// enable-migrations -ProjectName Domain -StartUpProjectName WebUI -Verbose

// Add-Migration "DataMigration" -ProjectName Domain -StartUpProjectName WebUI -Verbose

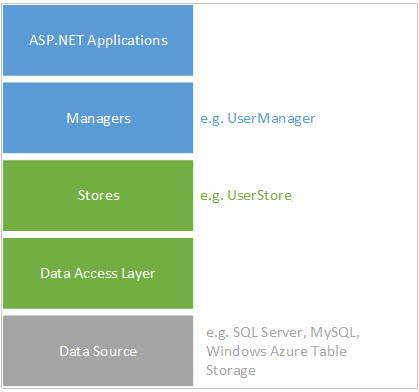
// update-database(any command) -ProjectName Domain -StartUpProjectName WebUI -Verbose

Предположим, мы решаем следующую задачу. У нас есть готовая БД HumanResource, и мы хотим прикрутить к ней asp.net identity. Возникает несколько проблем – PK во всех таблицах asp.net identity является строкой…, а нам нужен тип int.

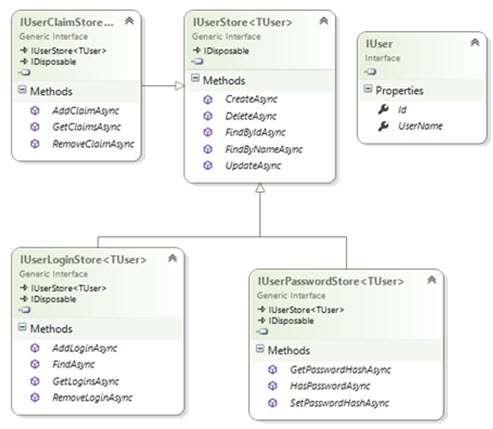
Решение будет следующее….

Чтобы разобраться как все будет (в случае с ключем целого типа – создадим пустой проект) и посмотрим, что из этого получится.

### схематическое представление



<http://ggs.grapecity.com/blog/web/securing-web-applications-using-aspnet-identity>



Предположим нам остается добавить в проект аутентификацию…

1. Install NuGet packages - ***Microsoft.Owin.Host.SystemWeb, Microsoft.Owin.Security.Cookies, Microsoft.AspNet.Identity.Owin, Microsoft.AspNet.Identity.EntityFramework, Microsoft.Owin.Security.OAuth***
2. ***Entity framework***
3. Добавляем строку подключения

<connectionStrings>

<add name="HrContext" connectionString="data source=localhost;initial catalog=HumanResourcesUser; integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" providerName="System.Data.SqlClient" />

</connectionStrings>

1. В папку Models добавляем класс пользователя

public class AppUser : IdentityUser

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

По умолчанию модель пользователя наследует класс

public class IdentityUser : IdentityUser<***string***, IdentityUserLogin, IdentityUserRole, IdentityUserClaim>, IUser, IUser<***string***>

{

public IdentityUser();

public IdentityUser(string userName);

}

Мы хотим вместо типа string – использовать тип int

public class AppUser : IdentityUser<int, CustomUserLogin, CustomUserRole,

CustomUserClaim>

{

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime DateBirthday { get; set; }

}

public class CustomUserRole : IdentityUserRole<int> { }

public class CustomUserClaim : IdentityUserClaim<int> { }

public class CustomUserLogin : IdentityUserLogin<int> { }

public class AppRole : IdentityRole<int, CustomUserRole>

{

public CustomRole() { }

public CustomRole(string name) { Name = name; }

}

public class CustomUserStore : UserStore<AppUser, AppRole, int,

CustomUserLogin, CustomUserRole, CustomUserClaim>

{

public CustomUserStore(ApplicationDbContext context)

: base(context)

{

}

}

public class CustomRoleStore : RoleStore< AppRole, int, CustomUserRole>

{

public CustomRoleStore(ApplicationDbContext context)

: base(context)

{

}

}

1. Нам также необходимо добавить файл Startup в корень нашего приложения

public class Startup

{

public void Configuration(IAppBuilder app)

{

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = "ApplicationCookie",

LoginPath = new PathString("/auth/login")

});

}

}

1. Добавим а наш проект HomeController, с атрибутом [Authorize] и создадим View Index. Если мы запустим наше приложение, то будем перенаправлены по адресу <http://localhost:9355/auth/login?ReturnUrl=%2FHome%2FIndex>

Т.к. мы не создали контроллера Auth c методом действия Login – то получаем ошибку!

