاده کنید در غیر اینصورت یک حساب جدید ایجاد کنید. مشترکین Windows Azure یک AD پیش فرض با نام **Default Directory** خواهند داشت.

در حساب کاربری AD خود یک کاربر جدید در نقش Global Administrator بسازید. اکانت AD خود را انتخاب کنید و **Add User** را کلیک کنید. برای اطلاعات کامل تر به [Managing Windows Azure AD from the Windows Azure Portal 1- Sign Up with an Organizational Account](#) مراجعه کنید.



یک نام کاربری انتخاب کرده و به مرحله بعد بروید.

×

ADD USER

Tell us about this user

TYPE OF USER ?

New user in your organization

USER NAME ?

AdminBob

@

rufuscornpone60gmail.onmicrosoft.

→

نام کاربری را وارد کنید و نقش **Global Administrator** را به آن اختصاص دهید. مدیران کلی به یک آدرس ایمیل متناوب هم نیاز دارند. به مرحله بعد بروید.

ADD USER

user profile

FIRST NAME LAST NAME

Bob Admin

DISPLAY NAME

Bob the Admin

ROLE ?

Global Administrator

ALTERNATE EMAIL ADDRESS

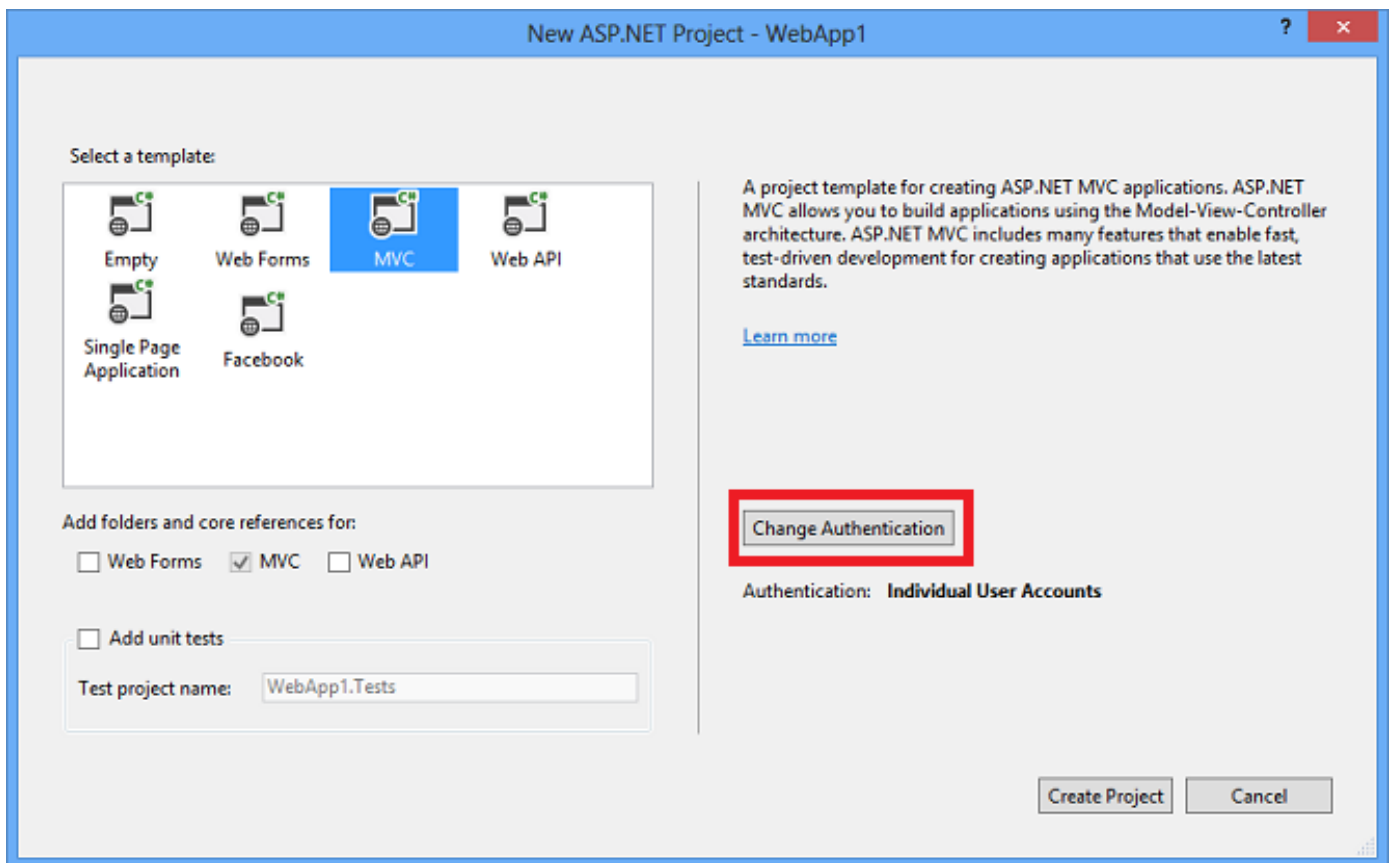
bob@contoso.com

☐ Require Multi-factor Authentication **PREVIEW**

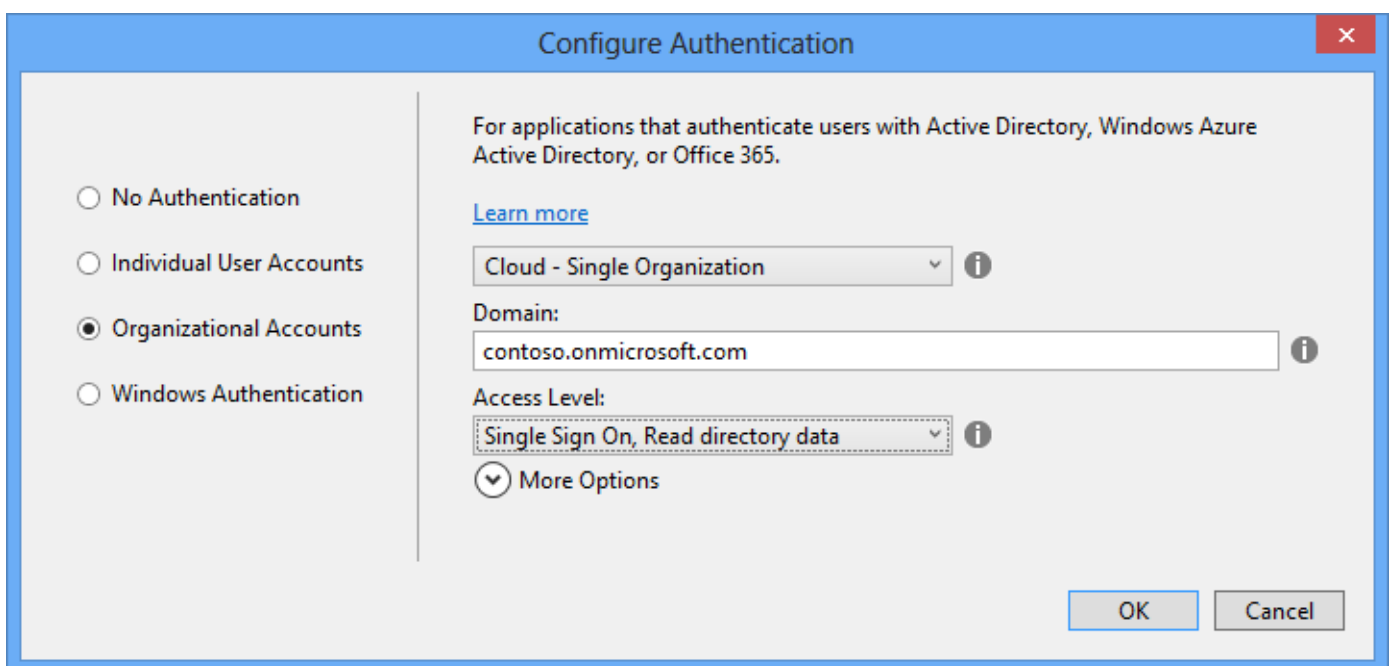
Navigation arrows: back and forward buttons, with the forward button highlighted by a red box.

