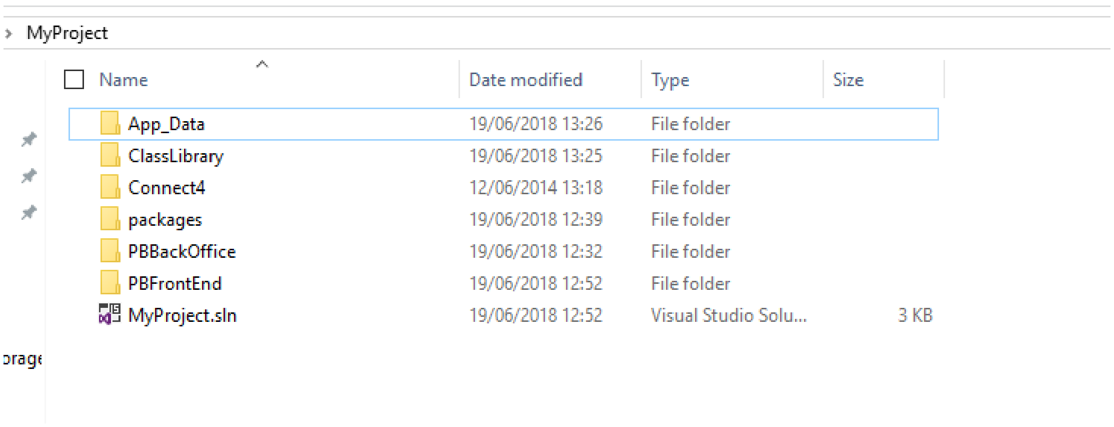
Configuring Visual Studio – Middle and Data Layers

Before we begin the next part of the work we need to think about where we are going with this.

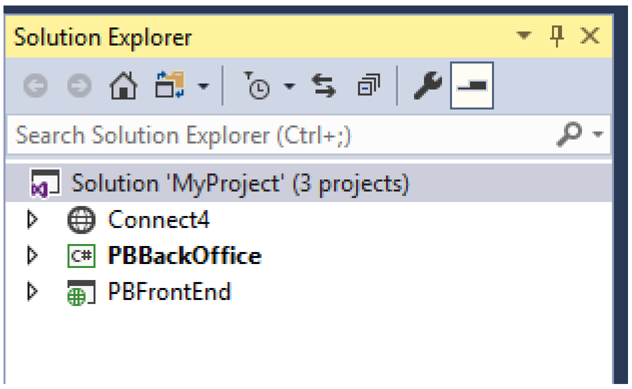
We are hoping to set up the following folder structure for the solution…



› MyProject

Name	Date modified	Type	Size
App_Data	19/06/2018 13:26	File folder	
ClassLibrary	19/06/2018 13:25	File folder	
Connect4	12/06/2014 13:18	File folder	
packages	19/06/2018 12:39	File folder	
PBBackOffice	19/06/2018 12:32	File folder	
PBFrontEnd	19/06/2018 12:52	File folder	
MyProject.sln	19/06/2018 12:52	Visual Studio Solu...	3 KB

So far in Visual Studio we have set up the following projects in the solution.



Solution Explorer

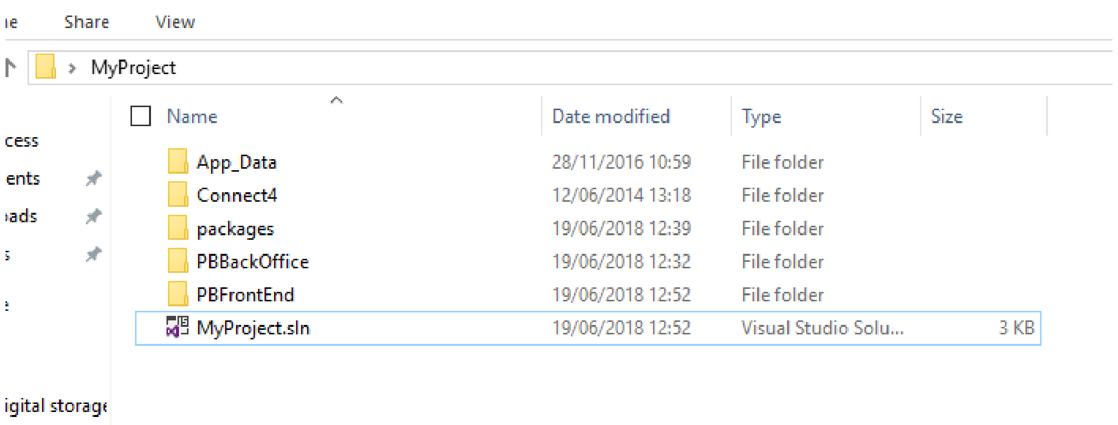
Search Solution Explorer (Ctrl+;)

Solution 'MyProject' (3 projects)
- Connect4
- **PBBackOffice**
- PBFrontEnd

Creating the Data Layer

A simple database file has been provided for you on the module web site.

Download it and extract it into your solution folder so that the folder structure looks like this…



Name	Date modified	Type	Size
App_Data	28/11/2016 10:59	File folder	
Connect4	12/06/2014 13:18	File folder	
packages	19/06/2018 12:39	File folder	
PBBackOffice	19/06/2018 12:32	File folder	
PBFrontEnd	19/06/2018 12:52	File folder	
MyProject.sln	19/06/2018 12:52	Visual Studio Solu...	3 KB

(Make sure at this point you don’t end up with an App_Data folder inside an App_Data folder!)

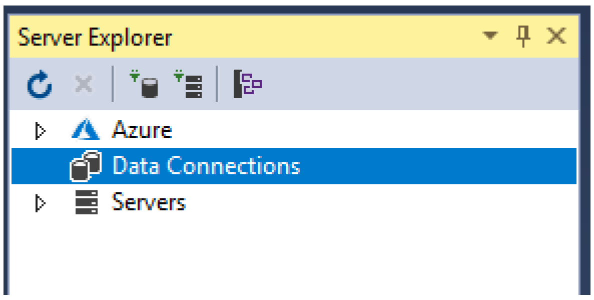
Last year we placed the App_Data folder in the same folder as our website.

For this work placing the database in that location won’t allow us to easily share the database across multiple presentation layers.

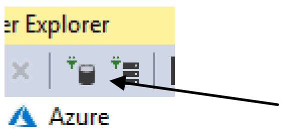
By placing the App_Data folder at this level of the folder structure it may be made available to multiple presentation layers within our solution.

The reality is that in a finished system the database would be hosted on a different computer i.e. a database server. But for this work we get to achieve the same effect by this folder structure.

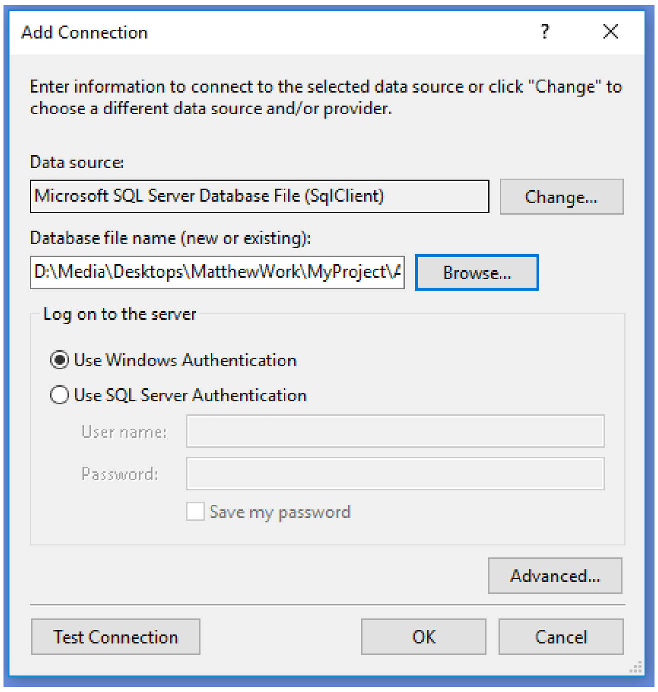
To access the database from within Visual Studio you will need to use Visual Studio’s server explorer.



