

Custom Role Provider. Часть 1

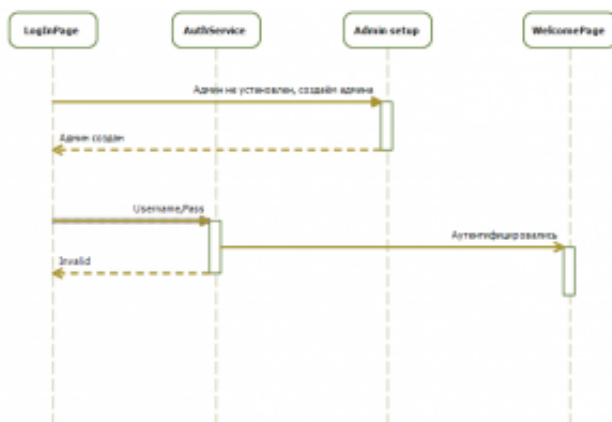
Итак. В прошлых статьях был описан процесс создания membership провайдера. С помощью него появилась возможность производить базовые операции, такие как регистрация пользователей, активация через почту, аутентификация. Такой набор функций подойдет разве что для чата или форума, и то без модерации. Для того чтобы система стала более полноценной, необходимо добавить в неё роли и администраторскую панель.

Следующие две статьи будут посвящены созданию ролей и администраторской панели для взаимодействия с аккаунтами пользователей.

Логика работы.

Так как у нас будет разделение ролей, нужно сделать админа, который будет управлять аккаунтами пользователей. Как и в большинстве cms, мы сделаем это через первую настройку.

Для полноты картины, нарисовал диаграмму последовательности. Извиняюсь, если сделал её неправильно, но думаю, смысл должен быть понятен.



Когда посетитель попадает на страницу логина, идёт проверка, настроена ли учётная запись админа. Если учётка не настроена, делаем редирект на страницу настройки админа. После того, как админ настроен —редирект обратно на логин.

Реализацию работы с ролями мы начнём сверху, то есть с представлений и контроллеров. Затем добавим свой провайдер ролей и внесем изменения в UserRepository.

Открываем **AccountModels.cs** и добавим интерфейс для провайдера ролей и назовём его **IRoleService**:

```
1 public interface IRoleService
4     void AddUsersToRoles( string [] usernames, string []
rolenames);
5     void RemoveUsersFromRoles( string [] usernames,
string [] rolenames);
6     void CreateRole( string roleName);
```

Через этот интерфейс будут работать методы для настройки учётной записи администратора.

Далее реализуем этот интерфейс:

```
01 public class AccountRoleService : IRoleService
```

```

03     private readonly RoleProvider _provider;
05     public AccountRoleService(): this ( null )
09     public AccountRoleService(RoleProvider provider)
11         _provider = provider;
14     public bool AdminExists()
16         var users = _provider.GetUsersInRole( "Admin" );
18         if (users.Count()==0)
19             return false ;
24     public void AddUsersToRoles( string [] usernames,
string [] rolenames)
26         _provider.AddUsersToRoles(usernames, rolenames);
29     public void RemoveUsersFromRoles( string [] usernames,
string [] rolenames)
31         _provider.RemoveUsersFromRoles(usernames, rolenames);
34     public void CreateRole( string roleName)
36         _provider.CreateRole(roleName);

```

Теперь необходимо добавить ссылку на класс в **AccountController.cs**.

Добавим следующее поле:

```

1     public IRoleService RoleService { get ; set ; }

```

А в метод инициализации контроллера добавляем следующий код:

```

1     if (RoleService == null ) { RoleService = new
AccountRoleService(); }

```

Как теперь видно, вся эта процедура была проделана по аналогии с **MembershipService** из стандартного шаблона.

