Entity Framework and WPF Example

Entity framework works nicely with SQL server as the database. A Repository can be easily created which uses LINQ to issue queries to the database. We will create a simple example for a bank application where a user can login, check balances and do a transfer from checking to saving as an atomic transaction. The front end for the application will be created in WPF. When a user logs in, he/she belongs to a certain role such as “Customer”, “Manager”, “Admin” etc.. So for this purpose, we will have three tables in the database, Users, Roles, and UserRoles in addition to the banking related tables.

Create a database called MyBank in SQL Server with the following tables.

Users

ALPHA.MyBank - dbo.Users | ALPHA.MyBank - dbo.Users

Column Name	Data Type
Username	varchar(50)
Password	varchar(50)

Roles

ALPHA.MyBank - dbo.Roles | ALPHA.MyBank - dbo.Roles

Column Name	Data Type
RoleID	int
RoleName	varchar(50)

UserRoles

ALPHA.MyBank - dbo.UserRoles | ALPHA.MyBank - dbo.UserRoles | ALPHA.MyBank - dbo.Roles

Column Name	Data Type	Allow Nulls
Username	varchar(50)	
RoleID	int	

CheckigAccounts

ALPHA.MyBank - dbo.CheckingAccounts | ALPHA.MyBank - dbo.Transaction

Column Name	Data Type	Allow Nulls
Username	varchar(50)	
CheckingAccountNumber	bigint	
Balance	money	

SavingAccounts

ALPHA.MyBank - dbo.SavingAccounts | ALPHA.MyBank - dbo.CheckingAc

Column Name	Data Type	Allow Nulls
Username	varchar(50)	☐
SavingAccountNumber	bigint	☐
Balance	money	☐

TransactionTypes

ALPHA.MyBank - dbo.TransactionTypes | ALPHA.MyBank - db

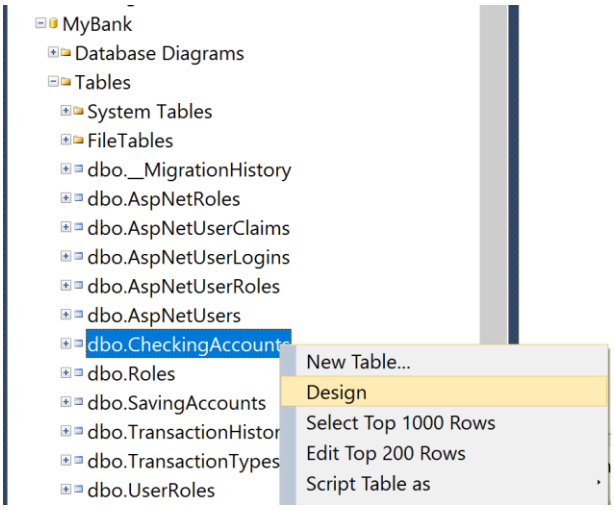
Column Name	Data Type	Allow Nulls
TransactionTypeId	int	☐
TransactionTypeName	varchar(100)	☐

TransactionHistories

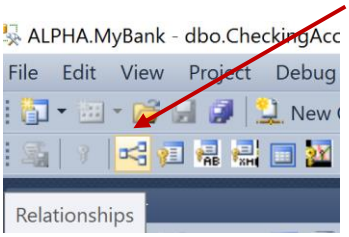
ALPHA.MyBank - db...ansactionHistories | ALPHA.MyBank - dbo.Saving

Column Name	Data Type	Allow Nulls
TransactionId	bigint	☐
TrasactionDate	datetime	☐
CheckingAccountNumber	bigint	☐
SavingAccountNumber	bigint	☐
Amount	money	☐
TransactionFee	money	☐
TransactionTypeId	int	☐

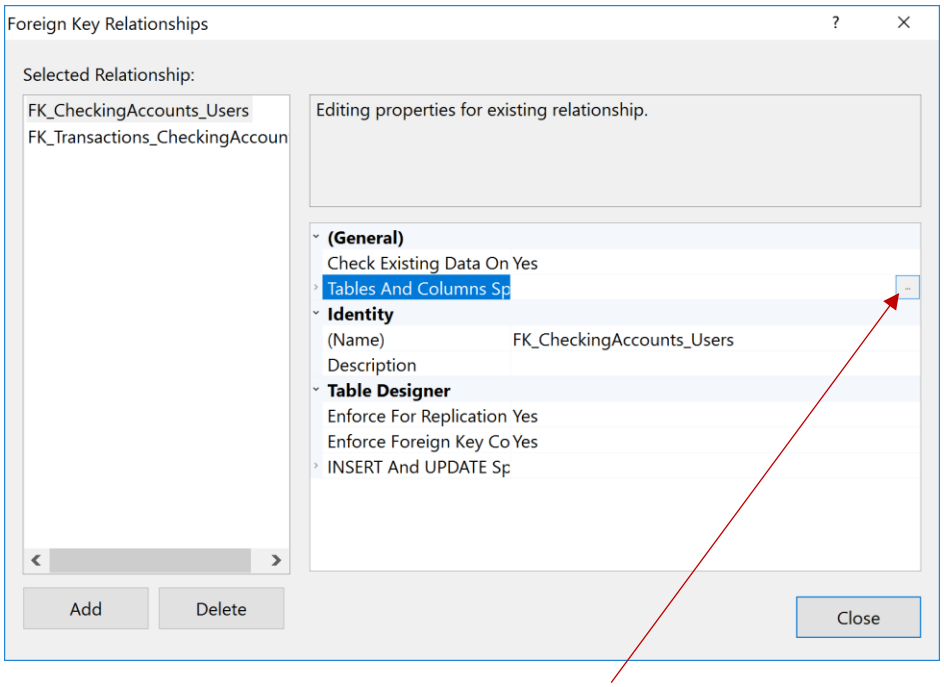
Link the tables appropriately by putting primary keys on each and table and creating primary foreign key relationships. For example, the Username is a primary key in the Users table. In the CheckingAccounts table, the CheckingAccountNumber is a foreign key whereas the Username is a foreign key. To set this primary-foreign key relationship, right click on the CheckingAccounts table in the SQL server Management Studio, and choose “design”.



