

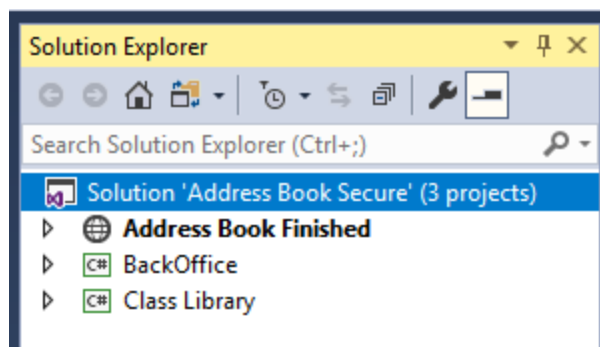
## Using clSecurity

This class has been created to try and take some of the headache out of creating your own security code. The design allows for multiple users with functionality for a web front-end and a windows desktop back-office. There is the option for assigning administrators within the system but there are some limitations on this implementation! Firstly users are either administrators or simple users; there are no multiple levels of authorisation. The design will need extending to allow for this. Management of accounts is limited in some cases this will require direct editing of the data in the table. There is no differentiation between staff and customers. A person who signs up via the web interface automatically has an account on the back office application.

### Overview of Features

Perhaps the best place to start is to take a look at the final implementation and see what the class has to offer. You will need to download and extract Address Book Secure.

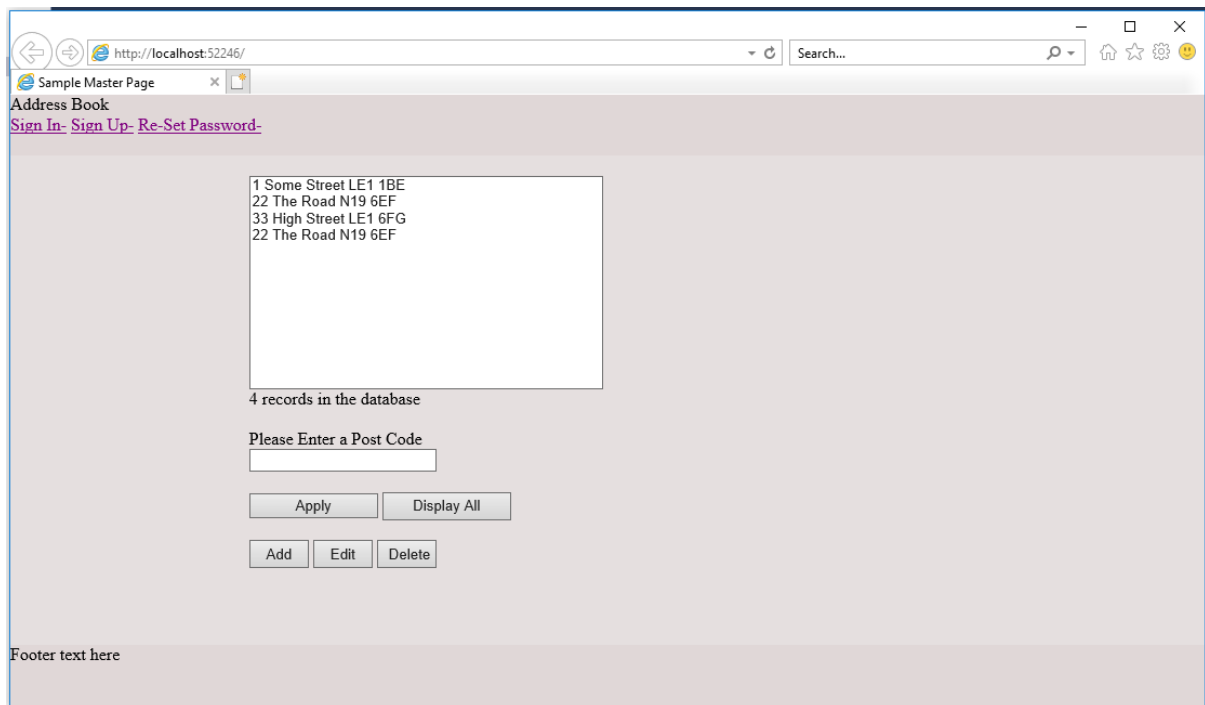
Once started in Visual Studio take a look at the configuration...



Within the solution there are three projects.

- Address Book Finished which is the web based front end
- BackOffice the staff facing backend admin system running as a windows desktop application
- Class Library containing all of the middle layer code

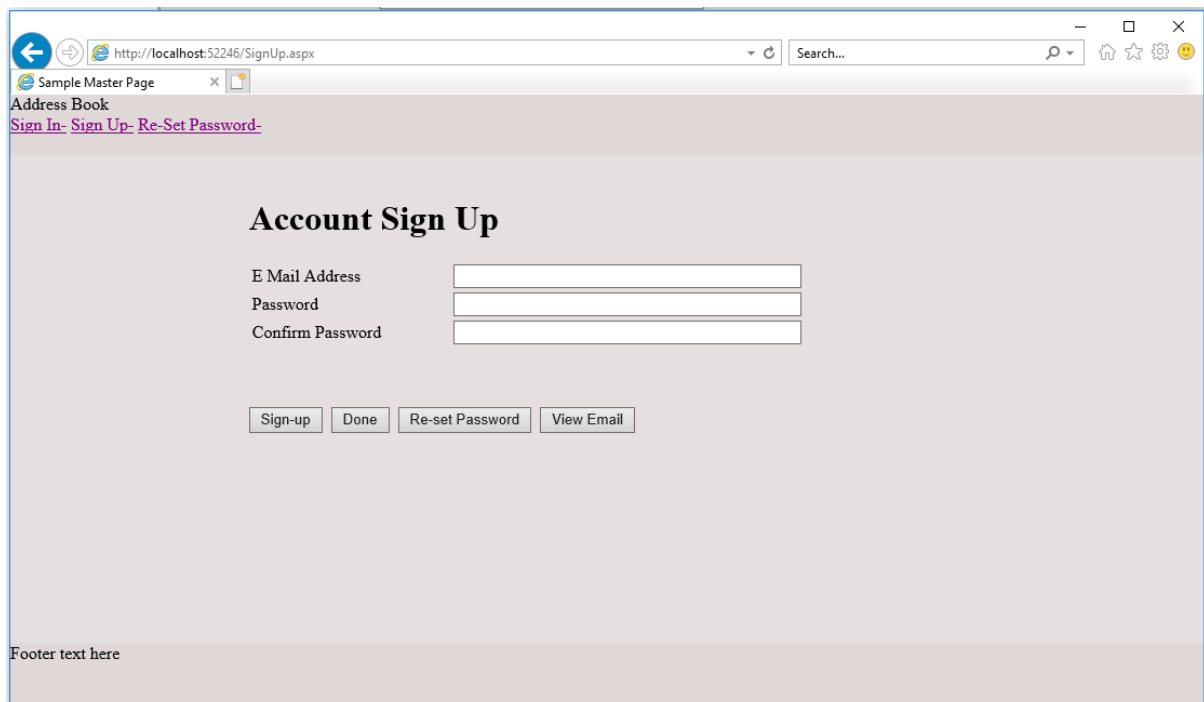
Out of the box the front end should be the start up project – run the program to see the main screen.



There are probably more features in the implementation than we shall cover here but first things to note are the following...

- The web application incorporates a master page and external CSS to provide consistent layout across all pages
- There is at this point no authorisation as such every user sees the same data regardless of who they are
- The system does change based on authentication. Note that the links at the top left will change as soon as somebody has logged in
- The interface isn't terribly pretty and needs a good tidy up!

To use the system you will need to create an account, so click on the link for "Sign-up". This should take you to the following screen...




Before doing anything, press sign-up whilst leaving the field blank!

Note that the class already incorporates a good set of validation messages for you to use in your own application.

A close-up view of the 'Account Sign Up' form. The form has a light gray background. At the top, there is a heading 'Account Sign Up'. Below the heading, there are three input fields with labels 'E Mail Address', 'Password', and 'Confirm Password' to their left. Below these fields, there is a validation message: 'Your password must be at least 7 characters your password must contain a number'. At the bottom of the form, there are four buttons: 'Sign-up', 'Done', 'Re-set Password', and 'View Email'.

To sign up for an account you will need to enter your email address and type your password twice like so...

# Account Sign Up

E Mail Address	<input type="text" value="fred@fred.com"/>
Password	<input type="password" value="....."/>
Confirm Password	<input type="password" value="....."/> 

Your password must be at least 7 characters your password must contain a number

[Sign-up](#)[Done](#)[Re-set Password](#)[View Email](#)

Press Sign-up again and the account should be created for you.

# Account Sign Up

E Mail Address	<input type="text" value="fred@fred.com"/>
Password	<input type="password"/>
Confirm Password	<input type="password"/>

An email has been sent to your account allowing you to activate the account

[Sign-up](#)[Done](#)[Re-set Password](#)[View Email](#)

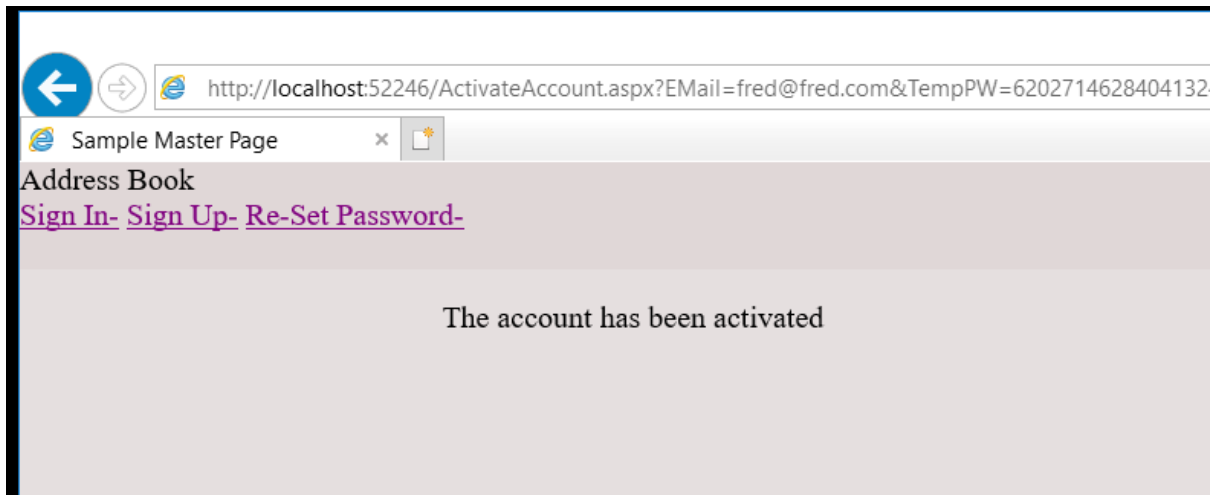
One issue with web based sign up is that we don't want people to sign up to our site with fake email addresses. The system is able to send verification emails to the user in order to activate the account. If this process is not followed then the user will not be able to sign in.