Press the button to create a new connection…



Complete the options to point at the database file…



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server Database File (SqlClient)

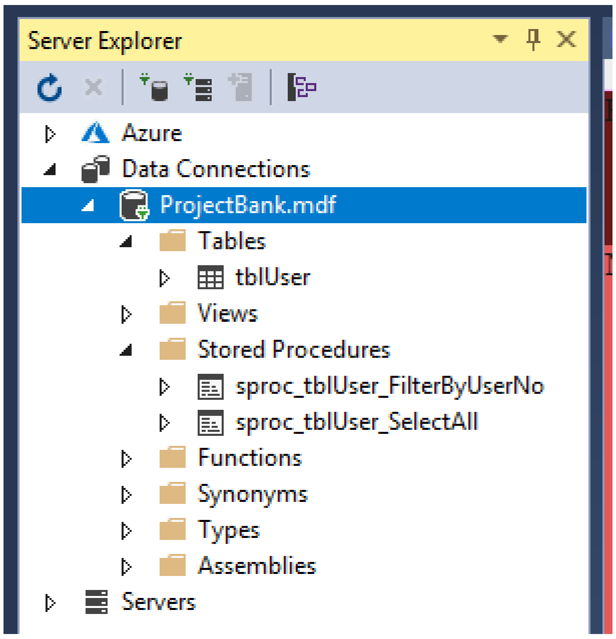
Database file name (new or existing):
D:\Media\Desktops\MatthewWork\MyProject\A

Log on to the server

[● Use Windows Authentication]
○ Use SQL Server Authentication

User name:
Password:
☐ Save my password

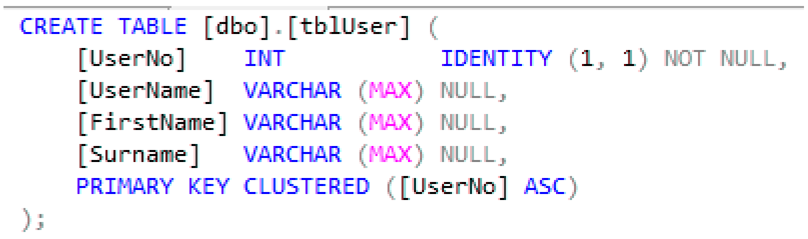
And press OK. You should now be able to access the database structure in the server explorer…



```
Server Explorer
▷ Azure
◢ Data Connections
  ◢ ProjectBank.mdf
    ◢ Tables
      ▷ tblUser
    ▷ Views
    ◢ Stored Procedures
      ▷ sproc_tblUser_FilterByUserNo
      ▷ sproc_tblUser_SelectAll
    ▷ Functions
    ▷ Synonyms
    ▷ Types
    ▷ Assemblies
▷ Servers
```

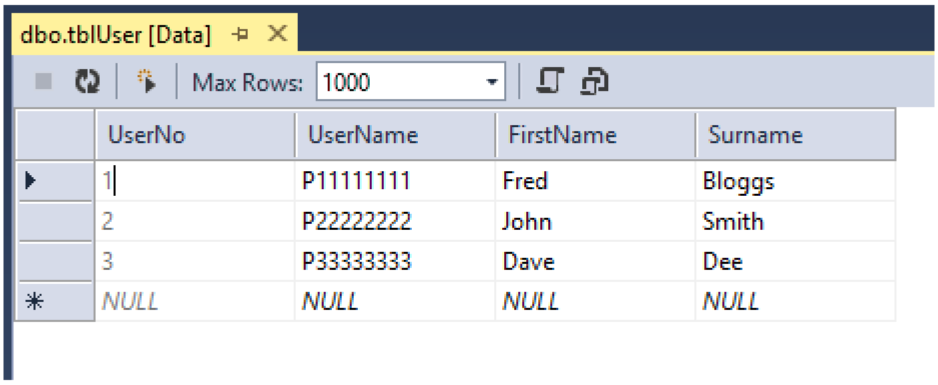
It is a good idea to see what the database contains.

We have a table with the following structure…



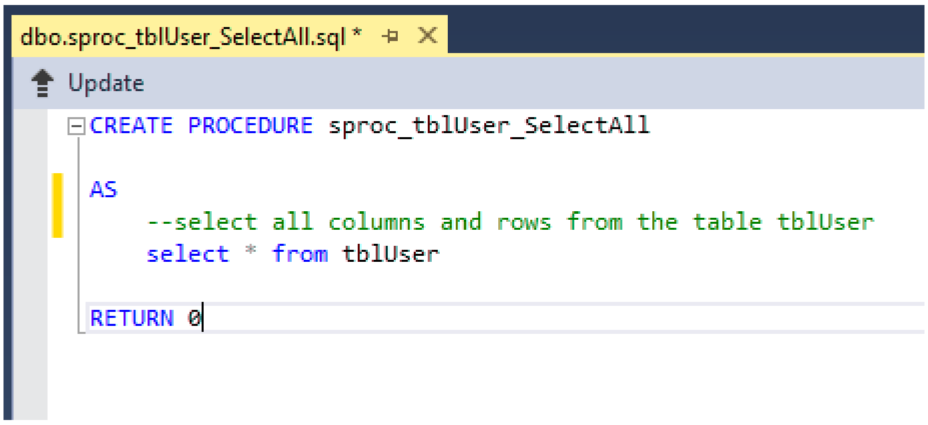
```sql
CREATE TABLE [dbo].[tblUser] (
    [UserNo]     INT           IDENTITY (1, 1) NOT NULL,
    [UserName]  VARCHAR (MAX) NULL,
    [FirstName] VARCHAR (MAX) NULL,
    [Surname]   VARCHAR (MAX) NULL,
    PRIMARY KEY CLUSTERED ([UserNo] ASC)
);
```

And the following data…



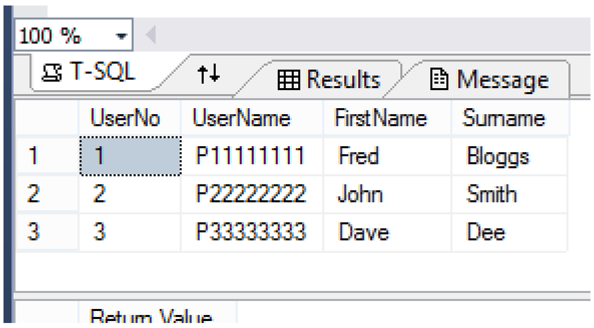
	UserNo	UserName	FirstName	Surname
▶	1	P11111111	Fred	Bloggs
	2	P22222222	John	Smith
	3	P33333333	Dave	Dee
*	NULL	NULL	NULL	NULL

We also have a stored procedure with the following SQL…



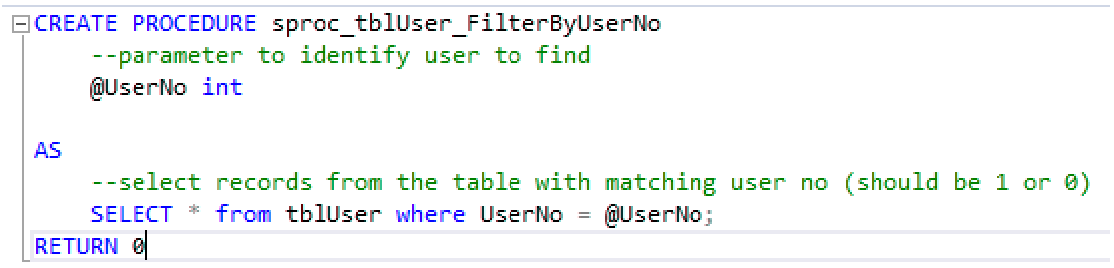
```sql
CREATE PROCEDURE sproc_tblUser_SelectAll

AS
    --select all columns and rows from the table tblUser
    select * from tblUser

RETURN 0
```

Which when executed produces the following list…



	UserNo	UserName	FirstName	Surname
1	1	P11111111	Fred	Bloggs
2	2	P22222222	John	Smith
3	3	P33333333	Dave	Dee

Along with a second stored procedure sproc_tblUser_FilterByUserNo



```
CREATE PROCEDURE sproc_tblUser_FilterByUserNo
    --parameter to identify user to find
    @UserNo int

AS
    --select records from the table with matching user no (should be 1 or 0)
    SELECT * from tblUser where UserNo = @UserNo;
RETURN 0
```

Which requires a parameter allowing us to find specific users.

Creating the Middle Layer – The Class Library

To create the middle layer we are going to create a new project called a ClassLibrary.

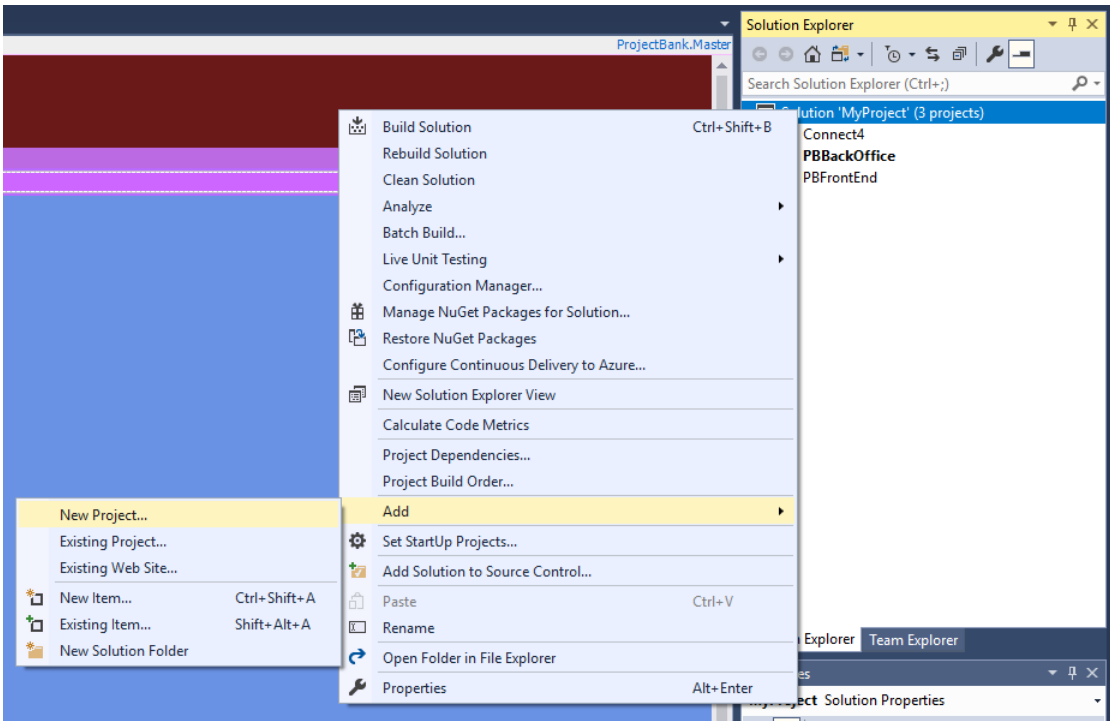
Why?

