Practical Exercise 7

Q1. Creating SQL Server Express Database

- Open the given [**Practical7**] ASP.NET web application project in Visual Studio.
- Create a new SQL Server Express database named [**UserDB.mdf**] in the [**App_Data**] folder.
- Create a new table named [**Member**] with the following fields:

Name	Data Type	Allow Nulls
Id	char(2)	☐
Name	varchar(50)	☐

- Fill the [**Member**] table with the following records:

Id	Name
IA	Information Systems Engineering
IB	Business InformationSystems
IT	Internet Technology

- Create a new table named [**Admin**] with the following fields.

Name	Data Type	Allow Nulls
Id	char(10)	☐
Name	varchar(30)	☐
Gender	char(1)	☐
ProgramId	char(2)	☐

- Fill the [**Admin**] table with the following records:

Id	Name	Gender	ProgramId
12ABC00001	Ng Swee Chin	F	IA
12ABC00002	Lim Mei Shyan	F	IA
12ABC00003	Low Kok Han	M	IB
12ABC00004	Ting Hie Choon	F	IB
12ABC00005	Teh Boon Chuan	M	IT
12ABC00006	Teo Soon Beng	M	IT

Q2. Creating Database View

- Create a new database view named [**User**] that extract login data from both tables. The SQL required is as follows:

```sql
CREATE VIEW [dbo].[User]
AS

SELECT [Username], [Hash], 'Member' AS [Role]
FROM [Member]

UNION

SELECT [Username], [Hash], 'Admin' AS [Role]
FROM [Admin]
```

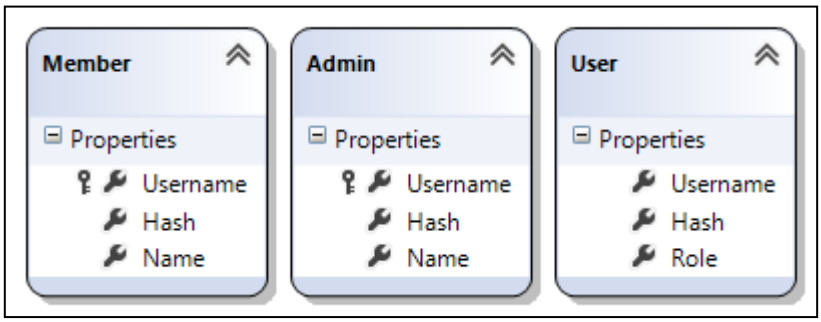
- Right-click the [**User**] view and select [**Show Results**] to examine its data:

Username	Hash	Role
potato	XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=	Member
tomato	XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=	Member
admin1	XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=	Admin
admin2	XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=	Admin

Q3. Adding Database View to OR Designer

- Add a new DBML (LINQ to SQL Classes) file named [**UserDB.dbml**] to the project.

- From **Server Explorer**, drag-and-drop the [**Member**] table, the [**Admin**] table and **[User]** view to the **Object Relational (OR) Designer**. The following entity classes should be generated:



Q4. Configuring Authentication

- In **Solution Explorer**, open the [**Web.config**] file under the root folder.

- Add a new **<authentication>** element under the hierarchy [**configuration** > **system.web**]. Type the XML codes as shown below:

```
<configuration>
    ...
    <system.web>
        ...
        <!-- Authentication -->
        <authentication mode="Forms">
            <forms defaultUrl="Home.aspx"
                   loginUrl="Login.aspx"
                   timeout="20160" />
        </authentication>
        ...
    </system.web>
    ...
</configuration>
```

Other sections are not shown for brevity

We are implementing **Forms Authentication**

URL for the default landing page (after login) and the login page

Remember the user login session for 14 days (14 X 24 X 60 = 20160 minutes) if he opts for

Q5. Configuring Authorization

- Within the same [**Web.config**] file, add a new **<location>** element under the hierarchy [**configuration**]. Write authorization rules to prevent **Anonymous** users from accessing the [**Protected.aspx**] page:

```
<configuration>
    ...
    <system.web>
        ...
    </system.web>
    ...
    <!-- Authorization -->
    <location path="Protected.aspx">
        <system.web>
            <authorization>
                <deny users="?" />
            </authorization>
        </system.web>
    </location>
    ...
</configuration>
```