There is a problem though. Without setting this up on a real server it can be a bit of a problem closing the loop between sign-in, email confirmation and final verification, to fix this we have added the "View Email" button.

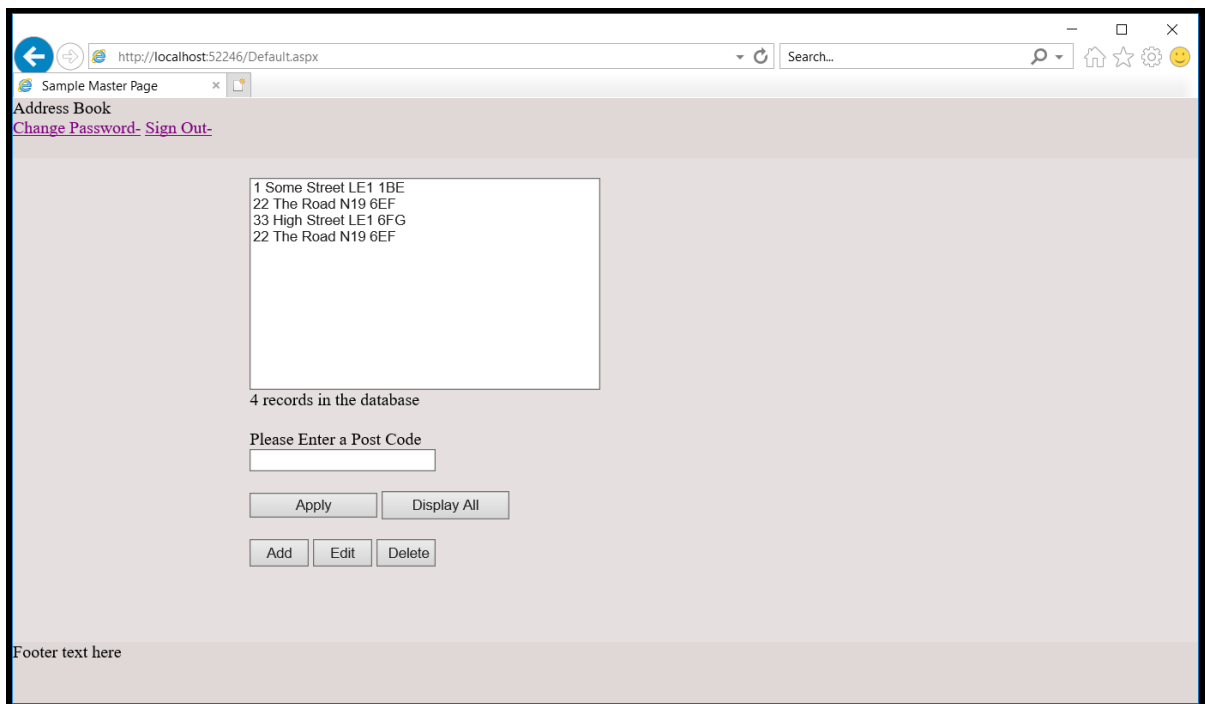
Press View-Email and you will be taken to a page which displays the email that would have been sent to your account...

To: fred@fred.com  
From: noreply@dmu.ac.uk  
Subject: Instructions for activating your account  
Body: [Activate Account](#)

In order to complete the activation process, click the Activate Account link...



Now you should be able to sign-in without any problem...



The system now recognises the user as authenticated and modifies the range of options available on the top left of the screen.

We are now able to...

- Change Password
- Sign Out

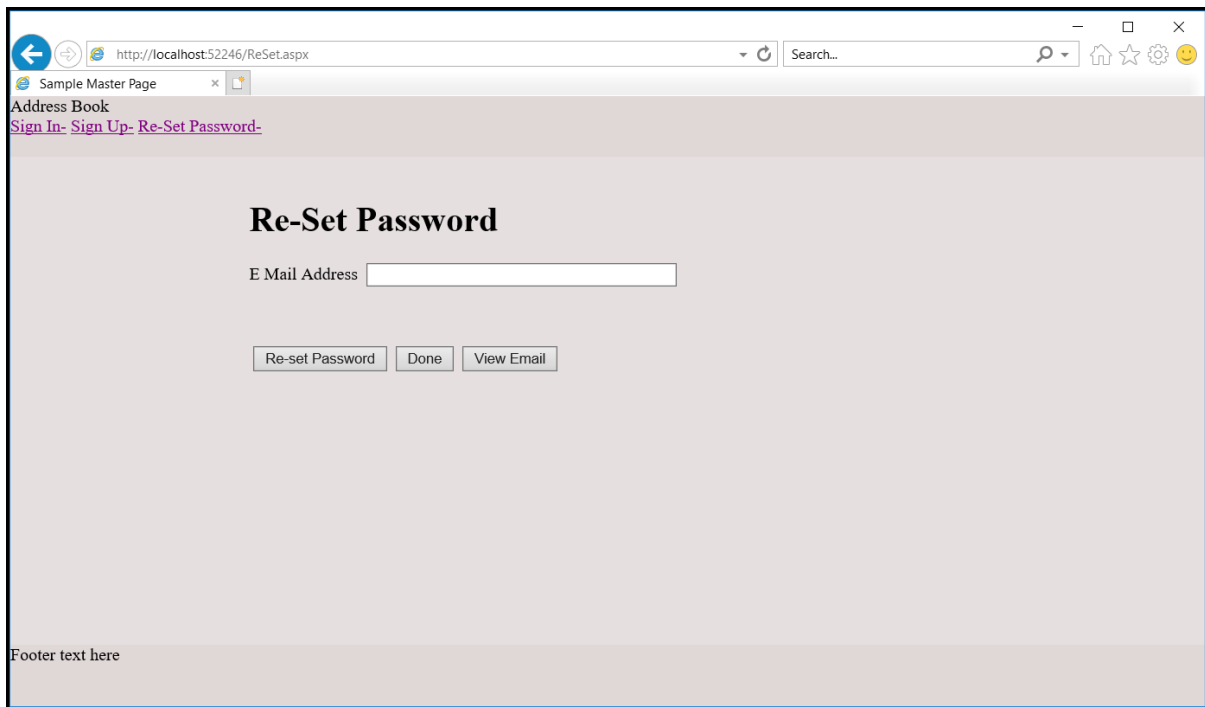
Change password takes you to the following screen which should hold no surprises...

The screenshot shows a web browser window with the URL `http://localhost:52246/ChangePassword.aspx`. The page has a header with a search bar and navigation icons. Below the header, there's a navigation bar with links: [Address Book](#), [Change Password- Sign Out-](#). The main content area is titled "Change Password" and contains three input fields labeled "Current Password", "Password", and "Confirm Password". Below these fields are two buttons: "Change Password" and "Done". The footer area contains the text "Footer text here".

Sign-out will take you back to the first unauthenticated main page...

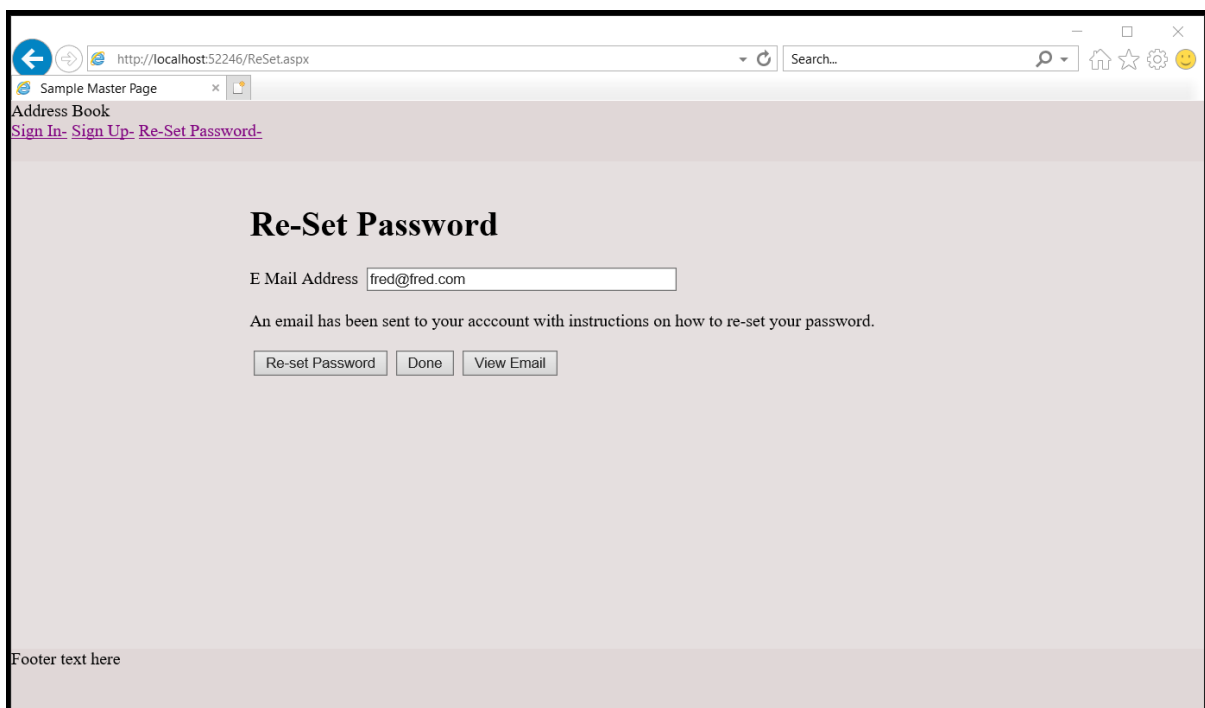
The screenshot shows a web browser window with the URL `http://localhost:52246/Default.aspx`. The page has a header with a search bar and navigation icons. Below the header, there's a navigation bar with links: [Sign In-](#), [Sign Up-](#), [Re-Set Password-](#). The main content area displays a list of addresses in a text box: "1 Some Street LE1 1BE", "22 The Road N19 6EF", "33 High Street LE1 6FG", "22 The Road N19 6EF". Below the list, it says "4 records in the database". There's a label "Please Enter a Post Code" followed by an input field. Below this are two buttons: "Apply" and "Display All". At the bottom of the main content area, there are three buttons: "Add", "Edit", and "Delete". The footer area contains the text "Footer text here".