Теперь нужно добавить методы для создания учётки админа. Берем код из метода **Register** и слегка изменяем его:

```

05     public ActionResult AdminSetup()
07         if (RoleService.AdminExists())
08             return RedirectToAction( "LogOn" );
14     public ActionResult AdminSetup(RegisterModel model)
16         if (ModelState.IsValid)
17             MembershipCreateStatus createStatus =
19 MembershipService.CreateUser(model.UserName, model.Password,
model.Email, true );
21         if (createStatus == MembershipCreateStatus.Success)
23             RoleService.CreateRole( "Admin" );
24             RoleService.AddUsersToRoles( new string []

```

```

    {model.Username}, new string[] { "Admin" });
25 FormsService.SignIn(model.UserName, true);
26 return RedirectToAction("Admin", "Home");
31 ModelState.AddModelError("", AccountValidation.ErrorCodeToString(createStatus));
34 return View(model);

```

Данный метод будет создавать роль админа и добавлять регистрируемого пользователя в эту роль.

Теперь нужно добавить код в Get-метод **LogOn**, для перенаправления на **AdminSetup**, если админ не создан:

```

1 if (!RoleService.AdminExists())
3     TempData["Message"] = "Необходимо настроить учётную запись администратора перед началом работы приложения.";
4     return RedirectToAction("AdminSetup");

```

Для того, чтобы отключить настройку админа, когда он уже есть, было добавлено условие **AdminExists**. Этот метод будет искать пользователей в базе данных с ролью администратора. Если такой найдется, будет редирект на страницу входа для предотвращения создания новых админов.

Теперь необходимо добавить еще кое-какие изменения в стандартный шаблон **AccountModels** для того, чтобы админ сразу входил в систему после настройки, минуя активацию через e-mail.

В стандартном Membership провайдере в методе **CreateUser** есть параметр **IsApproved**. Не знаю, как он работает в стандартной реализации, но мы будем использовать его для активации учётных записей. Возможно, там он служит для этих же целей.

Открываем **AccountModels.cs**. Находим интерфейс **IMembershipService** и меняем **CreateUser**:

```

1 MembershipCreateStatus CreateUser(string userName, string password, string email, bool isApproved);

```

То же самое проделываем в **AccountMembershipService**. Теперь реализованный метод должен выглядеть так:

```

01 public MembershipCreateStatus CreateUser(string userName, string password, string email, bool isApproved)
03     if (String.IsNullOrEmpty(userName)) throw new ArgumentException("Value cannot be null or empty.", "userName");
04     if (String.IsNullOrEmpty(password)) throw new ArgumentException("Value cannot be null or empty.", "password");
05     if (String.IsNullOrEmpty(email)) throw new

```

```

05 ArgumentException( "Value cannot be null or empty." ,
    "email" );
07 MembershipCreateStatus status;
08 null , isApproved, null , out status);

```

Так как мы изменили метод **CreateUser**, теперь необходимо найти его вызовы в **AccountController** и добавить параметр **IsApproved**.

В post-методе **AdminSetup** меняем createStatus на следующий:

```

    MembershipCreateStatus createStatus =
1 MembershipService.CreateUser(model.UserName,
    model.Password,model.Email, true );

```

А в post-методе **Register** меняем на этот:

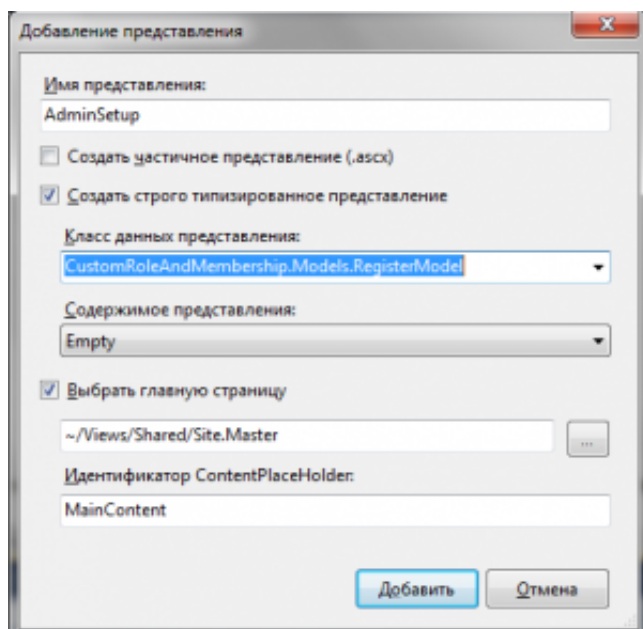
```

    MembershipCreateStatus createStatus =
1 MembershipService.CreateUser(model.UserName, model.Password,
    model.Email, false );

```

При настройке админа, его запись будет активирована и он будет автоматически залогинен.

