Реализация собственного Membership Provider (часть 2)

Вторая часть руководства по созданию собственного Membership Provider. Начало см. Реализация собственного Membership Provider (часть 1)

Вкратце рассмотрим, что мы уже сделали до этого. Мы создали готовый проект ASP.NET MVC 3 Web Application с готовым шаблоном, тем самым ускорив разработку.

Мы создали собственный Membership Provider и показали как переопределять его методы (реализовав свою логику метода **ValidateUser**).

В этой части руководства мы создадим БД, класс **UserRepository**, обеспечивающий логику работу с данными из БД (будем стараться сохранять провайдер настолько простым, насколько это возможно), и определим некоторые из основных методов Membership Provider'а.

Перво-наперво, хочу сказать, что всё что я делаю в этом руководстве, я делаю вместе с вами впервые, так что я открыт для советов как это может быть улучшено (и код и способ, с помощью которого это все работает).

Прежде чем мы начнем, я хотел бы быстро (извиняюсь за все эти задержки) рассказать, каким я вижу это руководство в следующих частях.

Мы реализуем все стандартные функции Membership Provider'а. И постараемся сделать их как можно более простыми и понятными. Даже если это означает всего лишь возвращение жестко закодированного значения, например формат пароля (нам не будет нужна возможность выбора между открытым/шифрованным паролем).

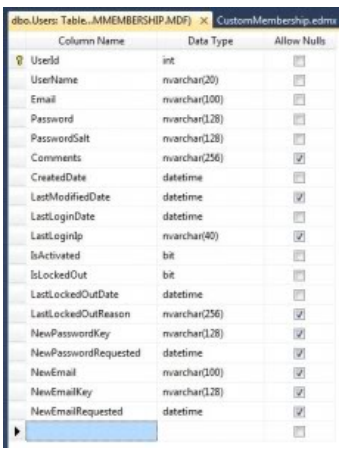
Мы также реализуем систему проверки email-адреса пользователя, требующую перехода по сгенерированной ссылке отправленной ему на почту при регистрации, сбросе пароля или изменении email адреса.

После всего этого мы реализуем Role Provider и создадим административный раздел (back-end) для нашего mvc приложения.

Возвращаясь к этой части руководства, первое что нам необходимо – таблица в БД для записи информации о пользователе. Таблица, которую я предлагаю, в основном, является сочетанием таблицы **aspnet_membership** из ASPNETDB бд и таблицы Users из **Tank_Auth -** библиотеки аутентификации из CodeInteger PHP Framework (перейдите по этой ссылке на Stack Overflow чтобы узнать больше о Tank_Auth, там все очень доступно написано - What Code Igniter authentication library is best? – Stack Overflow).

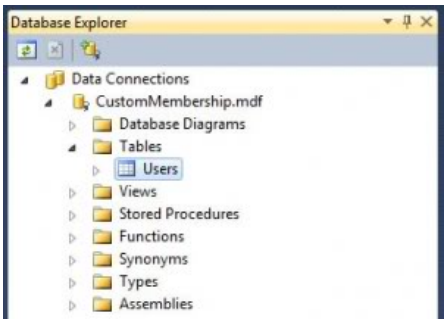
Давайте уже начнем, добавив базу данных в наш проект. Щелкаем правой кнопкой на папке **App_Data** и выбираем Add->New Item, здесь выбираем SQL Server Database. Назовем ее **CustomMembership.mdf** и нажимаем Add.

В Database Explorer, щелкаем п.к.м на узле Tables и выбираем Add New Table. Создаем вот такую вот таблицу:



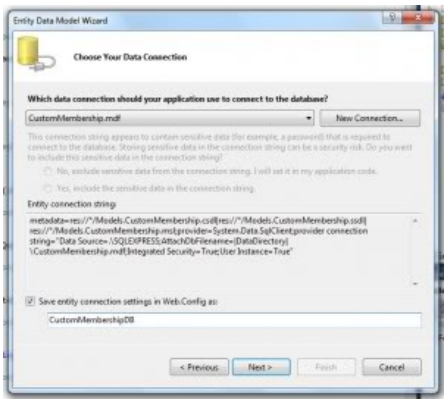
Column Name	Data Type	Allow Nulls
UserId	int	
UserName	nvarchar(20)	
Email	nvarchar(100)	
Password	nvarchar(128)	
PasswordSalt	nvarchar(128)	
Comments	nvarchar(256)	✓
CreatedDate	datetime	
LastModifiedDate	datetime	✓
LastLoginDate	datetime	
LastLoginIp	nvarchar(40)	✓
IsActivated	bit	
IsLockedOut	bit	
LastLockedOutDate	datetime	
LastLockedOutReason	nvarchar(256)	✓
NewPasswordKey	nvarchar(128)	✓
NewPasswordRequested	datetime	✓
NewEmail	nvarchar(100)	✓
NewEmailKey	nvarchar(128)	✓
NewEmailRequested	datetime	✓

Для UserId устанавливаем Primary Key и Identity Column и сохраняем таблицу под именем **Users**.

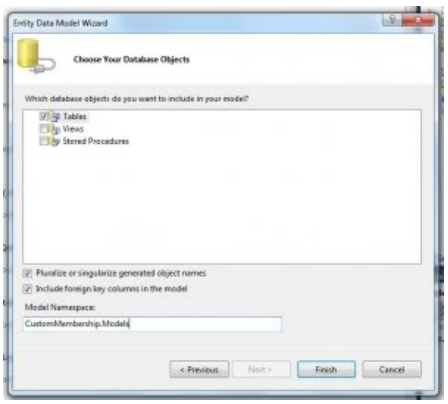


Возвращаеся обратно в Solution Explorer, жмем правой кнопкой мыши на папке Models и выбираем Add->New Item. Далее выбираем шаблон ADO.NET Entity Data Model, называем его **CustomMembership.edmx** и добавляем его к нашему проекту.

Выбираем «Создать из базы данных» и нажимаем «Далее». Изменим название соединения на **CustomMembershipDB** и идем дальше.



Теперь поставим галочку напротив «Таблицы» и изменим пространство имен на **CustomMembership.Models**. На этом завершим мастер.



Теперь нам нужно открыть файл **web.config** и удалить из него все соединения к базе ASPNETDB. Теперь, когда у нас уже есть строка соединения к собственной бд, которую, кстати, Membership Provider, не собирается использовать напрямую где-либо, мы спокойно удаляем соединения к ASPNETDB.

Изменим значение параметра **connectionStringName** в узлах Membership, Role и Profile, в файле **web.config**. Должно получиться что-то похожее на изображенное ниже:

Мы сразу можем запустить наше приложение чтобы убедиться, что мы не сломали что-то еще.

Давайте посмотрим, что случиться если мы нажмем на ссылку регистрации на странице авторизации.

Если бы мы хотели, мы могли бы просто жестко задать значение свойства **MinRequiredPasswordLength** в коде нашего Membership Provider'а.

Что происходить при инициализации объекта провайдера – правильно, вызывается метод **Initialize()**. Он считывает данные из **web.config** и устанавливаем свойства провайдера. Поэтому, давайте реализуем этот метод инициализации в классе **MyMembershipProvider**.

Сперва добавим ссылку на сборку **System.Collections.Specialized**:

```
1. using System.Collections.Specialized;
```

Далее, создадим функцию, которая поможет нам при чтение конфигурационных значений.

```
1. //
2. // A helper function to retrieve config values from the configuration file.
3. //
4.
5. private string GetConfigValue(string configValue, string defaultValue)
6. {
7.  return (string.IsNullOrEmpty(configValue)) ? defaultValue : configValue;
8. }
```