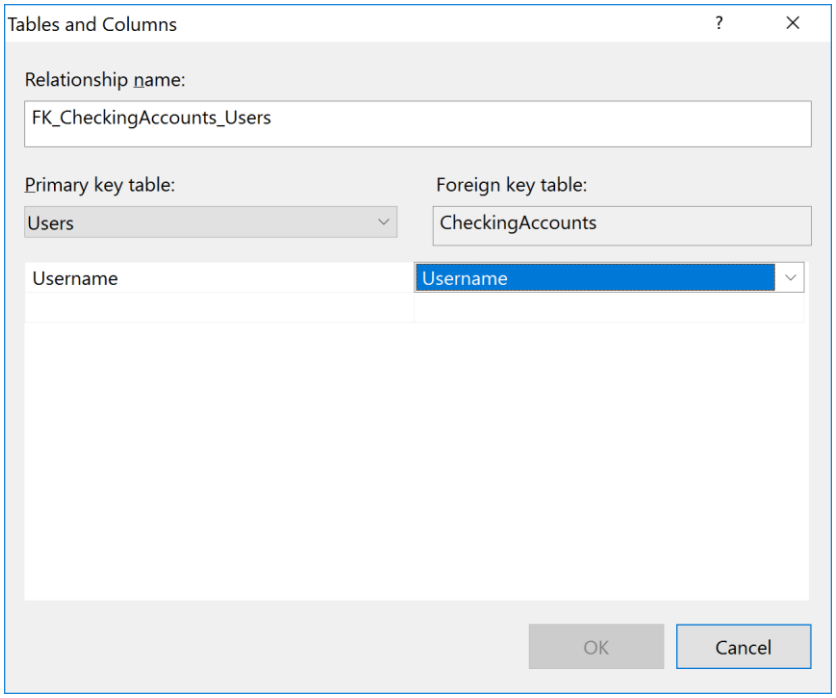
Then click on the toolbar button for relationships as shown below.



Then click on the Add button, as shown below.



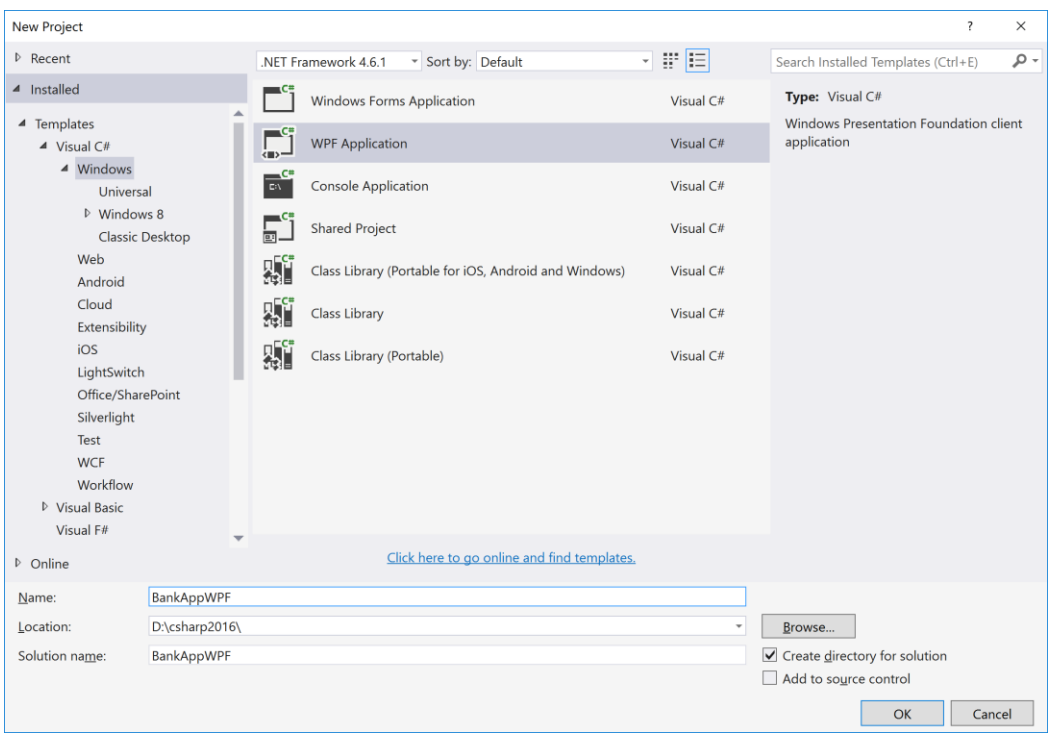
Then click on the three dotted button in the Tables And Columns Specification row and select the relationships as shown below.



Then click on OK, and close the designer, it will ask you to update the tables, click OK.

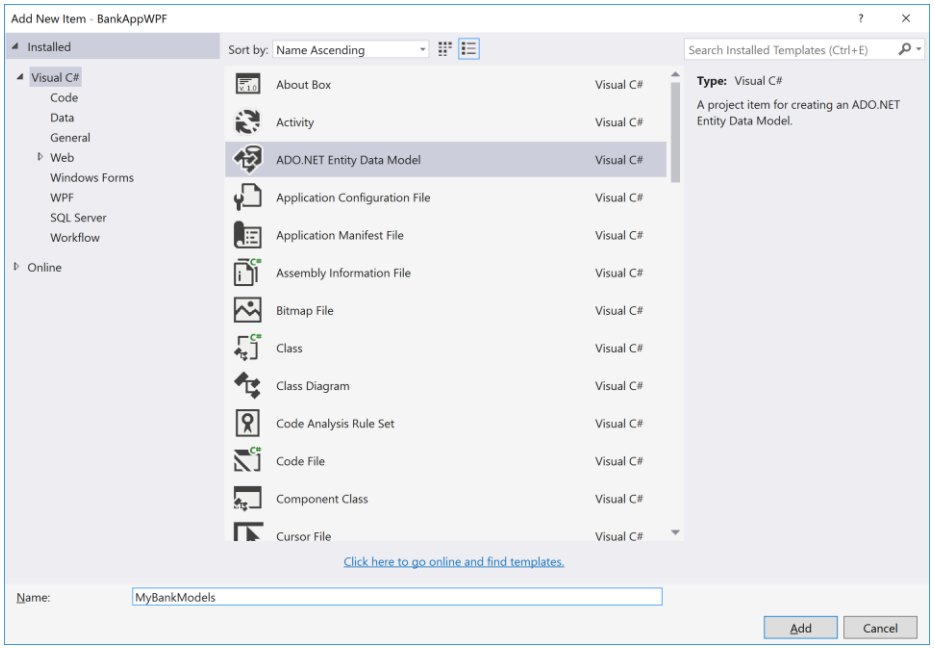
Similarly, add primary-foreign key relationships between different tables in your database design where ever it makes sense.

Now create a WPF application called “BankAppWPF” using Visual Studio as shown below.