The setting is applied to [**Protected.aspx**] page

Anonymous users are denied from accessing the page

- Create a new [**Web.config**] file under the [**MemberOnly**] folder. Add a new **<authorization>** element under the hierarchy [**configuration** > **system.web**]. Write authorization rules to allow <u>ONLY</u> **Member** users to access the folder:

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <authorization>
            <allow roles="Member" />
            <deny users="*" />
        </authorization>
    </system.web>
</configuration>
```

<u>ONLY</u> **Member** users are allowed to access to the folder (and its pages)

- Create a new [**Web.config**] file under the [**AdminOnly**] folder. Add a new **<authorization>** element under the hierarchy [**configuration** > **system.web**]. Write authorization rules to allow ONLY **Admin** users to access the folder:

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <authorization>
            <allow roles="Admin" />
            <deny users="*" />
        </authorization>
    </system.web>
</configuration>
```

ONLY **Admin** users are allowed to access to the folder (and its pages)

- Test the result. You should be denied from accessing the [**Protected.aspx**] page, as well as pages under the [**MemberOnly**] and [**AdminOnly**] folder.

Q6. Programming the Hash Function

- Add a new C# class file named [**Security.cs**] under the root folder:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Practical7
{
    public class Security
    {

    }
}
```

- Add the following additional **using** statements:

```
using System.Security.Cryptography;
using System.Text;
```

NOTE: You can also add an **using** statement automatically by placing the mouse cursor on the targeted class (e.g. **Encoding**) and press [**Ctrl** + **.**] to activate and select an option (e.g. add **using** statement) from the smart tag menu.

- Within the class, create a new method named **GetHash**. The method should convert a plaintext password to its base64 encoded hash by using the SHA-256 hash function:

```
public static string GetHash(string strPass)
{
    byte[] binPass = Encoding.Default.GetBytes(strPass);

    SHA256 sha = SHA256.Create();
    byte[] binHash = sha.ComputeHash(binPass);
    string strHash = Convert.ToBase64String(binHash);

    return strHash;
}
```

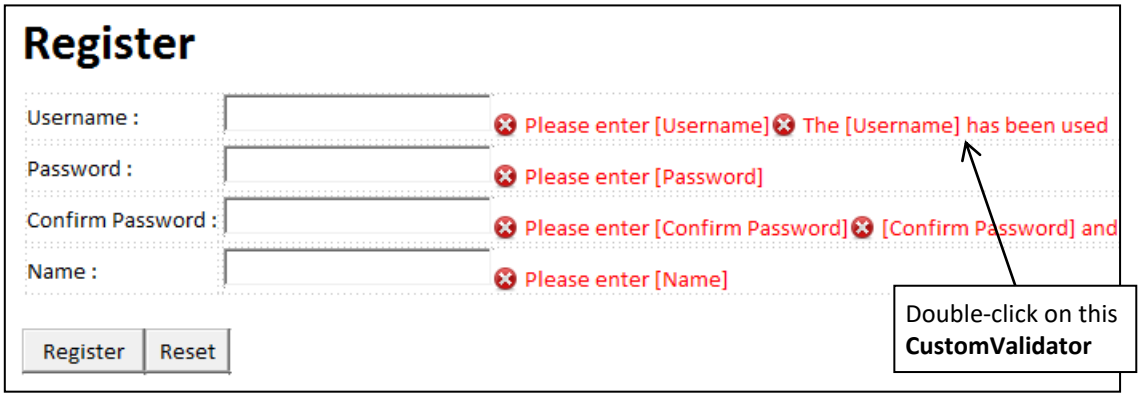
Convert string to byte array

Compute the hash, which works with byte array only

Convert byte array to base64 encoded string

Q7. Programming the Register Page

- Open the [**Register.aspx**] page:



- Double-click on the **CustomValidator** with error message [**The [Username] has been used**]. Program its **ServerValidate** event handler to detect duplicated username:

```
protected void CustomValidator1_ServerValidate(object source,
                                               ServerValidateEventArgs args)
{
    string username = args.Value;

    // Check for duplicated username
    if (db.Users.Any(u => u.Username == username))
    {
        args.IsValid = false;
    }
}
```

Get the input

Reject the username if it has been used by any of the users (both **Member** and **Admin**)

- Double-click on **btnRegister**. Program the **btnRegister_Clicked** event handler to insert a new **Member** account:

```
protected void btnRegister_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string username = txtUsername.Text;
        string password = txtPassword.Text;
        string name = txtName.Text;

        // Insert new member account
        Member m = new Member
        {
            Username = username,
            Hash = Security.GetHash(password),
            Name = name
        };
        db.Members.InsertOnSubmit(m);
        db.SubmitChanges();

        Response.Redirect("Successful.aspx");
    }
}
```

Compute the password hash for storing in the database

Redirect user to the [**Successful.aspx**] page

- Open the [**Successful.aspx**] page:

Successful

Your registration is successful
You will be redirected to [Login] page after 3 seconds

- Switch to **Source View**. Write the following HTML and JavaScript codes to redirect the user to [**Login.aspx**] after 3 seconds:

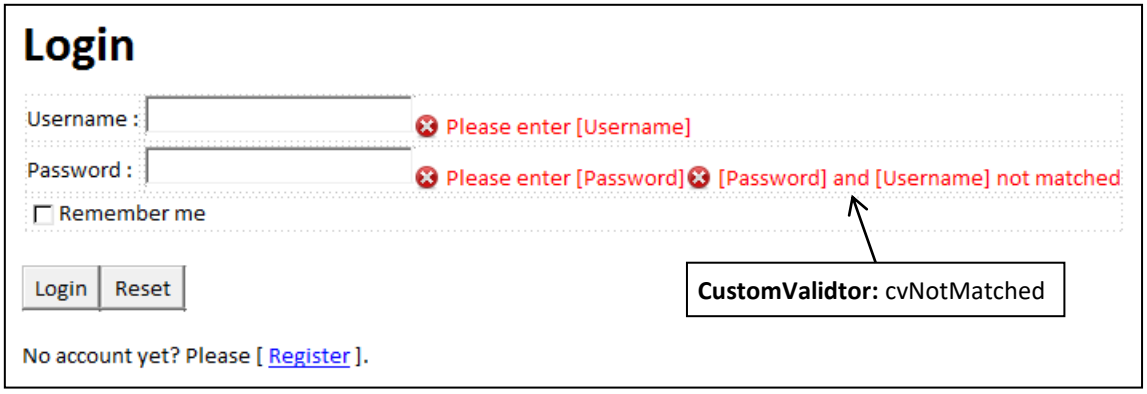
```
<!-- JavaScript -->
<script>
    setTimeout(" location = 'Login.aspx' ", 3000);
</script>
```

Perform the JavaScript codes (the 1st parameter) after 3000 milliseconds (3 seconds)

- Test the result. Try to register a new **Member** account. Once register successfully, you should be redirected to the [**Successful.aspx**] page. After 3 seconds, you will be redirected again to the [**Login.aspx**] page.

Q8. Programming the Login Page

- Open the [**Login.aspx**] page:



- Add the following additional **using** statement:

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

- Double-click on **btnLogin**. Program the **btnLogin_Clicked** event handler to login the user if the login credentials are matched. Otherwise, display the relevant error message:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string username = txtUsername.Text;
        string password = txtPassword.Text;
        bool rememberMe = chkRememberMe.Checked;

        // Login the user
        User u = db.Users.SingleOrDefault(
                    x => x.Username == username &&
                         x.Hash == Security.GetHash(password)
                );

        if (u != null)
        {
            FormsAuthentication.RedirectFromLoginPage(u.Username, rememberMe);
        }
        else
        {
            cvNotMatched.IsValid = false;
        }
    }
}
```

Use LINQ query to select the user record with matching username and password

Compute the password hash before comparison

If a matching user record is found, login the user

Otherwise, display the error message

NOTE: Unlike typical circumstances, the **CustomValidator** named **cvNotMatched** is not attached with any server-side validation code. This is because our login logic has already performed the necessary login credentials checking. Thus, the **CustomValidator** is mainly used for displaying the error message only in our scenario here.

