Gora LEYE ~ Expert C#.NET

ASP.NET MVC Custom Membership Password Hashing based on SALT key using SHA-3 Algorithm

***09** Lundi sept 2013*

PUBLIÉ PAR GORALEYE IN ASP.NET, C#.NET, CRYPTOGRAPHY, DATABASE, ENTITY FRAMEWORK, MEMBERSHIP, SECURITY

≈ **4 COMMENTAIRES**

Tags
.NET Framework, ASP.NET, C#.NET, Cryptography, database, Entity Framework, Hash, MD5, MemberShip, Password, Security, SHA-1, SHA-2, SHA-3

ASP.NET membership is designed to enable you to easily use a number of different membership providers for your ASP.NET applications. You can use the supplied membership providers that are included with the .NET Framework, or you can implement your own providers.

There are two primary reasons for creating a custom membership provider.

- You need to store membership information in a data source that is not supported by the membership providers included with the .NET Framework, such as a MysQL database, an Oracle database, or other data sources.
- You need to manage membership information using a database schema that is different from the database schema used by the providers that ship with the .NET Framework. A common example of this would be membership data that already exists in a SQL Server database for a company or Web site.

This tutorial is a continuation of our tutorial How to configure Custom Membership and Role Provider using ASP.NET MVC4

Let's talk about password encoding and decoding and take the opportunity to introduce the **Unit Of Work Pattern** used to insert and retrieve data.

There exists a sequence of hash functions SHA-0, SHA-1, SHA-2 and the most recent SHA-3.

SHA-3, is a new cryptographic hash function that has been selected by NIST in October 2012 following a public competition launched in 2007, this because the weaknesses discovered on MD-5 and SHA-1 let fear fragility SHA-2 is built on the same schéma. It has variations that can produce hashes 224, 256, 384 and 512 bits. It is built on a different principle from that of MD5, SHA-1 and SHA-2 functions.

So in our tutorial, we will use the SHA-3 512 bits,

For this, it is not intended to replace SHA-2, which is at present not been compromised by a significant attack, but to provide an alternative response to attacks against MD5 possibilities standards SHA-0 and SHA-1.

A. Unit Of Work Pattern

- We are using the default ASP.NET Membership database, so execute the script if it does not exist in your test database server, please see our tutorial

Creating the Application Services Database for SQL Server or download Sample.

The purpose of this article is not an introduction to **Unit Of Work Pattern** but a particular use of it. So for more information **Unit Of Work Pattern**, please read articles that deal with this subject such as http://msdn.microsoft.com/en-us/magazine/dd882510.aspx

- Create a class library Project and reference EntityFramwork.dll
- Generate the model and classes from membeship database (Model.Context.tt, Model.Entities.tt and Model.mapping.tt) . If you are newly in Entity Framwork, please read our tutorial **Introduction to entity framework Code first**
- Create IRepository interface

```
“public interface IRepository<T> where T : class
{
/// <summary>
/// Get the total objects count.
/// </summary>
int Count { get; }

/// <summary>
/// Gets all objects from database
/// </summary>
IQueryable<T> All();

/// <summary>
/// Gets object by primary key.
/// </summary>
/// <param name=”id”> primary key </param>
/// <returns> </returns>
T GetById(object id);
```

```
/// <summary>
/// Gets objects via optional filter, sort order, and includes
/// </summary>
/// <param name="filter"> </param>
/// <param name="orderBy"> </param>
/// <param name="includeProperties"> </param>
/// <returns> </returns>
IQueryable<T> Get(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>,
IOrderedQueryable<T>> orderBy = null, string includeProperties = "");

/// <summary>
/// Gets objects from database by filter.
/// </summary>
/// <param name="predicate"> Specified a filter </param>
IQueryable<T> Filter(Expression<Func<T, bool>> predicate);

/// <summary>
/// Gets objects from database with filting and paging.
/// </summary>
/// <param name="filter"> Specified a filter </param>
/// <param name="total"> Returns the total records count of the filter. </param>
/// <param name="index"> Specified the page index. </param>
/// <param name="size"> Specified the page size </param>
IQueryable<T> Filter(Expression<Func<T, bool>> filter, out int total, int index = 0, int size = 50);

/// <summary>
/// Gets the object(s) is exists in database by specified filter.
/// </summary>
/// <param name="predicate"> Specified the filter expression </param>
bool Contains(Expression<Func<T, bool>> predicate);

/// <summary>
/// Find object by keys.
/// </summary>
/// <param name="keys"> Specified the search keys. </param>
T Find(params object[] keys);

/// <summary>
/// Find object by specified expression.
/// </summary>
/// <param name="predicate"> </param>
T Find(Expression<Func<T, bool>> predicate);

/// <summary>
/// Create a new object to database.
/// </summary>
/// <param name="entity"> Specified a new object to create. </param>
T Create(T entity);
```

```
/// <summary>
/// Deletes the object by primary key
/// </summary>
/// <param name="id"> </param>
void Delete(object id);

/// <summary>
/// Delete the object from database.
/// </summary>
/// <param name="entity"> Specified a existing object to delete. </param>
void Delete(T entity);

/// <summary>
/// Delete objects from database by specified filter expression.
/// </summary>
/// <param name="predicate"> </param>
void Delete(Expression<Func<T, bool>> predicate);

/// <summary>
/// Update object changes and save to database.
/// </summary>
/// <param name="entity"> Specified the object to save. </param>
void Update(T entity);
/// <summary>
///
/// </summary>
/// <param name="query"></param>
/// <param name="parameters"></param>
/// <returns></returns>
IEnumerable<T> GetWithRawSql(string query, params object[] parameters);
}"
```