بر روی **Create** کلیک کنید و کلمه عبور موقتی را کپی کنید. پس از اولین ورود باید کلمه عبور را تغییر دهید.

یک اپلیکیشن ASP.NET بسازید
در ویژوال استودیو یک پروژه جدید ASP.NET Web Forms یا MVC بسازید و روی **Change Authentication** کلیک کنید.



گزینه **Organizational Accounts** را انتخاب کنید. نام دامنه خود را وارد کنید و سپس گزینه **Single Sign On, Read directory data** را انتخاب کنید. به مرحله بعد بروید.



نکته: در قسمت **More Options** می توانید قلمرو اپلیکیشن (Application ID URI) را تنظیم کنید. تنظیمات پیش فرض برای اکثر

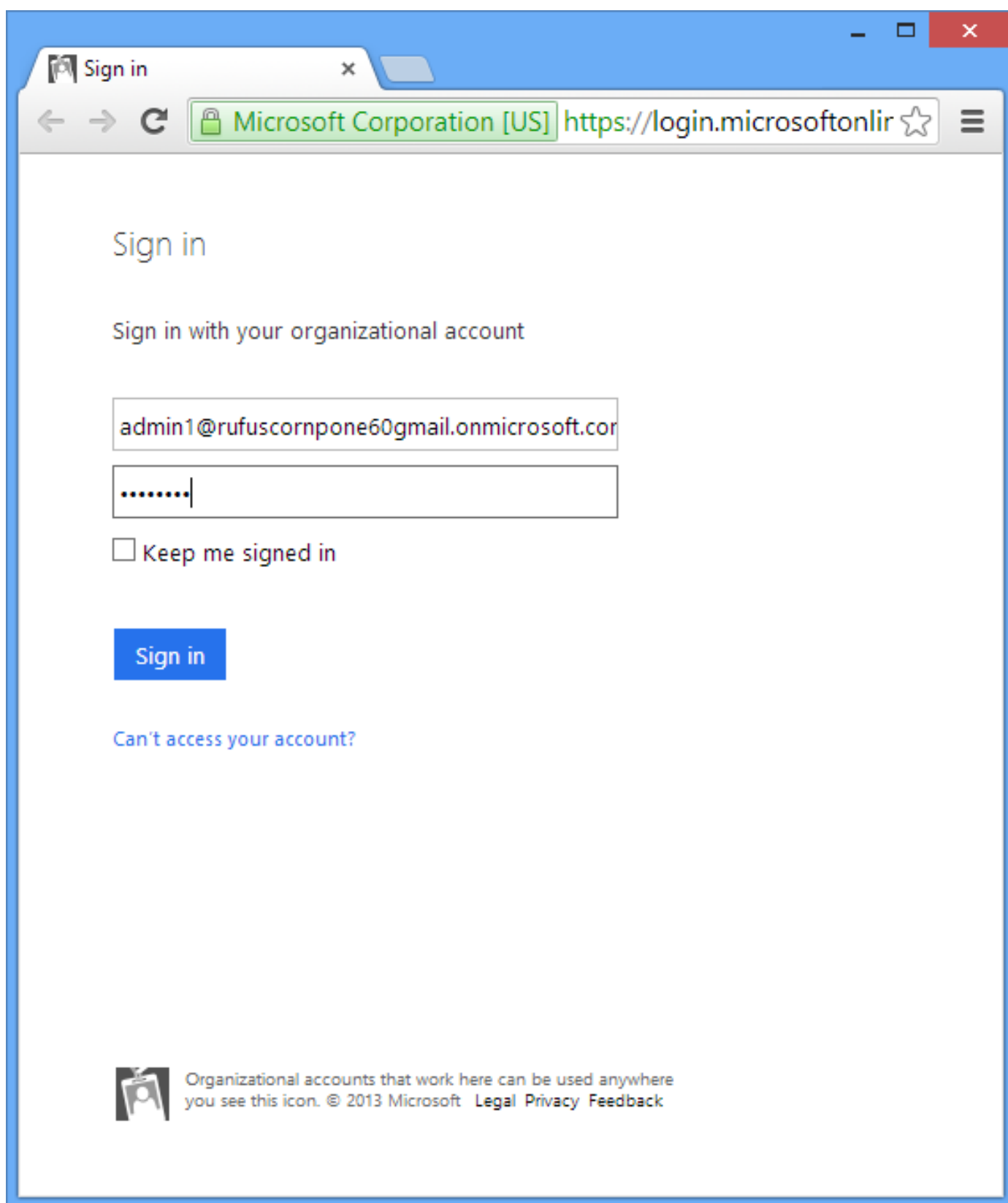
کاربران مناسب است اما در صورت لزوم می‌توانید آنها را ویرایش کنید، مثلاً از طریق Windows Azure Portal دامنه‌های سفارشی خودتان را تنظیم کنید.

اگر گزینه **Overwrite** را انتخاب کنید اپلیکیشن جدیدی در Windows Azure برای شما ساخته خواهد شد. در غیر اینصورت فریم ورک سعی می‌کند اپلیکیشنی با شناسه یکسان پیدا کند (در پست [متدهای احراز هویت در VS 2013](#) به تنظیمات این قسمت پرداخته شده).

اطلاعات مدیر کلی دامنه در Active Directory خود را وارد کنید (Credentials) و پروژه را با کلیک کردن بر روی **Create Project** بسازید.

با کلیدهای ترکیبی **Ctrl + F5**، اپلیکیشن را اجرا کنید. مرورگر شما باید یک اخطار SSL Certificate به شما بدهد. این بدین دلیل است که مدرک استفاده شده توسط IIS Express مورد اعتماد (trusted) نیست. این اخطار را بپذیرید و اجازه اجرا را به آن بدهید. پس از آنکه اپلیکیشن خود را روی Windows Azure منتشر کردید، این پیغام دیگر تولید نمی‌شود؛ چرا که Certificate‌های استفاده شده trusted هستند.

با حساب کاربری سازمانی (organizational account) که ایجاد کرده‌اید، وارد شوید.



Sign in

Sign in with your organizational account


admin1@rufuscornpone60gmail.onmicrosoft.com

.....

