عنوان: **Asp.Net Identity #3**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۴/۰۴/۱۷ ۱۹:۵۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, Security, ASP.NET Identity

در مقاله‌ی پیشین نگاهی داشتیم به نحوه‌ی برپایی سیستم Identity. در این مقاله به نحوه‌ی استفاده از این سیستم به منظور طراحی یک سیستم مدیریت کاربران خواهیم پرداخت و انشالله در مقاله‌های بعدی این سیستم را تکمیل خواهیم نمود. کار را با اضافه کردن یک کنترلر جدید به پروژه آغاز می‌کنیم.

```
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity.Owin;
using Users.Infrastructure;

namespace Users.Controllers
{
    public class HomeController : Controller
    {
        private AppUserManager UserManager
        {
            get { return HttpContext.GetOwinContext().GetUserManager<AppUserManager>(); }
        }
        // GET: Home
        public ActionResult Index()
        {
            return View(UserManager.Users);

        }

}
```

در خط 10 یک پروپرتی از نوع AppUserManager (کلاسی که مدیریت کاربران را برعهده دارد) ایجاد می‌کنیم. اسمبلی Microsoft.Owin.Host.SystemWeb یک سری متدهای الحاقی را به کلاس HttpContext اضافه می‌کند که یکی از آنها متد GetOwinContext می‌باشد. این متد یک شیء Per-Request Context را از طریق رابط IOwinContext به OwinApi ارسال می‌کند؛ با استفاده از متد الحاقی GetUserManager<T> که T همان کلاس AppUserManager می‌باشد. حال که نمونه‌ای از کلاس AppUserManager را بدست آوردیم، می‌توانیم درخواستهایی را به جداول کاربران بدهیم. مثلا در خط 17 با استفاده از پروپرتی Users میتوانیم لیست کاربران موجود را بدست آورده و آن را به ویو پاس دهیم.

```
@using Users.Models
@model IEnumerable<AppUser>
@{
    ViewBag.Title = "Index";
}
<div class="panel panel-primary">
    <div class="panel-heading">
        User Accounts
    </div>
    <table class="table table-striped">
        <tr><th>ID</th><th>Name</th><th>Email</th></tr>
        @if (!Model.Any())
        {
            <tr><td colspan="3" class="text-center">No User Accounts</td></tr>
        }
        else
        {
            foreach (AppUser user in Model)
            {
                <tr>
                    <td>@user.Id</td>
                    <td>@user.UserName</td>
                    <td>@user.Email</td>
                </tr>
            }
        }
    </table>
</div>
@Html.ActionLink("Create", "CreateUser", null, new { @class = "btn btn-primary" })
```

نحوه‌ی ساخت یک کاربر جدید

ابتدا در پوشه Models یک کلاس ایجاد کنید :

```
namespace Users.Models
    {
        public class CreateModel
        {
            [Required]
            public string Name { get; set; }
            [Required]
            public string Email { get; set; }
            [Required]
            public string Password { get; set; }
        }
    }
```

فقط دوستان توجه داشته باشید که در پروژه‌های حرفه‌ای و تجاری هرگز اطلاعات مهم مربوط به مدل‌ها را در پوشه‌ی Models قرار ندهید. ما در اینجا صرف آموزش و برای جلوگیری از پیچیدگی مثال این کار را انجام میدهیم. برای اطلاعات بیشتر به این مقاله مراجعه کنید.

حال در کنترلر برنامه کدهای زیر را اضافه می‌کنیم:

```
public ActionResult CreateUser()
        {
            return View();
        }

        [HttpPost]
        public async Task<ActionResult> CreateUser(CreateModel model)
        {
            if (!ModelState.IsValid)
                return View(model);

            var user = new AppUser { UserName = model.Name, Email = model.Email };
            var result = await UserManager.CreateAsync(user, model.Password);

            if (result.Succeeded)
            {
                return RedirectToAction("Index");
            }

            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
            return View(model);
        }
```

در اکشن CreateUser ابتدا یک شیء از کلاس AppUser ساخته و پروپرتی‌های مدل را به پروپرتی‌های کلاس AppUser انتساب می‌دهیم. در مرحله‌ی بعد یک شیء از کلاس IdentityResult به نام result ایجاد کرده و نتیجه‌ی متد CreateAsync را درون آن قرار می‌دهیم. متد CreateAsync از طریق پروپرتی از نوع AppUserManager قابل دسترسی است و دو پارامتر را دریافت می‌کند. پارامتر اول یک شیء از کلاس AppUser و پارامتر دوم یک رشته‌ی حاوی Password می‌باشد و خروجی متد یک شیء از کلاس IdentityResult است. در مرحله‌ی بعد چک می‌کنیم اگر Result، مقدار Succeeded را داشته باشد (یعنی نتیجه موفقیت آمیز بود) آن‌وقت ... در غیر اینصورت خطاهای موجود را به ModelState اضافه نموده و به View می‌فرستیم.

```
@model Users.ViewModels.CreateModel

@Html.ValidationSummary(false)

@using (Html.BeginForm())
{
    <div class="form-group">
        <label>Name</label>
        @Html.TextBoxFor(x => x.UserName, new { @class = "form-control" })
    </div>
    <div class="form-group">
        <label>Email</label>
        @Html.TextBoxFor(x => x.Email, new { @class = "form-control" })
    </div>
```

```
    <div class="form-group">
        <label>Password</label>
        @Html.PasswordFor(x => x.Password, new { @class = "form-control" })
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
    @Html.ActionLink("Cancel", "Index", null, new { @class = "btn btn-default" })
}
```

اعتبار سنجی رمز

عمومی‌ترین و مهمترین نیازمندی برای هر برنامه‌ای، اجرای سیاست رمزگذاری می‌باشد؛ یعنی ایجاد یک سری محدودیتها برای ایجاد رمز است. مثلا رمز نمی‌تواند از 6 کاراکتر کمتر باشد و یا باید حاوی حروف بزرگ و کوچک باشد و ... . برای اجرای سیاست‌های رمزگذاری از کلاس PasswordValidator استفاده میشود. کلاس PasswordValidator برای اجرای سیاستهای رمزگذاری از پروپرتی‌های زیر استفاده می‌کند.