1. В папку App\_Start добавляем

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

filters.Add(new AuthorizeAttribute());

}

}

В global.asax.cs регистрируем

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RouteConfig.RegisterRoutes(RouteTable.Routes);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

}

1. *В CustomUserStore и CustomRoleStore используется* ***ApplicationDbContext***

*Добавим класс* ***ApplicationDbContext***

public class ApplicationDbContext : IdentityDbContext<AppUser, AppRole, int, CustomUserLogin, CustomUserRole, CustomUserClaim>

{

public ApplicationDbContext()

: base("HrContext")

{ }

//Для создание при старте приложения

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

}

1. Добавим еще ViewModel для AppUser и AppRole

public class LogInModel

{

[Required]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[HiddenInput]

public string ReturnUrl { get; set; }

}

public class RegisterModel

{

[Required]

[DataType(DataType.EmailAddress)]

public string Email { 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]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

//[Required]

//public DateTime DateBirthday { get; set; }

}

Поставьте аттрибуты!

public class CreateRoleModel

{

public string Name { get; set; }

}

public class EditRoleModel

{

public string Id { get; set; }

public string Name { get; set; }

//public string Description { get; set; }

}

Теперь добавляем пустой AuthController контроллер. Для работы с пользователями нам нужен менеджер пользователей, а для работы с ролями менеджер ролей. Для менеджера пользователей создадим фабрику в классе Startup

public class Startup

{

**public static Func<UserManager<AppUser, int>> UserManagerFactory { get; private set; }**

public void Configuration(IAppBuilder app)

{

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = "ApplicationCookie",

LoginPath = new PathString("/auth/login")

});

**UserManagerFactory = () =>**

**{**

**var usermanager = new UserManager<AppUser, int>(**

**new CustomUserStore(new ApplicationDbContext()));**

**// добавляем при необходимости валидацию!!!**

**usermanager.UserValidator = new UserValidator<AppUser, int>(usermanager)**

**{**

**AllowOnlyAlphanumericUserNames = false**

**};**

**return usermanager;**

**};**

}

}

В контроллере AuthController добавим несколько закрытых методов

private string GetRedirectUrl(string returnUrl)

{

if (string.IsNullOrEmpty(returnUrl) || !Url.IsLocalUrl(returnUrl))

{

return Url.Action("index", "home");

}

return returnUrl;

}

private IAuthenticationManager GetAuthenticationManager()

{

var ctx = Request.GetOwinContext();

return ctx.Authentication;

}

## Роли

Сущность ролей в ASP.NET Identity представлена классом IdentityRole, который реализует интерфейс IRole. И мы можем продолжать использовать IdentityRole, но при необходимости также можем переопределить, добавив в него новые свойства. Добавим в наш функциональность управления ролями.

public class AppRole : IdentityRole<int, CustomUserRole>

{

public AppRole() { }

public AppRole(string name) { Name = name; }

}

Для управления ролями используется менеджер ролей RoleManager. Добавим в папку Models новый класс ApplicationRoleManager

public class ApplicationRoleManager : RoleManager<ApplicationRole,int>

{

//public ApplicationRoleManager(RoleStore<ApplicationRole> store) : base(store) !!!!!!!!!!!!!!!!!!!!!!!!!!

public ApplicationRoleManager(RoleStore<ApplicationRole, int, CustomUserRole> store)

: base(store)

{ }

public static ApplicationRoleManager Create(IdentityFactoryOptions<ApplicationRoleManager> options,

IOwinContext context)

{

return new ApplicationRoleManager(new

RoleStore<ApplicationRole, int, CustomUserRole>(context.Get<ApplicationDbContext>()));

}

}

Менеджер ролей будет использовать тот же контекст данных, что и менеджер пользователей.

## Миграции

Добавим к нашим ролям свойство

public class AppRole : IdentityRole<int, CustomUserRole>

{

public AppRole() { }

**public string Description { get; set; }**

public AppRole(string name) { Name = name; }

}

в окне Package Manager Console введем команду: enable-migrations

Командой Add-Migration "DataMigration" – создаем миграцию. Будет создан класс

public partial class DataMigration : DbMigration

{

public override void Up()

{

AddColumn("dbo.AspNetRoles", "Description", c => c.String());

}

public override void Down()

{

DropColumn("dbo.AspNetRoles", "Description");

}

}

Чтобы выполнить миграцию, применим этот класс, набрав в той же консоли команду:

PM > Update-Database

// update-database -ProjectName Domain -StartUpProjectName WebUI -Verbose