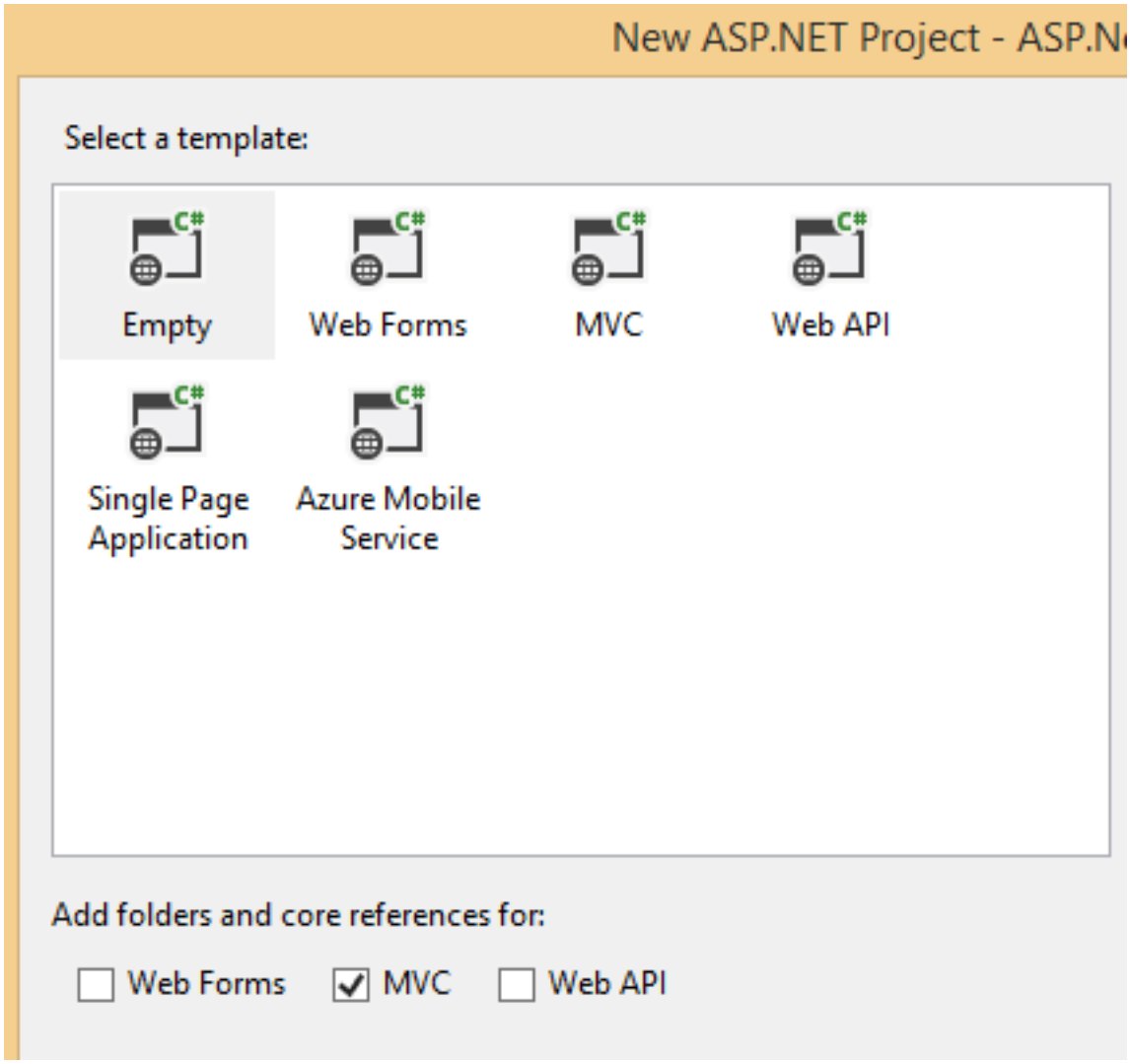
عنوان: **Asp.Net Identity #2**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۴/۰۴/۱۰ ۲۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Security, ASP.NET Identity