Having a single class library means that any code we create in one project may be re-used in other projects we create in the future.

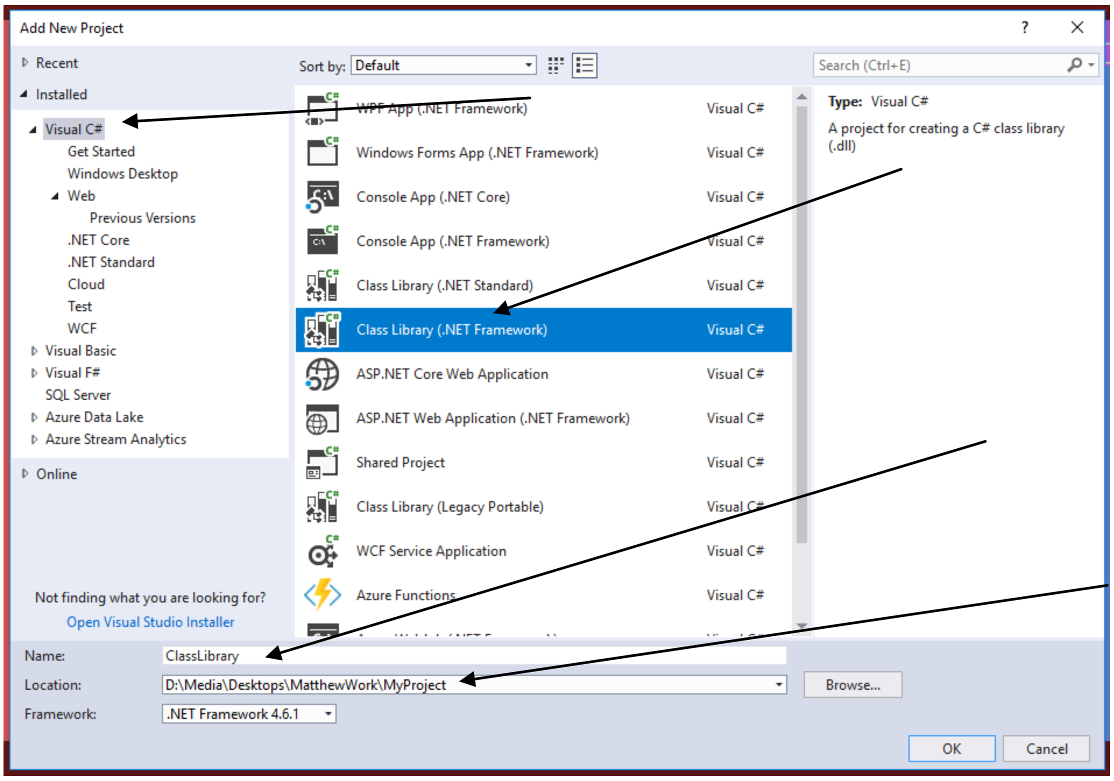
It saves a great deal of repeated work.

When you enter the final year of your degree this class library may be used as a toolbox for work you create as part of your final year project.

To create the class library right click on the solution and add a new project…



Set up the class library project as follows…


Add New Project
Recent
Installed
Visual C#
Get Started
Windows Desktop
Web
Previous Versions
.NET Core
.NET Standard
Cloud
Test
WCF
Visual Basic
Visual F#
SQL Server
Azure Data Lake
Azure Stream Analytics
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
ASP.NET Core Web Application Visual C#
ASP.NET Web Application (.NET Framework) Visual C#
Shared Project Visual C#
Class Library (Legacy Portable) Visual C#
WCF Service Application Visual C#
Azure Functions Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a C# class library (.dll)

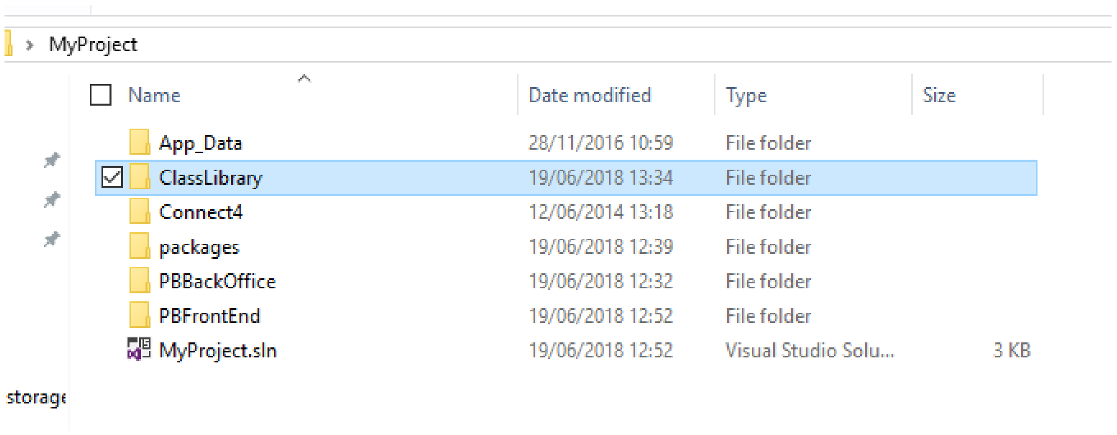
Name: ClassLibrary

Location: D:\Media\Desktops\MatthewWork\MyProject

Framework: .NET Framework 4.6.1

Make sure that the class library is in the same folder as your other projects in the solution.

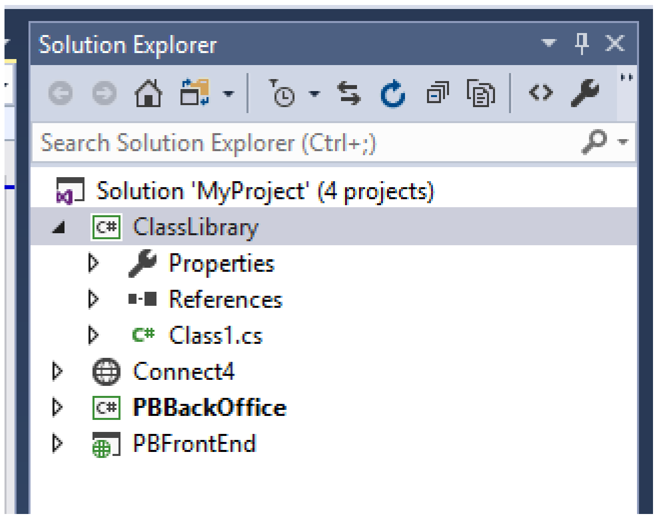
You folder structure should look something like this..



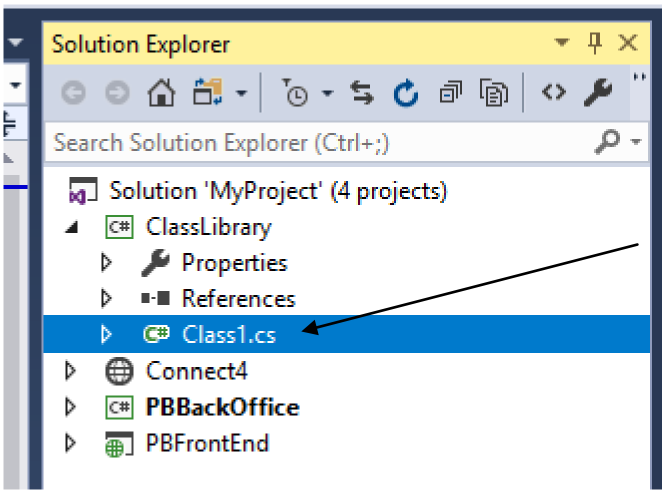
MyProject

Name	Date modified	Type	Size
App_Data	28/11/2016 10:59	File folder	
ClassLibrary	19/06/2018 13:34	File folder	
Connect4	12/06/2014 13:18	File folder	
packages	19/06/2018 12:39	File folder	
PBBackOffice	19/06/2018 12:32	File folder	
PBFrontEnd	19/06/2018 12:52	File folder	
MyProject.sln	19/06/2018 12:52	Visual Studio Solu...	3 KB