Следующее что мы сделаем, это добавим свойства из **web.config**. Мы собираемся жестко задать некоторые из них. Также мы хотим уникальные почтовые ящики для каждого пользователя и хеширование их паролей, и не будем позволять пользователям восстанавливать их пароли, (даже при использовании «Ответа на секретный вопрос»). Если вы захотите, вы без особого труда сможете сделать это сами.

```
01. //
02. // Properties from web.config, default all to False
03. //
04. private string _ApplicationName;
05. private bool _EnablePasswordReset;
06. private bool _EnablePasswordRetrieval = false;
07. private bool _RequiresQuestionAndAnswer = false;
08. private bool _RequiresUniqueEmail = true;
09. private int _MaxInvalidPasswordAttempts;
10. private int _PasswordAttemptWindow;
11. private int _MinRequiredPasswordLength;
12. private int _MinRequiredNonalphanumericCharacters;
13. private string _PasswordStrengthRegularExpression;
14. private MembershipPasswordFormat _PasswordFormat = MembershipPasswordFormat.Hashed;
```

И, наконец, добавим метод инициализации

```
01. public override void Initialize(string name, NameValueCollection config)
02. {
03.  if (config == null)
04.  throw new ArgumentNullException("config");
05.
06.  if (name == null || name.Length == 0)
07.  name = "CustomMembershipProvider";
08.
09.  if (String.IsNullOrEmpty(config["description"]))
10.  {
11.   config.Remove("description");
12.   config.Add("description", "Custom Membership Provider");
13.  }
14.
15.  base.Initialize(name, config);
16.
17.  _ApplicationName = GetConfigValue(config["applicationName"],
```

```
                System.Web.Hosting.HostingEnvironment.ApplicationVirtualPath);
 _MaxInvalidPasswordAttempts = Convert.ToInt32(
                GetConfigValue(config["maxInvalidPasswordAttempts"], "5"));
 _PasswordAttemptWindow = Convert.ToInt32(
                GetConfigValue(config["passwordAttemptWindow"], "10"));
 _MinRequiredNonalphanumericCharacters = Convert.ToInt32(
                GetConfigValue(config["minRequiredNonalphanumericCharacters"], "1"));
 _MinRequiredPasswordLength = Convert.ToInt32(
                GetConfigValue(config["minRequiredPasswordLength"], "6"));
 _EnablePasswordReset = Convert.ToBoolean(
                GetConfigValue(config["enablePasswordReset"], "true"));
_PasswordStrengthRegularExpression = Convert.ToString(
                GetConfigValue(config["passwordStrengthRegularExpression"], ""));

}
```

Я думаю эта функция довольно проста и не требует объяснения.
Теперь мы можем отредактировать узел Membership в файле **web.config** и удалить все жестко заданные параметры.

```
<membership defaultProvider="CustomMembershipProvider">
 <providers>
 <clear />
 <add name="CustomMembershipProvider"
      type="MyMembershipProvider"
      connectionStringName="CustomMembershipDB"
      enablePasswordReset="true"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="6"
      minRequiredNonalphanumericCharacters="0"
      passwordAttemptWindow="10"
      applicationName="/" />
 </providers>
</membership>
```

Следующее, что будет делать – реализуем все все свойства в нашем Membership Provider.

Откроем файл **MyMembershipProvider** и начнем редакитровать свойства и методы, как делется ниже:

```
public override string ApplicationName
{
 get { return _ApplicationName; }
 set { _ApplicationName = value; }
}
```