Name	Description
RequiredLength	Specifies the minimum length of a valid passwords.
RequireNonLetterOrDigit	When set to true, valid passwords must contain a character that is neither a letter nor a digit.
RequireDigit	When set to true, valid passwords must contain a digit.
RequireLowercase	When set to true, valid passwords must contain a lowercase character.
RequireUppercase	When set to true, valid passwords must contain a uppercase character.

```
var manager = new AppUserManager(new UserStore<AppUser>(db))
            {
                PasswordValidator = new PasswordValidator
                {
                    RequiredLength = 6,
                    RequireNonLetterOrDigit = false,
                    RequireDigit = false,
                    RequireLowercase = true,
                    RequireUppercase = true
                }
            };
```

فقط دوستان توجه داشته باشید که کد بالا را در متد Create از کلاس AppUserManager استفاده کنید.

اعتبار سنجی نام کاربری

برای اعبارسنجی نام کاربری از کلاس UserValidator به صورت زیر استفاده می‌کنیم:

```
manager.UserValidator = new UserValidator<AppUser>(manager)
            {
                AllowOnlyAlphanumericUserNames = true,
                RequireUniqueEmail = true
            };
```

کد بالا را نیز در متد Create از کلاس AppUserManager قرار می‌دهیم.

نظرات خوانندگان

نویسنده: محمد بیات
تاریخ: ۱۳۹۴/۰۴/۲۶ ۱۴:۵

با سلام و خسته نباشید، تشکر بابت مطالب خوبتون یک سوال برای من پیش اومده و اینکه اگر بخواهیم کلاسهای دیگه ای داشته باشیم مثلا یک کلاس برای جدول کالاها که اصلا ربطی به جدول Users ما نداشته باشه چطور باید به دیتابیس کانتکس وصلش کنیم یا به عبارتی چطور می‌تونیم توی Identity بهش بفهمونیم که همچین جدولی رو هم برای ما بسازه! و بتونیم از طریق صدا زدن Databasecontext اطلاعاتش رو بخونیم....
من تمامی مطالب شما رو انجام دادم تا الان درست کار کرده و میتونم لاگین و ریجستر رو برای کاربرهام داشته باشم ولی نمیدونم چطور باید با جداول دیگه ارتباط برقرار کنم و بسازمشون.
ممنون میشم توضیح بدید.
موفق و پیروز باشید.

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۴/۲۶ ۱۴:۳۱

در قسمت دوم Asp.Net Identity #2 این کلاس جدید رو باید به AppIdentityDbContext اضافه کنید (مثل روش معمول EF Code first توسط DbSetها). بعدش هم باید با مباحث migrations آشنایی داشته باشید.

نویسنده: احمد نواصری
تاریخ: ۱۳۹۴/۰۴/۳۱ ۰:۲۸

سلام. کلاس AppIdentityDbContext تعریف شده در Asp.net Identity #2 هیچ تفاوتی با dbContext تعریف شده توسط EF ندارد. شما میتونید DbSetهای خودتون رو توی کلاس AppIdentityDbContext تعریف کنید.

```
ublic class AppIdentityDbContext : IdentityDbContext<AppUser>
{
   public AppIdentityDbContext()
               : base("IdentityDb") { }

    // DbSet Definition
   public DbSet<Product> Products { get; set; }

    static AppIdentityDbContext()
    {
        Database.SetInitializer<AppIdentityDbContext>(new IdentityDbInit());
    }
    public static AppIdentityDbContext Create()
    {
        return new AppIdentityDbContext();
    }
 }
```

البته همونطور که دوستمون محسن خان فرمودن بایستی با مباحث Ef Code First آشنایی داشته باشید که در اینجا یک دوره کامل تدارک دیده شده.

نویسنده: روناک تابعی
تاریخ: ۱۳۹۴/۰۵/۰۹ ۱۵:۳۵

با سلام و تشکر. آیا سیستم Identity هیچ تنظیماتی در web.config نداره؟
نیازی به تنظیمات قسمت authentication نیست دیگه؟

نویسنده: روناک تابعی

تاریخ: ۱۳۹۴/۰۵/۰۹ ۱۵:۳۹

آیا استفاده از یک DbContext برای Identity و دیگر مدلهای برنامه خوبه یا اینکه برای دیگر مدلهای برنامه از DbContext جدا استفاده کنیم؟

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۵/۰۹ ۱۵:۵۹

می‌تونید استفاده کنید. فقط باید به نکات استفاده از چندین Context در EF 6 Code first دقت داشته باشید.

نویسنده: احمد نواصری
تاریخ: ۱۳۹۴/۰۵/۱۰ ۲۰:۴۱

سلام. مراجعه کنید به Asp.Net Identity #2 . فقط کافیه که یک Connection String تعریف کنید واسه ارتباط به پایگاه داده و یک کلید که معرف کلاس شروع Owin هست . نیاز به تنظیمات اضافه‌تری نداره.