Теперь добавим представление **AdminSetup**. В качестве модели будем использовать стандартную модель RegisterModel:



В само представление добавим слегка изменённый код из Register.aspx

```

01 < asp:Content ID = "adminTitle"
    ContentPlaceHolderID = "TitleContent" runat = "server" >
02     Настройка администратора
05 < asp:Content ID = "adminContent"
    ContentPlaceHolderID = "MainContent" runat = "server" >
06     < h2 >Настройка учётной записи администратора</ h2 >
07     <%: TempData["Message"] %>

```

```

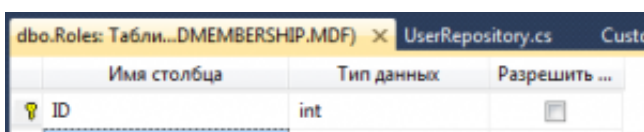
08     <% using (Html.BeginForm()) { %>
09         <%= Html.ValidationSummary(true) %>
12         < legend >Информация</ legend >
14         < div class = "editor-label" >
15             <%= Html.LabelFor(m => m.UserName) %>
17         < div class = "editor-field" >
18             <%= Html.TextBoxFor(m => m.UserName) %>
19             <%= Html.ValidationMessageFor(m =>
22         < div class = "editor-label" >
23             <%= Html.LabelFor(m => m.Email) %>
25         < div class = "editor-field" >
26             <%= Html.TextBoxFor(m => m.Email) %>
27             <%= Html.ValidationMessageFor(m => m.Email)
30         < div class = "editor-label" >
31             <%= Html.LabelFor(m => m.Password) %>
33         < div class = "editor-field" >
34             <%= Html.PasswordFor(m => m.Password) %>
35             <%= Html.ValidationMessageFor(m =>
38         < div class = "editor-label" >
39             <%= Html.LabelFor(m => m.ConfirmPassword) %>
41         < div class = "editor-field" >
42             <%= Html.PasswordFor(m => m.ConfirmPassword)
43             <%= Html.ValidationMessageFor(m =>
47         < input type = "submit"
         value = "Сохранить" />
49     </ fieldset >



```

Ну а теперь настала очередь провайдера ролей. Перед тем как его делать, необходимо создать две таблицы в базе данных и обновить схему EDM.

Так как у пользователей может быть несколько ролей, а в одной роли может быть множество пользователей, необходимо реализовать связь многие-ко-многим, поэтому и нужно добавить две таблицы. Одна из них будет связывать пользователей с ролями.

Создаём таблицу **Roles** со следующими полями:



Имя столбца	Тип данных	Разрешить ...
ID	int	
Name	nvarchar(50)	

Не забываем указывать первичный ключ и идентифицирующую спецификацию для ID!

Теперь создадим таблицу для связи ролей и

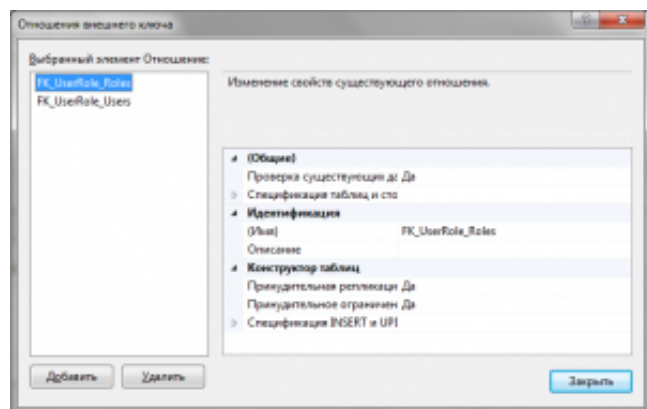
Name	nvarchar(10)	<input type="checkbox"/>
		<input type="checkbox"/>

теперь создадим таблицу для связи ролей и пользователей и назовём её **UserRole**:

Имя столбца	Тип данных	Разрешить ...
RoleID	int	<input type="checkbox"/>
UserID	int	<input type="checkbox"/>
		<input type="checkbox"/>