Q9. Programming the Logout Page

- Open the [**Logout.aspx**] page:

Logout

You have been logout

- Add the following additional **using** statement:

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

- Program the **Page_Load** event handler to logout the user if he is an authenticated user. Force the browser to reload the page after logout:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Logout the user
    if (User.Identity.IsAuthenticated)
    {
        FormsAuthentication.SignOut();
        Response.Redirect("Logout.aspx");
    }
}
```

If it is an authenticated user

Logout the user and reload the page

- Test the result. Try to login by using any of the **Member** or **Admin** accounts available in the database. Passwords for all the default accounts are [**password**]. After login, you should be able to access to the [**Protected.aspx**] page. Try to logout also.

- However, you still unable to access to the pages under the [**MemberOnly**] and [**AdminOnly**] folders even if you login with the relevant **Member** or **Admin** account. This is because forms authentication does not role-based authorization by default.

Q10. Enabling Role-based Authorization

- Open the [**Security.cs**] C# class file.

- Add the following additional **using** statements:

```
using System.Security.Principal;
using System.Threading;
using System.Web.Security;
```

- Open the data file named [**Code Snippets.txt**] as given to you. Copy all the codes in the data file and paste them within the **Security** class block. You should now have 2 additional public static methods in the class: **LoginUser** and **ProcessRoles**.

- Open the [**Login.aspx**] page. Modify the **btnLogin_Clicked** event handler as shown below:

```
// Login the user
User u = db.Users.SingleOrDefault(
             x => x.Username == username &&
                  x.Hash == Security.GetHash(password)
         );

if (u != null)
{
    //FormsAuthentication.RedirectFromLoginPage(u.Username, rememberMe);
    Security.LoginUser(u.Username, u.Role, rememberMe);
}
else
{
    cvNotMatched.IsValid = false;
}
```

Remove or comment out the original login function

Replace the original login function with this call to the **LoginUser** method method

- In **Solution Explorer**, add the [**Global.asax**] file under the root folder.

- Add the following **Application_PostAuthenticateRequest** event handler to the file:

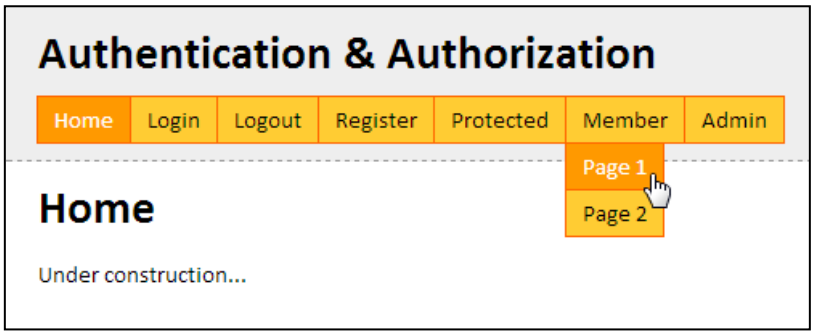
```
protected void Application_PostAuthenticateRequest(object sender, EventArgs e)
{
    Security.ProcessRoles();
}
```

NOTE: The event handler is not in the [**Global.asax**] file by default. You will have to add it manually (you can copy-and-paste other event handler and modify accordingly).

- Test the result. Try to login by using any of the **Member** or **Admin** accounts available in the database. Passwords for all the default accounts are [**password**]. After login, you should be able to access to the pages under the [**MemberOnly**] or [**AdminOnly**] folder now (depending if you are using **Member** or **Admin** account).

Q11. Enabling Sitemap Security Trimming

- Currently, the menu displays all menu items regardless if the user has access to them. We want to show ONLY menu items that are accessible by the current user:



- In **Solution Explorer**, open the [**Web.config**] file under the root folder.

- Add a new **<siteMap>** element under the hierarchy [**configuration** > **system.web**]. Type the XML codes as shown below:

```
<configuration>
    ...
    <system.web>
        ...
        <!-- Sitemap -->
        <siteMap defaultProvider="sitemap1">
            <providers>
                <add name="sitemap1"
                     siteMapFile="Web.sitemap"
                     type="System.Web.XmlSiteMapProvider"
                     securityTrimmingEnabled="true" />
            </providers>
        </siteMap>
        ...
    </system.web>
    ...
</configuration>
```

Indicate the default sitemap provider is **sitemap1** (which is to be added below)

A name for the sitemap provider (or settings)

Get sitemap data from the default [**Web.sitemap**] file

Enable sitemap security trimming

Type of the sitemap provider

- Within the same [**Web.config**] file, locate the **<location>** element we have created earlier. Add 3 more **<location>** elements to refine our authorization rules:

```
<configuration>
    ...
    <system.web>
        ...
    </system.web>
    ...
    <!-- Authorization -->
    <location path="Protected.aspx">
        <system.web>
            <authorization>
                <deny users="?" />
            </authorization>
        </system.web>
    </location>

    <location path="Logout.aspx">
        <system.web>
            <authorization>
                <deny users="?" />
            </authorization>
        </system.web>
    </location>
```

```
    <location path="Login.aspx">
        <system.web>
            <authorization>
                <allow users="?" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>

    <location path="Register.aspx">
        <system.web>
            <authorization>
                <allow users="?" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    ...
</configuration>
```

- Open the [**Logout.aspx**] page. Modify the **Page_Load** event handler as shown below:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Logout the user
    if (User.Identity.IsAuthenticated)
    {
        FormsAuthentication.SignOut();
        //Response.Redirect("Logout.aspx");
        Response.Redirect("Home.aspx");
    }
}
```

Remove or comment out this original line

We redirect the user to [**Home.aspx**] after logout (as now **Anonymous** users can longer access to the logout page)

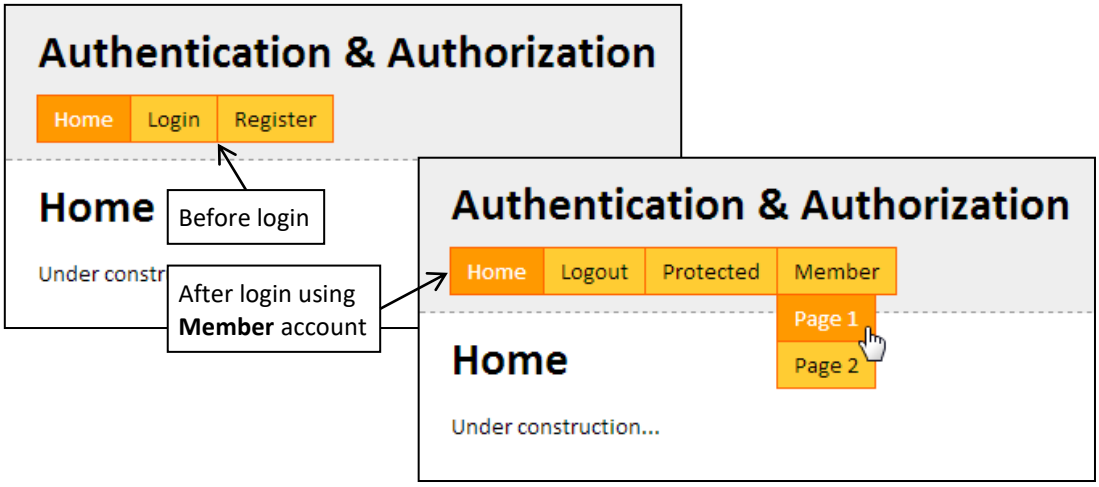
- Open the [**Web.sitemap**] XML file. Add the **roles** attribute to the **Member** and **Admin** sitemap nodes:

```
<siteMapNode title="Home" url="Home.aspx">
    <siteMapNode title="Login" url="Login.aspx" />
    <siteMapNode title="Logout" url="Logout.aspx" />
    <siteMapNode title="Register" url="Register.aspx" />
    <siteMapNode title="Protected" url="Protected.aspx" />

    <siteMapNode title="Member" roles="Member">
        <siteMapNode title="Page 1" url="MemberOnly/Page1.aspx" />
        <siteMapNode title="Page 2" url="MemberOnly/Page2.aspx" />
    </siteMapNode>

    <siteMapNode title="Admin" roles="Admin">
        <siteMapNode title="Page 1" url="AdminOnly/Page1.aspx" />
        <siteMapNode title="Page 2" url="AdminOnly/Page2.aspx" />
    </siteMapNode>
</siteMapNode>
```