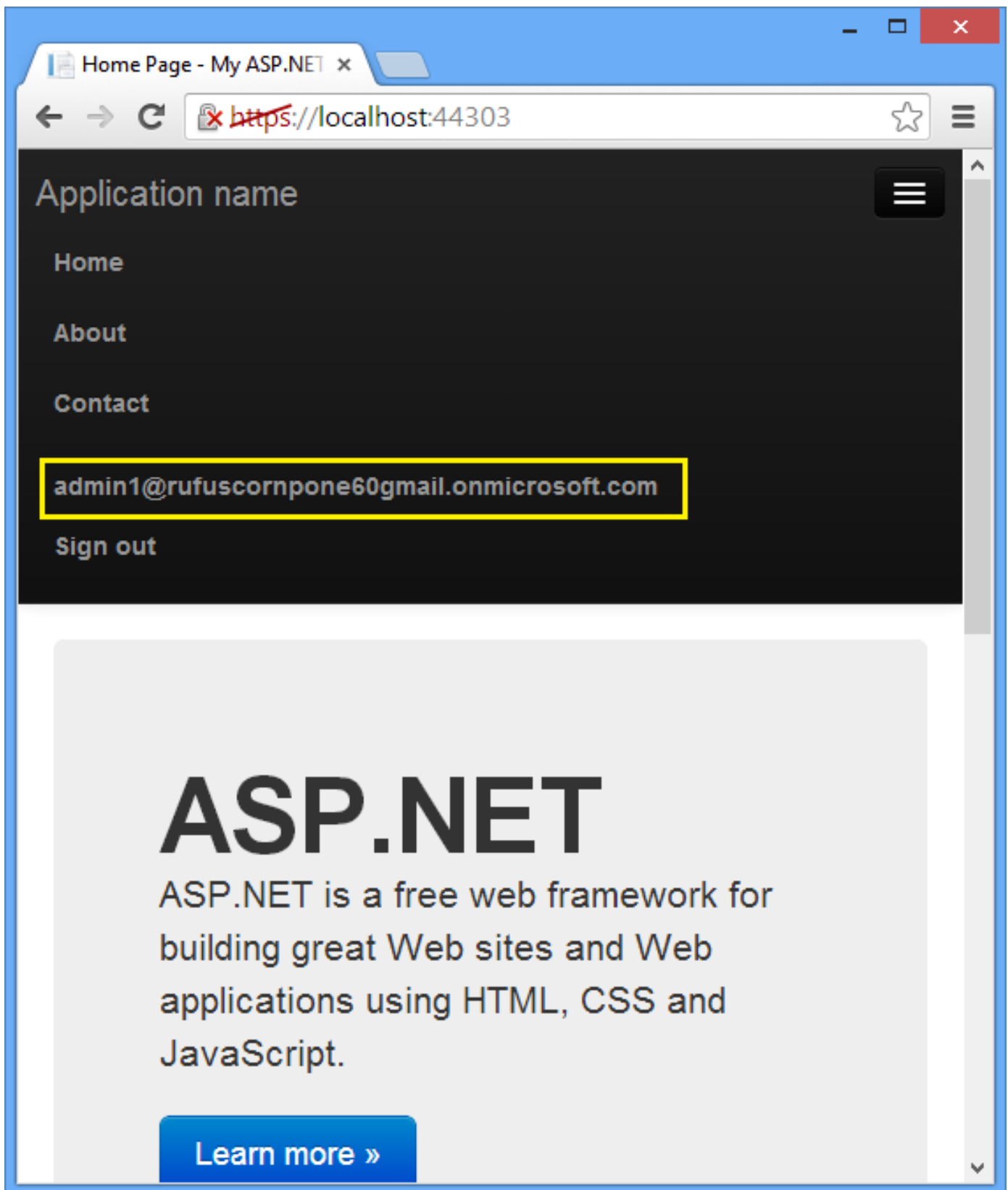
☐ Keep me signed in

[Sign in](#)

[Can't access your account?](#)

 Organizational accounts that work here can be used anywhere you see this icon. © 2013 Microsoft [Legal](#) [Privacy](#) [Feedback](#)

همانطور که مشاهده می‌کنید هم اکنون به سایت وارد شده اید.



توزیع اپلیکیشن روی Windows Azure

در Windows Azure Portal یک وب سایت را به همراه یک دیتابیس، ایجاد کنید. در پانل سمت چپ صفحه روی **Websites** کلیک کنید و بعد **New** را انتخاب کنید. سپس گزینه **Custom Create** را برگزینید.

NEW WEB SITE - CUSTOM CREATE

Create Web Site

URL

Contoso8✓.azurewebsites.net

REGION

West US

DATABASE

Create a free 20 MB SQL database

DB CONNECTION STRING NAME ?

DefaultConnection

☐ Publish from source control ?

→ 2

اپلیکیشن را روی Windows Azure منتشر کنید. روی پروژه کلیک راست کرده و **Publish** را انتخاب کنید. در مرحله تنظیمات (Settings) مشاهده می کنید که احراز هویت حساب های سازمانی (organizational accounts) فعال است. همچنین سطح دسترسی به خواندن تنظیم شده است. در قسمت Database رشته اتصال دیتابیس را تنظیم کنید.

Publish Web

Profile

Connection

Settings

Preview

Contoso8 *

Configuration: Release

File Publish Options

☒ Enable Organizational Authentication

Directory Access Level:
☒ Read ☐ Read and write

Databases

TenantDbContext (DefaultConnection)

Data Source=tcp:ox6jx6raw7.database.windows.net,1433;Initial Catalog=Cont

☒ Use this connection string at runtime (update destination web.config)

☐ Execute Code First Migrations (runs on application start)