Lastly there is the option to allow a user to re-set their password like so...

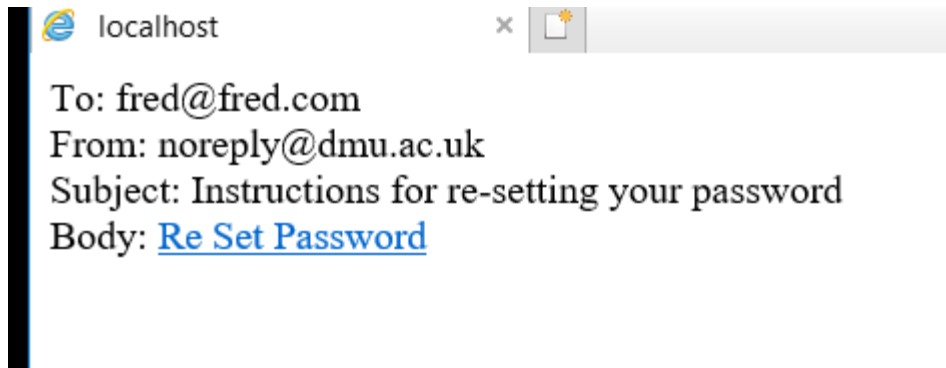


Typing in the email address and pressing re-set will trigger an email containing instruction on how to re-set the password.

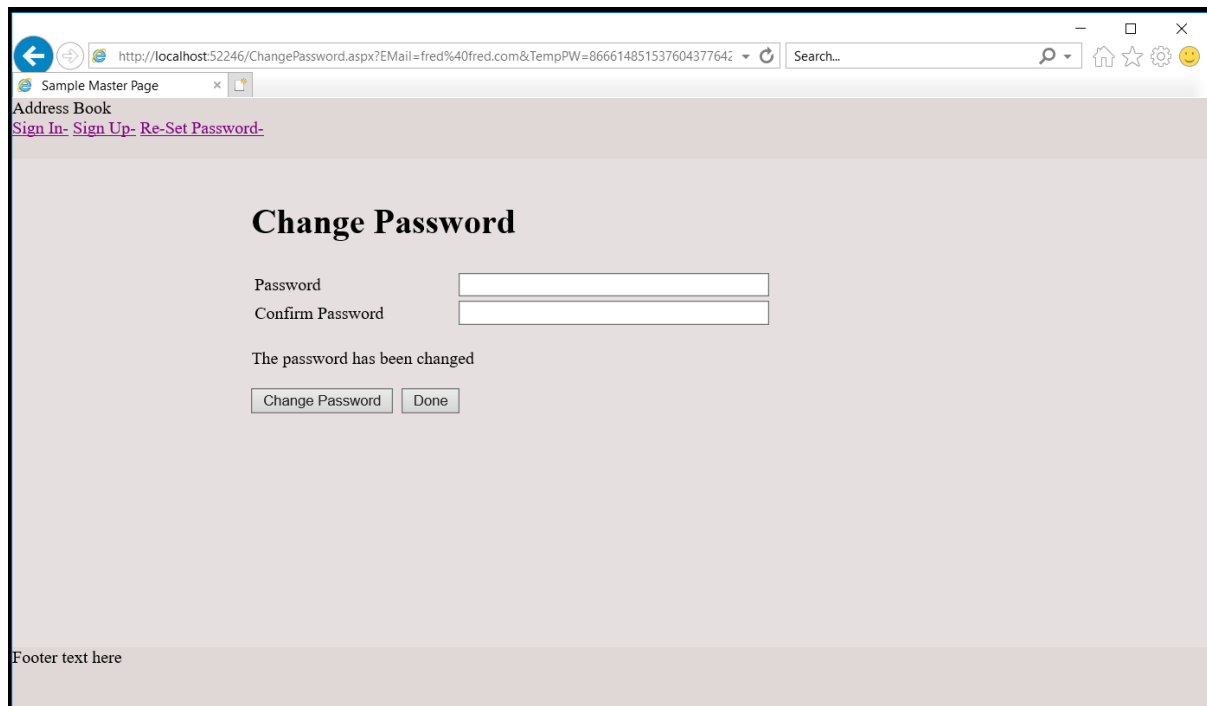
So enter an email address and re-set the password...



As above we need to take into account the limitations on emails, so to view the message click on View Email...



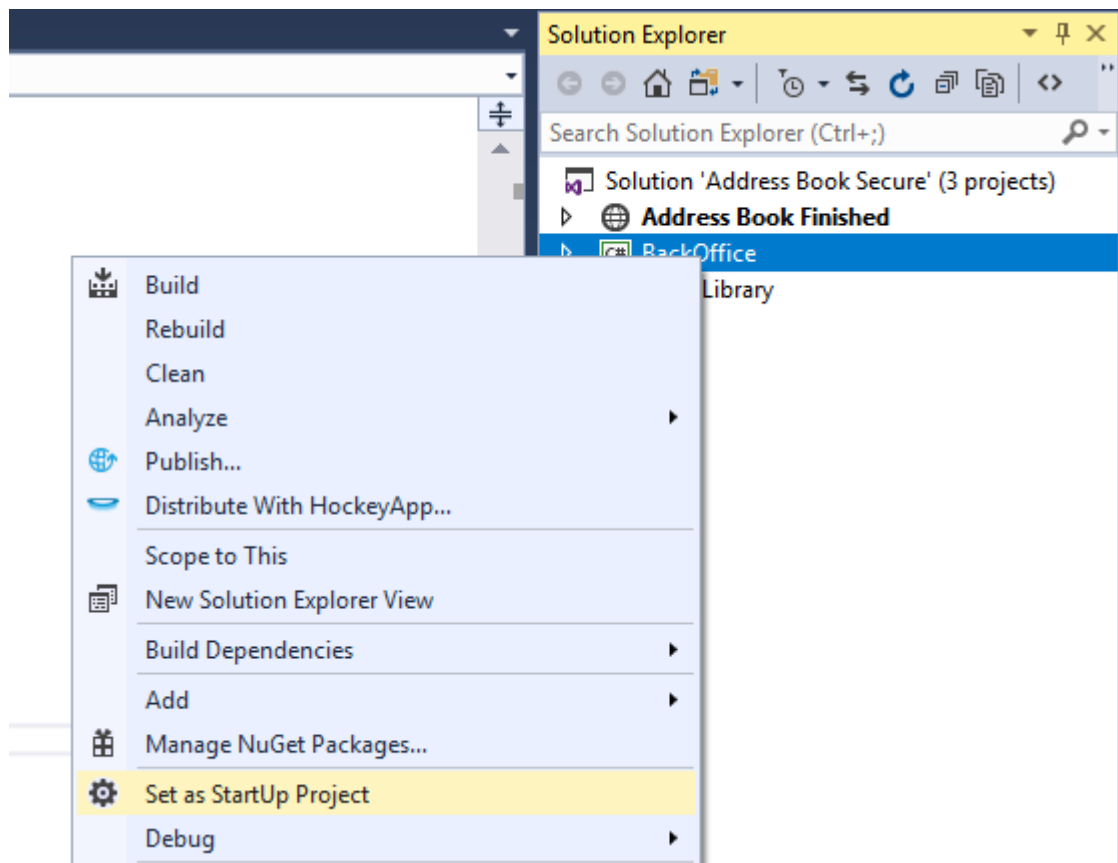
Follow the link to complete the process...