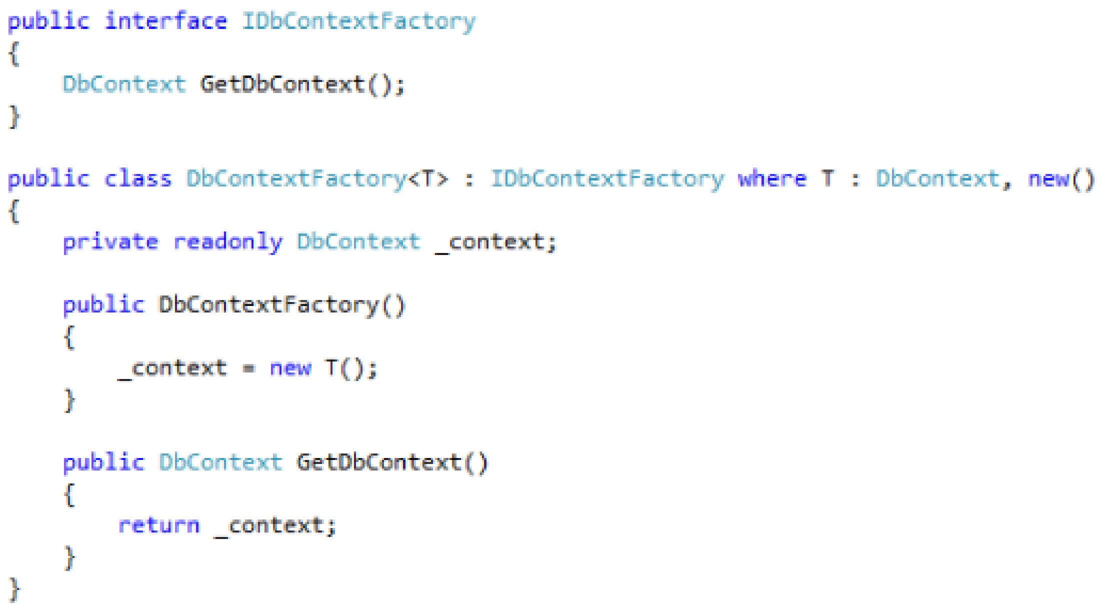
- Next, we are going to implement the IRepository interface

```
"public class Repository<T> : IRepository<T> where T : class
{
protected readonly DbContext _dbContext;
protected readonly DbSet<T> _dbSet;

public Repository(DbContext dbContext)
{
_dbContext = dbContext;
_dbSet = _dbContext.Set<T>();
}

public virtual int Count
{
get { return _dbSet.Count(); }
}
```

```
public virtual IQueryable<T> All()
{
return _dbSet.AsQueryable();
}

public virtual T GetById(object id)
{
return _dbSet.Find(id);
}

public virtual IQueryable<T> Get(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>,
IOrderedQueryable<T>> orderBy = null, string includeProperties = "")
{
IQueryable<T> query = _dbSet;

if (filter != null)
{
query = query.Where(filter);
}

if (!String.IsNullOrWhiteSpace(includeProperties))
{
foreach (var includeProperty in includeProperties.Split(new[] { ',' },
StringSplitOptions.RemoveEmptyEntries))
{
query = query.Include(includeProperty);
}
}

if (orderBy != null)
{
return orderBy(query).AsQueryable();
}
else
{
return query.AsQueryable();
}
}

public virtual IQueryable<T> Filter(Expression<Func<T, bool>> predicate)
{
return _dbSet.Where(predicate).AsQueryable();
}

public virtual IQueryable<T> Filter(Expression<Func<T, bool>> filter, out int total, int index = 0,
int size = 50)
{
int skipCount = index * size;
var resetSet = filter != null ? _dbSet.Where(filter).AsQueryable() : _dbSet.AsQueryable();
resetSet = skipCount == 0 ? resetSet.Take(size) : resetSet.Skip(skipCount).Take(size);
```

```
total = resetSet.Count();
return resetSet.AsQueryable();
}

public bool Contains(Expression<Func<T, bool>> predicate)
{
return _dbSet.Count(predicate) > 0;
}

public virtual T Find(params object[] keys)
{
return _dbSet.Find(keys);
}

public virtual T Find(Expression<Func<T, bool>> predicate)
{
return _dbSet.FirstOrDefault(predicate);
}

public virtual T Create(T entity)
{
var newEntry = _dbSet.Add(entity);
return newEntry;
}

public virtual void Delete(object id)
{
var entityToDelete = _dbSet.Find(id);
Delete(entityToDelete);
}

public virtual void Delete(T entity)
{
if (_dbContext.Entry(entity).State == EntityState.Detached)
{
_dbSet.Attach(entity);
}
_dbSet.Remove(entity);
}

public virtual void Delete(Expression<Func<T, bool>> predicate)
{
var entitiesToDelete = Filter(predicate);
foreach (var entity in entitiesToDelete)
{
if (_dbContext.Entry(entity).State == EntityState.Detached)
{
_dbSet.Attach(entity);
}
_dbSet.Remove(entity);
}
}

public virtual void Update(T entity)
{
var entry = _dbContext.Entry(entity);
_dbSet.Attach(entity);
entry.State = EntityState.Modified;
}

public virtual IEnumerable<T> GetWithRawSql(string query, params object[] parameters)
{
return _dbSet.SqlQuery(query, parameters).ToList();
}
}"
```

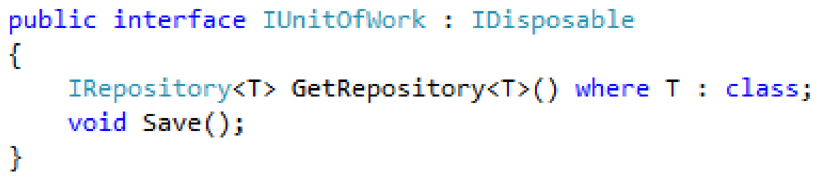
- Lets create a **IDbContextFactory** interface interface and implement it



```
public interface IDbContextFactory
{
    DbContext GetDbContext();
}

public class DbContextFactory<T> : IDbContextFactory where T : DbContext, new()
{
    private readonly DbContext _context;

    public DbContextFactory()
    {
        _context = new T();
    }

    public DbContext GetDbContext()
    {
        return _context;
    }
}
```