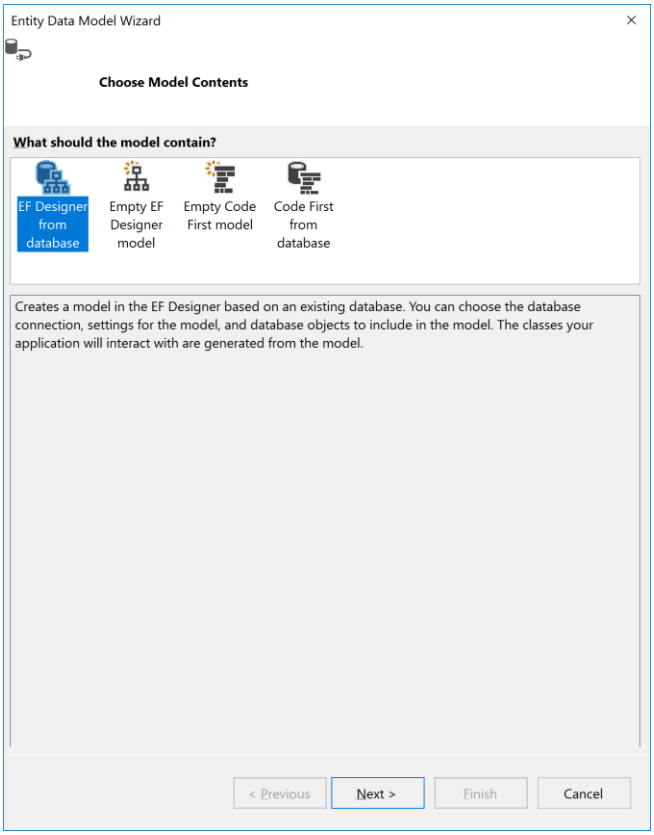


Add a folder called EFModels by right clicking on the name of the project and choosing new folder.

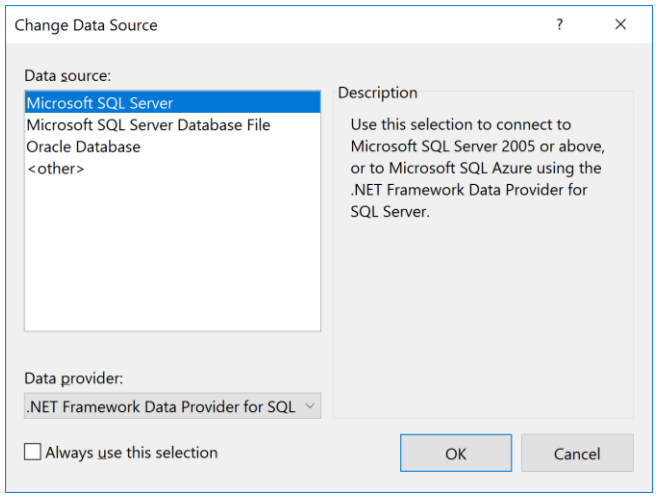
Then right click on the EFModels folder and choose, add new item, and select “ADO.NET Entity Data Model” called “MyBankModels” as shown below.



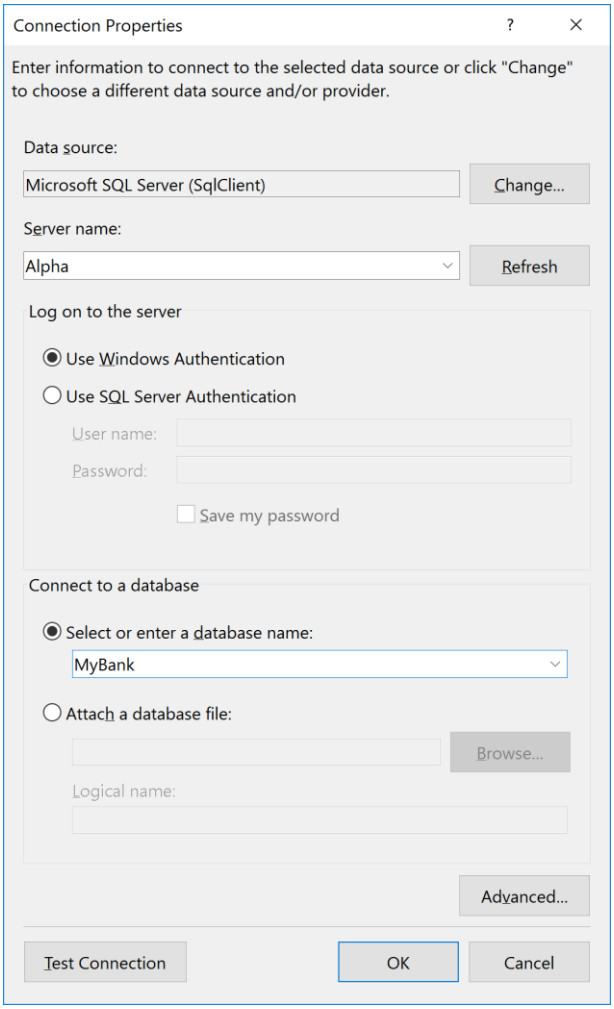
Select EF Designer from Database

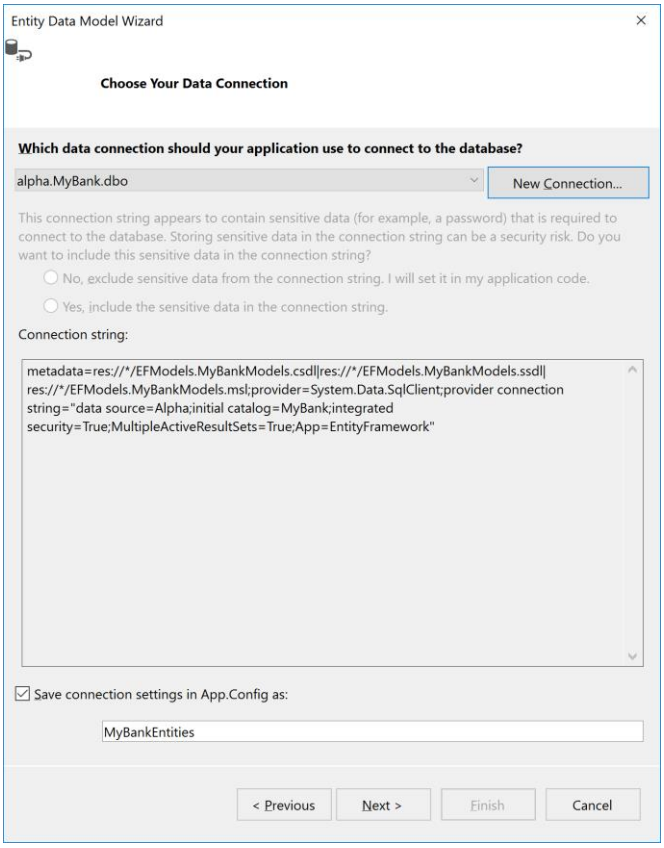


In the next screen, click on “New Connection” button and select the Microsoft SQL server as shown below.