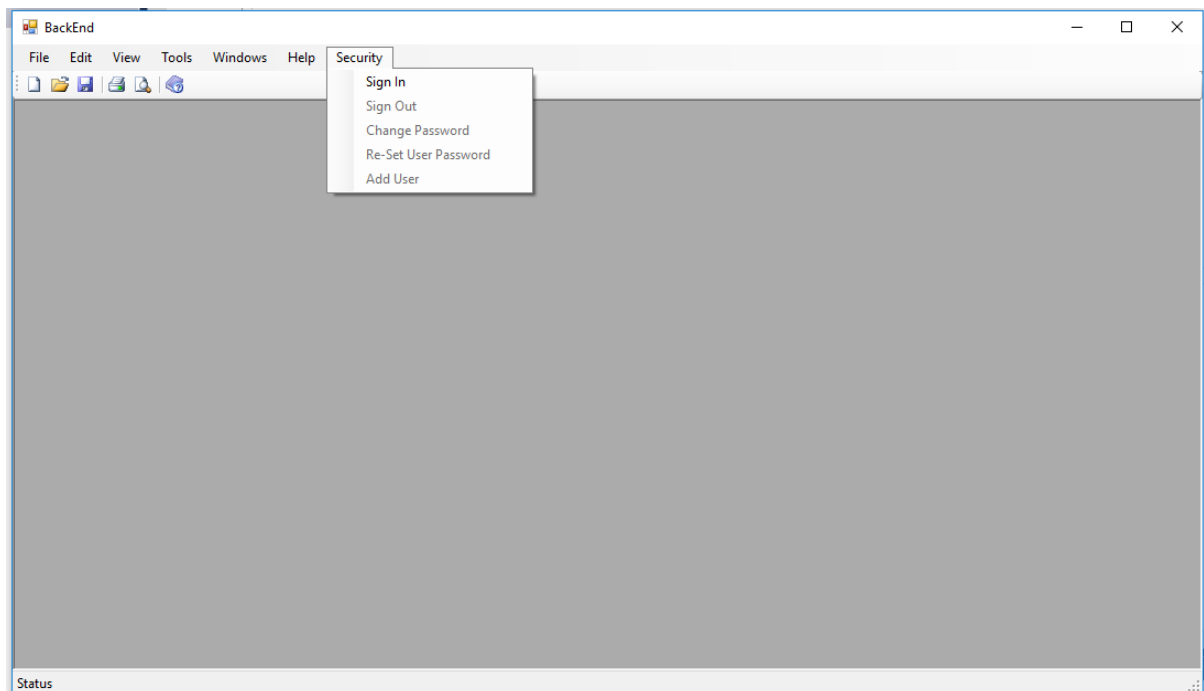
That concludes the overview of the customer facing front end, now we will have a play with the back office.

Right click on the back office and make it the start-up project...





Now run the solution to see what it does...

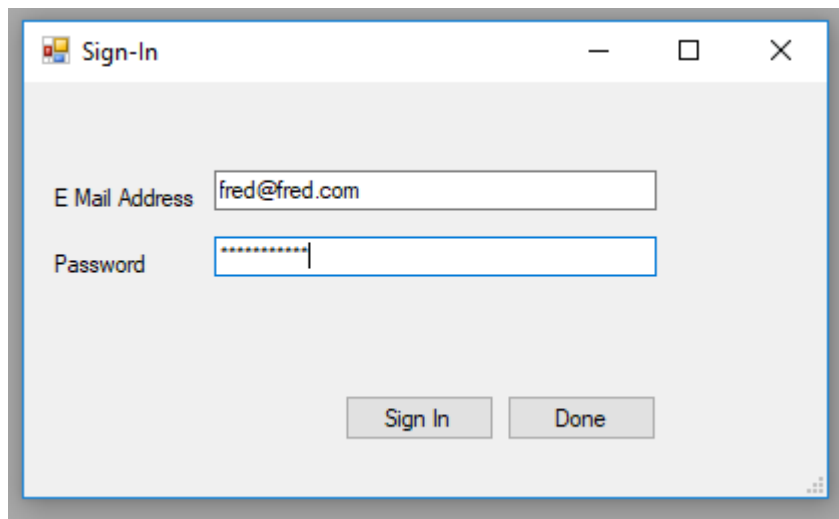


It is worth pointing out at this stage that the business rules / use cases for a back office system are slightly different for a web based front end.

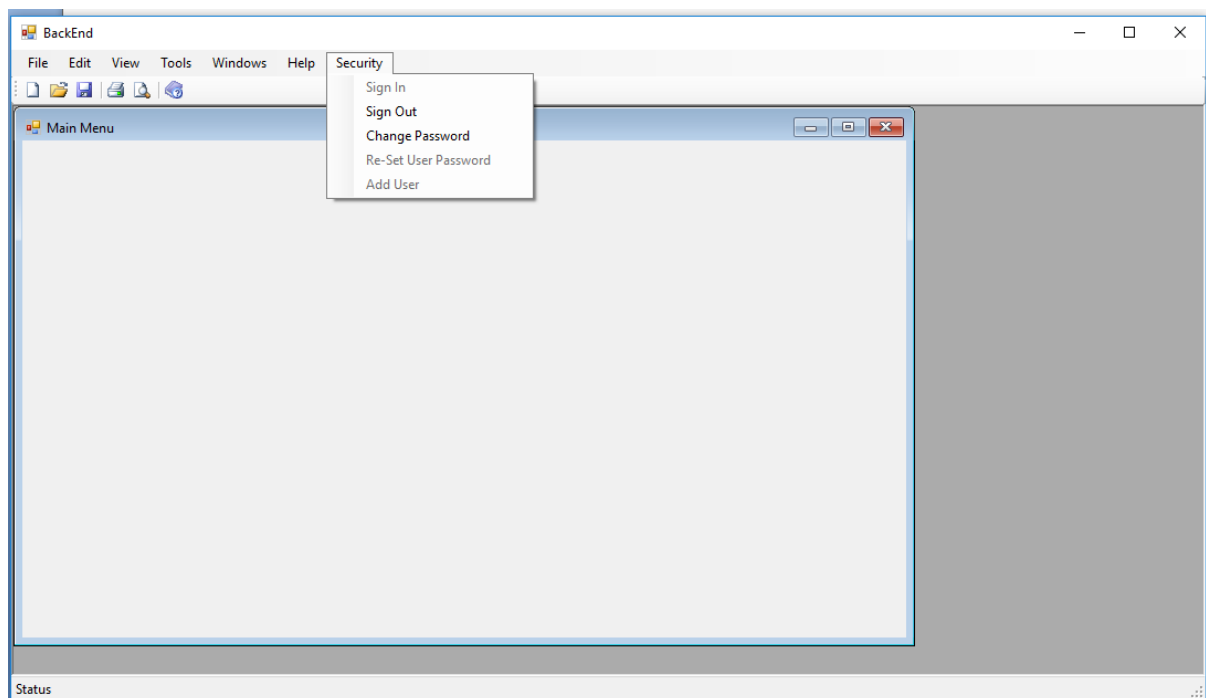
The main difference, is that back office systems don't allow users to sign themselves up to the system; this is handled by the administrator. This being the case it also isn't necessary to include some sort of email verification. In a similar vein if a person forgets their password it is usually down to the admin to fix this rather than allowing the user to do so via an email.

Given that we need to include admin and non admin users on the back end we need to include not just authentication but also some measure of authorisation to control access to admin functions.

From the security menu sign-in as our existing user like so...



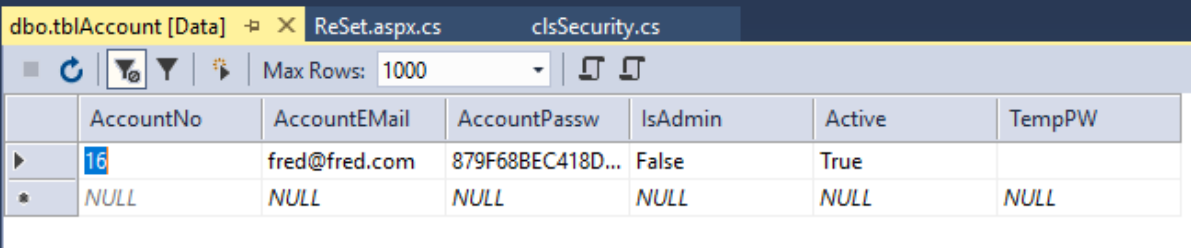
Since this presentation layer uses the same class library as the web interface, they both share the same database. This means that any users added via the front office will also be able to log in via the back office...



You should see the dummy Main Menu pop up but also note that now different options have been activated on the security menu, namely “sign-out” and “change password. Play with these options to see that they work as you expect.

More importantly note that “re-set user password” and “add user” are not enabled as these are both admin functions. To enable these we will need to create an admin user and unfortunately the only way at the moment to do that is to modify the database directly.

Stop the program and open the table tblAccount to view the data...



The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the file explorer with 'dbo.tblAccount [Data]', 'ReSet.aspx.cs', and 'clsSecurity.cs'. The bottom pane shows the data for the 'tblAccount' table. The table has seven columns: AccountNo, AccountEMail, AccountPassw, IsAdmin, Active, and TempPW. The first row shows a user with AccountNo 16, AccountEMail fred@fred.com, AccountPassw 879F68BEC418D..., IsAdmin False, Active True, and TempPW. The second row shows a user with AccountNo NULL, AccountEMail NULL, AccountPassw NULL, IsAdmin NULL, Active NULL, and TempPW NULL.

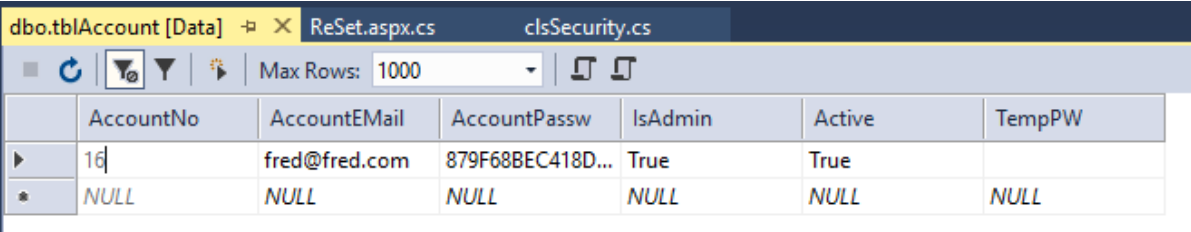
	AccountNo	AccountEMail	AccountPassw	IsAdmin	Active	TempPW
▶	16	fred@fred.com	879F68BEC418D...	False	True	
★	NULL	NULL	NULL	NULL	NULL	NULL

To turn our existing user into an admin change IsAdmin from false to true.

While we are here it may be worth taking a first look at some of the other features of the table.