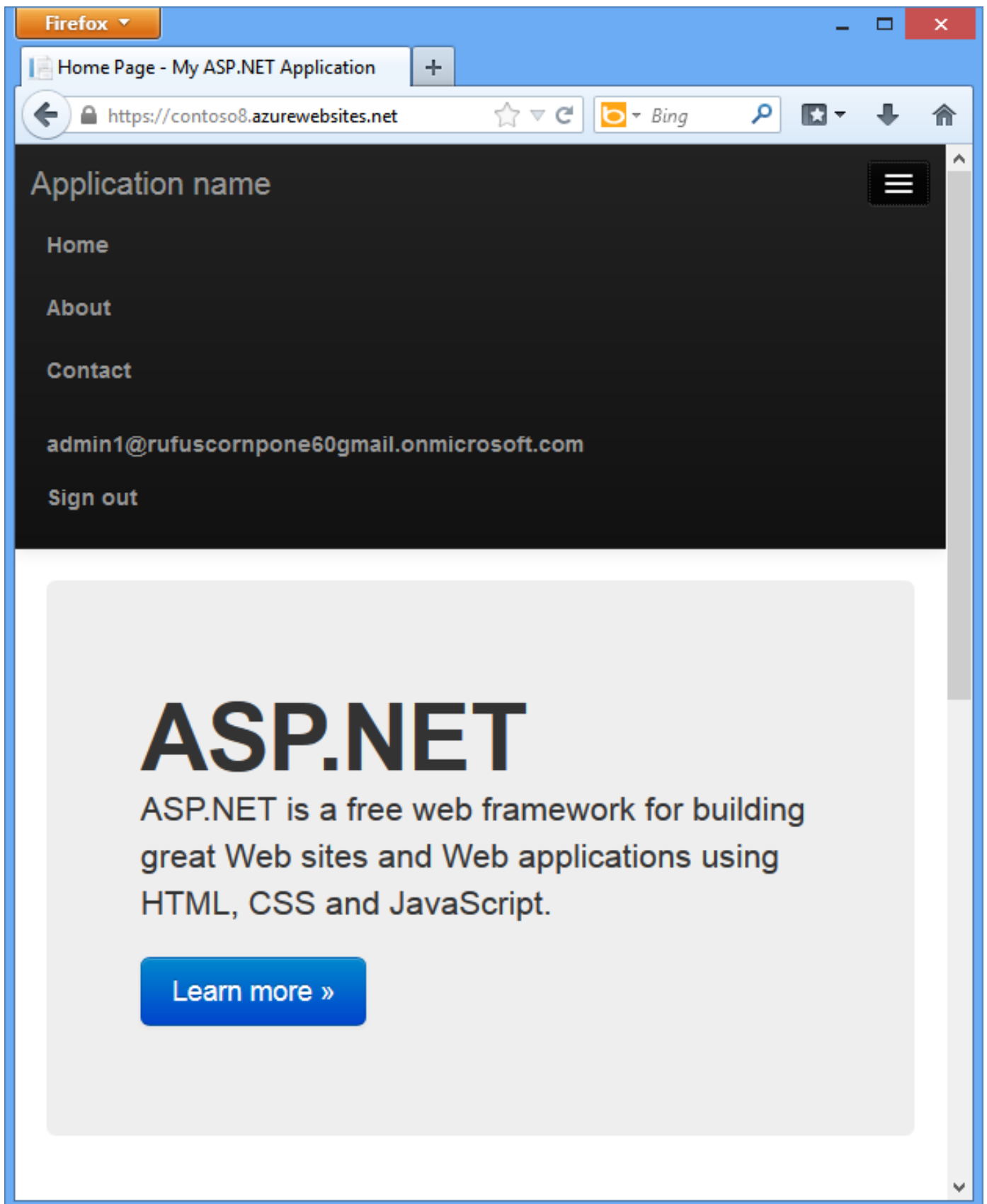
< Prev

Next >

Publish

Close

حال به وب سایت Windows Azure خود بروید و توسط حساب کاربری ایجاد شده وارد سایت شوید. در این مرحله دیگر نباید خطای امنیتی SSL را دریافت کنید.



خواندن اطلاعات پروفایل کاربران توسط Graph API

قالب پروژه ویژوال استودیو برای organizational accounts یک متد و نما بنام UserProfile به پروژه اضافه کرده است.

[Authorize]

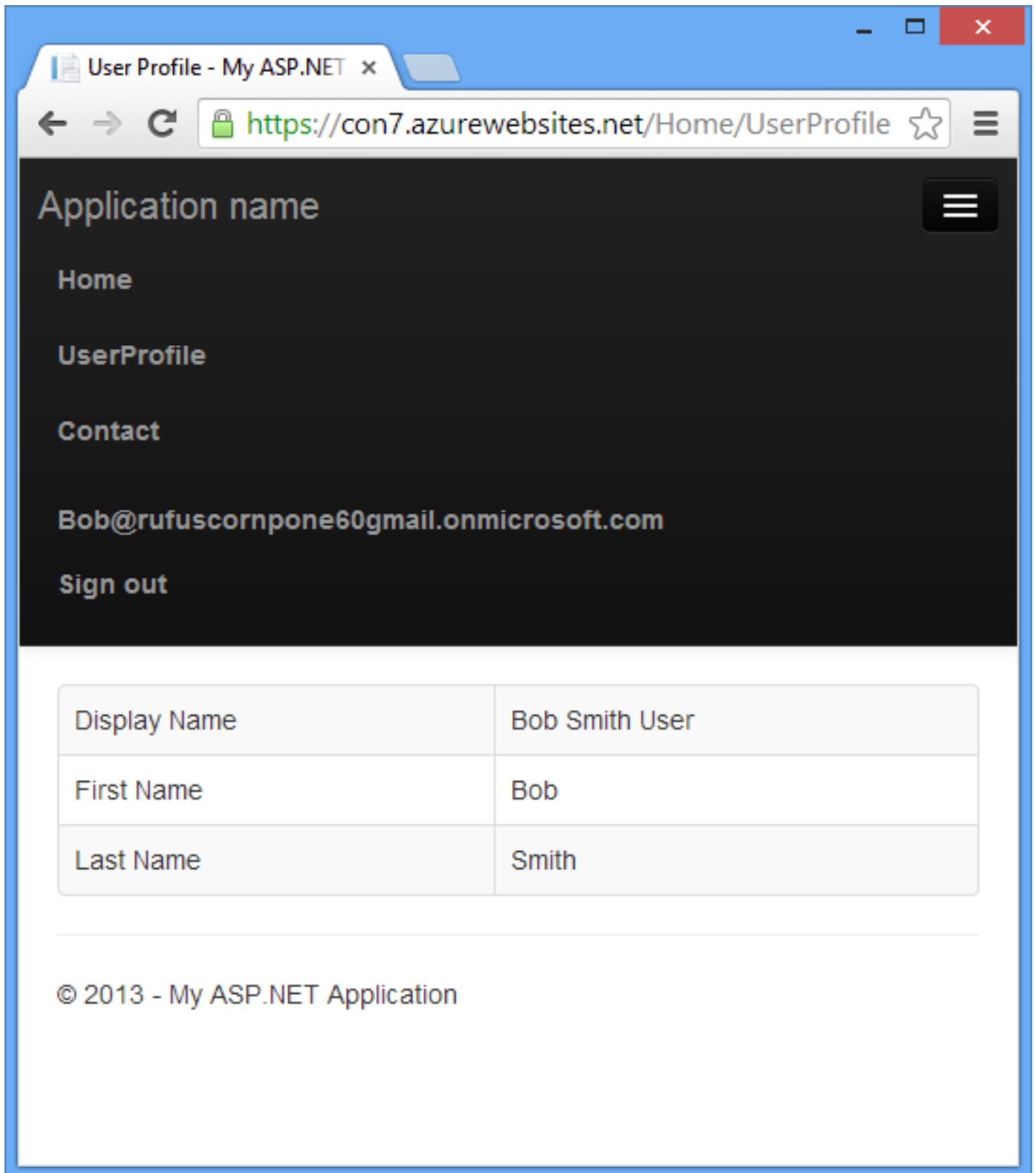
```
public async Task<ActionResult> UserProfile()
{
    string tenantId = ClaimsPrincipal.Current.FindFirst(TenantSchema).Value;

    // Get a token for calling the Windows Azure Active Directory Graph
    AuthenticationContext authContext = new
AuthenticationContext(String.Format(CultureInfo.InvariantCulture, LoginUrl, tenantId));
    ClientCredential credential = new ClientCredential(AppPrincipalId, AppKey);
    AuthenticationResult assertionCredential = authContext.AcquireToken(GraphUrl, credential);
    string authHeader = assertionCredential.CreateAuthorizationHeader();
    string requestUrl = String.Format(
        CultureInfo.InvariantCulture,
        GraphUserUrl,
        HttpUtility.UrlEncode(tenantId),
        HttpUtility.UrlEncode(User.Identity.Name));

    HttpClient client = new HttpClient();
    HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, requestUrl);
    request.Headers.TryAddWithoutValidation("Authorization", authHeader);
    HttpResponseMessage response = await client.SendAsync(request);
    string responseString = await response.Content.ReadAsStringAsync();
    UserProfile profile = JsonConvert.DeserializeObject<UserProfile>(responseString);

    return View(profile);
}
```

کلیک کردن لینک **UserProfile** اطلاعات پروفایل کاربر جاری را نمایش می‌دهد.