پیشتر در اینجا در مورد تاریخچه‌ی سیستم Identity مطالبی را عنوان کردیم. در این مقاله می‌خواهیم نحوه‌ی برپایی سیستم Identity را بحث کنیم.
ASP.NET Identity مانند ASP.NET Membership به اسکیمای SQL Server وابسته نیست؛ اما Relational Storage همچنان واحد ذخیره سازی پیش فرض و آسانی می‌باشد. بدین جهت که تقریبا بین همه‌ی توسعه دهندگان جا افتاده است. ما در این نوشتار از LocalDB جهت ذخیره سازی جداول استفاده می‌کنیم. ذکر این نکته ضروری است که سیستم Identity از Entity Framework Code First جهت ساخت اسکیما استفاده می‌کند.
پیش از هر چیز، ابتدا یک پروژه‌ی وب را ایجاد کنید؛ مانند شکل زیر:




در مرحله‌ی بعد سه پکیج زیر را باید نصب کنیم :

```
Microsoft.AspNet.Identity.EntityFramework -
```

- Microsoft.AspNet.Identity.OWIN

- Microsoft.Owin.Host.SystemWeb

بعد از اینکه پکیج‌های بالا را نصب کردیم، باید فایل Web.config را بروز کنیم. اولین مورد، تعریف یک Connection String می‌باشد:

```
<connectionStrings>
 <add name="IdentityDb"
      providerName="System.Data.SqlClient"
      connectionString="Data Source=(localdb)\v11.0;Initial Catalog=IdentityDb;Integrated
Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;MultipleActiveResultSets=True"/>
 </connectionStrings>
```

بعد از آن، تعریف یک کلید در قسمت AppSettings تحت عنوان Owin:AppStartup است:

```
<appSettings>
 <add key="webpages:Version" value="3.0.0.0" />
 <add key="webpages:Enabled" value="false" />
 <add key="ClientValidationEnabled" value="true" />
 <add key="UnobtrusiveJavaScriptEnabled" value="true" />
```

**</ "add key="owin:AppStartup" value="Users.IdentityConfig>**

```
</appSettings>
```

Owin مدل شروع برنامه (Application Startup Model) خودش را تعریف می‌کند که از کلاس کلی برنامه (منظور Global.asax) جداست. AppSetting تعریف شده با نام owin:Startup شناخته می‌شود و مشخص کننده کلاسی است که Owin وهله سازی خواهد کرد. وقتی برنامه شروع به کار می‌کند، تنظیمات خودش را از این کلاس دریافت خواهد کرد (در این نوشتار کلاس IdentityConfig می‌باشد).

**ساخت کلاس User**

مرحله‌ی بعد ساخت کلاس User می‌باشد. این کلاس بیانگر یک کاربر می‌باشد. کلاس User باید از کلاس IdentityUser ارث بری کند که این کلاس در فضای نام Microsoft.AspNet.Identity.EntityFramework قرار دارد. کلاس IdentityUser فراهم کننده‌ی یک کاربر عمومی و ابتدایی است. اگر بخواهیم اطلاعات اضافی مربوط به کاربر را ذخیره کنیم باید آنها در کلاسی که از کلاس IdentityUser ارث بری می‌کند قرار دهیم. کلاس ما در اینجا AppUser نام دارد.

```
using System;
using Microsoft.AspNet.Identity.EntityFramework;
namespace Users.Models
{
   public class AppUser : IdentityUser
  {
      // پروپرتی‌های اضافی در اینجا
  }
}
```