- AccountEMail is currently used as the user name of the account
- AccountPassword is a salted hash
- IsAdmin needs to be changed manually (until I get round to changing things!)
- Active is used as part of the email activation process and may also be used for disabling an account
- TempPW is used on the web front end as part of re-setting a forgotten password

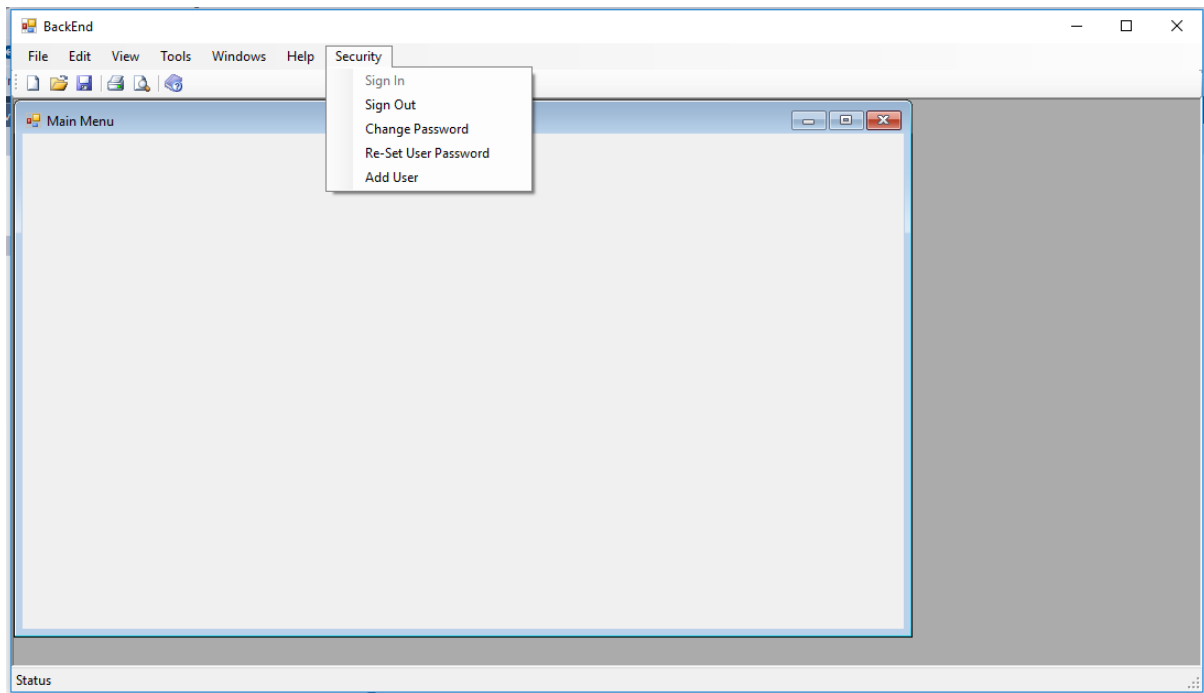
So, assuming you have modified the user to make them admin, like so...



The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the file explorer with 'dbo.tblAccount [Data]', 'ReSet.aspx.cs', and 'clsSecurity.cs'. The bottom pane shows the data for the 'tblAccount' table. The table has seven columns: AccountNo, AccountEMail, AccountPassw, IsAdmin, Active, and TempPW. The first row shows a user with AccountNo 16, AccountEMail fred@fred.com, AccountPassw 879F68BEC418D..., IsAdmin True, Active True, and TempPW. The second row shows a user with AccountNo NULL, AccountEMail NULL, AccountPassw NULL, IsAdmin NULL, Active NULL, and TempPW NULL.

	AccountNo	AccountEMail	AccountPassw	IsAdmin	Active	TempPW
▶	16	fred@fred.com	879F68BEC418D...	True	True	
★	NULL	NULL	NULL	NULL	NULL	NULL

Run the program again, sign in and then see what the menu options look like...



This time due to being admin all of the menu options should be available.

Let's add a new user as admin, so select "Add-User"...

A screenshot of a dialog box titled "Add User". It contains three input fields: "E Mail Address" with the value "bob@fred.com", "Password" with masked characters "\*\*\*\*\*", and "Confirm Password" with masked characters "\*\*\*\*\*". At the bottom of the dialog are two buttons: "Add" and "Done".

It is often the case with back end systems that the admin will add them with the new person physically present, perhaps as a new member of staff. If this is the case then the new user with the administrator's supervision would add the new account themselves. There are potentially other sets of business rules but this is one possible example.

Note that once the account is created there is the option to continue adding staff until the process is completed.

Sign out as admin and now sign in as the new user. Since they are not admin they should only see a limited set of menu options...



So what happens if they forget their password? In this case rather than re-setting it themselves they need to ask the admin to do it for them.

Sign back in as the admin and access “re-set user password”...



This displays the following screen...

A screenshot of a 'Change Password' dialog box. The dialog has a title bar with a minimize, maximize, and close button. It contains three input fields: 'E Mail Address', 'New Password', and 'Confirm Password'. Below the fields are two buttons: 'Change' and 'Done'.

Since I am the admin I am able to re-set the password for any user on the system.

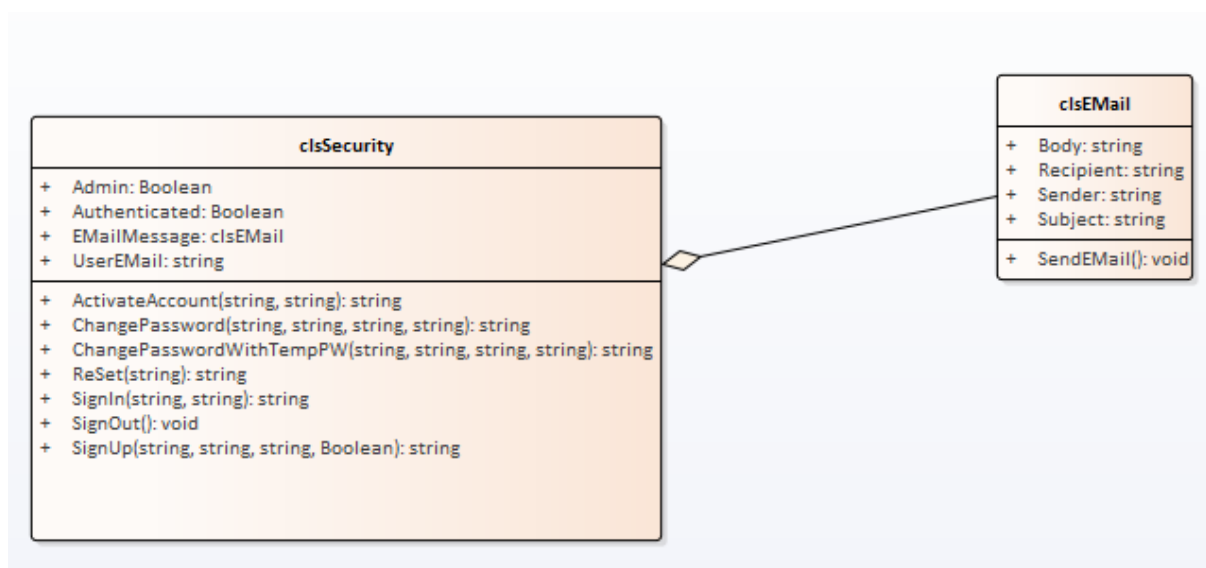
Compare this with the screen when I want to change my own password...

Note that I can only change the password for my own account whereas an admin can do this for all accounts.

We have now looked at the functionality available for both customer facing front end and staff facing back end.

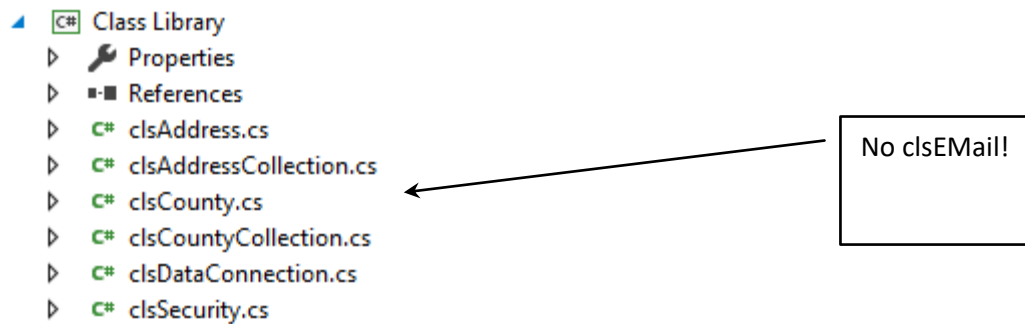
## clsSecurity Internal Design

As a class diagram we have the following to consider...



## clsEmail

Given that I have only given you the code for clsSecurity it is a fair question to ask what about the email class?



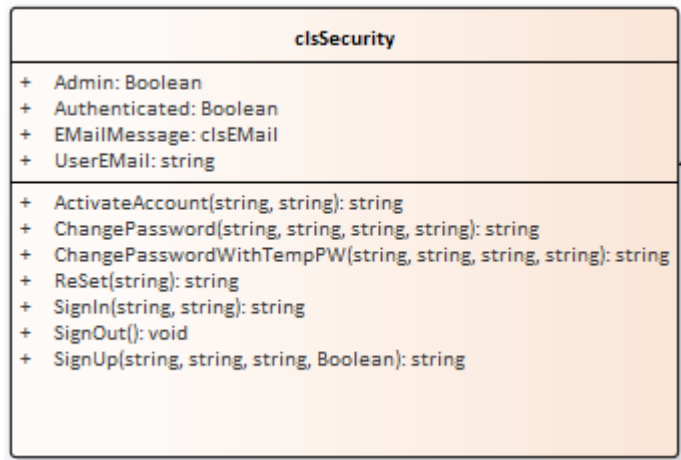
This is a little naughty but clsEMail is actually embedded in clsSecurity. Take a look at the code for clsSecurity and you will see the class definition at the top...