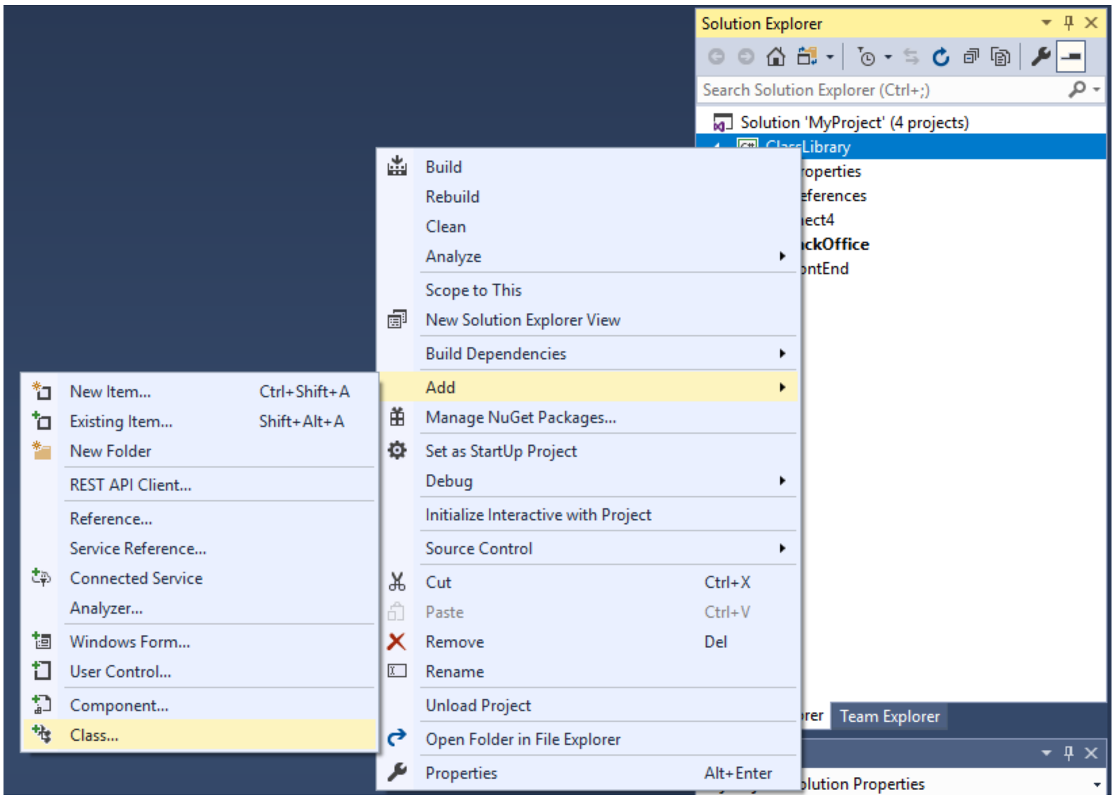
And Visual Studio should look something like this…



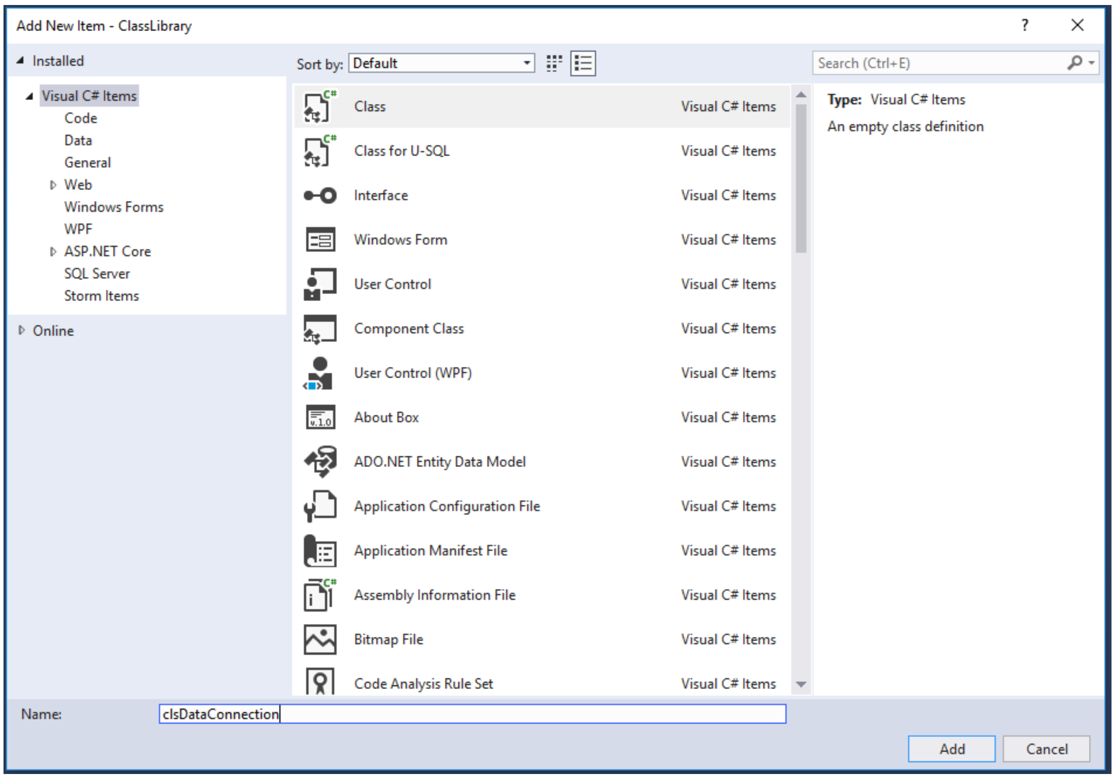
Visual Studio will create a default class template for you. Delete this…



Right click on the class library and add a new class…



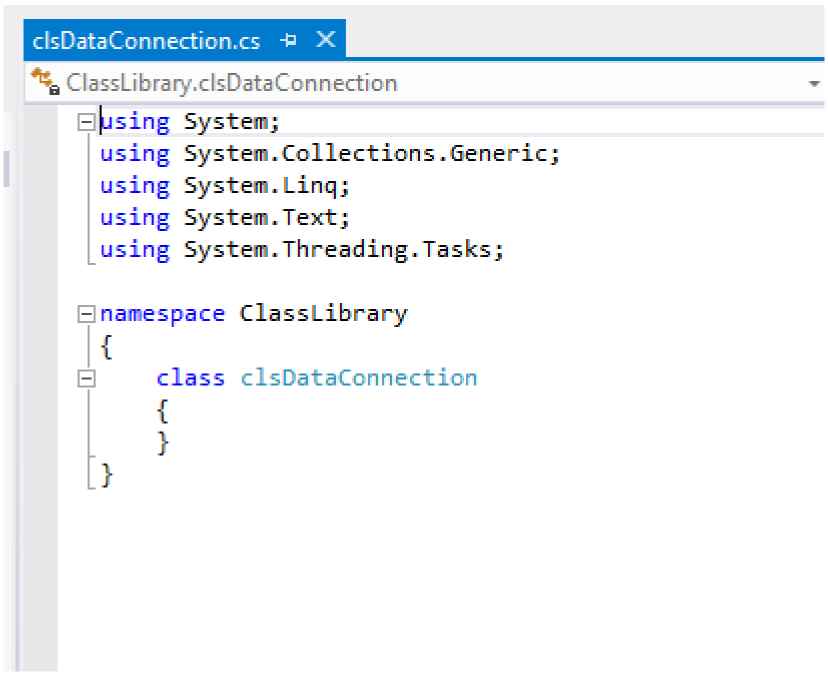
Set the name of the class as clsDataConnection…


Add New Item - ClassLibrary
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
ASP.NET Core
SQL Server
Storm Items
Online
Sort by: Default
Search (Ctrl+E)
Class Visual C# Items
Class for U-SQL Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File Visual C# Items
Assembly Information File Visual C# Items
Bitmap File Visual C# Items
Code Analysis Rule Set Visual C# Items
Add
Cancel

Type: Visual C# Items
An empty class definition

Name: clsDataConnection

Press Add and the class will be created with some default code…



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClassLibrary
{
    class clsDataConnection
    {
    }
}
```

Namespaces

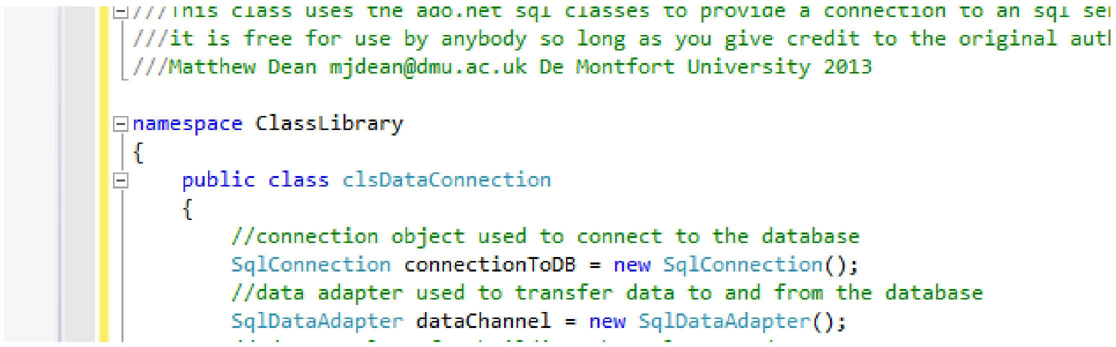
Notice in the above code that there is a namespace called ClassLibrary. The namespace is a way of organising classes in the library such that they are grouped together. For example you could have a namespace called graphics with all classes related to generating images organised in that namespace.

We are not interested in looking at namespaces in this module but we will need to make sure that the ClassLibrary namespace is included in our classes.

Copy the code for the class from the module web site.

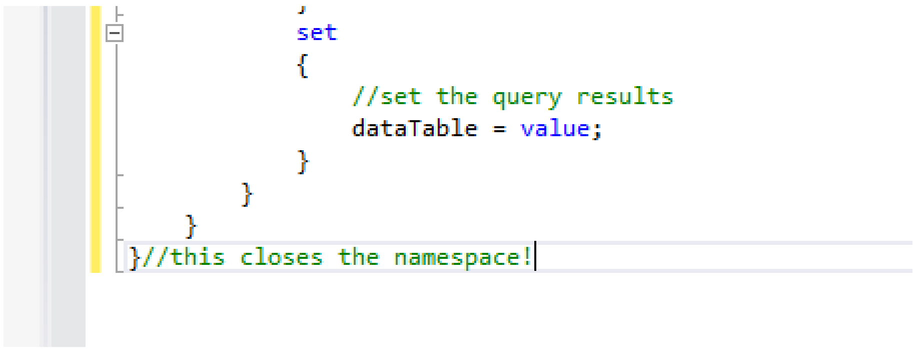
Delete the default code and replace it with the code from the module web site.

Don’t forget to add in your namespace at the top of the code…



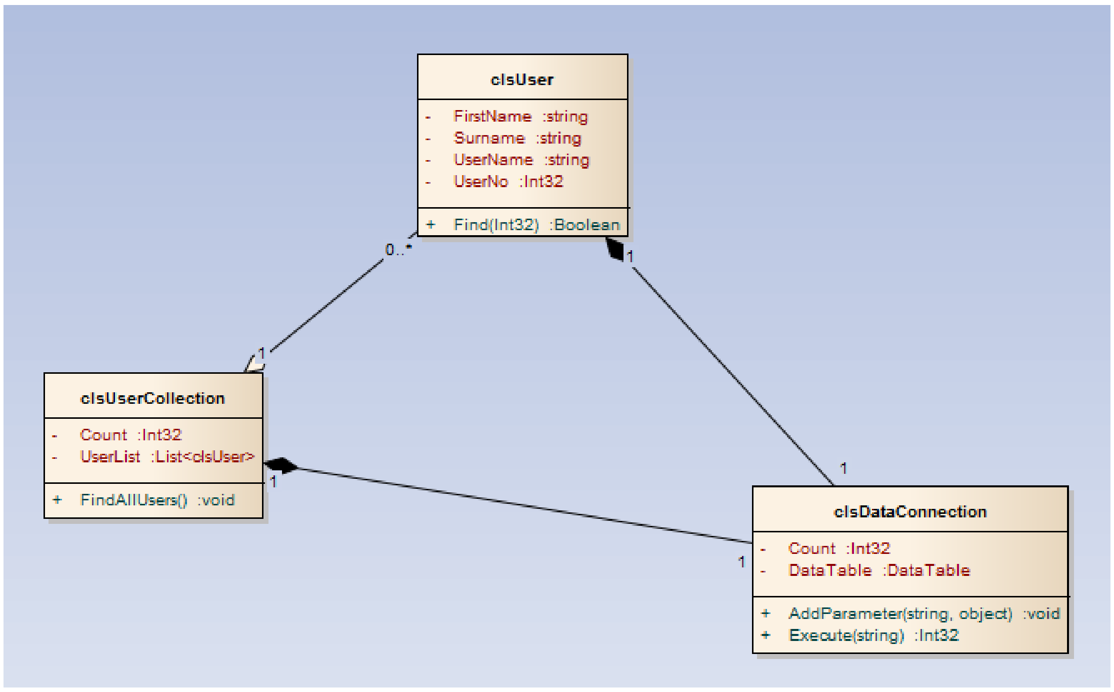
```
///This class uses the ado.net sql classes to provide a connection to an sql se
///it is free for use by anybody so long as you give credit to the original autl
///Matthew Dean mjdean@dmu.ac.uk De Montfort University 2013

