|  |  |
| --- | --- |
|  | Белявский Иван  (Фамилия Имя Отчество) |
| 3/6/2016 |  |
| Задание на день | исправление ошибок в авторизации, добавление ролей, изучение identity |

Результаты работы: Пока буксую, в одном примере он создает классы наследуемые непонятно от чего ,в другом просит создать *IdentityModels.cs,* хотя у нас уже есть готовый *AccountModels.cs ,* а если использовать модель Аккаунт то опять просят создать непонятный файл типа этого:

Так как WSAT использовать не получится, есть 2 способа создания пользователей и ролей. Если используется модель EntityFramework Code First, удобно будет добавить Micragion с созданием пользователей по умолчанию:

public partial class AddDefaultUser : DbMigration

{

public override void Up()

{

if (!WebSecurity.Initialized)

{

WebSecurity.InitializeDatabaseConnection("DefaultConnection", "Users", "Id", "UserName",

autoCreateTables:

true);

}

var roles = (SimpleRoleProvider)Roles.Provider;

var membership = (SimpleMembershipProvider)Membership.Provider;

if (!roles.RoleExists("Admin"))

{

roles.CreateRole("Admin");

}

if (membership.GetUser("Admin", false) == null)

{

membership.CreateUserAndAccount("Admin", "SuperAdminPassword");

}

if (!roles.GetRolesForUser("Admin").Contains("Admin"))

{

roles.AddUsersToRoles(new[] { "Admin" }, new[] { "Admin" });

}

}

public override void Down()

{

throw new NotImplementedException();

}

}

Листинг первого примера

Начнем с создания класса идентификации пользователя, который будет доступен через сведения о безопасности текущего Http запроса HttpContext.User.Identity.

[Serializable]

//TAccount - Тип аккаунта в бизнес логике.

//TRole - Тип роли.

public abstract class AbstractIdentity<TAccount, TRole>: MarshalByRefObject, IIdentity

{

protected AbstractIdentity()

{

Id = long.MinValue;

}

private bool \_isInitialized = false;

public long Id { get; set; }

public string Name { get; set; }

public string AuthenticationType

{

get

{

return String.Format("CustomizeAuthentication\_{0}", typeof(TAccount).Name);

}

}

public string[] Role { get; set; }

public TRole[] Roles { get; set; }

public bool IsAuthenticated

{

get { return Id != long.MinValue; }

}

public bool CheckRole(TRole role)

{

return Role.All(r => r.Equals(role.ToString()));

}

public void SetAccount(TAccount account)

{

Id = GetId(account);

Name = GetName(account);

Roles = GetRole(account);

Role = Roles.Select(c=>c.ToString()).ToArray();

InitializeMoreFields(account);

\_isInitialized = true;

}

protected virtual void InitializeMoreFields(TAccount account) { }

protected abstract long GetId(TAccount account);

protected abstract string GetName(TAccount account);

protected abstract TRole[] GetRole(TAccount account);

public string Serialize()

{

if (!\_isInitialized)

throw new AccountNotSetException();

using (var stream = new MemoryStream())

{

var formatter = new XmlSerializer(GetType());

formatter.Serialize(stream, this);

return Encoding.UTF8.GetString(stream.ToArray());

}

}

public static TIdenty Deserialize<TIdenty>(string value)

where TIdenty : AbstractIdentity<TAccount, TRole>

{

using (var stream = new MemoryStream(Encoding.UTF8.GetBytes(value)))

{

var formatter = new XmlSerializer(typeof(TIdenty));

return (TIdenty)formatter.Deserialize(stream);

}

}

}