Перед тем, как

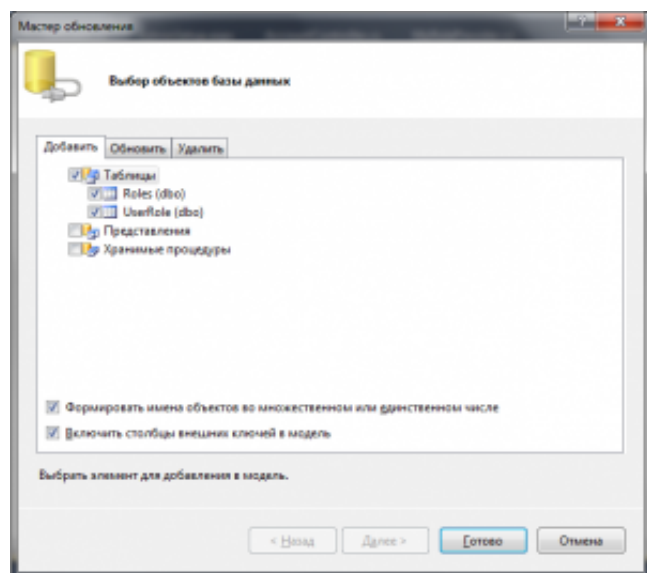
задавать первичный ключ, необходимо установить отношения с таблицами Users и Roles, и сохранить таблицу.



Теперь нужно задать первичный ключ. **Важно**, чтобы первичным ключом являлись оба столбца. Чтобы это сделать, нужно выбрать их оба и в контекстном меню выбрать “задать первичный ключ”.

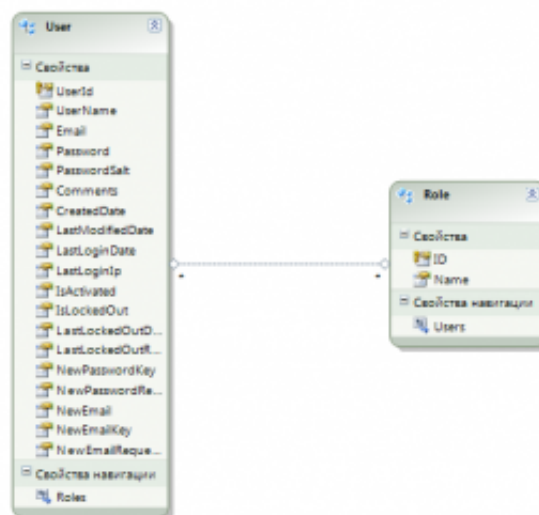
После того, как таблицы были созданы, необходимо обновить модель EDM. Открываем CustomRoleAndMembership.edmx, щелкаем правой

кнопкой на любом месте схемы и выбираем “Обновить модель из базы данных”. Выбираем созданные таблицы и включаем опцию формирования имён объектов:



EF распознал замудрённую связь и замаппил её на классы! В итоге схема вышла вот такой:

СВЯЗЬ



многие-ко-многим

Теперь, когда в базе есть таблицы для ролей и создан класс Role, можно заняться провайдером.

Создадим класс

Создаём **MyRoleProvider.cs** со следующим содержимым:

```
2  using System.Collections.Generic;
5  using System.Web.Security;
7  public class MyRoleProvider : RoleProvider
```