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Security.Cryptography;
6
7 /// <summary>
8 /// basic security class - free to use so long as you credit the author i.e. me
9 /// Matthew Dean mjdean@dmu.ac.uk
10 /// last update 17/8/2018
11 /// </summary>
12 public class clsSecurity
13 {
14     public class clsEMail
15     {
16         ///this class is internal to clsSecurity just to make it simpler to use
17         ///you may want to consider using this definition to create your own class
18         ///within your class library and then get rid of this definition
19         public string Subject { get; internal set; }
20         public string Recipient { get; internal set; }
21         public string Sender { get; internal set; }
22         public string Body { get; internal set; }
23
24         public void SendEmail()
25         {
26             //sends an email
27             //this function won't work till all the code is activated
28             System.Net.Mail.MailMessage eMail = new System.Net.Mail.MailMessage();
29             System.Net.Mail.SmtpClient Server = new System.Net.Mail.SmtpClient("mail.dmu.ac.uk");
30             eMail = new System.Net.Mail.MailMessage(this.Sender, this.Recipient, this.Subject, this.Body);
31             //Server.Send(eMail);
32         }
33     }
34 }
35
```

This embedded class comes free of charge and is designed like this to make including the features simpler. If you want at some point to take the class definition out of here and give it its own file, please feel free to do so.

clsEMail is used to represent an email, provide basic send functionality and is also used on the view email feature we looked at in the first examples above.

## clsSecurity



This class is the one that handles all of the security features we have seen so far.

There are the following attributes to make use of...

- Admin indicates true if the authenticated user is an administrator
- Authenticated is used to see if the current user is in fact signed in
- EMailMessage exposes the last sent email message
- UserEMail allows access to the email address of the current user which is also their user ID in this design

There are also the following operations along with a brief description...

**Activate Account:** Used as part of the email activation process, accepts the user email and a temporary password as parameters

**ChangePassword:** Accepts the email address and current password for authentication along with two copies of the new password to reduce the risk of typos

**ChangePasswordWithTempPW:** accepts the email address and a system generated temporary password along with two copies of the new password. This function is used when a user needs to re-set their password on the web

**ReSet:** Sends out an email containing a temporary password to the email address specified as a parameter

**SignIn:** accepts the email address and password of the user to be authenticated setting the classes authenticated attribute to true if successful

**SignOut:** Sets the class to an non-authenticated state

**SignUp:** Used to create a new user on the system. Parameters required are email address, duplicate password and a Boolean active parameter. The active parameter determines if an activation email is sent to the new user. In the case of web based sign up active should be set to false to force a



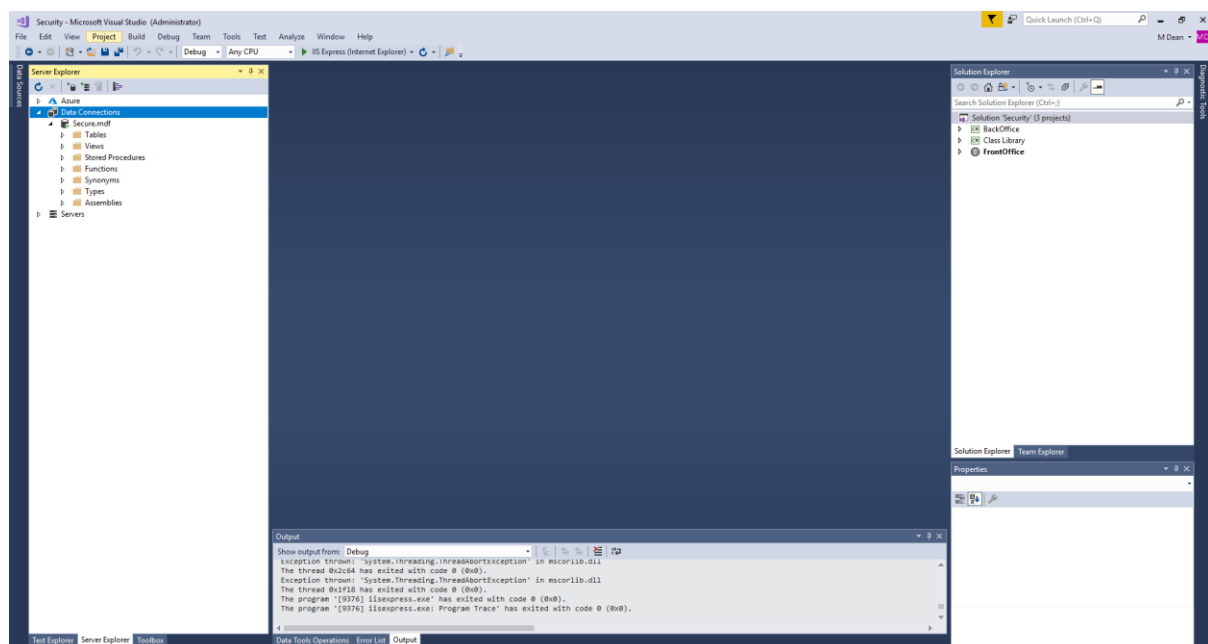
verification email. In the case of a back office sign up this may be set to true to bypass this step of the process.

Below is a screen shot of all attributes / parameters along with their data types...

◆ ActivateAccount	Email: string, TempPW: string	string	Public
◆ ChangePassword	Email: string, CurrentPassword: string, Password1: string, Password2: string	string	Public
◆ ChangePasswordWithTempPW	Email: string, TempPW: string, Password1: string, Password2: string	string	Public
◆ ReSet	Email: string	string	Public
◆ SignIn	Email: string, Password: string	string	Public
◆ SignOut		void	Public
◆ SignUp	Email: string, Password1: string, Password2: string, Active: Boolean	string	Public

## Implementing clsSecurity

Rather than using the Address Book as your starting point for adding security to your system I have also provided a more stripped down version of the system called Security. Unzip the file and look at the contents in Visual Studio...