namespace ClassLibrary
{
    public class clsDataConnection
    {
        //connection object used to connect to the database
        SqlConnection connectionToDB = new SqlConnection();
        //data adapter used to transfer data to and from the database
        SqlDataAdapter dataChannel = new SqlDataAdapter();
```

With a closing bracket at the very end…



```
                }
                set
                {
                    //set the query results
                    dataTable = value;
                }
            }
        }
}//this closes the namespace!
```

We are going to set up the following classes in the class library…



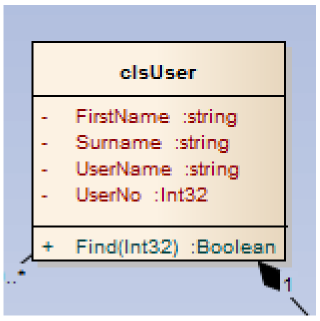
clsDataConnection has been created above.

We need to create the two other classes clsUser & clsUserCollection.

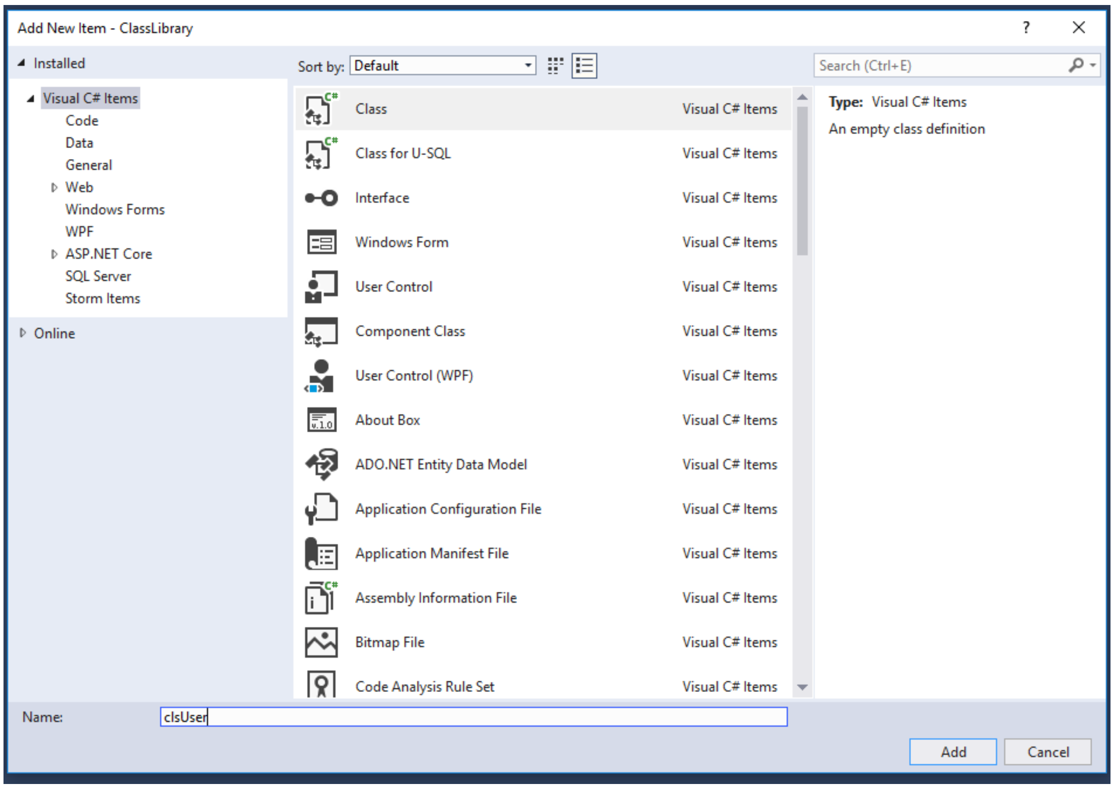
clsUser

clsUser will allow us to store data for a single record of data.

We will be implementing the following methods and properties for the class…

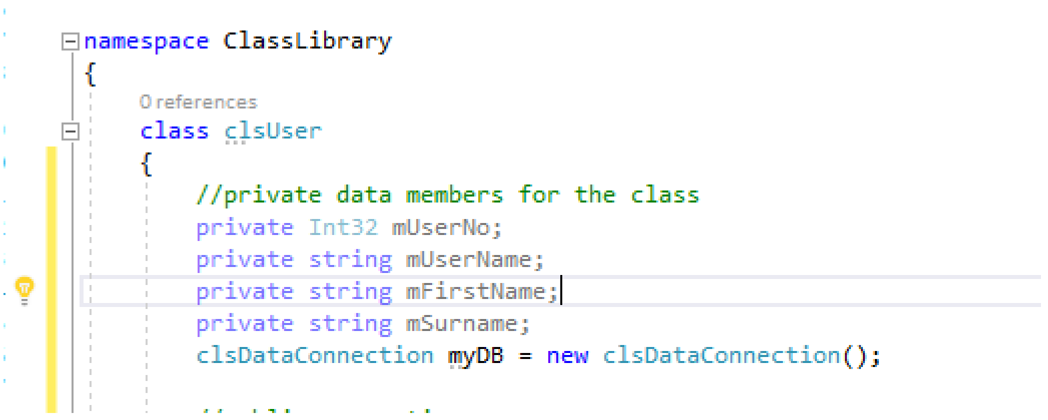


Right click on the class library and create the class…



Name: clsUser

To create the properties we will need to set up private data members. These data members are the variables that store the data for each column in the associated table.