اطلاعات بیشتر

[Managing Windows Azure AD from the Windows Azure Portal 1- Sign Up with an Organizational Account](#)

[Adding Sign-On to Your Web Application Using Windows Azure AD](#)

[Using the Graph API to Query Windows Azure AD](#)

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۲/۱۰/۱۵ ۱۲:۲۰

خیلی ممنون از زحمات شما
چند وقت پیش قصد داشتم ثبت نام کنم، زمانی که میخواهید حساب کاربری ایجاد کنید نیاز به شماره موبایل داره تا اجازه بده
مراحل ثبت نام کامل بشه ولی اصلا امکان انتخاب کشور ایران وجود ندارد و موفق نشدم

نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۱۶ ۱۳:۱۴

از [حساب های کاربری مایکروسافت](#) می تونید استفاده کنید.

نویسنده: بهروز
تاریخ: ۱۳۹۲/۱۰/۱۹ ۱۲:۱

ممنون بابت مطلب خوبتون.

لطفا یکم شفاف تر بگین که چطور برا تو حساب کاربری ایجاد کنیم؟ من هرکاری میکنم این صفحه میاد!
<https://manage.windowsazure.com/Error/NoSubscriptions>

نویسنده: کامران سادین
تاریخ: ۱۳۹۲/۱۱/۱۲ ۴:۲۲

سلام. من شماره موبایل هم دادم اما اطلاعات مستر کارت و ویزا کارت میخواد!

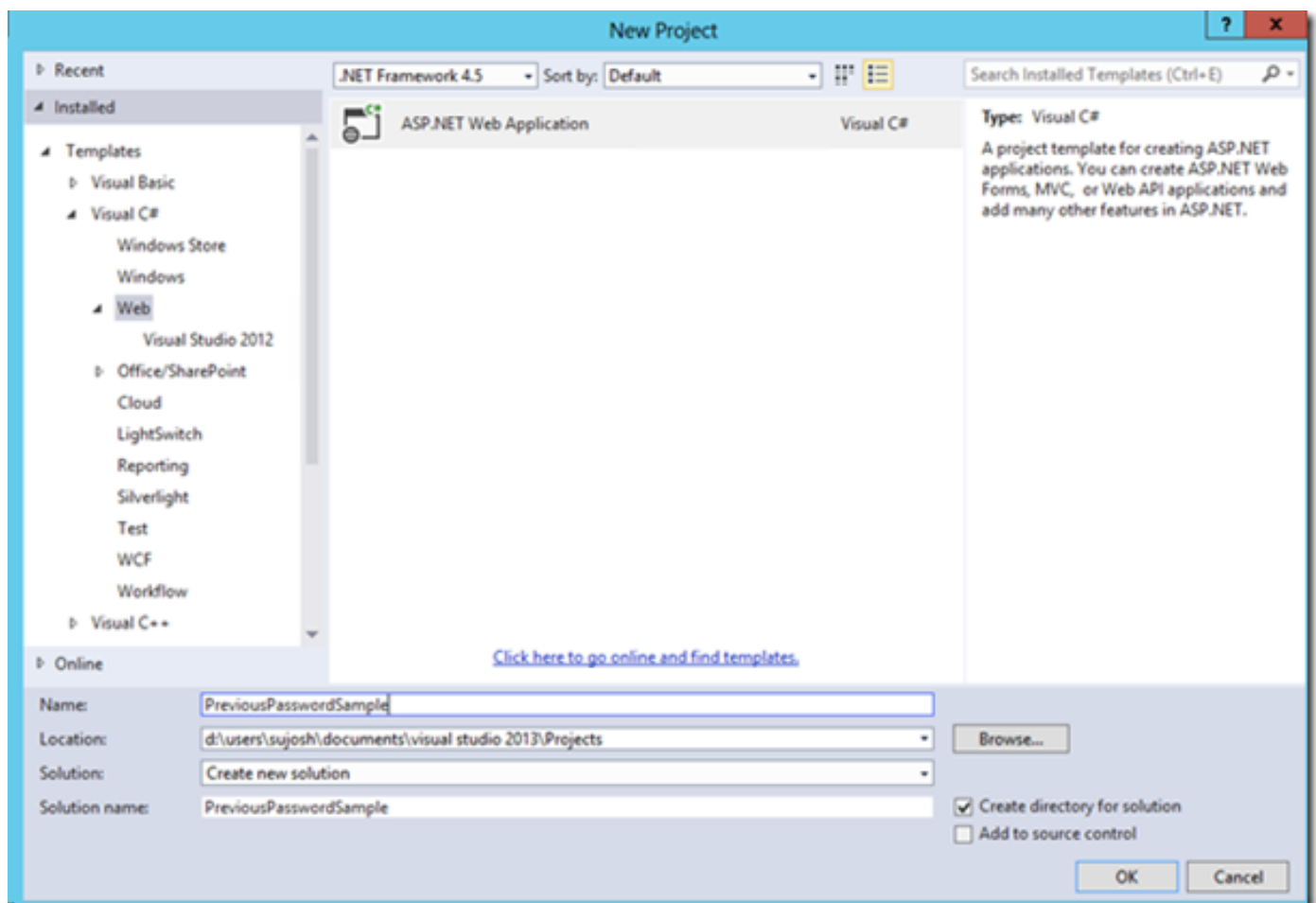
برای فراهم کردن یک تجربه کاربری ایمن‌تر و بهتر، ممکن است بخواهید پیچیدگی password policy را سفارشی سازی کنید. مثلاً ممکن است بخواهید حداقل تعداد کاراکترها را تنظیم کنید، استفاده از چند حروف ویژه را اجباری کنید، جلوگیری از استفاده نام کاربر در کلمه عبور و غیره. برای اطلاعات بیشتر درباره سیاست‌های کلمه عبور به [این لینک](#) مراجعه کنید. بصورت پیش فرض ASP.NET Identity کاربران را وادار می‌کند تا کلمه‌های عبوری بطول حداقل 6 کاراکتر وارد نمایند. در ادامه نحوه افزودن چند خط مشی دیگر را هم بررسی می‌کنیم.