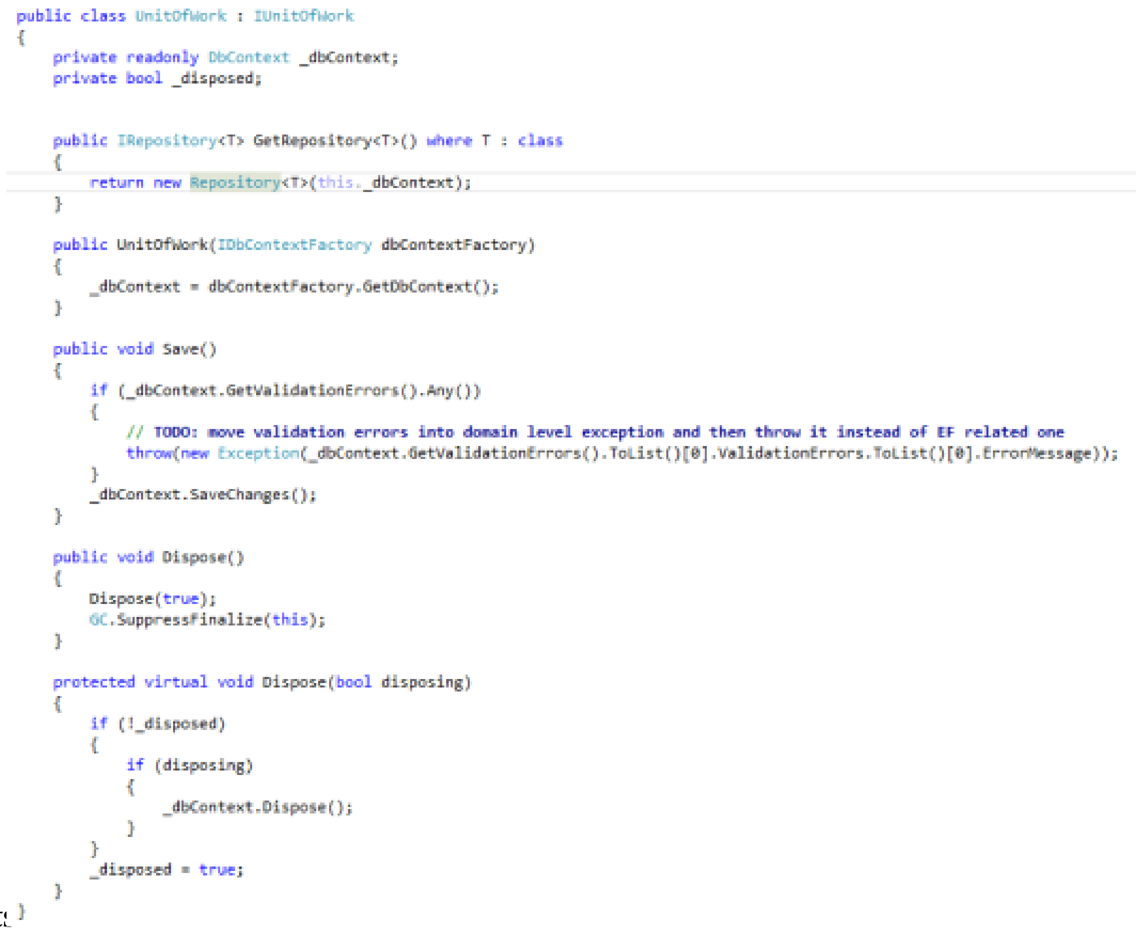
We will use the **DBContextFactory** interface to handle multiple database or schemas, for example if a database contains multiple schemas, we will have the opportunity to work with multiple schemas within a single context.

- Lets create a **IUnitOfWork** interfaceand implement it



```
public interface IUnitOfWork : IDisposable
{
    IRepository<T> GetRepository<T>() where T : class;
    void Save();
}
```

IUnitOfWork interface will be used only to get Repositories, save context and finally dispose



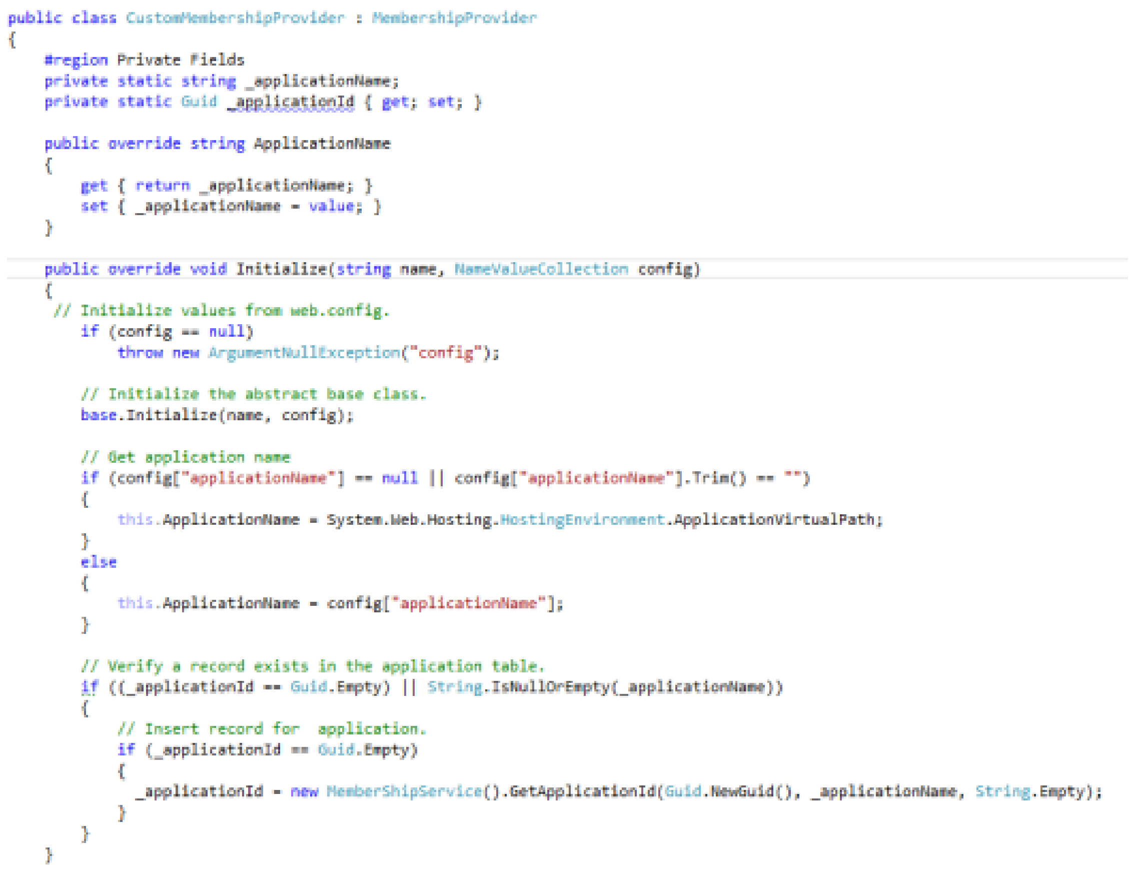
```
public class UnitOfWork : IUnitOfWork
{
    private readonly DbContext _dbContext;
    private bool _disposed;


    public IRepository<T> GetRepository<T>() where T : class
    {
        return new Repository<T>(this._dbContext);
    }

    public UnitOfWork(IDbContextFactory dbContextFactory)
    {
        _dbContext = dbContextFactory.GetDbContext();
    }

    public void Save()
    {
        if (_dbContext.GetValidationErrors().Any())
        {
            // TODO: move validation errors into domain level exception and then throw it instead of EF related one
            throw(new Exception(_dbContext.GetValidationErrors().ToList()[0].ValidationErrors.ToList()[0].ErrorMessage));
        }
        _dbContext.SaveChanges();
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _dbContext.Dispose();
            }
        }
        _disposed = true;
    }
}
```

