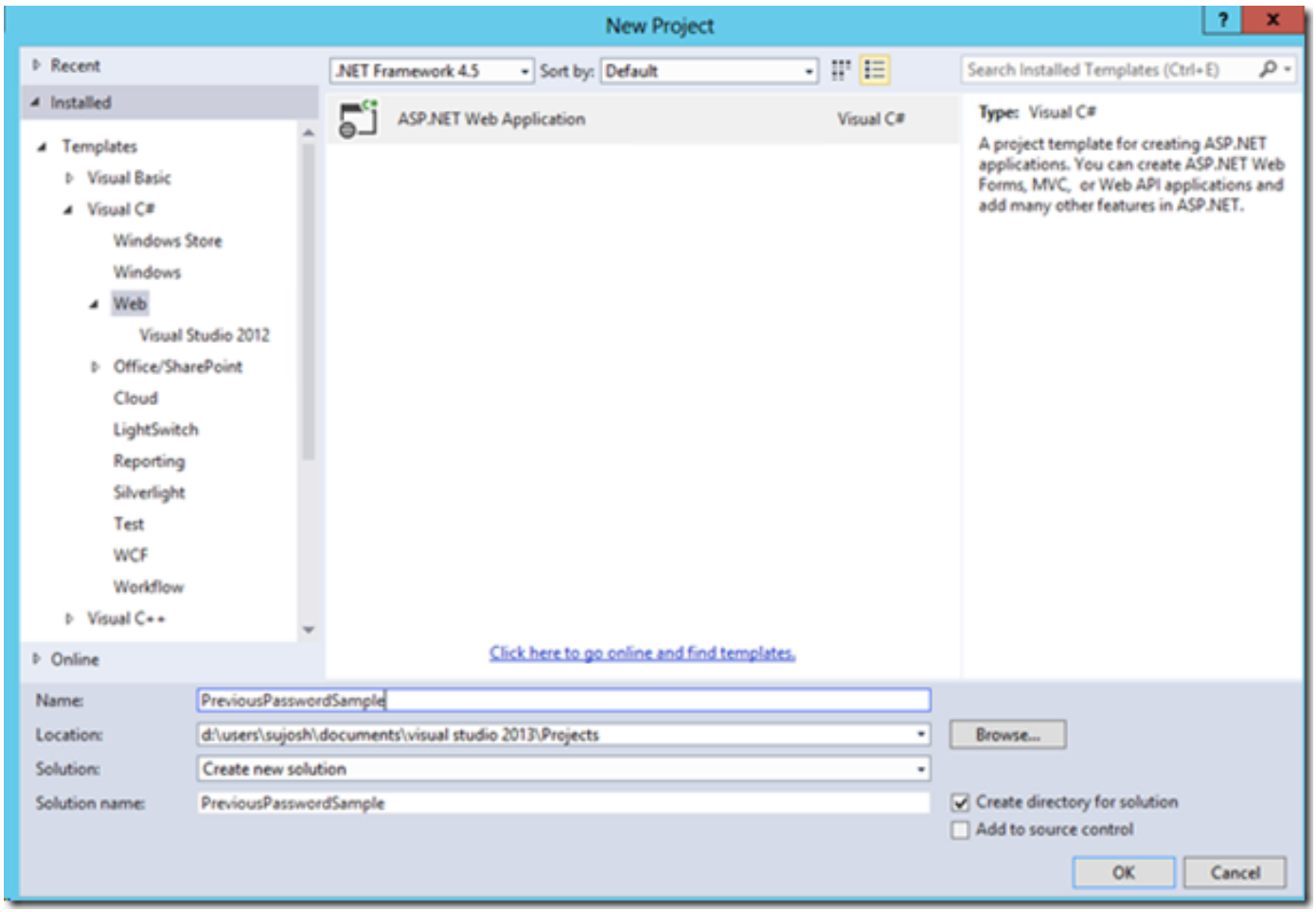
عنوان: **پیاده سازی Password Policy سفارشی توسط ASP.NET Identity**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۰/۲۲ ۲۱:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Tips, Identity, Visual Studio 2013, ASP.NET Identity

برای فراهم کردن یک تجربه کاربری ایمن‌تر و بهتر، ممکن است بخواهید پیچیدگی password policy را سفارشی سازی کنید. مثلا 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