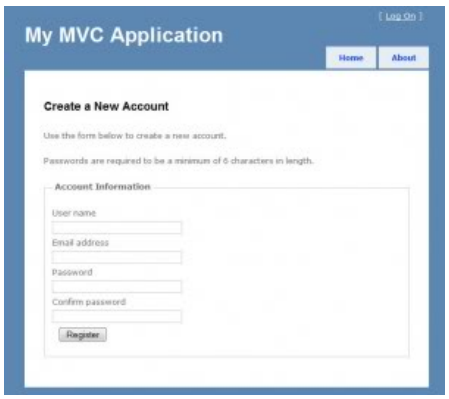
Т.к. мы не будем реализовывать восстановсление пароля используя секретный вопрос и ответ, мы можем задать:

```
public override bool ChangePasswordQuestionAndAnswer(string username,
                                                     string password,
                                                     string newPasswordQuestion,
                                                     string newPasswordAnswer)
{
 return false;
}
```

```
public override bool EnablePasswordReset
{
 get { return _EnablePasswordReset; }
}

public override bool EnablePasswordRetrieval
{
 get { return _EnablePasswordRetrieval; }
}

public override int MaxInvalidPasswordAttempts
{
 get { return _MaxInvalidPasswordAttempts; }
}

public override int MinRequiredNonAlphanumericCharacters
{
 get { return _MinRequiredNonalphanumericCharacters; }
}

public override int MinRequiredPasswordLength
{
 get { return _MinRequiredPasswordLength; }
}

public override int PasswordAttemptWindow
{
 get { return _PasswordAttemptWindow; }
}

public override MembershipPasswordFormat PasswordFormat
{
 get { return _PasswordFormat; }
}

public override string PasswordStrengthRegularExpression
{
 get { return _PasswordStrengthRegularExpression; }
}
```

```
40.
41. public override bool RequiresQuestionAndAnswer
42. {
43.  get { return _RequiresQuestionAndAnswer; }
44. }
45.
46. public override bool RequiresUniqueEmail
47. {
48.  get { return _RequiresUniqueEmail; }
49. }
```

После этого мы можем запустить наше приложение, перейти в раздел Log In->Register и увидеть:



В которой значение длины минимального пароля будет взято из **web.config**.

В случае, если я сделал где-то опечатки в **MyMembershipProvider.cs**, вы можете загрузить исходный код по этой ссылке: MyMembershipProvider.cs

Если мы попытаемся зарегистрировать нового пользователя, мы получем сообщение, что метод **CreateUser** еще не реализован.

Перед тем, как мы реалиуем **CreateUser**, нам нужно создать еще две дополнительные функции, которые будут им использоваться: **GetUser** и **GetUsernameByEmail**.

Они понадобятся нам для проверки имени пользователя и его почтового адреса на существование в базе данных.

Давайте начнем, создав класс **UserRepository**.

Щелкаем правой клавишей на папке Models и выбираем Add->Class, называем его **UserRepository** и нажимаем Добавить.

В только что созданном классе добавляем ссылку на сборку **System.Web.Security**.

```
1. using System.Web.Security;
```

И добавим поле, отвечающее за связь с базой данных:

```
1. private CustomMembershipDB _db;
2. public CustomMembershipDB db {
3.   get {
4.     if(_db == null) _db = new CustomMembershipDB();
5.     return _db;
6.   }
7. }
```

На этом шаге, нам нужно реализовать три метода:

CreateUser:

```
01. public MembershipUser CreateUser(string username, string password, string email)
02. {
03.   var user = new User
04.   {
05.     UserName = username,
06.     Email = email,
07.     PasswordSalt = "1234",
08.     CreatedDate = DateTime.Now,
09.     IsActivated = false,
10.     IsLockedOut = false,
11.     LastLockedOutDate = DateTime.Now;
```

```
    LastLockedOutDate = DateTime.Now,
    LastLoginDate = DateTime.Now,
    Password = password
  };

  db.AddToUsers(user);
  db.SaveChanges();

  return GetUser(username);
}
```

Вы можете видеть, что наш **CreateUser** метод из **UserRepository** использует Entity Data Model (**CustomMembershipDB**), созданную нами ранее. Мы вернемся в следующей части руководства к нему для шифрования пароля и активации email'а.

В случае, если вы удивлены, почему мы присвоили значение **DateTime.Now** свойствам **LastLockedOutDate** и **LastLoginDate**, дело в том, что по-умолчанию **MembershipUser** не позволяем им быть **Nullable**, это значит, они не могут быть записанны как NULL в базу данных. Для простоты, мы, при создании нового пользователя, просто задаем эти поля текущем временем.

GetUsernameByEmail:

```
public string GetUserNameByEmail(string email)
{
  var user = db.Users.FirstOrDefault(x => x.Email == email);

  return (user != null) ? user.UserName : string.Empty;
}
```

GetUser:

```
public MembershipUser GetUser(string username)
{
  var dbuser = db.Users.FirstOrDefault(u.UserName == username);
  if (dbuser != null)
  {
   string _username = dbuser.UserName;
   int _providerUserKey = dbuser.UserId;
   string _email = dbuser.Email;
   string _passwordQuestion = string.Empty;
   string _comment = dbuser.Comments;
   bool _isApproved = dbuser.IsActivated;
   bool _isLockedOut = dbuser.IsLockedOut;
   DateTime _creationDate = dbuser.CreatedDate;
   DateTime _lastLoginDate = dbuser.LastLoginDate;
   DateTime _lastActivityDate = DateTime.Now;
   DateTime _lastPasswordChangedDate = DateTime.Now;
   DateTime _lastLockedOutDate = dbuser.LastLockedOutDate;

   MembershipUser user = new MembershipUser("CustomMembershipProvider",
                                            _username,
                                            _providerUserKey,
                                            _email,
                                            _passwordQuestion,
                                            _comment,
                                            _isApproved,
                                            _isLockedOut,
                                            _creationDate,
                                            _lastLoginDate,
                                            _lastActivityDate,
                                            _lastPasswordChangedDate,
                                            _lastLockedOutDate);

   return user;
  }

   return null;
}
```

Снова, мы видим, что некоторые из свойств заданые значения DateTime.Now, и снова по той же причине.

Вкратце расскажем что делает этот метод. Он подключается к базе данных, считывает из нее данные о пользователе. Затем создается новый объект MembershipUser со свойствами из базы данных, и, наконец, он же и возвращается.

UserRepository класс может быть загружен ниже
UserRepository.cs

Теперь вернемся обратно в **MyMembershipProvider.cs** и добавим в нем ссылку на пространство имен **CustomMembership.Models**.

```
using CustomMembership.Models;
```

Добавим свойство для работы с данными из бд:

```
private CustomMembershipDB _db;
public CustomMembershipDB db {
  get {
    if(_db == null) _db = UserRepository();
    return _db;
  }
}
```

И реализуем следующие методы:

GetUsernameByEmail:

```
public override string GetUserNameByEmail(string email)

{
```

```
{
 return db.GetUserNameByEmail(email);
}
```

GetUser(string username, bool userIsOnline):

```
public override MembershipUser GetUser(string username, bool userIsOnline)
{
 return db.GetUser(username);
}
```

И, наконец, сам **CreateUser:**

```
public override MembershipUser CreateUser(string username,
                                          string password,
                                          string email,
                                          string passwordQuestion,
                                          string passwordAnswer,
                                          bool isApproved,
                                          object providerUserKey,
                                          out MembershipCreateStatus status)
{
 ValidatePasswordEventArgs args = new ValidatePasswordEventArgs(username,
                                                                password,
                                                                true);

 OnValidatingPassword(args);

 if (args.Cancel)
 {
  status = MembershipCreateStatus.InvalidPassword;
  return null;
 }

 if (RequiresUniqueEmail && string.IsNullOrEmpty(GetUserNameByEmail(email)))
 {
  status = MembershipCreateStatus.DuplicateEmail;
  return null;
 }

 MembershipUser u = GetUser(username, false);

 if (u == null)
 {
  db.CreateUser(username, password, email);
  status = MembershipCreateStatus.Success;

  return GetUser(username, false);
 }
 else
 {
  status = MembershipCreateStatus.DuplicateUserName;
 }

 return null;
}
```

Класс **MyMembershipProvider** может быть загружен ниже:
MyMembershipProvider.cs

После всего этого мы можем зарегистрироваться используя регистрационную форму (Log in->Register или по адресу /Account/Register) и проверить, что новый пользователь появился в базе данных.

Достаточно просто будет модифицировать метод ValidateUser, но прежде чем к этому приступить, нам нужно сделать еще несколько вещей в процессе регистрации (зашифровать пароль и создать для него соль(salt)). Мы оставим это до следующей части руководства.

Продолжение см. Реализация собственного Membership Provider (часть 3)

azhidkov 05.05.2011 17:13

11 RSS 4269

Теги: custom membership provider, asp.net mvc