objects

B. Implementation of MembershipProvider

Our **CustomMembershipProvider** derives from **MembershipProvider**

The first thing we are doing is to override **Initialize(string name, NameValueCollection config)** so as to get config parameters and also get ApplicationId or create it if does not exist



```
public class CustomMembershipProvider : MembershipProvider
{
    #region Private Fields
    private static string _applicationName;
    private static Guid _applicationId { get; set; }

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

    public override void Initialize(string name, NameValueCollection config)
    {
     // Initialize values from web.config.
        if (config == null)
            throw new ArgumentNullException("config");

        // Initialize the abstract base class.
        base.Initialize(name, config);

        // Get application name
        if (config["applicationName"] == null || config["applicationName"].Trim() == "")
        {
            this.ApplicationName = System.Web.Hosting.HostingEnvironment.ApplicationVirtualPath;
        }
        else
        {
            this.ApplicationName = config["applicationName"];
        }

        // Verify a record exists in the application table.
        if ((_applicationId == Guid.Empty) || String.IsNullOrEmpty(_applicationName))
        {
            // Insert record for  application.
            if (_applicationId == Guid.Empty)
            {
              _applicationId = new MemberShipService().GetApplicationId(Guid.NewGuid(), _applicationName, String.Empty);
            }
        }
    }
```

Try it :

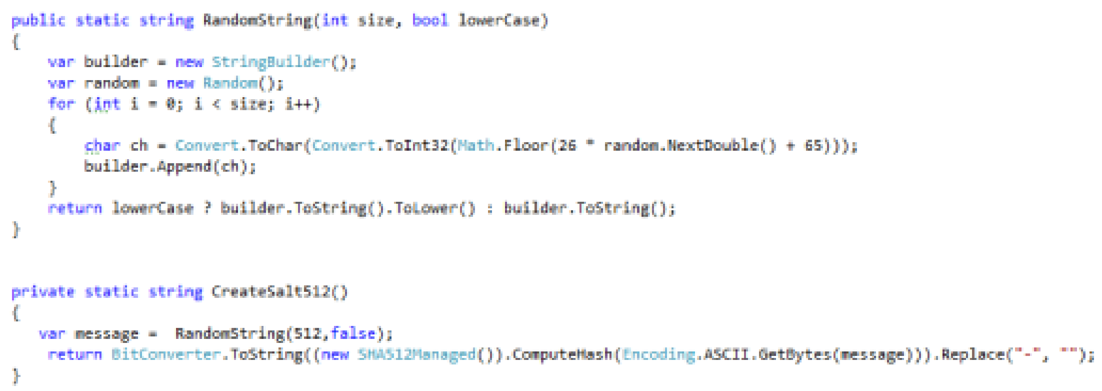
```
“public override void Initialize(string name, NameValueCollection config)
{
// Initialize values from web.config.
if (config == null)
throw new ArgumentNullException("config");

// Initialize the abstract base class.
base.Initialize(name, config);

// Get application name
if (config["applicationName"] == null || config["applicationName"].Trim() == "")
{
ApplicationName = System.Web.Hosting.HostingEnvironment.ApplicationVirtualPath;
}
else
{
ApplicationName = config["applicationName"];
}

// Verify a record exists in the application table.
if ((_applicationId == Guid.Empty) || String.IsNullOrEmpty(_applicationName))
{
// Insert record for application.
if (_applicationId == Guid.Empty)
{
_applicationId = new MemberShipService().GetApplicationId(Guid.NewGuid(), _applicationName,
String.Empty);
}
}
}”
```

B.1 Encoding Password

Now lets create a random salt using SHA 512 string format **CreateSalt512**

*In cryptography, a **salt** is random data that are used as an additional input to a one-way function that hashes a password or passphrase. The primary function of salts is to defend against dictionary attacks and pre-computed rainbow table attacks.*

A new salt is randomly generated for each password. In a typical setting, the salt and the password are concatenated and processed with a cryptographic hash function, and the resulting output (but not the original password) is stored with the salt in a database. Hashing allows for later authentication while defending against compromise of the plaintext password in the event that the database is somehow compromised.

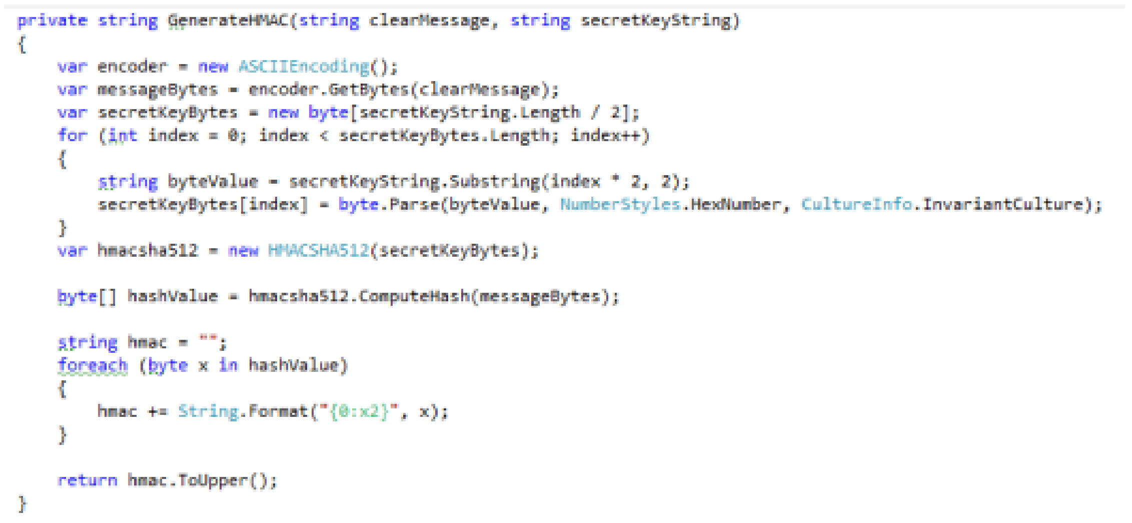


```
public static string RandomString(int size, bool lowerCase)
{
    var builder = new StringBuilder();
    var random = new Random();
    for (int i = 0; i < size; i++)
    {
        char ch = Convert.ToChar(Convert.ToInt32(Math.Floor(26 * random.NextDouble() + 65)));
        builder.Append(ch);
    }
    return lowerCase ? builder.ToString().ToLower() : builder.ToString();
}


private static string CreateSalt512()
{
   var message =  RandomString(512,false);
    return BitConverter.ToString((new SHA512Managed()).ComputeHash(Encoding.ASCII.GetBytes(message))).Replace("-", "");
}
```

Try it :