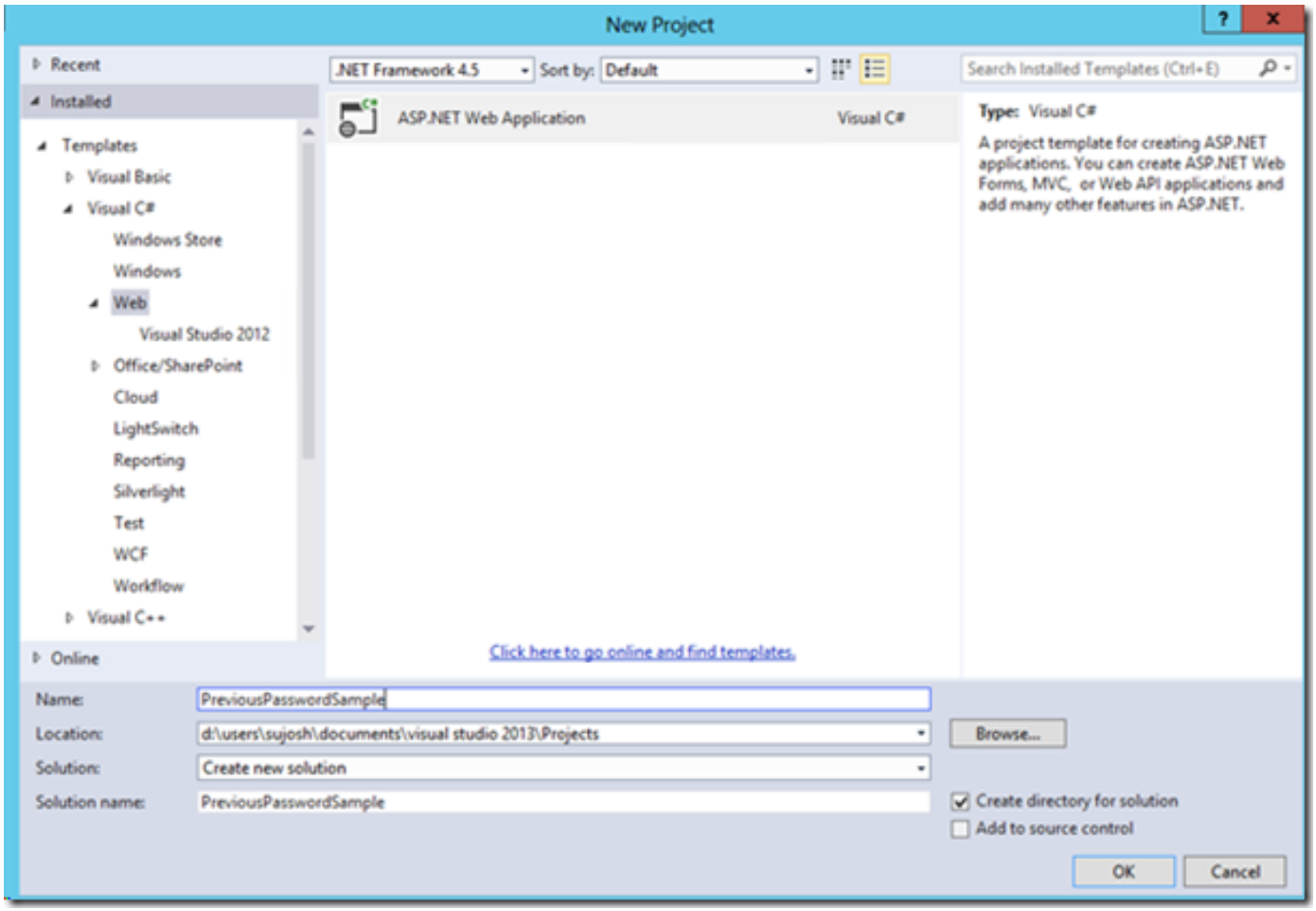
در آخر اپلیکیشن را اجرا می‌کنیم و عملکرد این قوانین جدید را بررسی خواهیم کرد.

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 کاربران جدید در اپلیکیشن نگاهی بیاندازیم. این به ما در شناسایی نیازهای جدیدمان کمک می‌کند. در پوشه Controllers فایلی بنام AccountController.cs وجود دارد که 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 وهله 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 وهله 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 APIهای 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.