Имплементируем методы стандартного провайдера и добавим работу с конфигом из **MyMembershipProvider.cs**. В общем, должно выйти вот так:

```
02  using System.Collections.Generic;
05  using System.Web.Security;
06  using System.Collections.Specialized;
08  public class MyRoleProvider : RoleProvider
11      private string _ApplicationName;
12      public override void Initialize( string name,
13      NameValueCollection config)
14      {
15          if (config == null )
16              throw new ArgumentNullException( "config" );
17          if (name == null || name.Length == 0)
18              name = "CustomRoleProvider" ;
19          if (String.IsNullOrEmpty(config[ "description" ]))
20              config.Remove( "description" );
21              config.Add( "description" , "Custom Role
22      Provider" );
23      base .Initialize(name, config);
24      _ApplicationName =
25      GetConfigValue(config[ "applicationName" ],
26      System.Web.Hosting.HostingEnvironment.Applic
27      ationName);
28      public override string ApplicationName
29      {
30          get { return _ApplicationName; }
31          set { _ApplicationName = value; }
32      }
33      private string GetConfigValue( string configValue,
34      string defaultValue)
35      {
36          if ( string.IsNullOrEmpty(configValue))
37              return defaultValue;
38          return configValue;
39      }
40      public override void AddUsersToRoles( string []
41      usernames, string [] roleNames)
42      {
43          throw new NotImplementedException();
44      }
45      public override void CreateRole( string roleName)
46      {
47          throw new NotImplementedException();
48      }
49      public override bool DeleteRole( string roleName,
50      bool throwOnPopulatedRole)
```

```

60         throw new NotImplementedException();
63     public override string [] FindUsersInRole( string
    roleName, string usernameToMatch)
65         throw new NotImplementedException();
68     public override string [] GetAllRoles()
70         throw new NotImplementedException();
73     public override string [] GetRolesForUser( string
    username)
75         throw new NotImplementedException();
78     public override string [] GetUsersInRole( string
    roleName)
80         throw new NotImplementedException();
83     public override bool IsUserInRole( string username,
    string roleName)
85         throw new NotImplementedException();
88     public override void RemoveUsersFromRoles( string []
    usernames, string [] roleNames)
90         throw new NotImplementedException();
93     public override bool RoleExists( string roleName)
95         throw new NotImplementedException();

```

Перемещаем его в App_Data и переходим к web.config. Открываем web.config, ищем раздел **roleManager** и меняем его код на следующий:

```

1  <roleManager enabled= "true"
    defaultProvider= "CustomRoleProvider" >
4      <add name= "CustomRoleProvider" type= "MyRoleProvider"
        connectionStringName= "CustomRoleAndMembershipDB"
        applicationName= "/" />

```

Теперь нужно вернуться к **AccountModels**. Ищем конструктор класса **AccountRoleService**, который принимает **RoleProvider**, вносим изменения в код:

```

1  public AccountRoleService(RoleProvider provider)
3      _provider = provider ?? new MyRoleProvider();

```

Теперь этот сервис будет работать с нашим провайдером. Попробуем запустить приложение и залогиниться. Появляется ошибка — не реализован метод **GetUsersInRole**. Отлично! Займемся реализацией методов провайдера.

Наш провайдер работает с **UserRepository**, поэтому мы будем параллельно добавлять код и в него. Но чтобы продолжить, необходимо исправить кое-какие недостатки в **UserRepository**. Этот класс содержит методы, которые работают в разных контекстах **CustomRoleAndMembershipDB**. В дальнейшем нужно, чтобы все методы работали в едином контексте экземпляра класса **UserRepository**.

Для того чтобы не вставлять весь код сюда, скачайте [UserRepository.cs](#) и замените его у себя в проекте.

Вернемся обратно к провайдеру. Нужно добавить ссылку на Models:

```
1 using CustomRoleAndMembership.Models;
```

И поле UserRepository:

```
1 private UserRepository userRepo = new UserRepository();
```

Теперь займемся реализацией методов `GetUsersInRole` и `GetRolesForUser`. Чтобы их сделать, нужно добавить несколько методов в **UserRepository**:

```
01 public User GetDBUser( string username)
02 {
03     return db.Users.SingleOrDefault(x=>x.UserName==username);
04 }
05
06 public Role GetRole( string name)
07 {
08     return db.Roles.SingleOrDefault(x => x.Name == name);
09 }
10 public List<User> GetAllUsers()
11 {
12     return db.Users.ToList();
13 }
```

Теперь сами методы **GetUsersInRole** и **GetRolesForUser** в **MyRoleProvider.cs**:

```
01 public override string [] GetUsersInRole( string roleName)
02 {
03     var role = userRepo.GetRole(roleName);
04     var usernames = userRepo.GetAllUsers()
05         .Where(x => x.Roles.Contains(role))
06         .Select(x => x.UserName);
07     return usernames.ToArray();
08 }
09
10 public override string [] GetRolesForUser( string username)
11 {
12     var user = userRepo.GetDBUser(username);
13     return user.Roles
14         .Select(x => x.Name).ToArray();
15 }
```

Опять пробуем залогиниться. Теперь идёт перенаправление на AdminSetup. Это значит, что метод `GetUsersInRole` работает корректно.