> *"public static string RandomString(int size, bool lowerCase)*
> *{*
> *var builder = new StringBuilder();*
> *var random = new Random();*
> *for (int i = 0; i < size; i++)*
> *{*
> *char ch = Convert.ToChar(Convert.ToInt32(Math.Floor(26 * random.NextDouble() + 65)));*
> *builder.Append(ch);*
> *}*
> *return lowerCase ? builder.ToString().ToLower() : builder.ToString();*
> *}*
> *private static string CreateSalt512()*
> *{*
> *var message = RandomString(512,false);*
> *return BitConverter.ToString((new SHA512Managed()).ComputeHash(Encoding.ASCII.GetBytes(message))).Replace("-", "");*
> *}"*

- Now we will generate a hashed Password using our clear password and our resultes salt (secret key)



```
private string GenerateHMAC(string clearMessage, string secretKeyString)
{
    var encoder = new ASCIIEncoding();
    var messageBytes = encoder.GetBytes(clearMessage);
    var secretKeyBytes = new byte[secretKeyString.Length / 2];
    for (int index = 0; index < secretKeyBytes.Length; index++)
    {
        string byteValue = secretKeyString.Substring(index * 2, 2);
        secretKeyBytes[index] = byte.Parse(byteValue, NumberStyles.HexNumber, CultureInfo.InvariantCulture);
    }
    var hmacsha512 = new HMACSHA512(secretKeyBytes);

    byte[] hashValue = hmacsha512.ComputeHash(messageBytes);

    string hmac = "";
    foreach (byte x in hashValue)
    {
        hmac += String.Format("{0:x2}", x);
    }

    return hmac.ToUpper();
}
```

Try it :

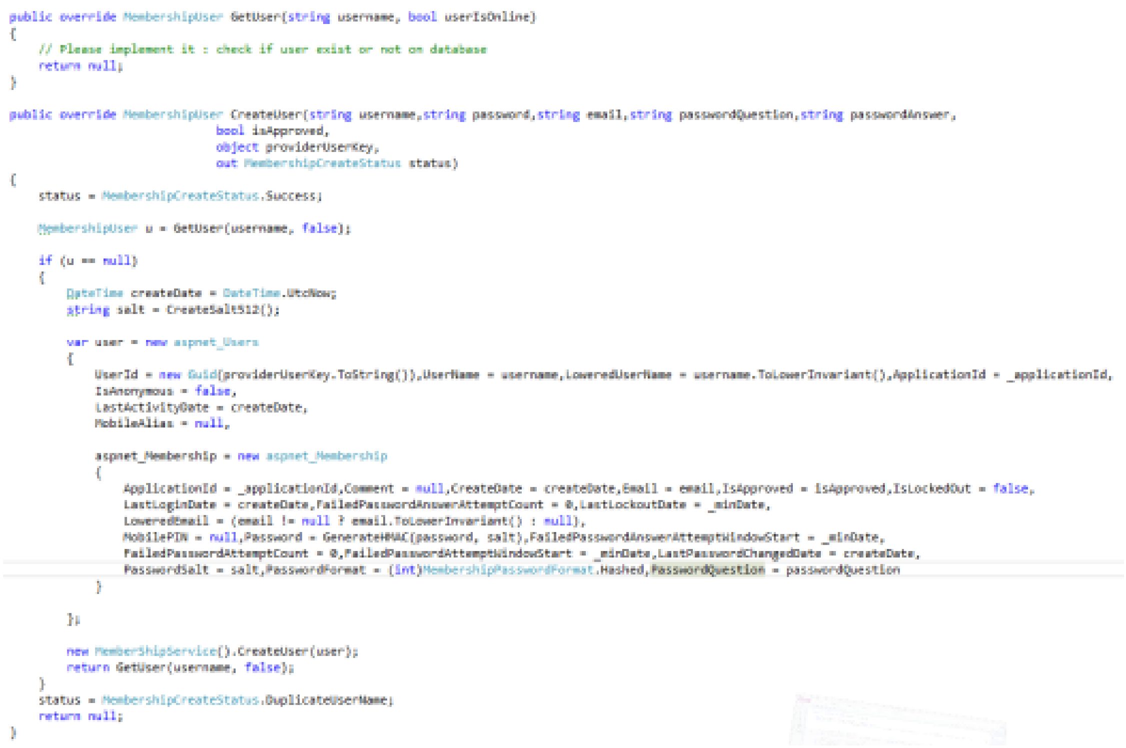
"private string GenerateHMAC(string clearMessage, string secretKeyString)
{
var encoder = new ASCIIEncoding();
var messageBytes = encoder.GetBytes(clearMessage);
var secretKeyBytes = new byte[secretKeyString.Length / 2];
for (int index = 0; index < secretKeyBytes.Length; index++)
{
*string byteValue = secretKeyString.Substring(index * 2, 2);*
secretKeyBytes[index] = byte.Parse(byteValue, NumberStyles.HexNumber, CultureInfo.InvariantCulture);
}
var hmacsha512 = new HMACSHA512(secretKeyBytes);

byte[] hashValue = hmacsha512.ComputeHash(messageBytes);

string hmac = "";
foreach (byte x in hashValue)
{
hmac += String.Format("{0:x2}", x);
}

return hmac.ToUpper();
}"

- Now we 'll use the GenerateHMAC function to Hash password and put in on database table