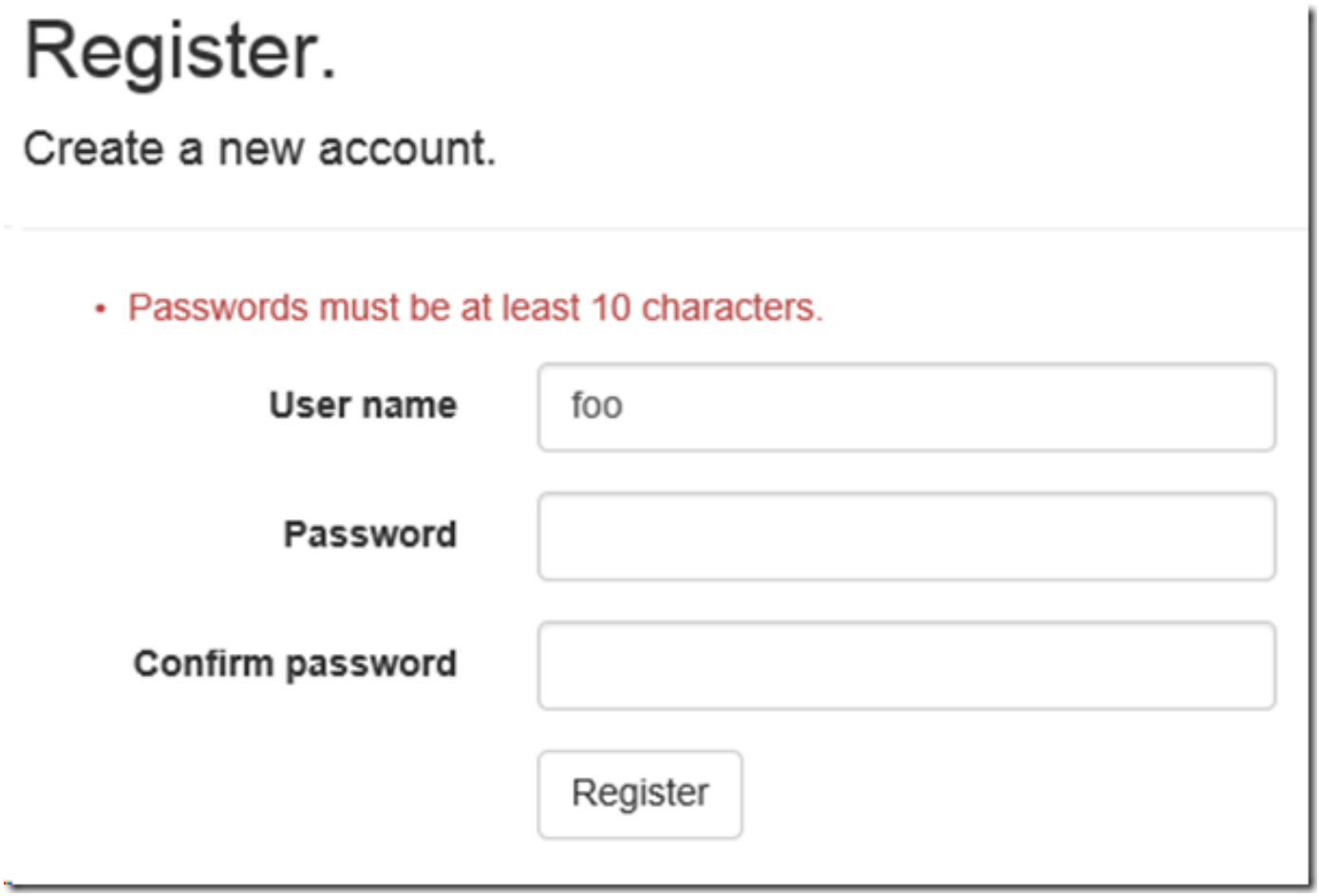
یک مزیت دیگر این روش این است که می‌توانیم متدهای UserManager را بازنویسی (overwrite) کنیم. برای پیاده سازی نیازمندهای بعدی دقیقا همین کار را خواهیم کرد.

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.



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. نهایتا 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.

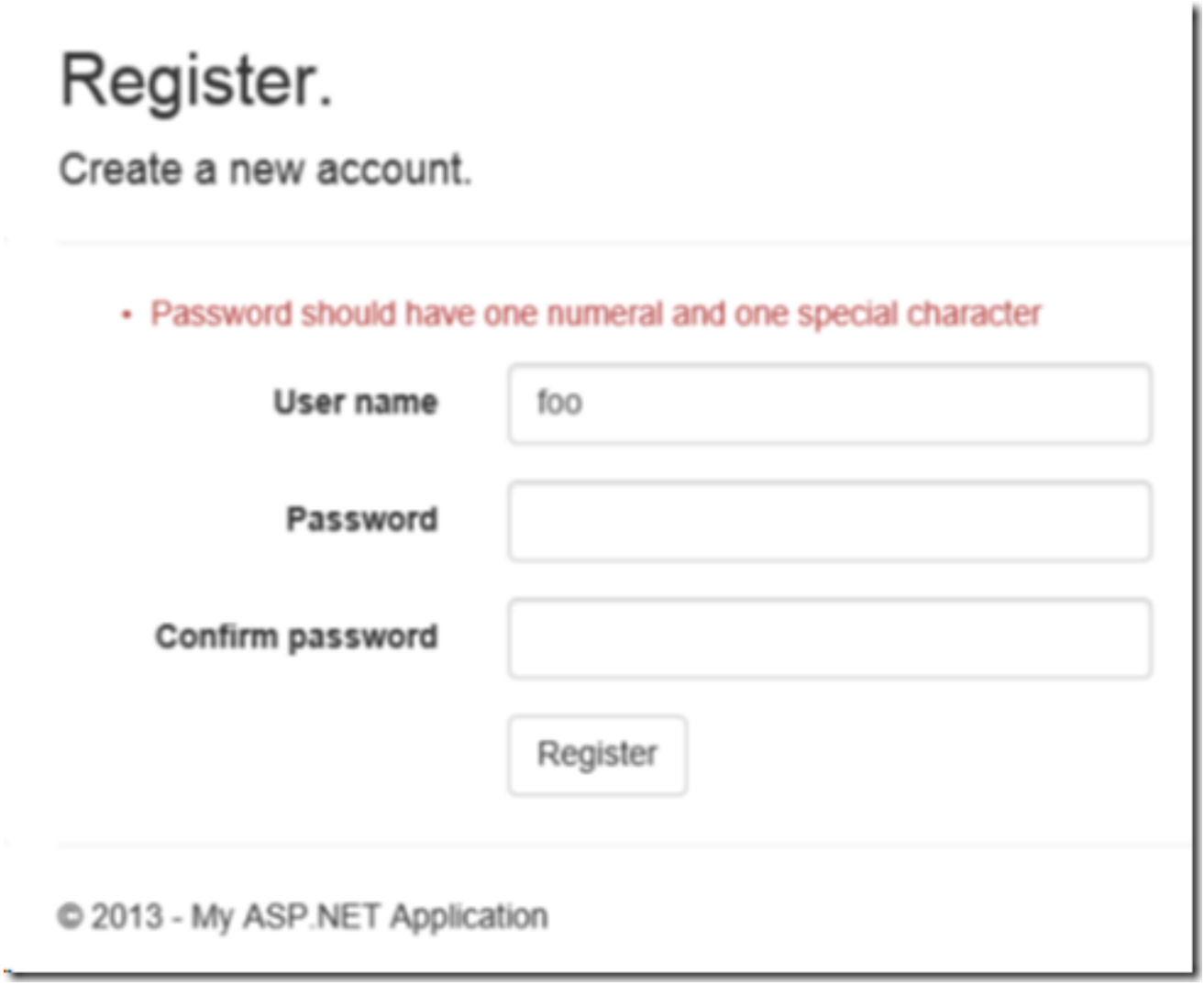
```
public class CustomPasswordValidator : IIdentityValidator<string>
{
    public int RequiredLength { get; set; }

    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** بررس 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.



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

هنگامی که کاربران سیستم، کلمه عبور خود را بازنشانی (reset) می‌کنند یا تغییر می‌دهند، می‌توانیم بررسی کنیم که آیا مجددا از یک کلمه عبور پیشین استفاده کرده اند یا خیر. این بررسی بصورت پیش فرض انجام نمی‌شود، چرا که سیستم Identity تاریخچه کلمه‌های عبور کاربران را ذخیره نمی‌کند. می‌توانیم در اپلیکیشن خود جدول جدیدی بسازیم و تاریخچه کلمات عبور کاربران را در آن ذخیره کنیم. هربار که کاربر سعی در بازنشانی یا تغییر کلمه عبور خود دارد، مقدار Hash شده را در جدول تاریخچه بررسی میکنیم.

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 می‌شود. فیلد 'CreateDate' یک مقدار timestamp ذخیره می‌کند که تاریخ ثبت کلمه عبور را مشخص می‌نماید. توسط این فیلد می‌توانیم تاریخچه کلمات عبور را فیلتر کنیم و مثلا 5 رکورد آخر را بگیریم.

Entity Framework Code First جدول 'PreviousPasswords' را می‌سازد و با استفاده از فیلدهای 'UserId' و 'Password' کلید اصلی (composite primary key) را ایجاد می‌کند. برای اطلاعات بیشتر درباره قرارداهای 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 APIهای لازم برای مدیریت کاربران را در بر می‌گیرد. هنگامی که کاربر برای نخستین بار در سایت ثبت می‌شود 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 نمی‌کند، 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 می‌دهد. سپس از این کلاس سفارشی در ApplicationUserManager بعنوان پیاده سازی پیش فرض UserStore استفاده می‌کنیم. 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);
     }
 }
```