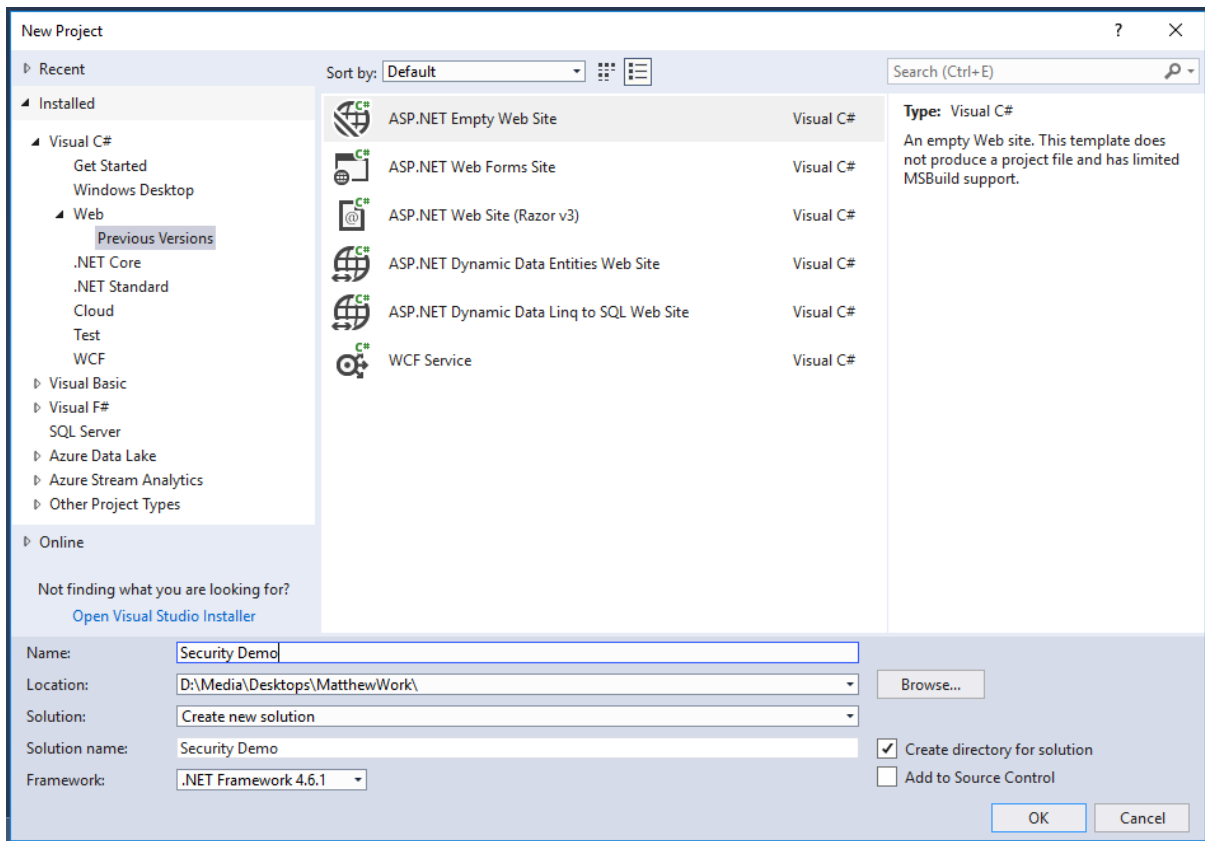


In this solution you have the three projects, BackOffice, FrontOffice and Class Library. This time I have not included the extra features of the address book.

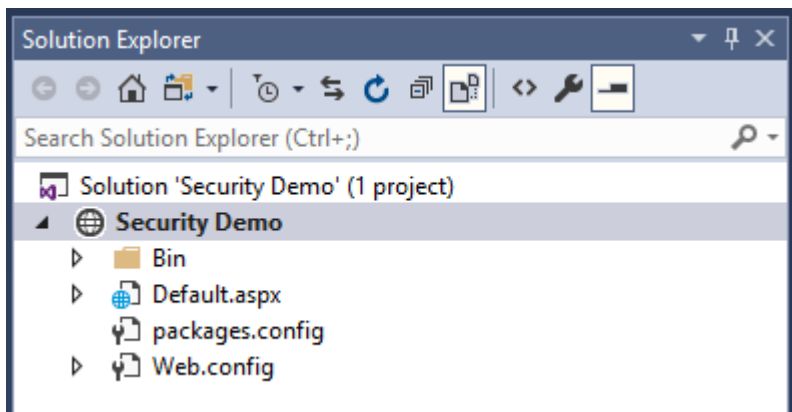
## Adding the Code to a Simple Web Site

For this example we shall add the code to a simple single project web site.

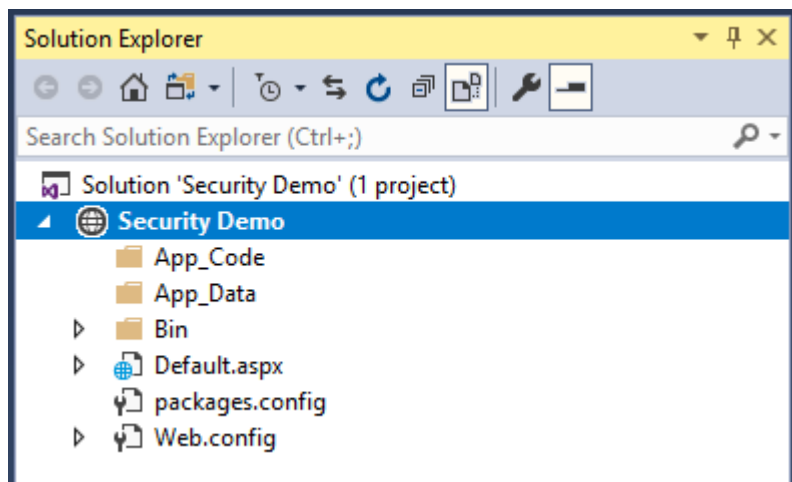
Create a new web site called security demo like so...



Within the web site create the new Default.asp page like so...



Next create the App\_Data and App\_Code folders...



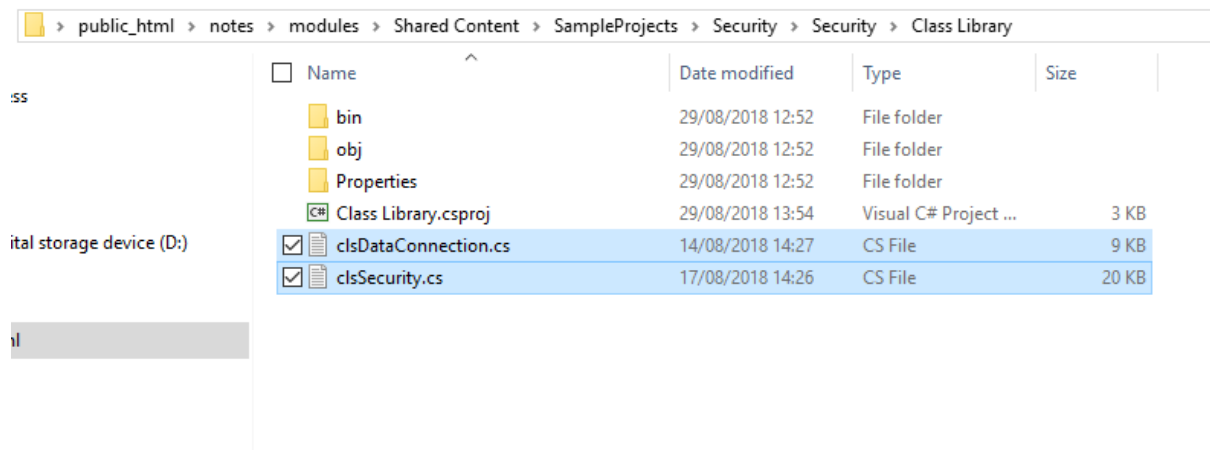
The next step is to copy the required files into our web site using file explorer.

With the contents of the zip file on screen with the folders for your new web site copy the following files from the front office to your new web site...

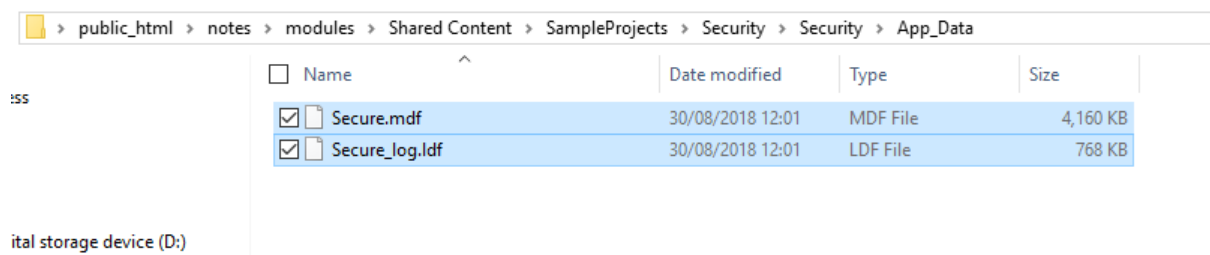
> public_html > notes > modules > Shared Content > SampleProjects > Security > Security > FrontOffice					
<input type="checkbox"/>	Name	Date modified	Type	Size	
	Bin	29/08/2018 12:52	File folder		
<input checked="" type="checkbox"/>	ActivateAccount.aspx	30/08/2018 12:10	ASP.NET Server Pa	1 KB	
<input checked="" type="checkbox"/>	ActivateAccount.aspx.cs	30/08/2018 12:10	CS File	1 KB	
<input checked="" type="checkbox"/>	ChangePassword.aspx	30/08/2018 12:12	ASP.NET Server Pa	3 KB	
<input checked="" type="checkbox"/>	ChangePassword.aspx.cs	30/08/2018 12:12	CS File	4 KB	
	Default.aspx	29/08/2018 14:00	ASP.NET Server Pa	2 KB	
	Default.aspx.cs	30/08/2018 12:07	CS File	2 KB	
<input checked="" type="checkbox"/>	EMailViewer.aspx	17/08/2018 11:28	ASP.NET Server Pa	1 KB	
<input checked="" type="checkbox"/>	EMailViewer.aspx.cs	30/08/2018 11:15	CS File	1 KB	
	packages.config	10/08/2018 12:38	CONFIG File	1 KB	
<input checked="" type="checkbox"/>	ReSet.aspx	30/08/2018 12:14	ASP.NET Server Pa	1 KB	
<input checked="" type="checkbox"/>	ReSet.aspx.cs	30/08/2018 12:14	CS File	2 KB	
<input checked="" type="checkbox"/>	SignIn.aspx	30/08/2018 12:16	ASP.NET Server Pa	2 KB	
<input checked="" type="checkbox"/>	SignIn.aspx.cs	30/08/2018 12:16	CS File	2 KB	
<input checked="" type="checkbox"/>	SignOut.aspx	30/08/2018 12:18	ASP.NET Server Pa	1 KB	
<input checked="" type="checkbox"/>	SignOut.aspx.cs	30/08/2018 12:18	CS File	1 KB	
<input checked="" type="checkbox"/>	SignUp.aspx	30/08/2018 12:19	ASP.NET Server Pa	3 KB	
<input checked="" type="checkbox"/>	SignUp.aspx.cs	30/08/2018 12:19	CS File	2 KB	
	Web.config	10/08/2018 12:42	CONFIG File	1 KB	

(Don't copy the page Default.aspx or it will overwrite the one you have just created)

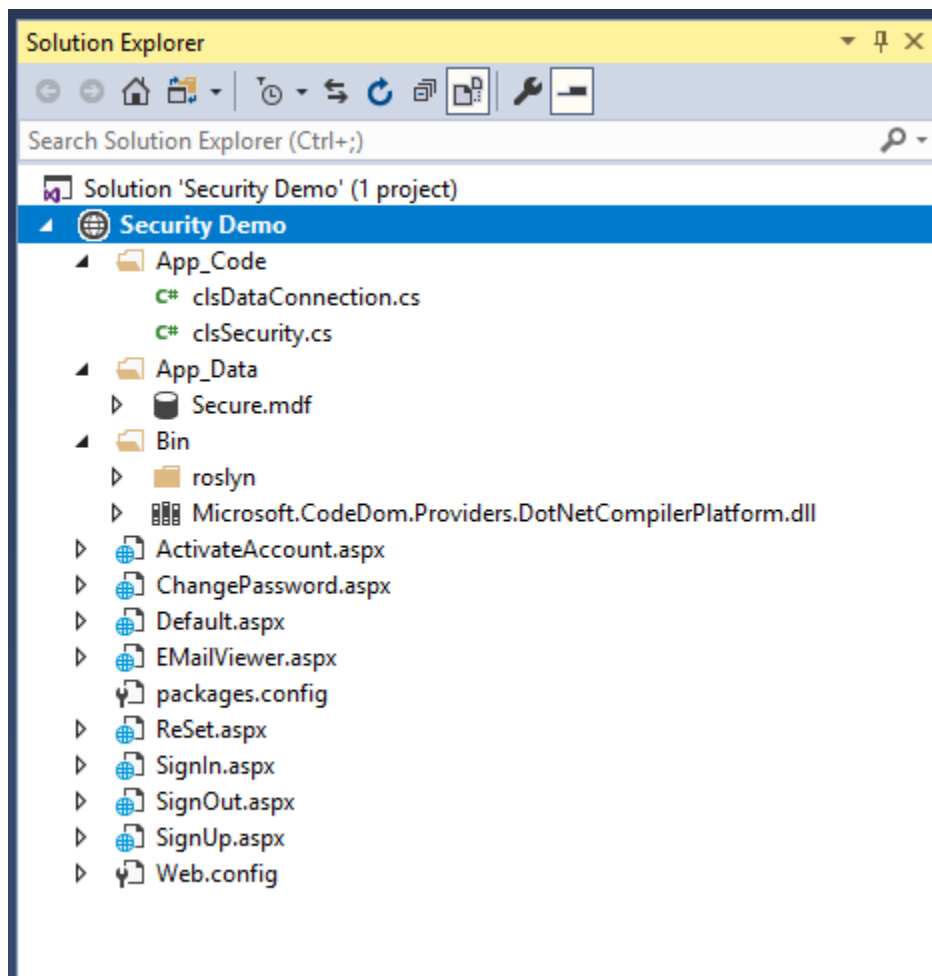
Next copy the classes clsSecurity and clsDataConnection (if you don't already have it) from the Class Library folder to your App\_Code folder...



Lastly copy the database file to your App\_Data folder...



If this has been done correctly then you will see the following in Visual Studio once you refresh the web site contents...



Now run the web site to make sure there are no errors.

## Adding the Code to Default.aspx

The code for implementing basic security is mostly handled by the class.

The presentation layer code is pretty simple.

Create a basic series of buttons on your ASPX page using the following code...

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html>

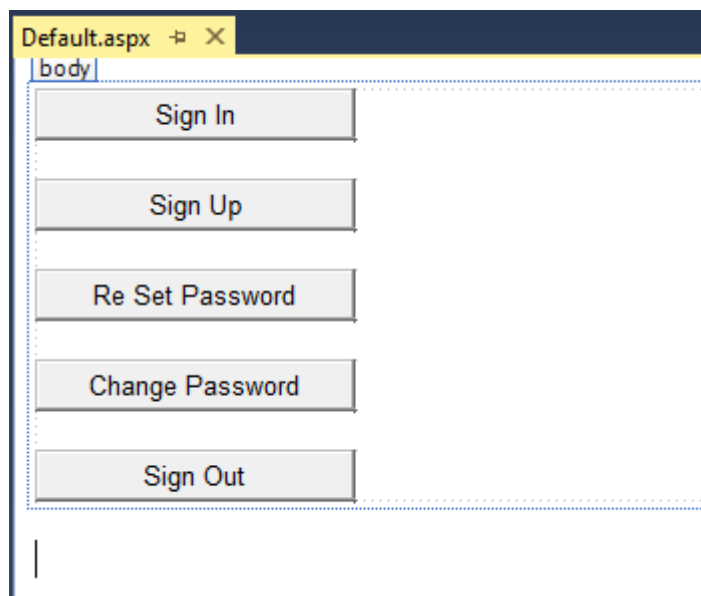
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="btnSignIn" runat="server" OnClick="btnSignIn_Click"
Text="Sign In" width="160px" />
            <br />
            <br />
            <asp:Button ID="btnSignUp" runat="server" OnClick="btnSignUp_Click"
Text="Sign Up" width="160px" />
        </div>
    </form>
</body>
</html>
```

```

        <br />
        <br />
        <asp:Button ID="btnReSet" runat="server" OnClick="btnReSet_Click" Text="Re
Set Password" width="160px" />
        <br />
        <br />
        <asp:Button ID="btnChangePassword" runat="server"
OnClick="btnChangePassword_Click" Text="Change Password" Width="160px" />
        <br />
        <br />
        <asp:Button ID="btnSignOut" runat="server" OnClick="btnSignOut_Click"
Text="Sign Out" width="160px" />
    </div>
</form>
</body>
</html>