```
namespace ClassLibrary
{
    0 references
    class clsUser
    {
        //private data members for the class
        private Int32 mUserNo;
        private string mUserName;
        private string mFirstName;
        private string mSurname;
        clsDataConnection myDB = new clsDataConnection();
```

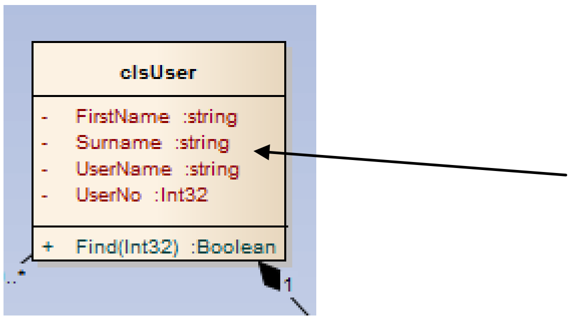
```
        //private data members for the class
```

```
private Int32 mUserNo;
private string mUserName;
private string mFirstName;
private string mSurname;
clsDataConnection myDB = new clsDataConnection();
```

Now that we have private variables set up we need to allow access to them via public properties…

```
//public properties
public Int32 UserNo
{
    get
    {
        return mUserNo;
    }
    set
    {
        mUserNo = value;
    }
}

public string UserName
{
    get
    {
        return mUserName;
    }
    set
    {
        mUserName = value;
    }
}

public string FirstName
{
    get
    {
        return mFirstName;
    }
    set
    {
        mFirstName = value;
    }
}

public string Surname
{
    get
    {
        return mSurname;
    }
    set
    {
        mSurname = value;
    }
}
```

This gives us the top section of the class – the attributes / properties.



The next step is to create the operations / methods for the class – in this case our Find method.

The code for this is here…

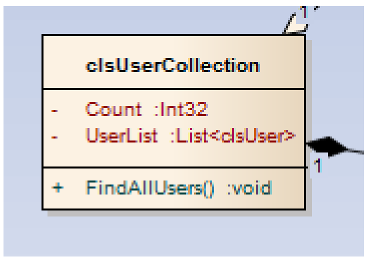
```
///public find method
public Boolean Find(Int32 UserNo)
{
    //re set the connection to the database
    myDB = new clsDataConnection();
    //pass the parameter to the stored procedure
    myDB.AddParameter("@UserNo", UserNo);
    //execute the stored procedure
    myDB.Execute("sproc_tblUser_FilterByUserNo");
    //check to see if we found anything
    if (myDB.Count == 1)
    {
        //set the private data members with the data from the database
        //private Int32 userNo;
        mUserNo = Convert.ToInt32(myDB.DataTable.Rows[0]["UserNo"]);
        //private string userName;
        mUserName = Convert.ToString(myDB.DataTable.Rows[0]["UserName"]);
        //private string firstName;
        mFirstName = Convert.ToString(myDB.DataTable.Rows[0]["FirstName"]);
        //private string surname;
        mSurname = Convert.ToString(myDB.DataTable.Rows[0]["Surname"]);
        //return success
        return true;
    }
    else //user no was invalid
    {
        //return that there was a problem
        return false;
    }
}
```

At this stage it isn't important that you write this code from scratch (copy and paste the examples here). However it is important that you understand what it does!

clsUserCollection

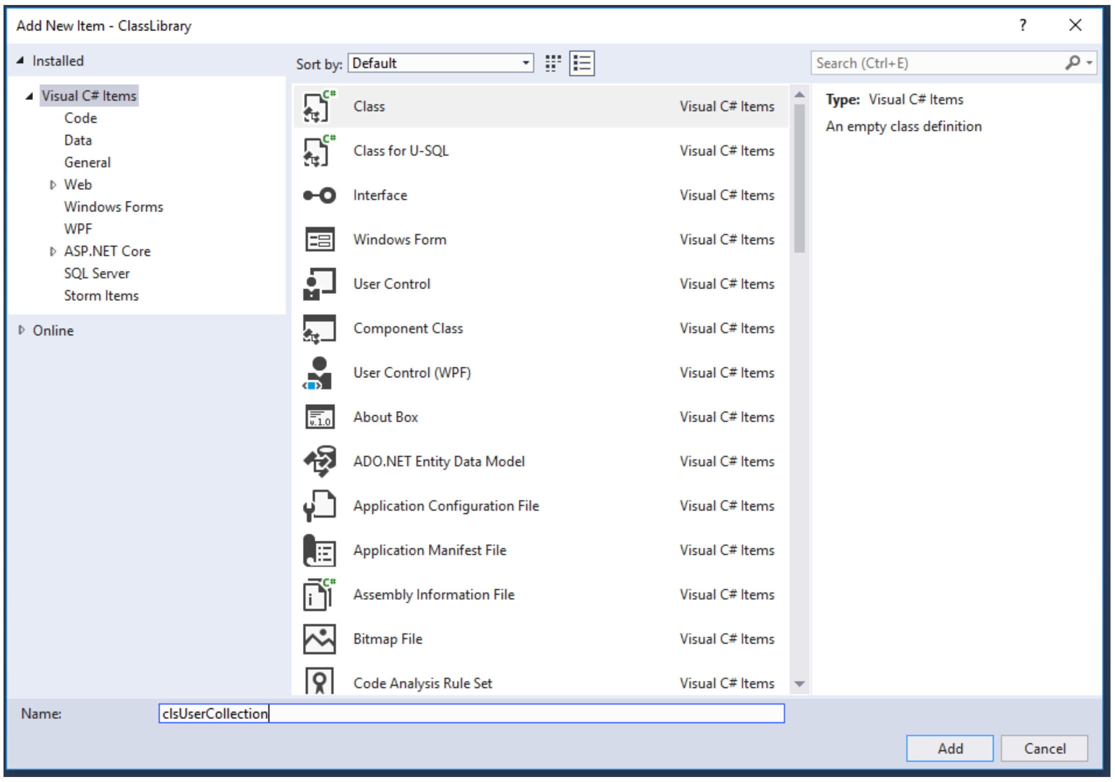
This class will allow us to manage multiple instances of clsUser.

In the finished code it will provide functions for Add, Update and Delete. At this stage we will concentrate on the following…



A public list of users and a function to implement the public method FindAllUsers.

Create the class within the class library…

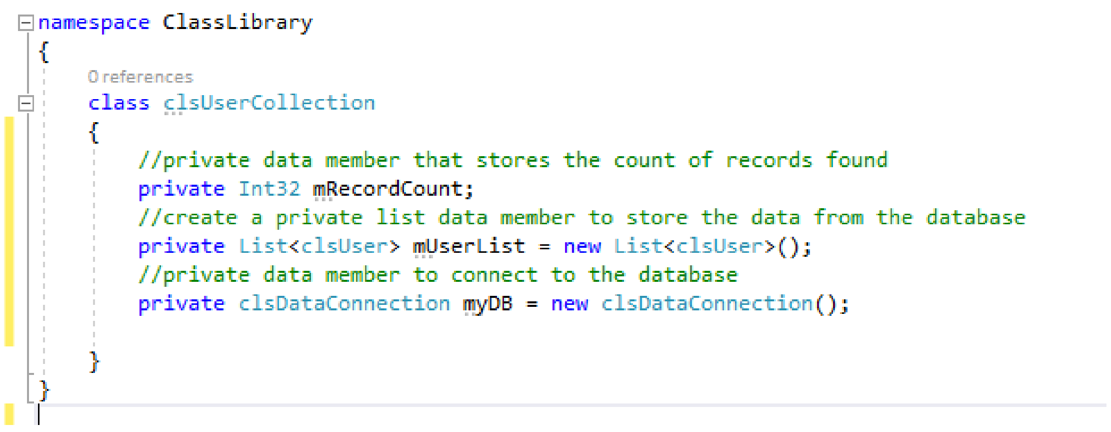

Add New Item - ClassLibrary

Name: clsUserCollection

Type: Visual C# Items
An empty class definition

Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
ASP.NET Core
SQL Server
Storm Items
Online
Sort by: Default
Search (Ctrl+E)
Class	Visual C# Items
Class for U-SQL	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
Component Class	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File	Visual C# Items
Assembly Information File	Visual C# Items
Bitmap File	Visual C# Items
Code Analysis Rule Set	Visual C# Items
Add
Cancel

As with most classes there will need to be private data members


```
namespace ClassLibrary
{
    0 references
    class clsUserCollection
    {
        //private data member that stores the count of records found
        private Int32 mRecordCount;
        //create a private list data member to store the data from the database
        private List<clsUser> mUserList = new List<clsUser>();
        //private data member to connect to the database
        private clsDataConnection myDB = new clsDataConnection();

    }
}
```

```
        //private data member that stores the count of records found
        private Int32 mRecordCount;
        //create a private list data member to store the data from the database
        private List<clsUser> mUserList = new List<clsUser>();
        //private data member to connect to the database
        private clsDataConnection myDB = new clsDataConnection();
```

In the case of this class there are two public properties

```
        //public property returning the count of records
        public Int32 Count
        {
            get
            {
                //return record count;
                return mRecordCount;
            }
        }