**ساخت کلاس Database Context برنامه**

مرحله‌ی بعد ساخت کلاس DbContext برنامه می‌باشد که بر روی کلاس AppUser عمل میکند. کلاس Context برنامه که ما در اینجا آن را AppIdentityDbContext تعریف کرده‌ایم، از کلاس IdentityDbContext<T> ارث بری می‌کند که T همان کلاس User می‌باشد (در مثال ما AppUser) .

```
using System.Data.Entity;
using Microsoft.AspNet.Identity.EntityFramework;
using Users.Models;
```

```
namespace Users.Infrastructure {
 public class AppIdentityDbContext : IdentityDbContext<AppUser>
{
   public AppIdentityDbContext()
               : base("IdentityDb") { }

  static AppIdentityDbContext()
 {
    Database.SetInitializer<AppIdentityDbContext>(new IdentityDbInit());
  }
 public static AppIdentityDbContext Create() {
 return new AppIdentityDbContext();
 }
 }
public class IdentityDbInit
 : DropCreateDatabaseIfModelChanges<AppIdentityDbContext> {
 protected override void Seed(AppIdentityDbContext context) {
 PerformInitialSetup(context);
 base.Seed(context);
 }
 public void PerformInitialSetup(AppIdentityDbContext context) {
 // initial configuration will go here
 }
 }
}
```

**ساخت کلاس User Manager**

یکی از مهمترین کلاسهای Identity کلاس User Manager (مدیر کاربر) می‌باشد که نمونه‌هایی از کلاس User را مدیریت می‌کند. کلاسی را که تعریف می‌کنیم، باید از کلاس UserManager<T> ارث بری کند که T همان کلاس User می‌باشد. کلاس UserManager خاص EF نمی‌باشد و ویژگی‌های عمومی بیشتری برای ساخت و انجام عملیات بر روی داده‌های کاربر را فراهم می‌نماید.

| Name | Description |
|---|---|
| ChangePasswordAsync(id, old, new) | Changes the password for the specified user. |
| CreateAsync(user) | Creates a new user without a password. See Chapter 15 for an example. |
| CreateAsync(user, pass) | Creates a new user with the specified password. See the “Creating Users” section. |
| DeleteAsync(user) | Deletes the specified user. See the “Implementing the Delete Feature” section. |
| FindAsync(user, pass) | Finds the object that represents the user and authenticates their password. See Chapter 14 for details of authentication. |
| FindByIdAsync(id) | Finds the user object associated with the specified ID. See the “Implementing the Delete Feature” section. |
| FindByNameAsync(name) | Finds the user object associated with the specified name. I use this method in the “Seeding the Database” section of Chapter 14. |
| UpdateAsync(user) | Pushes changes to a user object back into the database. See the “Implementing the Edit Feature” section. |
| Users | Returns an enumeration of the users. See the “Enumerating User Accounts” section. |

کلاس UserManager حاوی متدهای بالا است. اگر دقت کنید، می‌بینید که تمامی متدهای بالا به کلمه‌ی Async ختم می‌شوند. زیرا Asp.Net Identity تقریبا کل ویژگیهای برنامه نویسی Async را پیاده سازی کرده است و این بدین معنی است که عملیات میتوانند به صورت غیر همزمان اجرا شده و دیگر فعالیت‌ها را بلوکه نکنند.

```
using Microsoft.AspNet.Identity;
```

```
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using Users.Models;

namespace Users.Infrastructure
{
 public class AppUserManager : UserManager<AppUser>
{

 public AppUserManager(IUserStore<AppUser> store)
 : base(store) {
 }
 public static AppUserManager Create( IdentityFactoryOptions<AppUserManager> options, IOwinContext
context)
{
 AppIdentityDbContext db = context.Get<AppIdentityDbContext>();
 AppUserManager manager = new AppUserManager(new UserStore<AppUser>(db));
 return manager;
 }
 }
}
```

زمانی که Identity نیاز به وهله‌ای از کلاس AppUserManager داشته باشد، متد استاتیک Create را صدا خواهد زد. این عمل زمانی اتفاق می‌افتد که عملیاتی بر روی داده‌های کاربر انجام گیرد. برای ساخت وهله‌ای از کلاس AppUserManager نیاز به کلاس <UserStor<AppUser دارد. کلاس <UserStore<T یک پیاده سازی از رابط <IUserStore<T توسط EF میباشد که وظیفه‌ی آن فراهم کننده‌ی پیاده سازی Storage-Specific متدهای تعریف شده در کلاس User Manager (که در اینجا AppUserManager ) می‌باشد. برای ساخت <UserStore<T نیاز به وهله‌ای از کلاس AppIdentityDbContext می‌باشد که از طریق Owin به صورت زیر قابل دریافت است:

```
AppIdentityDbContext db = context.Get<AppIdentityDbContext>();
```