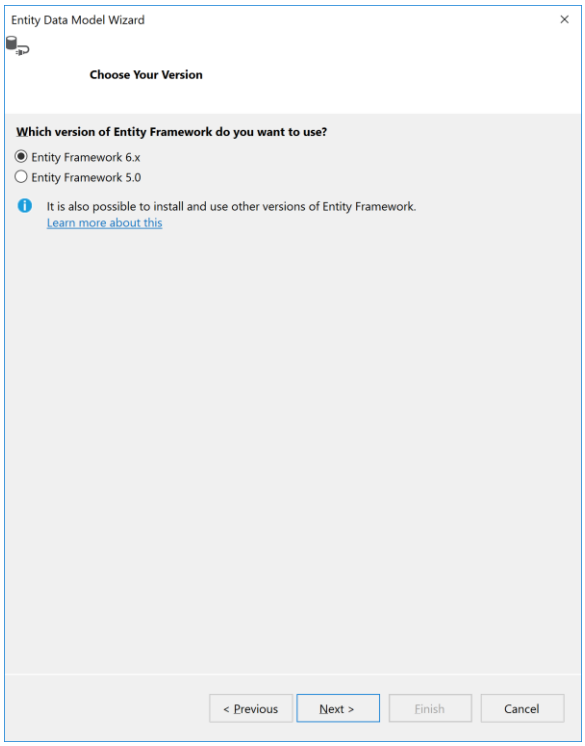


Then in the next screen, choose the name of your database server and the name of the database as shown below.

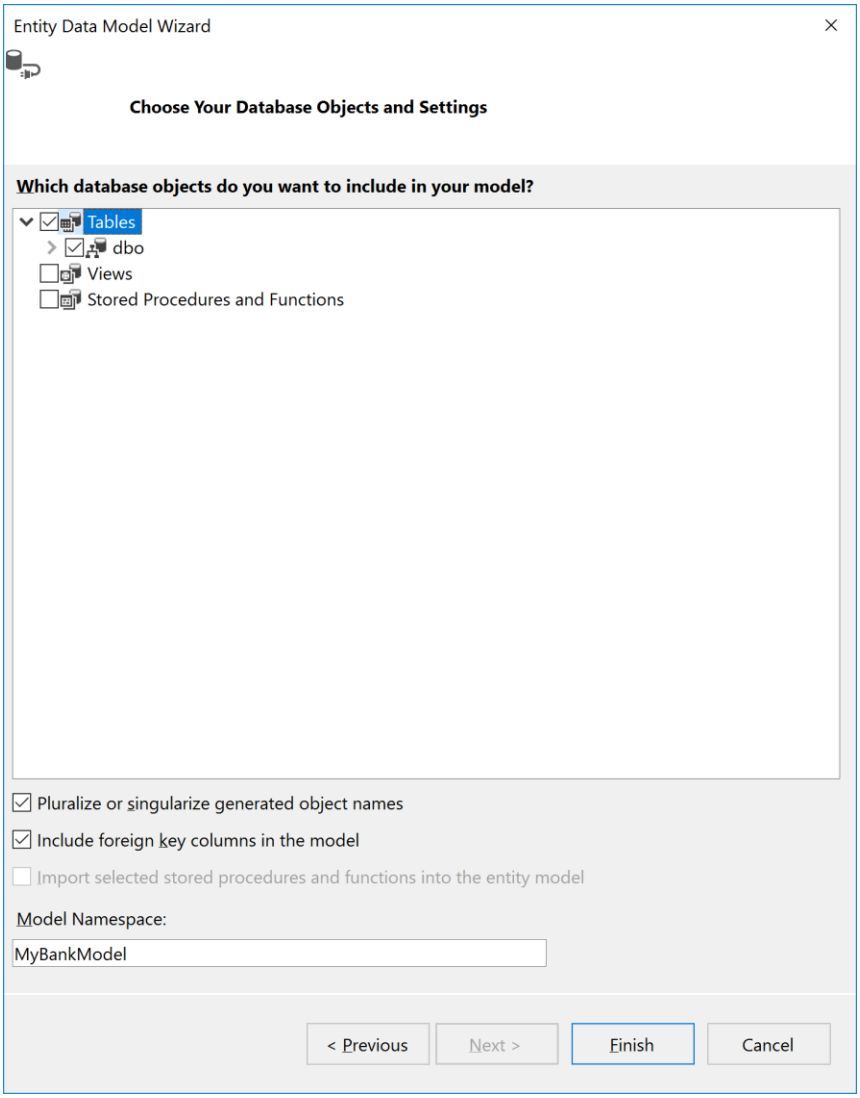




Select the latest Entity Framework in the next screen.



Select all the tables to be included in the output, then click finish.



You may get a few warnings, click OK when you get these.

Add a folder called DataLayer to the project. Then add an interface called IRepositoryAuthentication with the following code in it.

```
namespace BankAppWPF.DataLayer
{
    public interface IRepositoryAuthentication
    {
        bool CheckIfValidUser(string username, string password);
        string GetRolesForUser(string username);
        bool ChangePassword(string username, string oldPassword, string newPassword);

    }
}
```

Add an interface called IRepositoryBanking with the following interface in it.

```
namespace BankAppWPF.DataLayer
{
    public interface IRepositoryBanking
    {
        decimal GetCheckingBalance(long checkingAccountNum);
        decimal GetSavingBalance(long savingAccountNum);
        long GetCheckingAccountNumForUser(string username);
        long GetSavingAccountNumForUser(string username);
        bool TransferCheckingToSaving(long checkingAccountNum, long savingAccountNum,
decimal amount, decimal transactionFee);
        bool TransferSavingToChecking(long checkingAccountNum, long savingAccountNum,
decimal amount, decimal transactionFee);
        List<TransactionHistoryModel> GetTransactionHistory(long checkingAccountNum);
    }
}
```

Add a folder to the project called ViewModels. Then add a class called TransaferHistoryModel with the following code in it.

```
namespace BankAppWPF.ViewModels
{
    public class TransactionHistoryModel
    {
        public long CheckingAccountNumber { get; set; }
        public long SavingAccountNumber { get; set; }
        public decimal Amount { get; set; }
        public decimal TransactionFee { get; set; }
        public DateTime TransactionDate { get; set; }
        public string TransactionTypeName { get; set; } // added field
    }
}
```

Add a class **Repository** to the DataLayer folder with the following code in it.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using BankAppWPF.EFModels;
using System.Data.Entity;
using BankAppWPF.ViewModels;

namespace BankAppWPF.DataLayer
{
    public class Repository : IRepositoryAuthentication, IRepositoryBanking
    {
        MyBankEntities _dbContext = null;
        public Repository()
        {
            _dbContext = new MyBankEntities();
        }

        public Repository(DbContext dbContext)
```

```
        {
            _dbContext = dbContext as MyBankEntities;
        }
        public bool ChangePassword(string username, string oldPassword, string
newPassword)
        {
            throw new NotImplementedException();
        }

        public bool CheckIfValidUser(string username, string password)
        {
            bool ret = false;
            try
            {
                var res = (from rec in _dbContext.Users
                           where rec.Username == username && rec.Password==password
                           select rec.Username).FirstOrDefault<string>();
                if (res != null)
                    ret = true;
            }
            catch (Exception)
            {
                throw;
            }
            return ret;
        }

        public long GetCheckingAccountNumForUser(string username)
        {
            try
            {
                var res = (from rec in _dbContext.CheckingAccounts
                           where rec.Username == username
                           select rec.CheckingAccountNumber).FirstOrDefault<long>();
                return res;
            }
            catch (Exception)
            {
                throw;
            }
        }

        public decimal GetCheckingBalance(long checkingAccountNum)
        {
            try
            {
                var res = (from rec in _dbContext.CheckingAccounts
                           where rec.CheckingAccountNumber == checkingAccountNum
                           select rec.Balance).FirstOrDefault<decimal>();
                return res;
            }
            catch (Exception)
            {
                throw;
            }
```