با استفاده از ویژوال استودیو 2013 پروژه جدیدی خواهیم ساخت تا از ASP.NET Identity استفاده کند. مواردی که درباره کلمه‌های عبور می‌خواهیم اعمال کنیم در زیر لیست شده اند.
تنظیمات پیش فرض باید تغییر کنند تا کلمات عبور حداقل 10 کاراکتر باشند
کلمه عبور حداقل یک عدد و یک کاراکتر ویژه باید داشته باشد
امکان استفاده از 5 کلمه عبور اخیری که ثبت شده وجود ندارد

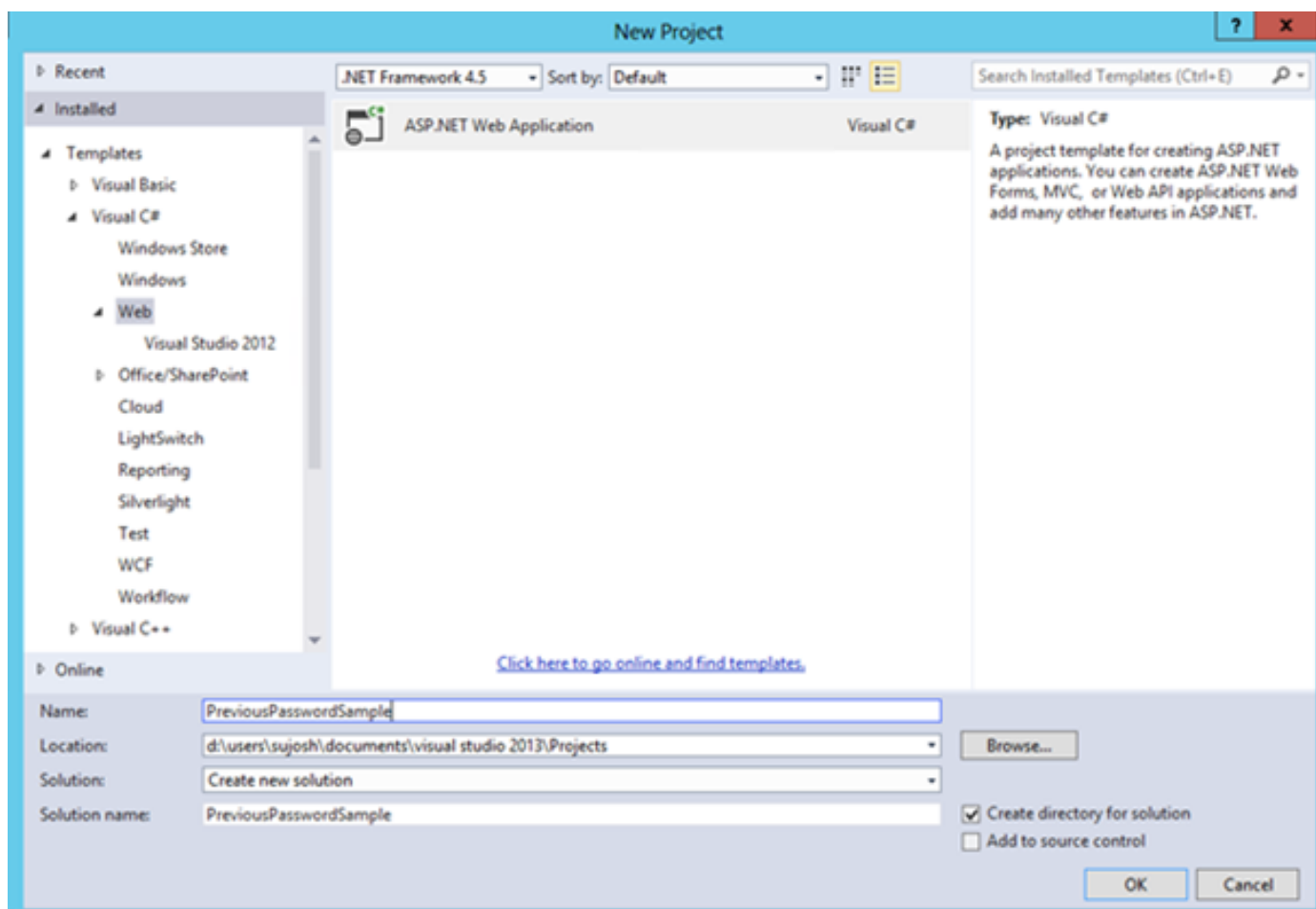
در آخر اپلیکیشن را اجرا می‌کنیم و عملکرد این قوانین جدید را بررسی خواهیم کرد.

ایجاد اپلیکیشن جدید

در Visual Studio 2013 اپلیکیشن جدیدی از نوع ASP.NET MVC 4.5 بسازید.



در پنجره Solution Explorer روی نام پروژه کلیک راست کنید و گزینه Manage NuGet Packages را انتخاب کنید. به قسمت **Update** بروید و تمام انتشارات جدید را در صورت وجود نصب کنید.



بگذارید تا به روند کلی ایجاد کاربر جدید در اپلیکیشن نگاهی بیاندازیم. این به ما در شناسایی نیازهای جدیدمان کمک می‌کند. پوشه Controllers حاوی متدهایی برای مدیریت کاربران است. کنترلر Account از کلاس UserManager استفاده می‌کند که در فریم ورک Identity تعریف شده است. این کلاس به نوبه خود از کلاس دیگری بنام UserStore استفاده می‌کند که برای دسترسی و مدیریت داده‌های کاربران استفاده می‌شود. در مثال ما این کلاس از Entity Framework استفاده می‌کند که پیاده سازی پیش فرض است. متد Register POST یک کاربر جدید می‌سازد. متد CreateAsync به طبع متد 'ValidateAsync' را روی خاصیت PasswordValidator فراخوانی می‌کند تا کلمه عبور دریافتی اعتبارسنجی شود.

```
var user = new ApplicationUser() { UserName = model.UserName };
var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)
{
    await SignInAsync(user, isPersistent: false);
    return RedirectToAction("Index", "Home");
}
```

در صورت موفقیت آمیز بودن عملیات ایجاد حساب کاربری، کاربر به سایت وارد می‌شود.

قانون 1: کلمه‌های عبور باید حداقل 10 کاراکتر باشند

بصورت پیش فرض خاصیت PasswordValidator در کلاس UserManager به کلاس MinimumLengthValidator تنظیم شده است، که اطمینان حاصل می‌کند کلمه عبور حداقل 6 کاراکتر باشد. هنگام و هله سازی UserManager می‌توانید این مقدار را تغییر دهید. مقدار حداقل کاراکترهای کلمه عبور به دو شکل می‌تواند تعریف شود. راه اول، تغییر کنترلر Account است. در متد سازنده این کنترلر کلاس UserManager و هله سازی می‌شود، همینجا می‌توانید این تغییر را اعمال کنید. راه دوم، ساختن کلاس جدیدی است که از UserManager ارث بری می‌کند. سپس می‌توان این کلاس را در سطح global تعریف کرد. در پوشه IdentityExtensions کلاس جدیدی با نام ApplicationUserManager بسازید.

```
public class ApplicationUserManager : UserManager<ApplicationUser>
{
    public ApplicationUserManager(): base(new UserStore<ApplicationUser>(new ApplicationDbContext()))
    {
        PasswordValidator = new MinimumLengthValidator (10);
    }
}
```

کلاس UserManager یک نمونه از کلاس IUserStore را دریافت می‌کند که پیاده سازی API های مدیریت کاربران است. از آنجا که کلاس UserStore مبتنی بر Entity Framework است، باید آبجکت DbContext را هم پاس دهیم. این کد در واقع همان کدی است که در متد سازنده کنترلر Account وجود دارد. یک مزیت دیگر این روش این است که می‌توانیم متدهای UserManager را بازنویسی (overwrite) کنیم. برای پیاده سازی نیازمندی‌های بعدی دقیقاً همین کار را خواهیم کرد.