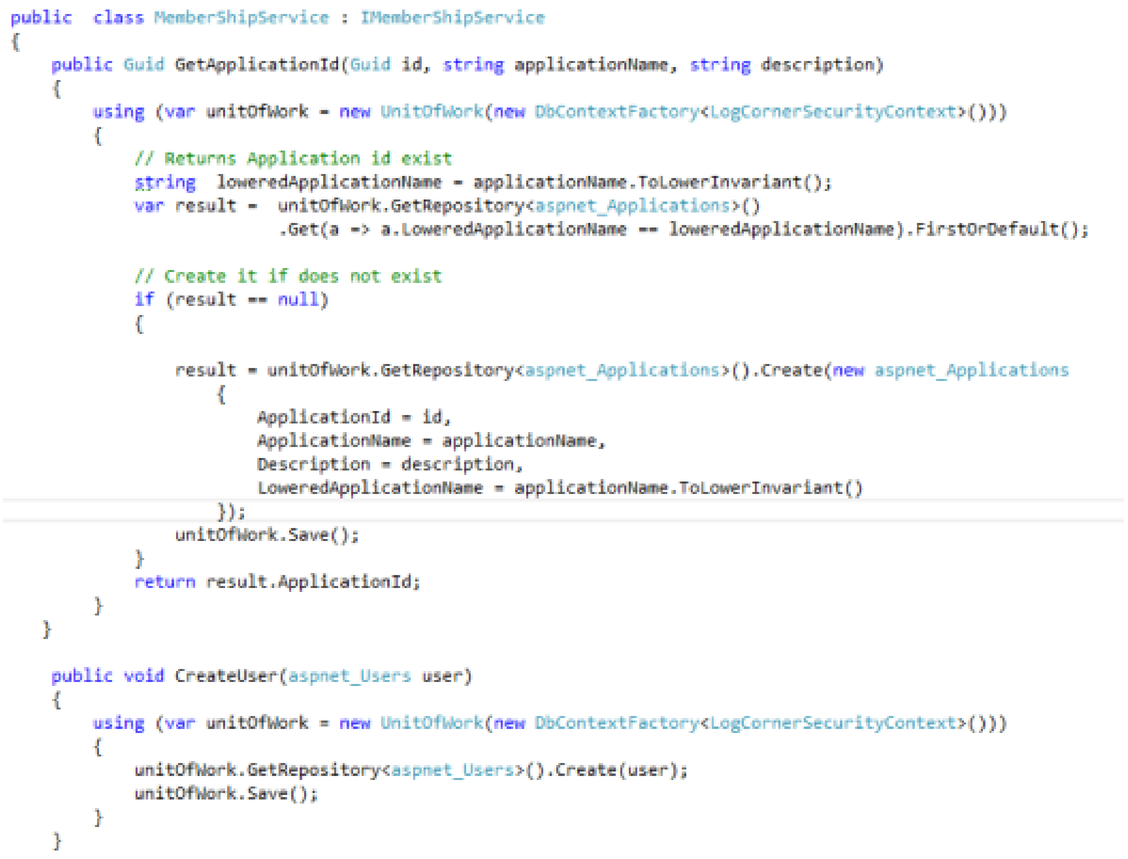
```
public override MembershipUser GetUser(string username, bool userIsOnline)
{
    // Please implement it : check if user exist or not on database
    return null;
}

public override MembershipUser CreateUser(string username,string password,string email,string passwordQuestion,string passwordAnswer,
                          bool isApproved,
                          object providerUserKey,
                          out MembershipCreateStatus status)
{
    status = MembershipCreateStatus.Success;

    MembershipUser u = GetUser(username, false);

    if (u == null)
    {
        DateTime createDate = DateTime.UtcNow;
        string salt = CreateSalt512();

        var user = new aspnet_Users
        {
            UserId = new Guid(providerUserKey.ToString()),UserName = username,LoweredUserName = username.ToLowerInvariant(),ApplicationId = _applicationId,
            IsAnonymous = false,
            LastActivityDate = createDate,
            MobileAlias = null,

            aspnet_Membership = new aspnet_Membership
            {
                ApplicationId = _applicationId,Comment = null,CreateDate = createDate,Email = email,IsApproved = isApproved,IsLockedOut = false,
                LastLoginDate = createDate,FailedPasswordAnswerAttemptCount = 0,LastLockoutDate = _minDate,
                LoweredEmail = (email != null ? email.ToLowerInvariant() : null),
                MobilePIN = null,Password = GenerateHMAC(password, salt),FailedPasswordAnswerAttemptWindowStart = _minDate,
                FailedPasswordAttemptCount = 0,FailedPasswordAttemptWindowStart = _minDate,LastPasswordChangedDate = createDate,
                PasswordSalt = salt,PasswordFormat = (int)MembershipPasswordFormat.Hashed,PasswordQuestion = passwordQuestion
            }

        };

        new MemberShipService().CreateUser(user);
        return GetUser(username, false);
    }
    status = MembershipCreateStatus.DuplicateUserName;
    return null;
}
```

Try it :

"public override MembershipUser GetUser(string username, bool userIsOnline)
{
// Please implement it : check if user exist or not on database

```
return null;
}

public override MembershipUser CreateUser(string username,string password,string email,string
passwordQuestion,string passwordAnswer,
bool isApproved,
object providerUserKey,
out MembershipCreateStatus status)
{
status = MembershipCreateStatus.Success;

MembershipUser u = GetUser(username, false);

if (u == null)
{
DateTime createDate = DateTime.UtcNow;
string salt = CreateSalt512();

var user = new aspnet_Users
{
UserId = new Guid(providerUserKey.ToString()),UserName = username,LoweredUserName =
username.ToLowerInvariant(),ApplicationId = _applicationId,
IsAnonymous = false,
LastActivityDate = createDate,
MobileAlias = null,

aspnet_Membership = new aspnet_Membership
{
ApplicationId = _applicationId,Comment = null,CreateDate = createDate,Email = email,IsApproved
= isApproved,IsLockedOut = false,
LastLoginDate = createDate,FailedPasswordAnswerAttemptCount = 0,LastLockoutDate =
_minDate,
LoweredEmail = (email != null ? email.ToLowerInvariant() : null),
MobilePIN = null,Password = GenerateHMAC(password,
salt),FailedPasswordAnswerAttemptWindowStart = _minDate,
FailedPasswordAttemptCount = 0,FailedPasswordAttemptWindowStart =
_minDate,LastPasswordChangedDate = createDate,
PasswordSalt = salt,PasswordFormat =
(int)MembershipPasswordFormat.Hashed,PasswordQuestion = passwordQuestion
}

};

new MemberShipService().CreateUser(user);
return GetUser(username, false);
}
status = MembershipCreateStatus.DuplicateUserName;
return null;
}”
```

- Now lets implement our operations of **MemberShipService** class :