Теперь реализуем добавление ролей для пользователей. За это отвечает метод `AddUsersToRoles`. Сначала добавим метод **AddUsersToRoles** в **UserRepository**:

```
01 public void AddUsersToRoles( string [] usernames,
02     string [] rolenames)
03 {
04     foreach (var username in usernames)
05     {
06         var user = GetDBUser(username);
07         if (user != null )
08         {
09             // ...
10         }
11     }
12 }
```

```

08         foreach (var rolename in rolenames)
09         {
10             var role = GetRole(rolename);
11             if (role != null)
12                 if (!user.Roles.Contains(role))
13                     user.Roles.Add(role);
14         }
15     }
16 }

```

Реализуем метод **AddUsersToRoles** в **MyRoleProvider**:

```

1  public override void AddUsersToRoles( string [] usernames,
2  string [] roleNames)
3  {
4      userRepo.AddUsersToRoles(usernames, roleNames);
5  }

```

Теперь нужно изменить метод **CreateUser**, чтобы добавить возможность активировать аккаунты сразу:

```

01  public MembershipUser CreateUser( string username, string
02  password, string email, bool isApproved)
03  {
04      User user = new User();
05      user.UserName = username;
06      user.Email = email;
07      user.PasswordSalt = CreateSalt();
08      user.Password = CreatePasswordHash(password,
09      user.PasswordSalt);
10      user.CreatedDate = DateTime.Now;
11      if (!isApproved)
12      {
13          user.IsActivated = false ;
14          user.NewEmailKey = GenerateKey();
15          user.IsActivated = true ;
16      }
17      user.IsLockedOut = false ;
18      user.LastLockedOutDate = DateTime.Now;
19      user.LastLoginDate = DateTime.Now;
20      db.AddToUsers(user);
21      db.SaveChanges();
22      if (!isApproved)
23      {
24          user.NewEmailKey = user.UserName + "/" +
25          user.NewEmailKey;
26          var message = new MailMessage( "EMAIL" , user.Email)
27          {
28              Subject = "Activate your account" ,
29              Body = ActivationLink
30          };
31          var client = new SmtpClient( "SERVER" );
32          client.Credentials = new
33          System.Net.NetworkCredential( "LOGIN" , "PASSWORD" );
34          client.EnableSsl = true ;
35          client.Send(message);
36      }
37  }

```

```
43 |         return GetUser(username);
```

Напомню, что тут нужно поменять значения PORT, EMAIL, SERVER, PASSWORD и LOGIN на собственные.

Далее, находим ссылку в **MyMembershipProvider.cs** где создаётся пользователь и исправляем вызов метода **CreateUser**:

```
1 | _user.CreateUser(username, password, email, isApproved);
```

Теперь настало время реализовать метод **CreateRole** в **MyRoleProvider**. Сначала добавим **CreateRole** в **UserRepository**:

```
1 | public void CreateRole( string roleName)
3 |     if (GetRole(roleName) == null )
4 |         db.AddToRoles( new Role { Name = roleName } );
```

Затем в сам провайдер:

```
1 | public override void CreateRole( string roleName)
3 |     userRepo.CreateRole(roleName);
```

Добавим простую страницу администратора. Открываем **HomeController** и добавляем метод **Admin** с атрибутом авторизации:

```
1 | [Authorize(Roles= "Admin" )]
2 | public ActionResult Admin()
```

И добавим простое представление без строгой типизации.

Если в базе данных остались пользователи, лучше их удалить. Запускаем наше приложение и пробуем залогиниться.

Создаём админа, когда будем перенаправлены на страницу настройки админа. Всё, админ создан и залогинен! Ура! В следующей части будет сделана небольшая администраторская панель для управления аккаунтами.

Скачать проект можно тут [CustomRoleAndMembership.zip](#)



shiftnotes.wordpress.com



<http://shiftnotes.wordpress.com/2011/04/22/custom-role-provider-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1/>



<http://goo.gl/q7voa>