حال باید کلاس ApplicationUserManager را در کنترلر Account استفاده کنیم. متد سازنده و خاصیت UserManager را مانند زیر تغییر دهید.

```
public AccountController() : this(new ApplicationUserManager())
{
}

public AccountController(ApplicationUserManager userManager)
{
    UserManager = userManager;
}

public ApplicationUserManager UserManager { get; private set; }
```

حالا داریم از کلاس سفارشی جدیدمان استفاده می‌کنیم. این به ما اجازه می‌دهد مراحل بعدی سفارشی سازی را انجام دهیم، بدون آنکه کدهای موجود در کنترلر از کار بیافتند. اپلیکیشن را اجرا کنید و سعی کنید کاربر محلی جدیدی ثبت نمایید. اگر کلمه عبور وارد شده کمتر از 10 کاراکتر باشد پیغام خطای زیر را دریافت می‌کنید.

Register.

Create a new account.

- Passwords must be at least 10 characters.

User name

Password

Confirm password

قانون 2: کلمه‌های عبور باید حداقل یک عدد و یک کاراکتر ویژه داشته باشند

چیزی که در این مرحله نیاز داریم کلاس جدیدی است که اینترفیس `IIdentityValidator` را پیاده سازی می‌کند. چیزی که ما می‌خواهیم اعتبارسنجی کنیم، وجود اعداد و کاراکترهای ویژه در کلمه عبور است، همچنین طول مجاز هم بررسی می‌شود. نهایتاً این قوانین اعتبارسنجی در متد `'ValidateAsync'` بکار گرفته خواهند شد. در پوشه `IdentityExtensions` کلاس جدیدی بنام `CustomPasswordValidator` بسازید و اینترفیس مذکور را پیاده سازی کنید. از آنجا که نوع کلمه عبور رشته (`string`) است از `IIdentityValidator<string>` استفاده می‌کنیم.

```
public class CustomPasswordValidator : IIdentityValidator<string>
{
    public int RequiredLength { get; set; }

    public CustomPasswordValidator(int length)
    {
        RequiredLength = length;
    }

    public Task<IdentityResult> ValidateAsync(string item)
    {
        if (String.IsNullOrEmpty(item) || item.Length < RequiredLength)
        {
            return Task.FromResult(IdentityResult.Failed(String.Format("Password should be of length {0}", RequiredLength)));
        }

        string pattern = @"^(?=[0-9])(?=.*[!@#$%^&*])[0-9a-zA-Z!@#$%^&*0-9]{10,}$";

        if (!Regex.IsMatch(item, pattern))
        {
            return Task.FromResult(IdentityResult.Failed("Password should have one numeral and one special character"));
        }
    }
}
```

```
return Task.FromResult(IdentityResult.Success);
}
```

در متد **ValidateAsync** بررسی می‌کنیم که طول کلمه عبور معتبر و مجاز است یا خیر. سپس با استفاده از یک RegEx وجود کاراکترهای ویژه و اعداد را بررسی می‌کنیم. دقت کنید که regex استفاده شده تست نشده و تنها بعنوان یک مثال باید در نظر گرفته شود.

قدم بعدی تعریف این اعتبارسنج سفارشی در کلاس UserManager است. باید مقدار خاصیت PasswordValidator را به این کلاس تنظیم کنیم. به کلاس ApplicationUserManager که پیشتر ساختید بروید و مقدار خاصیت PasswordValidator را به CustomPasswordValidator تغییر دهید.

```
public class ApplicationUserManager : UserManager<ApplicationUser>
{
    public ApplicationUserManager() : base(new UserStore<ApplicationUser>(new ApplicationDbContext()))
    {
        PasswordValidator = new CustomPasswordValidator(10);
    }
}
```

هیچ تغییر دیگری در کلاس AccountController لازم نیست. حال سعی کنید کاربر جدید دیگری بسازید، اما اینبار کلمه عبوری وارد کنید که خطای اعتبارسنجی تولید کند. پیغام خطایی مشابه تصویر زیر باید دریافت کنید.

Register.

Create a new account.

- Password should have one numeral and one special character

User name

Password

Confirm password

قانون 3: امکان استفاده از 5 کلمه عبور اخیر ثبت شده وجود ندارد

هنگامی که کاربران سیستم، کلمه عبور خود را بازنشانی (reset) می کنند یا تغییر می دهند، می توانیم بررسی کنیم که آیا مجدداً از یک کلمه عبور پیشین استفاده کرده اند یا خیر. این بررسی بصورت پیش فرض انجام نمی شود، چرا که سیستم Identity تاریخچه کلمه های عبور کاربران را ذخیره نمی کند. می توانیم در اپلیکیشن خود جدول جدیدی بسازیم و تاریخچه کلمات عبور کاربران را در آن ذخیره کنیم. هر بار که کاربر سعی در بازنشانی یا تغییر کلمه عبور خود دارد، مقدار Hash شده را در جدول تاریخچه بررسی می کنیم.

فایل IdentityModels.cs را باز کنید. مانند لیست زیر، کلاس جدیدی بنام 'PreviousPassword' بسازید.

```
public class PreviousPassword
{
    public PreviousPassword()
    {
        CreateDate = DateTimeOffset.Now;
    }

    [Key, Column(Order = 0)]
    public string PasswordHash { get; set; }
    public DateTimeOffset CreateDate { get; set; }

    [Key, Column(Order = 1)]
    public string UserId { get; set; }
    public virtual ApplicationUser User { get; set; }
}
```

در این کلاس، فیلد 'Password' مقدار Hash شده کلمه عبور را نگاه میدارد و توسط فیلد 'UserId' رفرنس می شود. فیلد 'CreateDate' یک مقدار timestamp ذخیره می کند که تاریخ ثبت کلمه عبور را مشخص می نماید. توسط این فیلد می توانیم تاریخچه کلمات عبور را فیلتر کنیم و مثلاً 5 رکورد آخر را بگیریم.