```
        }

        public string GetRolesForUser(string username)
        {
            string roles = "";
            try
            {
                var RList = (from u in _dbContext.Users join ur in
_dbContext.UserRoles on u.Username equals ur.Username
                             join r in _dbContext.Roles on ur.RoleID equals r.RoleID
                             where u.Username == username
                             select r.RoleName).ToList<string>();
                foreach (string r in RList)
                    roles += r + "|";
                roles = roles.Substring(0, roles.Length - 1);
            }
            catch (Exception)
            {
                throw;
            }
            return roles;
        }

        public long GetSavingAccountNumForUser(string username)
        {
            try
            {
                var res = (from rec in _dbContext.SavingAccounts
                           where rec.Username == username
                           select rec.SavingAccountNumber).FirstOrDefault<long>();
                return res;
            }
            catch (Exception)
            {
                throw;
            }
        }

        public decimal GetSavingBalance(long savingAccountNum)
        {
            try
            {
                var res = (from rec in _dbContext.SavingAccounts
                           where rec.SavingAccountNumber == savingAccountNum
                           select rec.Balance).FirstOrDefault<decimal>();
                return res;
            }
            catch (Exception)
            {
                throw;
            }
        }
```

```
        public List<TransactionHistoryModel> GetTransactionHistory(long
checkingAccountNum)
        {
            try
            {
                var res = (from rec in _dbContext.TransactionHistories
                           join trtype in _dbContext.TransactionTypes on
rec.TransactionTypeId equals
                           trtype.TransactionTypeId
                           where rec.CheckingAccountNumber == checkingAccountNum
                           select new TransactionHistoryModel
                           {
                               CheckingAccountNumber = rec.CheckingAccountNumber,
                               SavingAccountNumber = rec.SavingAccountNumber,
                               Amount = rec.Amount,
                               TransactionFee = rec.TransactionFee,
                               TransactionTypeName = trtype.TransactionTypeName,
                               TransactionDate = rec.TrasactionDate
                           }).ToList<TransactionHistoryModel>();
                return res;
            }
            catch (Exception)
            {
                throw;
            }
        }

        public bool TransferCheckingToSaving(long checkingAccountNum, long
savingAccountNum, decimal amount, decimal transactionFee)
        {
            bool ret = false;
            // transfer is done as a transaction
            using (var dbContextTransaction = _dbContext.Database.BeginTransaction())
            {
                try
                {
                    var reccheck = (from r in _dbContext.CheckingAccounts
                                    where r.CheckingAccountNumber == checkingAccountNum
                                    select r).FirstOrDefault<CheckingAccount>();
                    if (reccheck != null)
                        reccheck.Balance = reccheck.Balance - amount;
                    _dbContext.SaveChanges();
                    var bal = (from r in _dbContext.CheckingAccounts
                               where r.CheckingAccountNumber == checkingAccountNum
                               select r.Balance).FirstOrDefault<decimal>();
                    if (bal < 0)
                        throw new Exception("insufficient amount in Checking
Account..");
                    var recsav = (from r in _dbContext.SavingAccounts
                                  where r.SavingAccountNumber == savingAccountNum
                                  select r).FirstOrDefault<SavingAccount>();
                    if (recsav != null)
                        recsav.Balance = recsav.Balance + amount;
                    _dbContext.SaveChanges();
```

```
                    TransactionHistory th = new TransactionHistory();
                    th.CheckingAccountNumber = checkingAccountNum;
                    th.SavingAccountNumber = savingAccountNum;
                    th.Amount = amount;
                    th.TransactionTypeId = 100;
                    th.TransactionFee = transactionFee;
                    th.TrasactionDate = DateTime.Now;
                    _dbContext.TransactionHistories.Add(th);
                    _dbContext.SaveChanges();

                    dbContextTransaction.Commit();
                    ret = true;
                }
                catch (Exception ex)
                {
                    dbContextTransaction.Rollback();
                    throw;
                }
            }
            return ret;
        }

        public bool TransferSavingToChecking(long checkingAccountNum, long
savingAccountNum, decimal amount, decimal transactionFee)
        {
            throw new NotImplementedException();
        }
    }
}
```

Creating the WPF UI:

WPF using XML to describe the different elements in the UI. In our case, the main window will contain menu items for logging in and for banking activities such as transfer checking to saving and for viewing the transaction history.

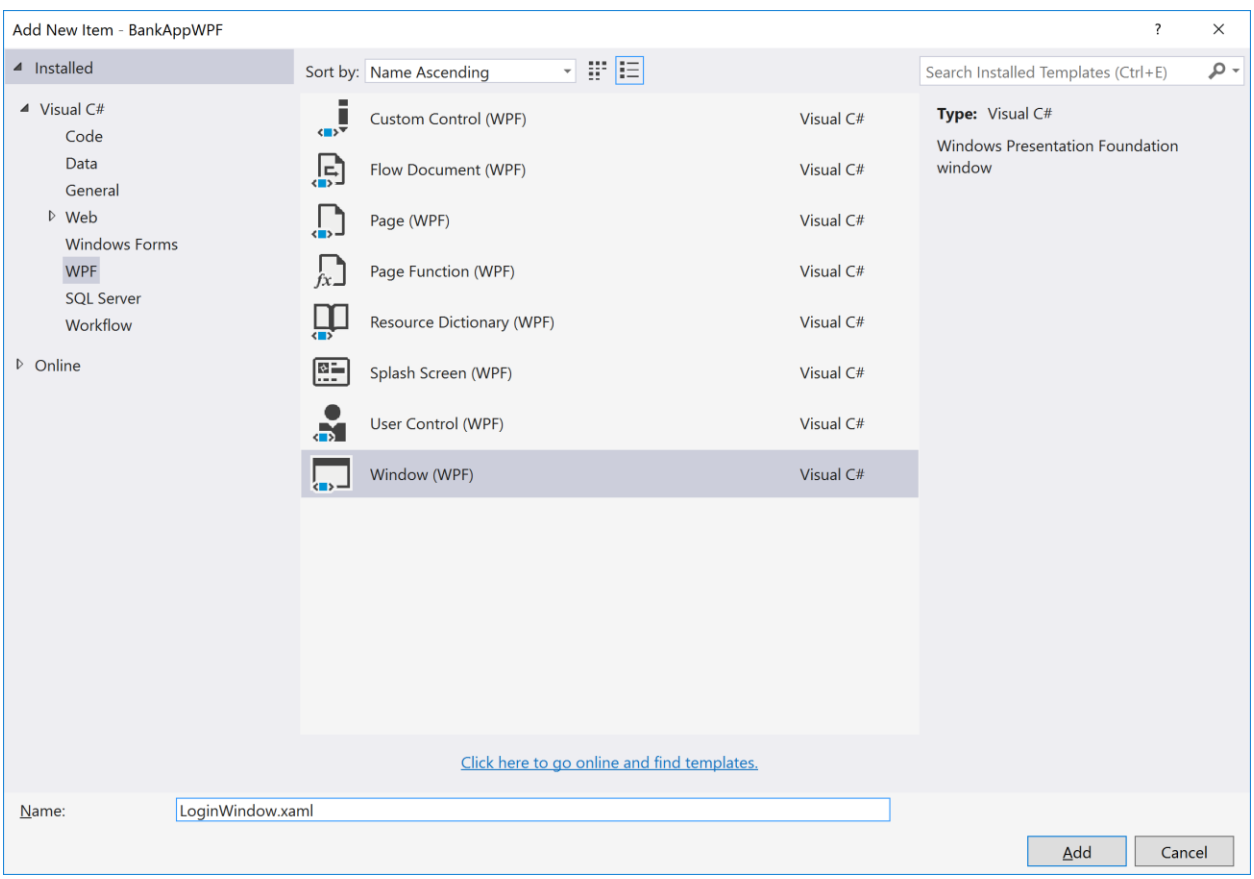
The XML for the main window is shown below. It has two top level menus System and Banking. Under system is the Login sub menu item, and under Banking, the sub menu items are Xfer Check To Sav and Transaction History