```

This will create the following layout...



All you need to do next is to add the appropriate code.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    //create an object based on the security class
    clsSecurity Sec;

    protected void Page_Load(object sender, EventArgs e)
    {
        //get the security data from the session
        Sec = (clsSecurity)Session["Sec"];
        //if it is null then this is the first run of the site
        if (Sec == null)
    }
}

```

```

    {
        //initialise the object
        Sec = new clsSecurity();
        //save it to the session
        Session["Sec"] = Sec;
    }
    //set the state of the menu buttons
    Authenticated(Sec.Authenticated);
}

protected void btnSignIn_Click(object sender, EventArgs e)
{
    //go to sign in
    Response.Redirect("SignIn.aspx");
}

protected void btnSignUp_Click(object sender, EventArgs e)
{
    //go to sign up
    Response.Redirect("SignUp.aspx");
}

protected void btnReSet_Click(object sender, EventArgs e)
{
    //go to re-set
    Response.Redirect("ReSet.aspx");
}

protected void btnChangePassword_Click(object sender, EventArgs e)
{
    //go to change password
    Response.Redirect("ChangePassword.aspx");
}

protected void btnSignOut_Click(object sender, EventArgs e)
{
    //go to sign out
    Response.Redirect("SignOut.aspx");
}

void Authenticated(Boolean Auth)
{
    //sets the visibility of the buttons on the main menu based on authentication state
    //if not logged in then display the following
    btnSignIn.Visible = !Auth;
    btnSignUp.Visible = !Auth;
    btnReSet.Visible = !Auth;
    //if logged in display the following
    btnChangePassword.Visible = Auth;
    btnSignOut.Visible = Auth;
}
}

```

There are a few things to note above.

To use a given feature in a button all you need to do is redirect to the appropriate web form. For example to access the sign in page we do the following...

```

protected void btnSignIn_Click(object sender, EventArgs e)
{
    //go to sign in

```

```

        Response.Redirect("SignIn.aspx");
    }

```

Note at the top of the code what is going on with the object called Sec...

```

public partial class _Default : System.Web.UI.Page
{
    //create an object based on the security class
    clsSecurity Sec;

```

First we create a non instantiated object with page level scope.

The next step is to check this object in the pages load event to see if it is set up correctly, i.e. not null...

```

protected void Page_Load(object sender, EventArgs e)
{
    //get the security data from the session
    Sec = (clsSecurity)Session["Sec"];
    //if it is null then this is the first run of the site
    if (Sec == null)
    {
        //initialise the object
        Sec = new clsSecurity();
        //save it to the session
        Session["Sec"] = Sec;
    }
    //set the state of the menu buttons
    Authenticated(Sec.Authenticated);
}

```

If the object is null then a copy is instantiated and saved to the session object called "Sec" this makes the data available to other web pages.

Finally the load event calls the function Authenticated passing as a parameter the Authenticated state of the object i.e. has the user logged in or not.

The Authenticated function turns buttons on the page on or off depending on if the user is logged in or not...

```

void Authenticated(Boolean Auth)
{
    //sets the visibility of the buttons on the main menu based on authentication state
    //if not logged in then display the following
    btnSignIn.Visible = !Auth;
    btnSignUp.Visible = !Auth;
    btnReSet.Visible = !Auth;
    //if logged in display the following
    btnChangePassword.Visible = Auth;
    btnSignOut.Visible = Auth;
}

```

This function uses a bit of Boolean logic to make it a bit cleaner.



For example the following line...

```
btnSignIn.Visible = !Auth;
```

Is the same as the following code...

```
If (Auth == false)
```

```
{
```

```
    btnSignIn.Visible = true;
```

```
}
```

## Using your own Database

One problem you will face as the design currently stands is when you want to have your own database as well as the security database. As things stand the class clsDataConnection will get upset if you have two databases in your App\_Data folder.

The simplest way to get round this is to create the tables and stored procedures from the security database in your own data file.

To make life slightly simpler I have provided the code to allow you to generate them yourselves (Don't copy and paste the hashes!)

```
#####
```

```
CREATE TABLE [dbo].[tblAccount] (
    [AccountNo]      INT          IDENTITY (1, 1) NOT NULL,
    [AccountEmail]   VARCHAR (50) NULL,
    [AccountPassword] VARCHAR (50) NULL,
    [IsAdmin]        BIT          NULL,
    [Active]         BIT          NULL,
    [TempPW]         VARCHAR (50) NULL,
    PRIMARY KEY CLUSTERED ([AccountNo] ASC)
);
```

```
#####
```

```
CREATE PROCEDURE sproc_tblAccount_UpdateTempPW
    @AccountEmail varchar(50),
    @TempPW varchar(50)
```

```
AS
```

```
    update tblAccount
    set TempPW=@TempPW
```

```
    where AccountEmail=@AccountEmail;
```

```
RETURN 0
```

```
#####
```

```
CREATE PROCEDURE sproc_tblAccount_UpdatePassword
```

```

        @AccountEmail varchar(50),
        @AccountPassword varchar(50)

AS
    update tblAccount
    set AccountPassword=@AccountPassword,
        TempPW=' '

    where AccountEmail=@AccountEmail;

RETURN 0

#####

CREATE PROCEDURE sproc_tblAccount_FilterByEmailAndTempPW
    @AccountEmail varchar(50),
    @TempPw varchar(50)
AS
    SELECT * from tblAccount where AccountEmail=@AccountEmail and TempPw=@TempPw;
RETURN 0

#####

CREATE PROCEDURE sproc_tblAccount_FilterByEmailAndPassword
    @AccountEmail varchar(50),
    @AccountPassword varchar(50)
AS
    SELECT * from tblAccount where AccountEmail=@AccountEmail and
AccountPassword=@AccountPassword and Active=1;
RETURN 0

#####

CREATE PROCEDURE sproc_tblAccount_FilterByEmail
    @AccountEmail varchar(50)

AS
    SELECT * from tblAccount where AccountEmail=@AccountEmail;
RETURN 0

#####

CREATE PROCEDURE sproc_tblAccount_Add
    @AccountEmail varchar(50),
    @AccountPassword varchar(50),
    @Active bit
AS
    insert into tblAccount (AccountEmail, AccountPassword, IsAdmin, Active)
    values (@AccountEmail, @AccountPassword,0, @Active);

RETURN @@Identity

#####

CREATE PROCEDURE sproc_tblAccount_Activate
    @AccountEmail varchar(50)

AS
    update tblAccount
    set Active=1,
        TempPW=' '

```

```
where AccountEMail=@AccountEMail;
```

```
RETURN 0
```

```
#####
```

Once you have created the above don't forget to make sure you only have your database in the App\_Data folder and not the extra security one.