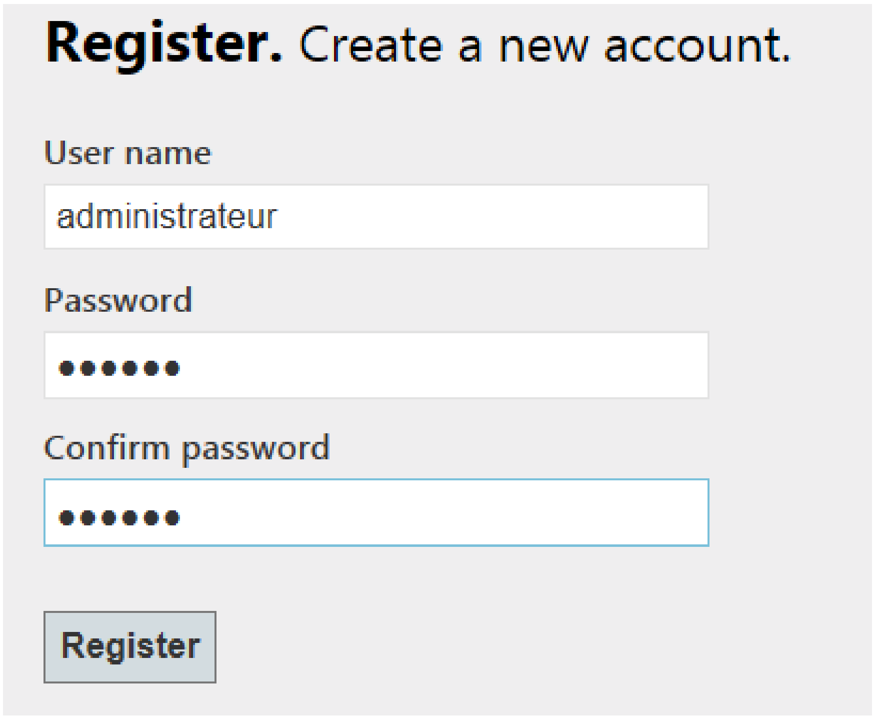
```
public  class MemberShipService : IMemberShipService
{
    public Guid GetApplicationId(Guid id, string applicationName, string description)
    {
        using (var unitOfWork = new UnitOfWork(new DbContextFactory<LogCornerSecurityContext>()))
        {
            // Returns Application id exist
            string  loweredApplicationName = applicationName.ToLowerInvariant();
            var result =  unitOfWork.GetRepository<aspnet_Applications>()
                          .Get(a => a.LoweredApplicationName == loweredApplicationName).FirstOrDefault();

            // Create it if does not exist
            if (result == null)
            {

                result = unitOfWork.GetRepository<aspnet_Applications>().Create(new aspnet_Applications
                    {
                        ApplicationId = id,
                        ApplicationName = applicationName,
                        Description = description,
                        LoweredApplicationName = applicationName.ToLowerInvariant()
                    });
                unitOfWork.Save();
            }
            return result.ApplicationId;
        }
   }

    public void CreateUser(aspnet_Users user)
    {
        using (var unitOfWork = new UnitOfWork(new DbContextFactory<LogCornerSecurityContext>()))
        {
            unitOfWork.GetRepository<aspnet_Users>().Create(user);
            unitOfWork.Save();
        }
    }
```

Try it :

```
“public Guid GetApplicationId(Guid id, string applicationName, string description)
{
using (var unitOfWork = new UnitOfWork(new DbContextFactory<LogCornerSecurityContext>
()))
{
// Returns Application id exist
string loweredApplicationName = applicationName.ToLowerInvariant();
var result = unitOfWork.GetRepository<aspnet_Applications>()
.Get(a => a.LoweredApplicationName == loweredApplicationName).FirstOrDefault();

// Create it if does not exist
if (result == null)
{

result = unitOfWork.GetRepository<aspnet_Applications>().Create(new aspnet_Applications
{
ApplicationId = id,
ApplicationName = applicationName,
Description = description,
LoweredApplicationName = applicationName.ToLowerInvariant()
});
unitOfWork.Save();
}
return result.ApplicationId;
}
}

public void CreateUser(aspnet_Users user)
{
using (var unitOfWork = new UnitOfWork(new DbContextFactory<LogCornerSecurityContext>
()))
{
unitOfWork.GetRepository<aspnet_Users>().Create(user);
unitOfWork.Save();
}
}”
```

- Run our sample application and enter user informations, so password is hashed and stored as follows :

Login informations



Register. Create a new account.

User name

administrateur

Password

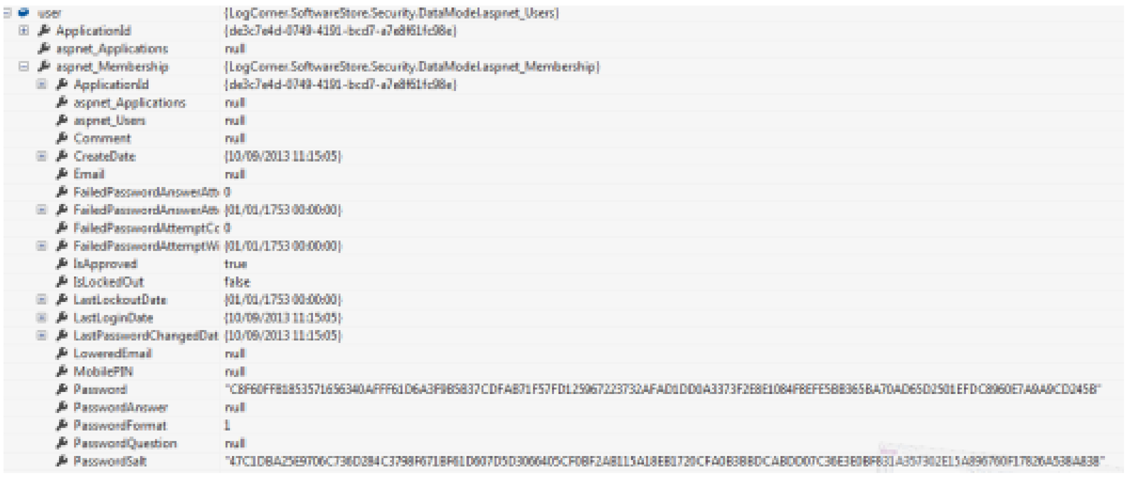
●●●●●●

Confirm password

●●●●●●

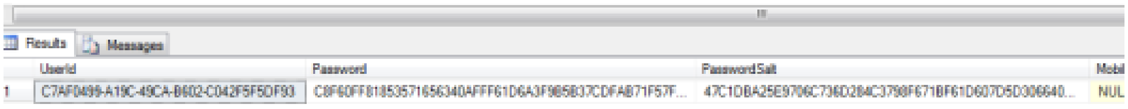
Register

Runtime values



```
user                        {LogCorner.SoftwareStore.Security.DataModel.aspnet_Users}
  ApplicationId             {de3c7e4d-0749-4191-bcd7-a7e8f61fc98e}
  aspnet_Applications       null
  aspnet_Membership         {LogCorner.SoftwareStore.Security.DataModel.aspnet_Membership}
    ApplicationId           {de3c7e4d-0749-4191-bcd7-a7e8f61fc98e}
    aspnet_Applications     null
    aspnet_Users            null
    Comment                 null
    CreateDate              {10/09/2013 11:15:05}
    Email                   null
    FailedPasswordAnswerAtt 0
    FailedPasswordAnswerAtt {01/01/1753 00:00:00}
    FailedPasswordAttemptCc 0
    FailedPasswordAttemptWi {01/01/1753 00:00:00}
    IsApproved              true
    IsLockedOut             false
    LastLockoutDate         {01/01/1753 00:00:00}
    LastLoginDate           {10/09/2013 11:15:05}
    LastPasswordChangedDat  {10/09/2013 11:15:05}
    LoweredEmail            null
    MobilePIN               null
    Password                "C8F60FF81853571656340AFFF61D6A3F9B5837CDFAB71F57FD125967223732AFAD1DD0A3373F2E8E1084F8EFE5BB365BA70AD65D2501EFDC8960E7A9A9CD245B"
    PasswordAnswer          null
    PasswordFormat          1
    PasswordQuestion        null
    PasswordSalt            "47C1DBA25E9706C736D284C3798F671BF61D607D5D3066405CF0BF2AB115A18EB1720CFA0B3BBDCABDD07C38E3E0BF831A357302E15A896760F17826A538A838"
```

Database table [dbo].[aspnet_Membership] values



Results | Messages

	UserId	Password	PasswordSalt	Mobil
1	C7AF0499-A19C-49CA-B602-C042F5F5DF93	C8F60FF81853571656340AFFF61D6A3F9B5837CDFAB71F57F...	47C1DBA25E9706C736D284C3798F671BF61D607D5D306640...	NUL

So we can see that password is hashed according to salt and stored in database.