        //public list of users
        public List<clsUser> Users
        {
            //getter for the property
            get
            {
                //return the list of users
                return mUserList;
            }
        }
```

And for this class a single method…

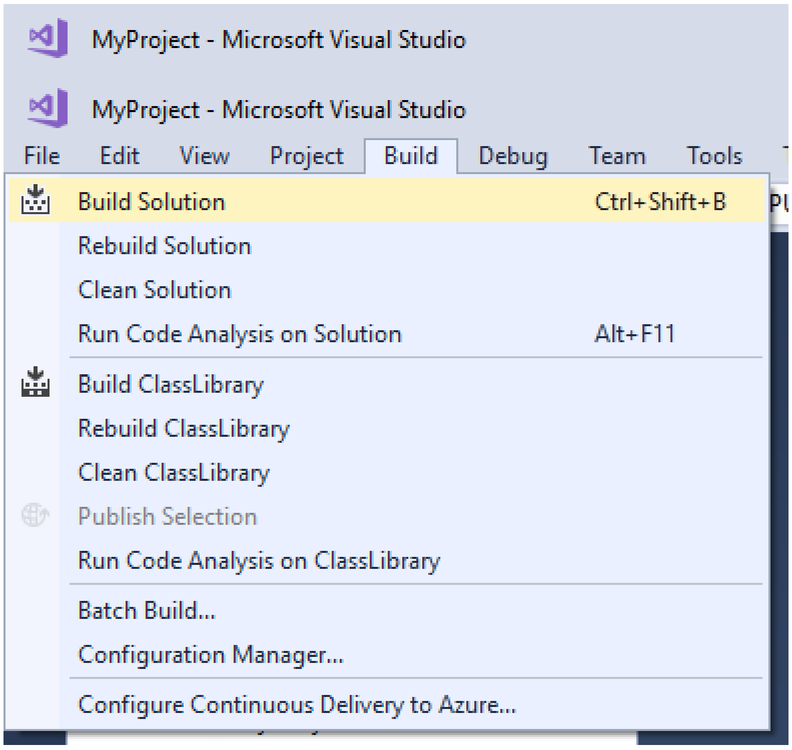
```
        public void FindAllUsers()
        {
            //re-set the connection
            myDB = new clsDataConnection();
            //var to store the index
            Int32 Index = 0;
            //var to store the user number of the current record
            Int32 UserNo;
            //var to flag that user was found
            Boolean UserFound;
            //execute the stored procedure
            myDB.Execute("sproc_tblUser_SelectAll");
            //get the count of records
            mRecordCount = myDB.Count;
            //while there are still records to process
            while (Index < myDB.Count)
            {
                //create an instance of the user class
                clsUser NewUser = new clsUser();
                //get the user number from the database
                UserNo = Convert.ToInt32(myDB.DataTable.Rows[Index]["UserNo"]);
                //find the user by invoking the find method
                UserFound = NewUser.Find(UserNo);
                if (UserFound == true)
                {
                    //add the user to the list
                    mUserList.Add(NewUser);
                }
                //increment the index
                Index++;
        }
    }
```

Linking the Class Library to the Web Site

Now that we have set up the code for the classes in the library we need to look at linking the class library to our other projects.

The first step is to build the solution which will set up the class library for us.

From the main menu select build and build solution.



This is the same as pressing F5 or the play button but it won't run the program.

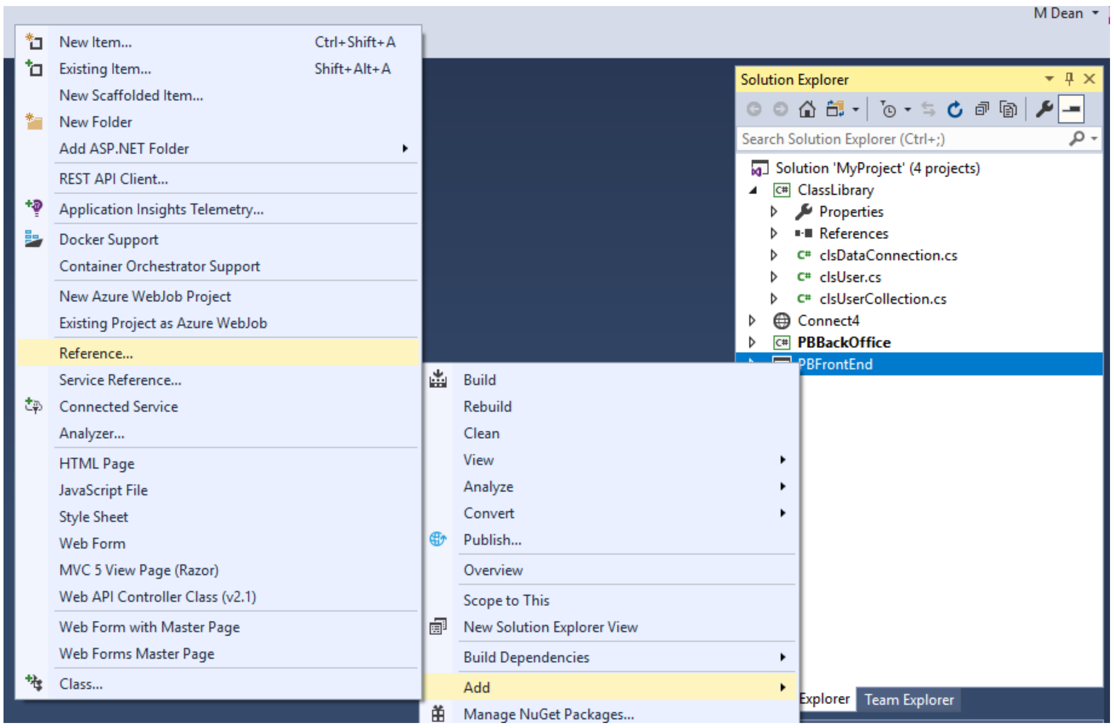
What this does is it checks our code for syntax errors. If there are any you will need to fix them before continuing to the next step.

It also creates a file in the class library called a Dynamic Link Library or DLL.

The DLL is a product of our class code compiled into machine language which the computer understands.

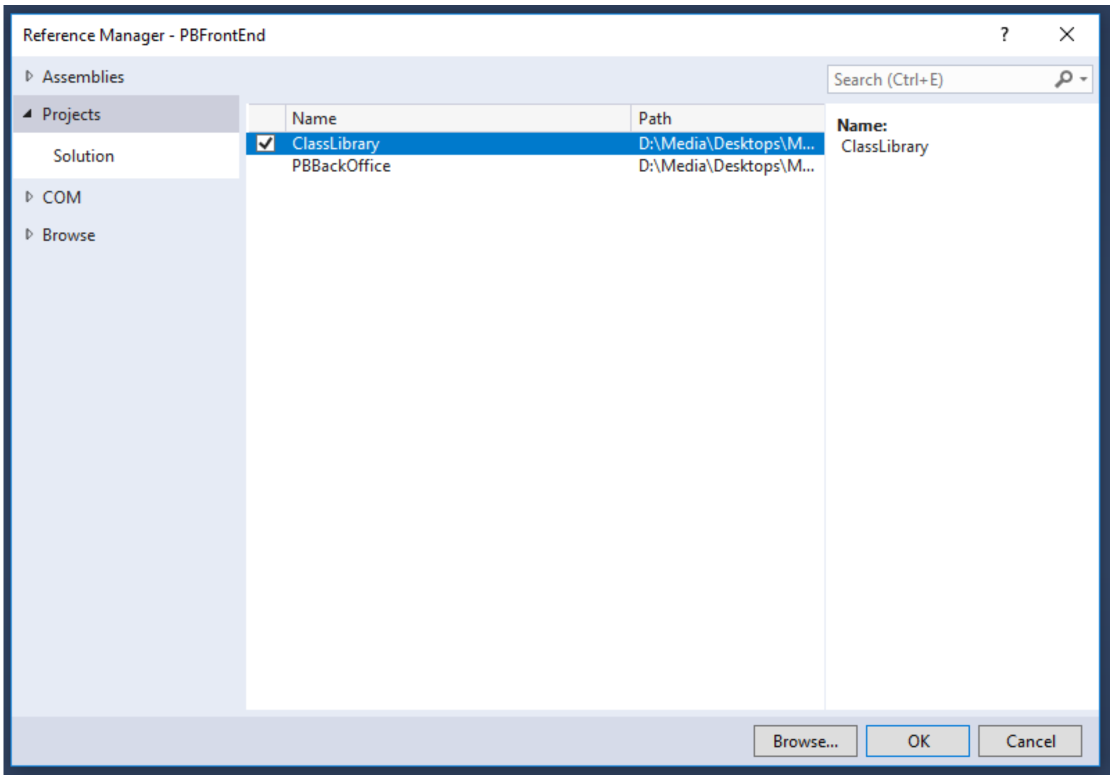
The next step is to link the class library to the front end and back end of the solution.

On the PBFrontEnd web site right click and select Add – Reference…



This will allow you to make the link between this project and the class library.

Make sure you have Solution selected on the right and then place a tick in the box next to the class library name…



	Name	Path
✓	ClassLibrary	D:\Media\Desktops\M...
	PBBackOffice	D:\Media\Desktops\M...

Name: ClassLibrary

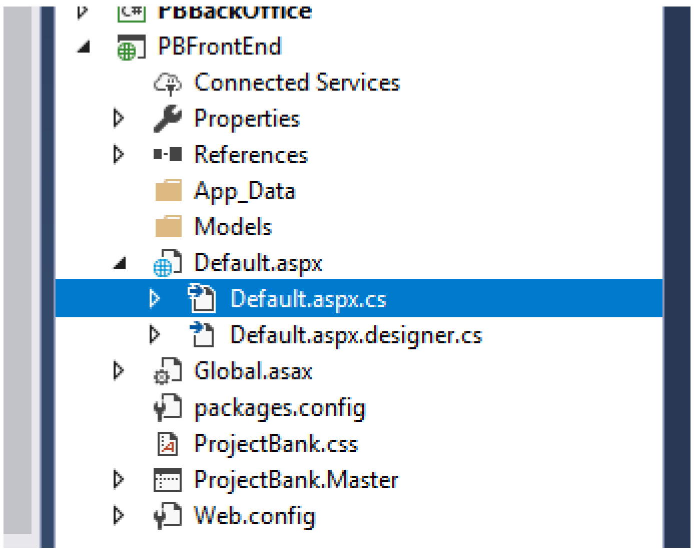
Now do the same for the project PBBackOffice…

This will configure Visual Studio such that any code in the class library is now available to the two projects in our solution.

Testing the Configuration

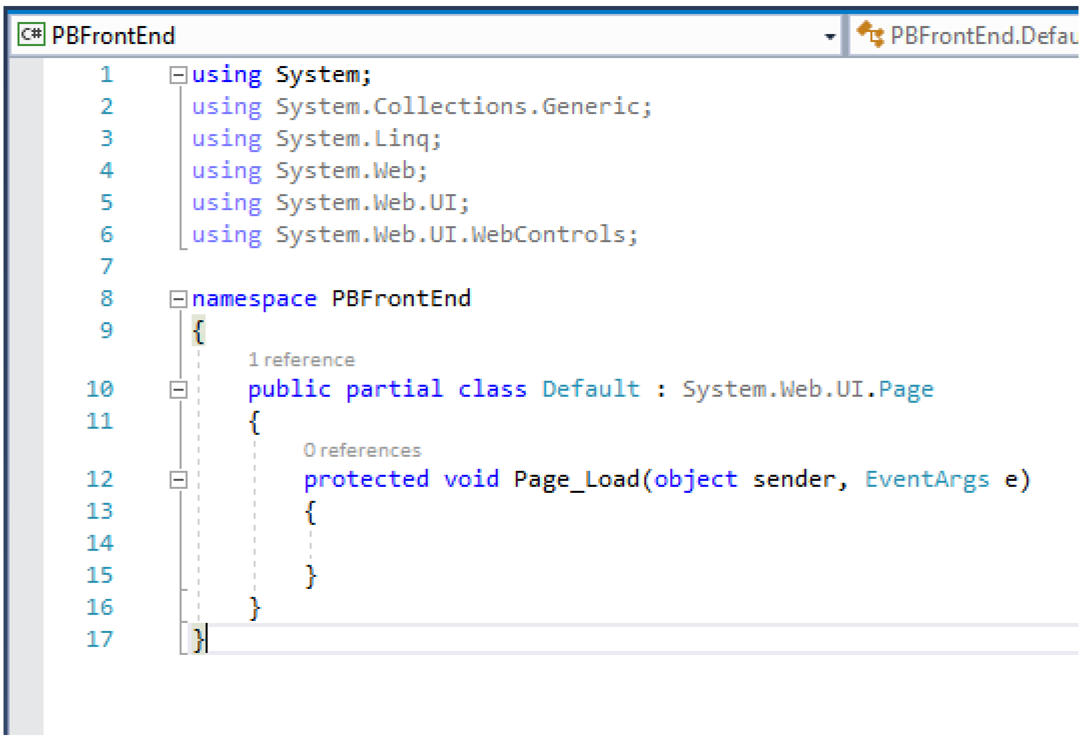
It would be a good idea at this stage to check if everything is working ok.

Open the C# code file for Default.aspx in the Front End web site like so…