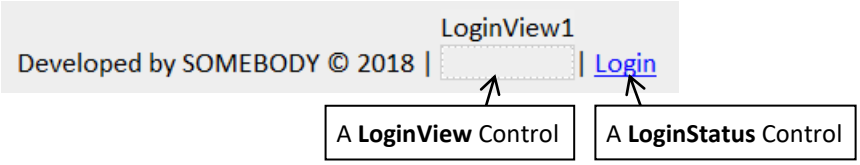
- Test the result. The menu should now display (or hide) menu items depending on the user login status and role:



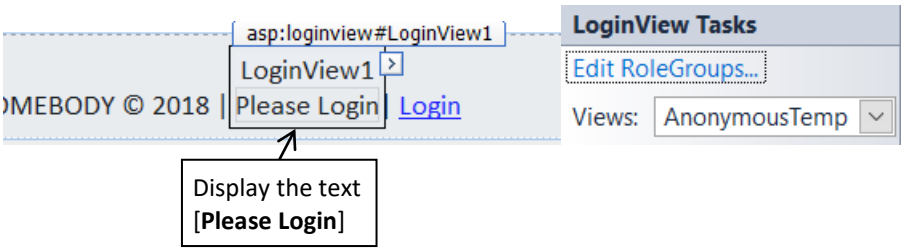
Q12. Using Login Controls

- Open the master page [**Site.Master**].

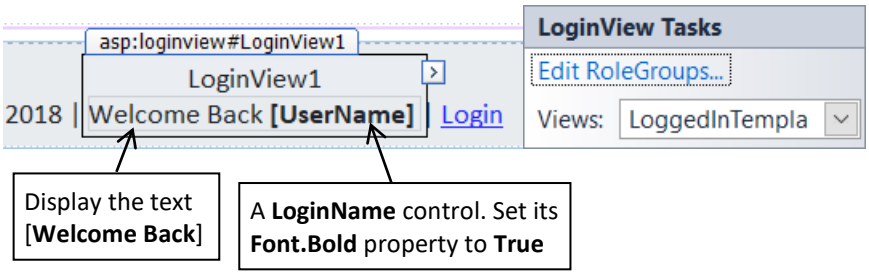
- Drag-and-drop a **LoginView** control and a **LoginStatus** control from the **Toolbox** to the footer section of the master page as shown below:



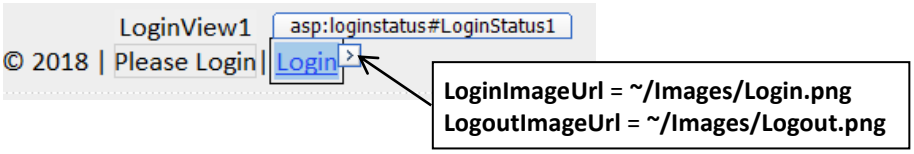
- Create the following contents in the **AnonymousTemplate** of the **LoginView** control:



- Create the following contents in the **LoggedInTemplate** of the **LoginView** Control:



- Select the **LoginStatus** control. Modify its properties as follows:



- Test the result. Try to login and examine the result at the footer section of the page: