Реализация собственного Membership Provider (часть 3)

Третья часть руководства по созданию собственного Membership Provider. Пред. см. Реализация собственного Membership Provider (часть 2)

В предыдущей части руководства мы реализовали метод **CreateUser**, который успешно добавляет нового пользователя в базу данных.

Первое, что мы хотим сделать сейчас – сгенерировать соль для пароля (password salt).
Откроем **UserRepository.cs** файл и добавим ссылку на System.Secority.Cryptography:

```
1. using System.Security.Cryptography;
```

и функцию для генерирования соли (в класс **UserRepository**):

```
1. private static string CreateSalt()
2. {
3.   RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
4.   byte[] buff = new byte[32];
5.   rng.GetBytes(buff);
6.
7.   return Convert.ToBase64String(buff);
8. }
```

Наконец, изменим метод CreateUser, для генерации и сохранения соли в базе данных, вместе с другими данными.

```
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 = CreateSalt(),
08.     CreatedDate = DateTime.Now,
09.     IsActivated = false,
10.     IsLockedOut = false,
11.     LastLockedOutDate = DateTime.Now,
12.     LastLoginDate = DateTime.Now,
13.     Password = password
14.   };
15.
16.   db.AddToUsers(user);
17.   db.SaveChanges();
18.
19.   return GetUser(username);
20. }
```

Давайте пойдем дальше и попробуем это. Запустим наше приложение и зарегистрируем нового пользователя. Если вы посмотрите внутрь таблицы в бд, вы увидите случайно сгенерированную, в процессе регистрации, соль.

Теперь давайте захешируем наш пароль. Добавьте следующий код в класс **UserRepository**:

```
1. private static string CreatePasswordHash(string pwd, string salt)
2. {
3.  string saltAndPwd = String.Concat(pwd, salt);
4.  string hashedPwd =
5.         FormsAuthentication.HashPasswordForStoringInConfigFile(
6.         saltAndPwd, "md5");
7.  return hashedPwd;
8. }
```

и снова изменим метод CreateUser для записи хешированного пароля. Т.к. мы собираемся использовать свойство **user.passwordSalt** как аргумент функции **CreatePasswordHash**, нам нужно изменить порядок инициализации свойств так, чтобы соль генерировалась перед хешированием пароля.

```
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 = CreateSalt(),
08.     CreatedDate = DateTime.Now,
09.     IsActivated = false,
10.     IsLockedOut = false,
11.     LastLockedOutDate = DateTime.Now,
12.     LastLoginDate = DateTime.Now
13.   };
14.   user.Password = CreatePasswordHash(password, user.PasswordSalt);
15.
16.   db.AddToUsers(user);
17.   db.SaveChanges();
18.
19.   return GetUser(username);
20. }
```

Я беспокоюсь, что это, возможно, не самый лучший способ хеширования пароля и генерации соли, и, уверен, они могут быть значительно улучшены. Но это не служит целью данного руководства, т.ч. оставим всё как есть.

Файл **UserRepository.cs** может быть загружен по адресу:
UserRepository.cs

У нас уже есть захешированный пароль и соль к нему в базе данных, т.ч. мы уже можем полностью реализовать метод **ValidateUser**.
Создадим метод **ValidateUser** в классе **UserRepository**.

```
public bool ValidateUser(string username, string password)
{
  var dbuser = db.Users.FirstOrDefault(x => x.UserName == username);

  return dbuser != null && dbuser.Password == CreatePasswordHash(password, dbuser.PasswordSalt;
}
```

и изменим метод **ValidateUser** в классе **MyMembershipProvider**:

```
public override bool ValidateUser(string username, string password)
{
 return db.ValidateUser(username, password);
}
```

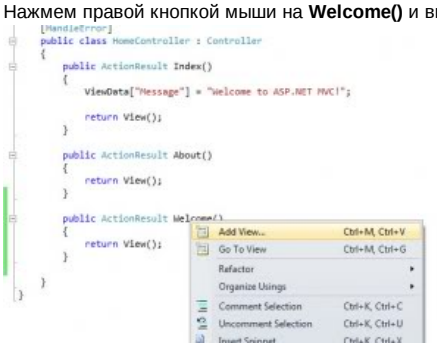
На данный момент мы можем запустить наше приложение, зарегистрировать нового пользователя и управлять его авторизацией.
Кстати, давайте сделаем чтобы наше приложение не авторизовывало пользователя сразу после регистрации.

Мы хотим, чтобы пользователь активировал свой аккаунт перейдя по ссылке в письме, отправленной ему после регистрации. До тех пор пока пользователь не активировал свою учетную запись он не сможет войти на сайт.

Откроем файл **HomeController.cs** и добавим в него метод **Welcome**:

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

Нажмем правой кнопкой мыши на **Welcome()** и выберем Add View...



В только-что созданном виде, добавим сообщение показываемое пользователю сразу после регистрации:

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

 <h2>Welcome</h2>

 <p>Thank you for registering!</p>

 <p>Activation email has been sent to you. Click on the link in
 the email to activate your account!</p>

</asp:Content>
```

Теперь откроем файл **AccountController.cs**, найдем метод **Register** и изменим:

```
if (createStatus == MembershipCreateStatus.Success)
{
 FormsService.SignIn(model.UserName, false /* createPersistentCookie */);
 return RedirectToAction("Index", "Home");
}
```

на

```
if (createStatus == MembershipCreateStatus.Success)
{
 return RedirectToAction("Welcome", "Home");
}
```

Если мы запустим наше приложение и зарегистрируем нового пользователя, мы будем перенаправлены на страницу с блогодарностями, и не будем авторизованы. То, что нам и нужно!

Давайте теперь сгенерируем ключ для активации аккаунта, и запишем его вместе с остальной информацией в БД.

Добавим следующий метод в класс **UserRepository**:

```
private static string GenerateKey()
{
```

```
  return Guid.NewGuid().ToString();
}
```

и изменим метод **CreateUser**, чтобы ключ сохранялся в поле **NewEmailKey**:

```
public MembershipUser CreateUser(string username, string password, string email)
{
  var user = new User
  {
    UserName = username,
    Email = email,
    PasswordSalt = CreateSalt(),
    CreatedDate = DateTime.Now,
    IsActivated = false,
    IsLockedOut = false,
    LastLockedOutDate = DateTime.Now,
    LastLoginDate = DateTime.Now,
    NewEmailKey = GenerateKey()
  };
  user.Password = CreatePasswordHash(password, user.PasswordSalt);

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

  return GetUser(username);
}
```

Теперь, когда у нас есть ключ для активации, мы можем отправить его пользователю на email.

В этом руководстве я реализую код отправки почты прямо в классе UserRepository. В серьезных приложения вам нужно будет хранить это в отдельном классе. Вам, также, нужно будет добавить обработчики ошибок в этот класс.

Также вам понадобится SMTP сервер позволяющий отсылать сообщения. Мы для этого будем использовать **GMail**.

Вот метод позволяющий нам отпралять сообщения используя нашу учетную запись на GMail.

```
private static void SendEmailThroughGmail(string messageBody, string emailTo)
        {
            SmtpClient client = new SmtpClient();
            NetworkCredential basicAuthenticationInfo =
                new NetworkCredential("blablabla@gmail.com", "password");
            client.Host = "smtp.gmail.com";
            client.UseDefaultCredentials = false;
            client.Credentials = basicAuthenticationInfo;
            client.EnableSsl = true;

            MailAddress to = new MailAddress(emailTo);
            MailAddress from = new MailAddress("blablabla@gmail.com", "Account activation",
              System.Text.Encoding.UTF8);

            MailMessage message = new MailMessage(from, to);
            message.Body = messageBody;
            message.IsBodyHtml = true;
            message.BodyEncoding = System.Text.Encoding.UTF8;
            message.Subject = "Account activation";
            message.SubjectEncoding = System.Text.Encoding.UTF8;

            client.Send(message);
        }
```

Чтобы функция выше работала вам вместо **"blablabla@gmail.com"** и **"password"** нужно, соответсветственно, указать данные от вашей учетной записи.

Сначала добавим ссылку на пространство имен **System.Net.Mail** (в **UserRepository.cs**):

```
using System.Net.Mail;
```

В метод CreateUser добавим следующий код:

```
public MembershipUser CreateUser(string username, string password, string email)
{
  var user = new User
  {
    UserName = username,
    Email = email,
    PasswordSalt = CreateSalt(),
    CreatedDate = DateTime.Now,
    IsActivated = false,
    IsLockedOut = false,
    LastLockedOutDate = DateTime.Now,
    LastLoginDate = DateTime.Now,
    NewEmailKey = GenerateKey()
  };
  user.Password = CreatePasswordHash(password, user.PasswordSalt);

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

  string ActivationLink = "http://localhost:PORT/Account/Activate/" +
                                    user.UserName + "/" + user.NewEmailKey;

```

```
23.   SendEmailThroughGmail(ActivationLink, user.Email);
24.
25.   return GetUser(username);
26. }
```

Вам нужно будет изменить PORT (на порт своего приложения, или удалить его на рабочем сервере).

Если в процессе регистрации у нас срабатывает исключение

Это означает, что ваш сервер не позволяет вам отсылать сообщения, и вам нужен другой.

Как только вы выполните эти шаги – вручную удалите все созданные до этого записи из базы данных, и впредь вы будете получать письмо для активации на email.

Довайте добавим функциональность для активации в **AccountController**.
Откроем **AccountController.cs** и добавим следующий код:

```
01. // ************************************
02. // URL: /Account/Activate/username/key
03. // ************************************
04.
05. public ActionResult Activate(string username, string key)
06. {
07.  UserRepository _user = new UserRepository();
08.  if (_user.ActivateUser(username, key) == false)
09.   return RedirectToAction("Index", "Home");
10.  else
11.   return RedirectToAction("LogOn");
12. }
```

Чтобы активационный Url работал нам нужно отредактировать файл **Global.asax.cs** добавив в него новый маршрут:

```
1. routes.MapRoute(
2.  "Activate",
3.  "Account/Activate/{username}/{key}",
4.  new { controller = "Account", action = "Activate",
5.  username = UrlParameter.Optional, key = UrlParameter.Optional } );
```

И, наконец, добавим метод **ActivateUser** в класс **UserRepository**:

```
01. public bool ActivateUser(string username, string key)
02. {
03.   var dbuser = db.Users.FirstOrDefault(x => x.UserName == username);
04.
05.   if (dbuser != null && dbuser.NewEmailKey == key)
06.   {
07.     dbuser.IsActivated = true;
08.     dbuser.LastModifiedDate = DateTime.Now;
09.     dbuser.NewEmailKey = null;
10.
11.     db.SaveChanges();
12.
13.     return true;
14.   }
15.
16.   return false;
17. }
```

В финале, изменим метод **ValidateUser** чтобы он авторизовывал только пользователей прошедших активацию.

```
1. public bool ValidateUser(string username, string password)
2. {
3.   var dbuser = db.Users.FirstOrDefault(x => x.UserName == username);
4.
5.   return dbuser != null && dbuser.Password == CreatePasswordHash(password, dbuser.PasswordSalt) &&
       dbuser.IsActivated;
6. }
```

Запустим приложение и зарегистрируем двух пользователей. Активируем одного из них, перейдя по ссылки в пришедшем на почту письме. И проверим, что мы можем заходить на сайт только с активированного аккаунта.

В следующей части руководства мы добавим несколько обработчиков ошибок и видов для поддержки реализованных процессов.

Файл из этого руководтсва могут быть загружены ниже (**прим. код загружается с сайта автора, поэтому возможны отличия в нем и данном переводе, как и во всех других файлах для загрузки*).
Global.asax.cs
AccountController.cs
UserRepository.cs

aznidkov 20.05.2011 16:15

Теги: asp.net mvc, custom membership provider