```
<Window x:Class="BankAppWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:BankAppWPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="550" Width="525">
    <Grid>
        <DockPanel>
            <Menu x:Name="menu" HorizontalAlignment="Left" Height="100"
VerticalAlignment="Top" Width="100">
            <MenuItem Header="System">
                <MenuItem Header="Login" Click="mnuLoginClick"
HorizontalAlignment="Right" Width="140"/>
```

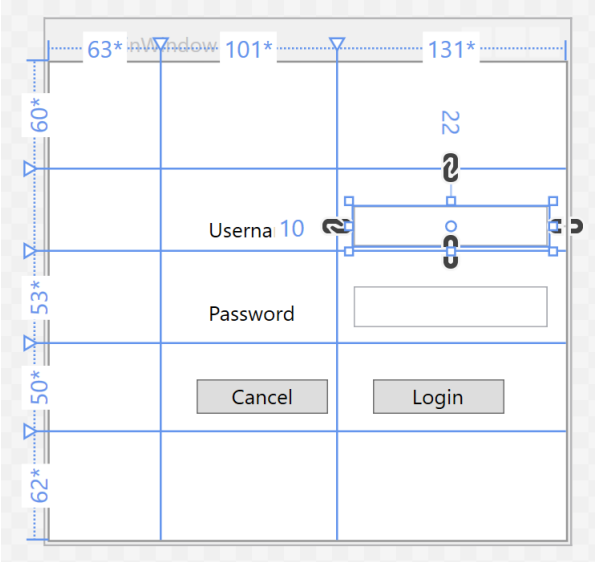
```
            </MenuItem>
        </Menu>
        <Menu x:Name="menu2" HorizontalAlignment="Left" Height="100"
VerticalAlignment="Top" Width="100">
                <Menu.ItemsPanel>
                    <ItemsPanelTemplate>
                        <StackPanel Orientation="Vertical"/>
                    </ItemsPanelTemplate>
                </Menu.ItemsPanel>
                <MenuItem Header="Banking">
                <MenuItem Header="Xfer Check To Sav" Click="mnuTransferCToS"
HorizontalAlignment="Right" Width="190"/>
                    <MenuItem Header="Transaction History" Click="mnuTransactionHistory"
HorizontalAlignment="Right" Width="192"/>
                </MenuItem>
            </Menu>

            </DockPanel>
    </Grid>
</Window>
```

The most of the above XML can be generated from the tool box, but some customization is needed in terms of layouts e.g., the dock panel for putting main menu items side by side and the stack panel with vertical orientation for putting sub menu items vertically.

Add a WPF window called LoginWindow.xaml as shown below.



Design the Login UI to appear as:



You can create a layout grid by clicking on the top portion of the window, and on the left side of the window (see blue lines in above window. Then you can drag and drop the labels, textboxes and buttons on the designer surface as shown above. In WPF, the label has a Content property, instead of the Text property.

You can write even handlers for buttons by double clicking on a button. Note that it writes the link for the button handler code in the xaml.

The code for the LoginWindow.xamls.cs is shown below.

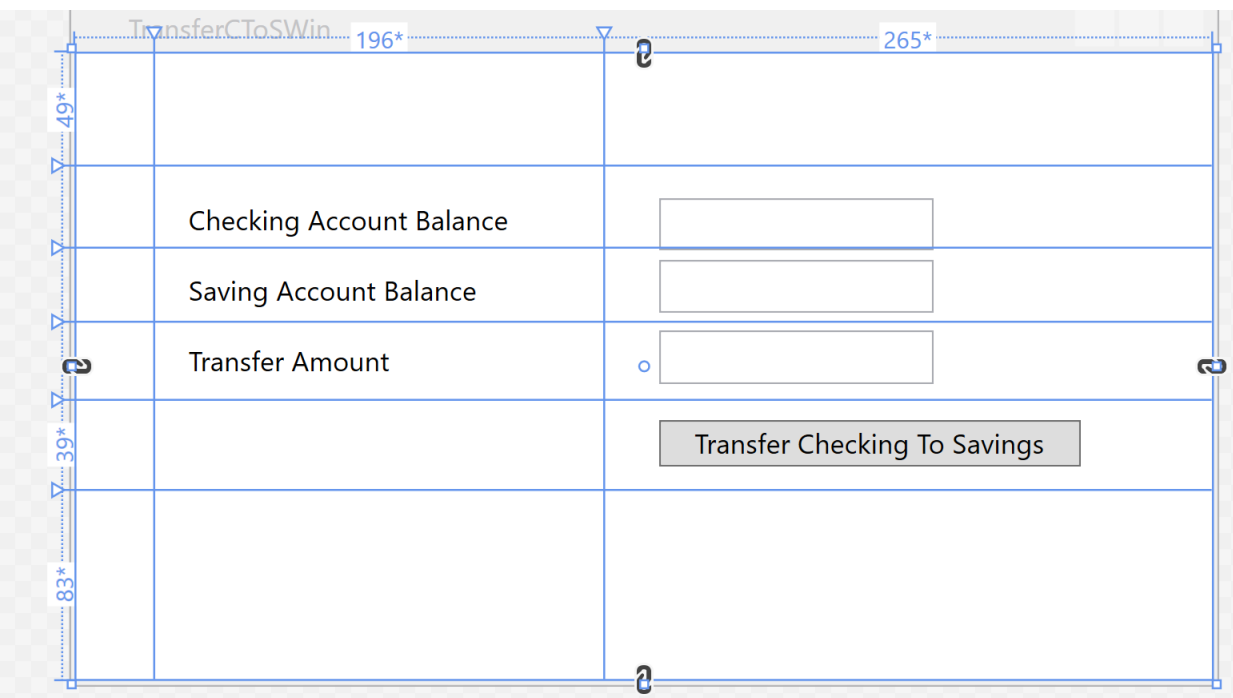
```
public partial class LoginWindow : Window
    {
        public string Username { get { return txtUsername.Text; }  }
        public string Password { get { return txtPassword.Text; } }
        public LoginWindow()
        {
            InitializeComponent();
        }