Entity Framework Code First جدول 'PreviousPasswords' را می سازد و با استفاده از فیلدهای 'Password' و 'UserId' کلید اصلی (composite primary key) را ایجاد می کند. برای اطلاعات بیشتر درباره قراردادهای EF Code First به [این لینک](#) مراجعه کنید. خاصیت جدیدی به کلاس ApplicationUser اضافه کنید تا لیست آخرین کلمات عبور استفاده شده را نگهداری کند.

```
public class ApplicationUser : IdentityUser
{
    public ApplicationUser() : base()
    {
        PreviousUserPasswords = new List<PreviousPassword>();
    }

    public virtual IList<PreviousPassword> PreviousUserPasswords { get; set; }
}
```

همانطور که پیشتر گفته شد، کلاس UserStore پیاده سازی API های لازم برای مدیریت کاربران را در بر می گیرد. هنگامی که کاربر برای نخستین بار در سایت ثبت می شود باید مقدار Hash کلمه عبورش را در جدول تاریخچه کلمات عبور ذخیره کنیم. از آنجا که UserStore بصورت پیش فرض متدی برای چنین عملیاتی معرفی نمی کند، باید یک override تعریف کنیم تا این مراحل را انجام دهیم. پس ابتدا باید کلاس سفارشی جدیدی بسازیم که از UserStore ارث بری کرده و آن را توسعه می دهد. سپس از این کلاس سفارشی در ApplicationUserManager بعنوان پیاده سازی پیش فرض UserStore استفاده می کنیم. پس کلاس جدیدی در پوشه IdentityExtensions ایجاد کنید.

```
public class ApplicationUserStore : UserStore<ApplicationUser>
{
    public ApplicationUserStore(DbContext context) : base(context) { }
```

```

public override async Task CreateAsync(ApplicationUser user)
{
    await base.CreateAsync(user);
    await AddToPreviousPasswordsAsync(user, user.PasswordHash);
}

public Task AddToPreviousPasswordsAsync(ApplicationUser user, string password)
{
    user.PreviousUserPasswords.Add(new PreviousPassword() { UserId = user.Id, PasswordHash = password });
    return UpdateAsync(user);
}
}

```

متد 'AddToPreviousPasswordsAsync' کلمه عبور را در جدول 'PreviousPasswords' ذخیره می‌کند. هرگاه کاربر سعی در بازنشانی یا تغییر کلمه عبورش دارد باید این متد را فراخوانی کنیم. API‌های لازم برای این کار در کلاس UserManager تعریف شده‌اند. باید این متدها را override کنیم و فراخوانی متد مذکور را پیاده کنیم. برای این کار کلاس ApplicationUserManager را باز کنید و متدهای ChangePassword و ResetPassword را بازنویسی کنید.

```

public class ApplicationUserManager : UserManager<ApplicationUser>
{
    private const int PASSWORD_HISTORY_LIMIT = 5;

    public ApplicationUserManager() : base(new ApplicationUserStore(new ApplicationDbContext()))
    {
        PasswordValidator = new CustomPasswordValidator(10);
    }

    public override async Task<IdentityResult> ChangePasswordAsync(string userId, string currentPassword, string newPassword)
    {
        if (await IsPreviousPassword(userId, newPassword))
        {
            return await Task.FromResult(IdentityResult.Failed("Cannot reuse old password"));
        }

        var result = await base.ChangePasswordAsync(userId, currentPassword, newPassword);

        if (result.Succeeded)
        {
            var store = Store as ApplicationUserStore;
            await store.AddToPreviousPasswordsAsync(await FindByIdAsync(userId), PasswordHasher.HashPassword(newPassword));
        }

        return result;
    }

    public override async Task<IdentityResult> ResetPasswordAsync(string userId, string token, string newPassword)
    {
        if (await IsPreviousPassword(userId, newPassword))
        {
            return await Task.FromResult(IdentityResult.Failed("Cannot reuse old password"));
        }

        var result = await base.ResetPasswordAsync(userId, token, newPassword);

        if (result.Succeeded)
        {
            var store = Store as ApplicationUserStore;
            await store.AddToPreviousPasswordsAsync(await FindByIdAsync(userId), PasswordHasher.HashPassword(newPassword));
        }

        return result;
    }

    private async Task<bool> IsPreviousPassword(string userId, string newPassword)
    {
        var user = await FindByIdAsync(userId);
    }
}

```



```

        if (user.PreviousUserPasswords.OrderByDescending(x => x.CreateDate).
            Select(x => x.PasswordHash).Take(PASSWORD_HISTORY_LIMIT)
            .Where(x => PasswordHasher.VerifyHashedPassword(x, newPassword) !=
PasswordVerificationResult.Failed).Any())
        {
            return true;
        }
        return false;
    }
}

```

فیلد 'PASSWORD_HISTORY_LIMIT' برای دریافت X رکورد از جدول تاریخچه کلمه عبور استفاده می‌شود. همانطور که می‌بینید از متد سازنده کلاس ApplicationUserStore برای گرفتن متد جدیدمان استفاده کرده ایم. هرگاه کاربری سعی می‌کند کلمه عبورش را بازنشانی کند یا تغییر دهد، کلمه عبورش را با 5 کلمه عبور قبلی استفاده شده مقایسه می‌کنیم و بر این اساس مقدار true/false بر می‌گردانیم.

کاربر جدیدی بسازید و به صفحه **Manage** بروید. حال سعی کنید کلمه عبور را تغییر دهید و از کلمه عبور فعلی برای مقدار جدید استفاده کنید تا خطای اعتبارسنجی تولید شود. پیامی مانند تصویر زیر باید دریافت کنید.

The screenshot shows the 'Manage Account' page. At the top, it says 'You're logged in as foo.' Below that is the 'Change Password Form'. A red error message states: '• Cannot reuse old password'. The form contains three input fields: 'Current password', 'New password', and 'Confirm new password'. Below these fields is a 'Change password' button. At the bottom of the form, there is a link that says 'Use another service to log in.'

سورس کد این مثال را می‌توانید از [این لینک](#) دریافت کنید. نام پروژه Identity-PasswordPolicy است، و زیر قسمت Samples/Identity قرار دارد.

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 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(new CookieAuthenticationOptions {
    AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
    LoginPath = new PathString("/Account/Login"),
    Provider = new CookieAuthenticationProvider {
        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
publicclass ApplicationUser : IdentityUser<int, CustomUserLogin, CustomUserRole, CustomUserClaim>
{
}
publicclass CustomRole : IdentityRole<int, CustomUserRole>
{
    public CustomRole() { }
    public CustomRole(string name) { Name = name; }
}
publicclass CustomUserRole : IdentityUserRole<int> { }
publicclass CustomUserClaim : IdentityUserClaim<int> { }
publicclass CustomUserLogin : IdentityUserLogin<int> { }

publicclass ApplicationDbContext : IdentityDbContext<ApplicationUser, CustomRole, int, CustomUserLogin, CustomUserClaim>
{
}

```

پشتیبانی از IQueryable روی Users و Roles

کلاس‌های UserStore و RoleStore حالا از IQueryable پشتیبانی می‌کنند، بنابراین می‌توانید ب راحتی 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(new UserManagerOptions<ApplicationUserManager>()
{
    DataProtectionProvider = app.GetDataProtectionProvider(),
    Provider = new UserManagerProvider<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 کلاسی ApplicationDbContext را رفرنس می‌کند. میان افزار DbContextFactory به ازای هر درخواست در اپلیکیشن یک وهله از ApplicationDbContext را به شما تحویل می‌دهد. می‌توانید 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. لطفاً 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 شده‌اند: '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 رسانی‌هایی 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 پیش‌نمایش‌های 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: ۲۰:۴۴ ۱۳۹۲/۱۰/۳۰

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، برنامه ها بصورت SSO کار می کرد و احتیاجی به هیچ کاری نداشت. حالا در سیستم جدید همون روش جواب میده؟ برای برنامه های با دامین های متفاوت چطور؟

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

[CookieDomain](#) رو باید تنظیم کنید.