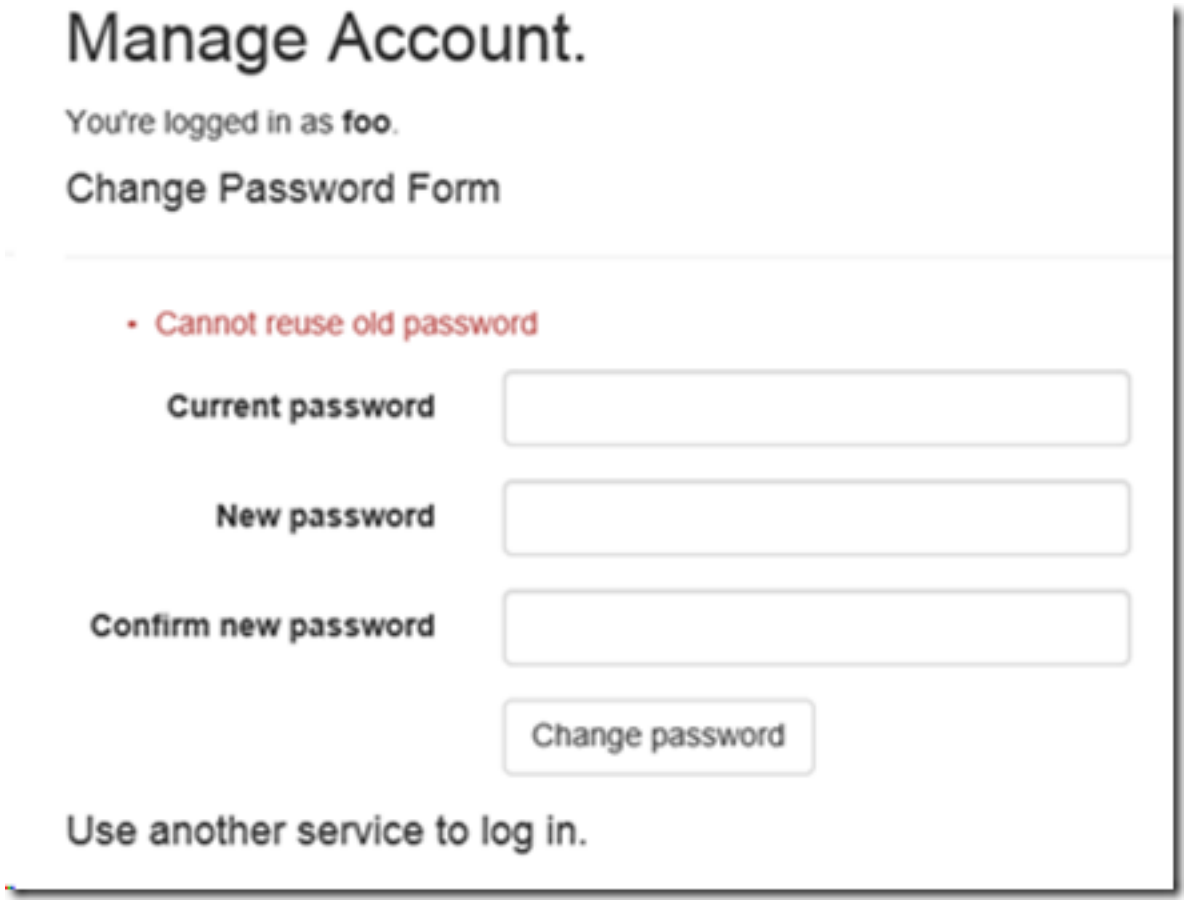
متد 'AddToPreviousPasswordsAsync' کلمه عبور را در جدول 'PreviousPasswords' ذخیره می‌کند.
هرگاه کاربر سعی در بازنشانی یا تغییر کلمه عبورش دارد باید این متد را فراخوانی کنیم. APIهای لازم برای این کار در کلاس UserManager تعریف شده اند. باید این متدها را override کنیم و فراخوانی متد مذکور را پیاده کنیم. 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. پیغامی مانند تصویر زیر باید دریافت کنید.



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