        private void btCancel_Click(object sender, RoutedEventArgs e)
        {
            this.DialogResult = false;
        }

        private void btnLogin_Click(object sender, RoutedEventArgs e)
        {
            this.DialogResult = true;
        }
    }
```

Now in the main window, double click on the Login menu, the Xfer Check To Sav menu and the Transaction History menu, and write the following code in the MainWindow.xaml.cs.

```
public partial class MainWindow : Window
    {
        string _username = "";
        long _checkingAccountNum = 0;
        long _savingAccountNum = 0;
        IRepositoryAuthentication _irepAuth = new Repository() as
IRepositoryAuthentication;
        IRepositoryBanking _irepBank = new Repository() as IRepositoryBanking;

        public MainWindow()
        {
            InitializeComponent();
        }

        private void mnuLoginClick(object sender, RoutedEventArgs e)
        {
            LoginWindow loginWin = new LoginWindow();
            if (loginWin.ShowDialog() == true)
            {
                try
                {
                    bool res = _irepAuth.CheckIfValidUser(loginWin.Username,
loginWin.Password);
                    if (res == true)
                    {
                        _username = loginWin.Username;
                        // obtain checking account and saving account infor for user
                        _checkingAccountNum =
_irepBank.GetCheckingAccountNumForUser(_username);
                        _savingAccountNum =
_irepBank.GetSavingAccountNumForUser(_username);
                        MessageBox.Show("Welome " + loginWin.Username);
                    }
                    else
                        MessageBox.Show("Invalid Login..");
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void mnuTransferCToS(object sender, RoutedEventArgs e)
        {
            if (_username == "")
            {
                MessageBox.Show("You must login first..");
                return;
            }
```

```
            TransferCToSWin tCToSwin = new
TransferCToSWin(_username,_checkingAccountNum,_savingAccountNum);
            tCToSwin.Show();
        }

        private void mnuTransactionHistory(object sender, RoutedEventArgs e)
        {
            if (_username == "")
            {
                MessageBox.Show("You must login first..");
                return;
            }
            try
            {
                TransactionHistoryWin thWin = new
TransactionHistoryWin(_username,_checkingAccountNum,_savingAccountNum);
                thWin.Show();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

    }
}
```

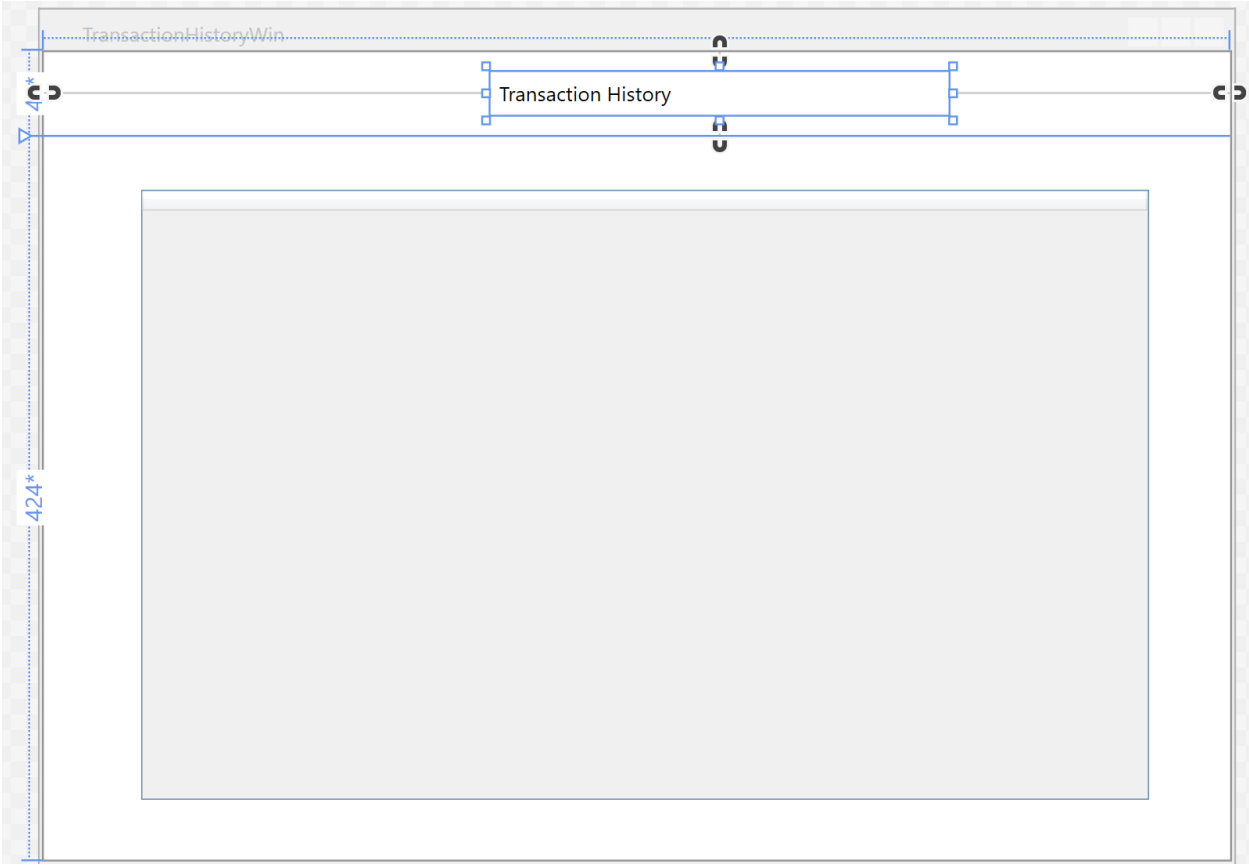
Add another WPF window called TransferCToSWin.xaml. Then create the following UI for it.



Double click on the Transfer Checking To Savings button. The code for the TransferWinCToS.xaml.cs is shown below.