یک پیاده سازی از رابط IOwinContext، به عنوان پارامتر به متد Create پاس داده می‌شود. در این پیاده سازی، یک تابع جنریک به نام Get تعریف شده که اقدام به برگشت وهله ای از اشیای ثبت شده‌ی در کلاس شروع Owin می‌نماید.

**ساخت کلاس شروع Owin**

اگر خاطرتان باشد یک کلید در قسمت AppSettings فایل Web.config به صورت زیر تعریف کردیم:

```
<add key="owin:AppStartup" value="Users.IdentityConfig" />
```

قسمت Value کلید بالا از دو قسمت تشکیل شده است: Users بیانگر فضای نام برنامه‌ی شماست و IdentityConfig بیانگر کلاس شروع می‌باشد. (البته شما می‌توانید هر نام دلخواهی را برای کلاس شروع بگذارید. فقط دقت داشته باشید که نام کلاس شروع و مقدار، با کلیدی که تعریف می‌کنید یکی باشد)

Owin مستقل از ASP.NET اعلام شده است و قراردادهای خاص خودش را دارد. یکی از این قراردادها تعریف یک کلاس و وهله سازی آن به منظور بارگذاری و پیکربندی میان افزار و انجام دیگر کارهای پیکربندی که نیاز است، می‌باشد. به طور پیش فرض این کلاس Start نام دارد و در پوشه‌ی App_Start تعریف می‌شود. این کلاس حاوی یک متد به نام Configuration می‌باشد که بوسیله زیرساخت Owin فراخوانی می‌شود و یک پیاده سازی از رابط Owin.IAppBuilder به عنوان آرگومان به آن پاس داده می‌شود که کار پشتیبانی از Setup میان افزار مورد نیاز برنامه را برعهده دارد.

```
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
using Users.Infrastructure;
namespace Users
{
 public class IdentityConfig
```

```
{
 public void Configuration(IAppBuilder app)
{
 app.CreatePerOwinContext<AppIdentityDbContext>(AppIdentityDbContext.Create);
 app.CreatePerOwinContext<AppUserManager>(AppUserManager.Create);
 app.UseCookieAuthentication(new CookieAuthenticationOptions {
 AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
 LoginPath = new PathString("/Account/Login"),
 });
 }
 }
}
```

رابط IAppBuilder بوسیله تعدادی متد الحاقی که در کلاسهایی که در فضای نام Owin تعریف شده‌اند، تکمیل شده است. متد CreatePerOwinContext کار ساخت وهله‌ای از کلاس AppUserManager و کلاس AppIdentityDbContext را برای هر درخواست بر عهده دارد. این مورد تضمین می‌کند که هر درخواست، به یک داده‌ی تمیز از Asp.Net Identity دسترسی خواهد داشت و نگران اعمال همزمانی و یا کش ضعیف داده نخواهد بود. متد UseCookieAuthentication به Asp.Net Identity می‌گوید که چگونه از کوکی‌ها برای تصدیق هویت کاربران استفاده کند که Optionهای آن از طریق کلاس CookieAuthenticationOptions مشخص می‌شود. مهمترین قسمت در اینجا پروپرتی LoginPath می‌باشد و مشخص می‌کند که کلاینت‌های تصدیق هویت نشده، هنگام دسترسی به یک منبع محافظت شده، به کدام URL هدایت شوند که توسط یک رشته به متد PathString پاس داده می‌شود.

خوب دوستان برپایی سیستم Identity به پایان رسید. انشالله در قسمت بعدی به چگونگی استفاده‌ی از این سیستم خواهیم پرداخت.