To better understand and right implement salt, please seek for more information about How Hashes are Cracked ? what happen if hacker try to decode password ?

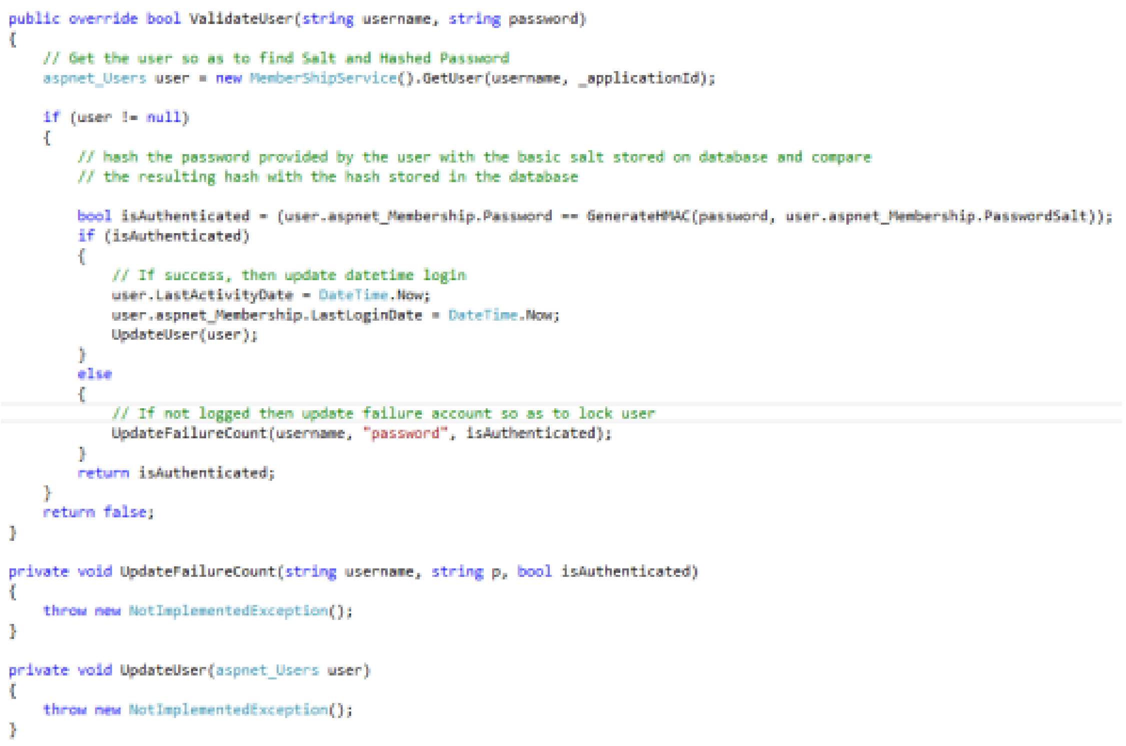
Before using Salt Password Hashing, we must understand what is **Dictionary and Brute Force Attacks**, **Reverse Lookup Tables**, **Lookup Tables** and **Rainbow Tables.**

B.2 << Matching >> Password

Now, to << decode >> the password (if one can speak of decoding) we can simply hash the password provided by the user with the basic salt stored on database and compare the resulting hash with the hash stored in the database.

So one and only the user knows the password. If any one else can discover the password then we are the victim of a successful attack.

The validate function can look like this



```
public override bool ValidateUser(string username, string password)
{
    // Get the user so as to find Salt and Hashed Password
    aspnet_Users user = new MemberShipService().GetUser(username, _applicationId);

    if (user != null)
    {
        // hash the password provided by the user with the basic salt stored on database and compare
        // the resulting hash with the hash stored in the database

        bool isAuthenticated = (user.aspnet_Membership.Password == GenerateHMAC(password, user.aspnet_Membership.PasswordSalt));
        if (isAuthenticated)
        {
            // If success, then update datetime login
            user.LastActivityDate = DateTime.Now;
            user.aspnet_Membership.LastLoginDate = DateTime.Now;
            UpdateUser(user);
        }
        else
        {
            // If not logged then update failure account so as to lock user
            UpdateFailureCount(username, "password", isAuthenticated);
        }
        return isAuthenticated;
    }
    return false;
}

private void UpdateFailureCount(string username, string p, bool isAuthenticated)
{
    throw new NotImplementedException();
}

private void UpdateUser(aspnet_Users user)
{
    throw new NotImplementedException();
}
```

Try it :

```
"public override bool ValidateUser(string username, string password)
{
// Get the user so as to find Salt and Hashed Password
aspnet_Users user = new MemberShipService().GetUser(username, _applicationId);

if (user != null)
{
// hash the password provided by the user with the basic salt stored on database and compare
// the resulting hash with the hash stored in the database

bool isAuthenticated = (user.aspnet_Membership.Password == GenerateHMAC(password,
user.aspnet_Membership.PasswordSalt));
if (isAuthenticated)
{
// If success, then update datetime login
user.LastActivityDate = DateTime.Now;
user.aspnet_Membership.LastLoginDate = DateTime.Now;
UpdateUser(user);
}
else
{
// If not logged then update failure account so as to lock user
UpdateFailureCount(username, "password", isAuthenticated);
}
return isAuthenticated;
}
return false;
}"
```

Now, what can happen if clear password travels on networks before being encoded ?

We will see this topic at next tutorial

Regards

réflexions à propos de " ASP.NET MVC Custom Membership Password Hashing based on SALT key using SHA-3 Algorithm "

1. Ping: How to configure Custom Membership Provider using ASP.NET MVC4 with external login like facebook, yahoo , google or other relying party accounts. | Gora LEYE
2. Ping: How to configure Custom Membership and Role Provider using ASP.NET MVC4 | Gora LEYE
3. Ping: ASP.NET MVC Tracking User Activity | Gora LEYE

4. Ping: Introduction to entity framework Code first | Gora LEYE

4