```
public partial class TransferCToSWin : Window
    {
        long _checkingAccountNum = 0;
        long _savingAccountNum = 0;
        string _username = "";
        IRepositoryBanking _irepBank = new Repository() as IRepositoryBanking;

        public TransferCToSWin(string username, long checkingAcctNum, long
savingAcctNum)
        {
            InitializeComponent();
            _checkingAccountNum = checkingAcctNum;
            _savingAccountNum = savingAcctNum;
            _username = username;
            txtCheckingBalance.Text =
_irepBank.GetCheckingBalance(_checkingAccountNum).ToString();
            txtSavingBalance.Text =
_irepBank.GetSavingBalance(_savingAccountNum).ToString();
        }

        private void btnTransfer_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                bool ret = _irepBank.TransferCheckingToSaving(_checkingAccountNum,
_savingAccountNum,
                    decimal.Parse(txtTransferAmount.Text),0);
                if (ret == true)
                {
                    txtCheckingBalance.Text =
_irepBank.GetCheckingBalance(_checkingAccountNum).ToString();
                    txtSavingBalance.Text =
_irepBank.GetSavingBalance(_savingAccountNum).ToString();
                    MessageBox.Show("Transfer successful..");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
```

Add another WPF window to the project called TransactionHistoryWin.xaml with the following UI.



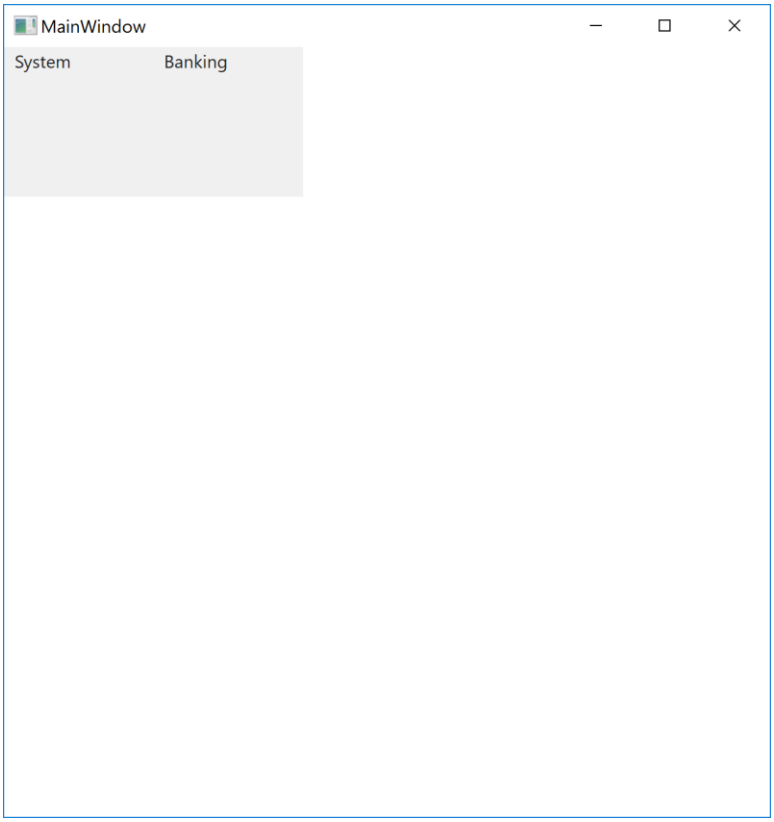
The gray box in the above window is the datagrid control.

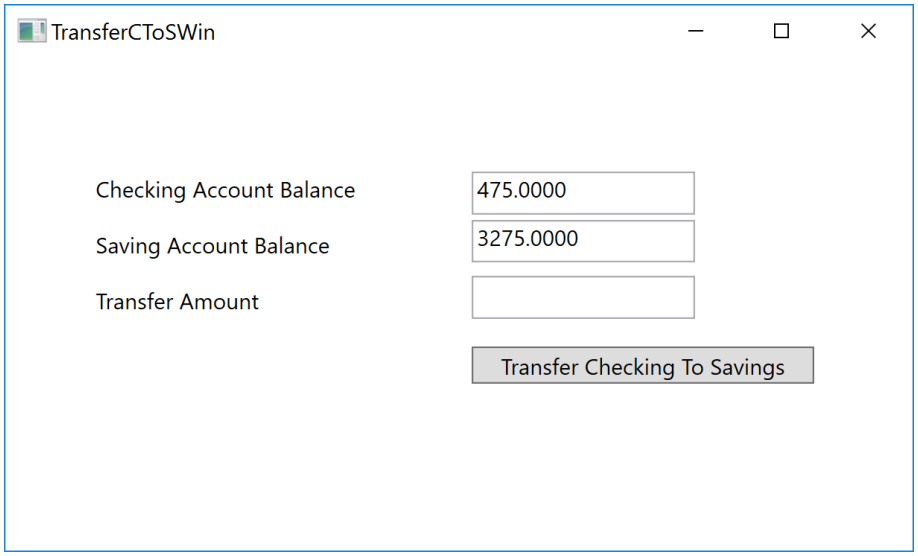
The code for the TransactionHistory.xaml.cs is shown below.

```
public partial class TransactionHistoryWin : Window
    {
        long _checkingAccountNum = 0;
        long _savingAccountNum = 0;
        string _username = "";
        IRepositoryBanking _irepBank = new Repository() as IRepositoryBanking;

        public TransactionHistoryWin(string username, long checkingAccountNum, long
savingAccountNum)
        {
            InitializeComponent();
            _username = username;
            _checkingAccountNum = checkingAccountNum;
            _savingAccountNum = savingAccountNum;
            List<TransactionHistoryModel> THList =
_irepBank.GetTransactionHistory(_checkingAccountNum);
            dgTransHist.ItemsSource = THList;
        }
    }
```

Build and test the application. Once you login, you can transfer from checking to savings and view your transaction history.

Note: To keep the example smple, the business layer was not used.





TransactionHistoryWin

Transaction History

CheckingAccountNumber	SavingAccountNumber	Amount	TransactionFee	TransactionDate	Tran
10000	100000	5.0000	0.0000	11/27/2016 4:44:55 PM	Chec
10000	100000	15.0000	0.0000	11/27/2016 5:31:03 PM	Chec
10000	100000	25.0000	0.0000	11/27/2016 9:30:45 PM	Chec
10000	100000	35.0000	0.0000	11/27/2016 9:38:07 PM	Chec
10000	100000	45.0000	0.0000	11/27/2016 9:39:30 PM	Chec
10000	100000	16.0000	0.0000	11/28/2016 11:39:42 PM	Chec
10000	100000	20.0000	0.0000	12/11/2016 7:16:44 PM	Chec
10000	100000	15.0000	0.0000	12/11/2016 7:54:50 PM	Chec