```
PBBackOffice
PBFrontEnd
    Connected Services
    Properties
    References
    App_Data
    Models
    Default.aspx
        Default.aspx.cs
        Default.aspx.designer.cs
    Global.asax
    packages.config
    ProjectBank.css
    ProjectBank.Master
    Web.config
```

You should see the following...

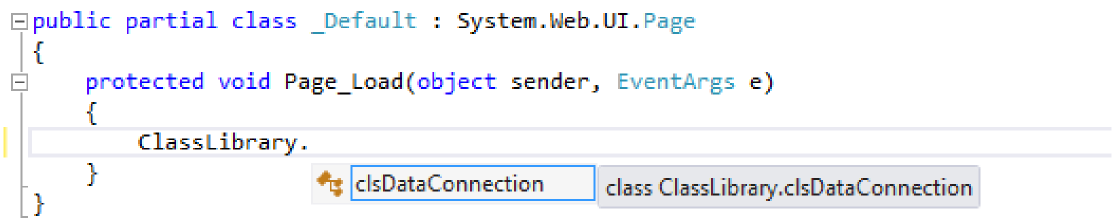


PBFrontEnd | PBFrontEnd.Defau

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

namespace PBFrontEnd
{
    1 reference
    public partial class Default : System.Web.UI.Page
    {
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

If everything is working correctly we should be able to access the classes in the class library.

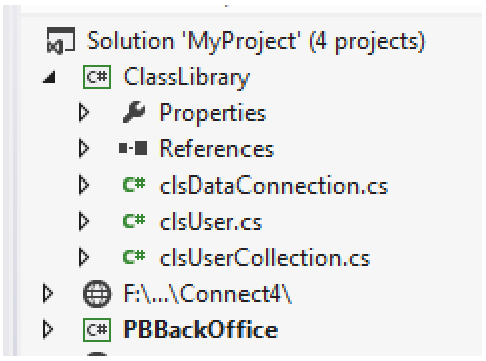
Try typing the name of the namespace followed by a dot….



```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        ClassLibrary.
    }
}
```

Can you see the problem?

In the class library we have three class files so far…

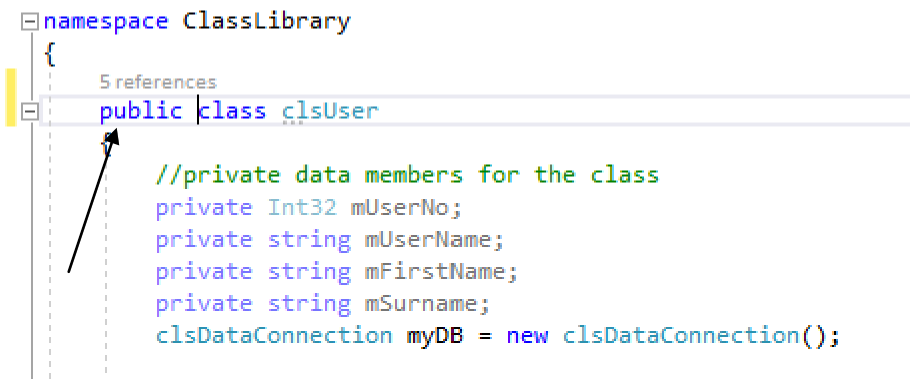


```
Solution 'MyProject' (4 projects)
  ClassLibrary
    Properties
    References
    clsDataConnection.cs
    clsUser.cs
    clsUserCollection.cs
  F:\...\Connect4\
  PBBackOffice
```

Why is it we can only see one?

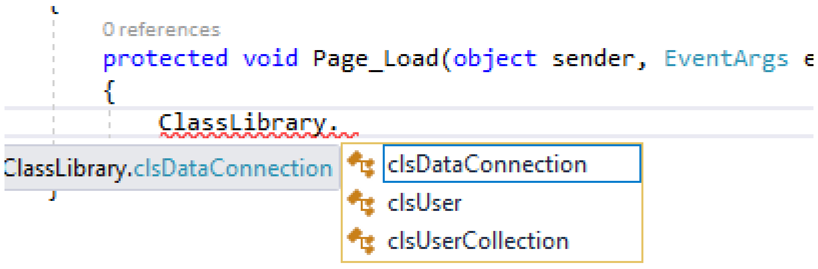
The problem is that we need to make the two other classes public as by default they are private.

In clsUser modify the class definition like so…



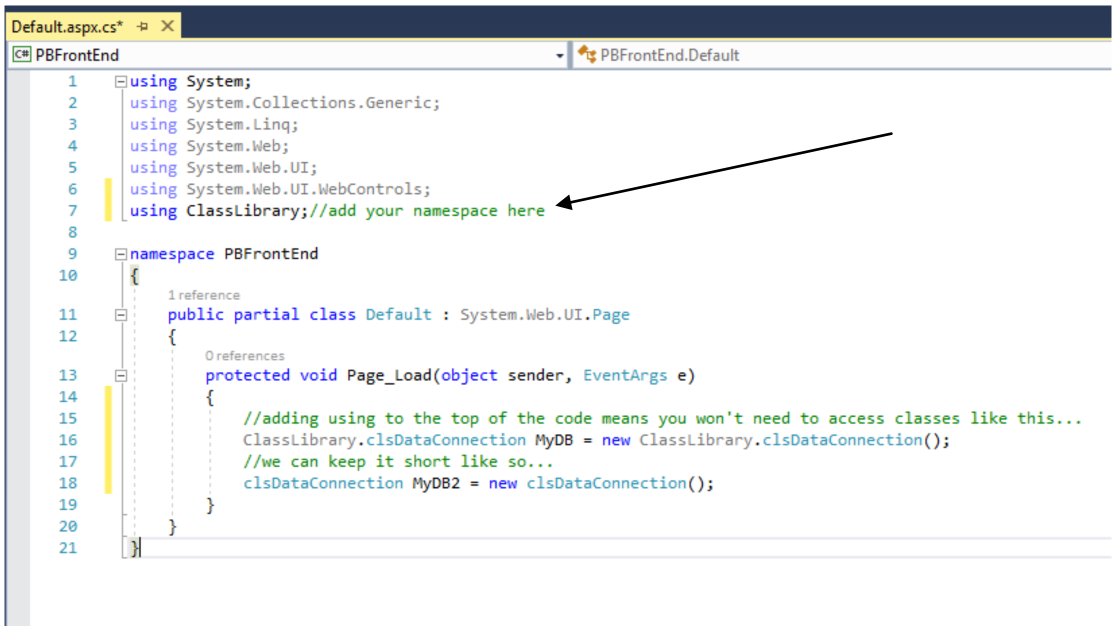
```
namespace ClassLibrary
{
    5 references
    public class clsUser
    {
        //private data members for the class
        private Int32 mUserNo;
        private string mUserName;
        private string mFirstName;
        private string mSurname;
        clsDataConnection myDB = new clsDataConnection();
```

Now try accessing the class from Default.aspx again. You should get the following results…



```
0 references
protected void Page_Load(object sender, EventArgs e
{
    ClassLibrary.
```

In future we can make the code a bit cleaner by adding the namespace to the top of our web form code we create like so…



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using ClassLibrary;//add your namespace here

namespace PBFrontEnd
{
    1 reference
    public partial class Default : System.Web.UI.Page
    {
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
            //adding using to the top of the code means you won't need to access classes like this...
            ClassLibrary.clsDataConnection MyDB = new ClassLibrary.clsDataConnection();
            //we can keep it short like so...
            clsDataConnection MyDB2 = new clsDataConnection();
        }
    }
}
```