#### Use of an IViewModel

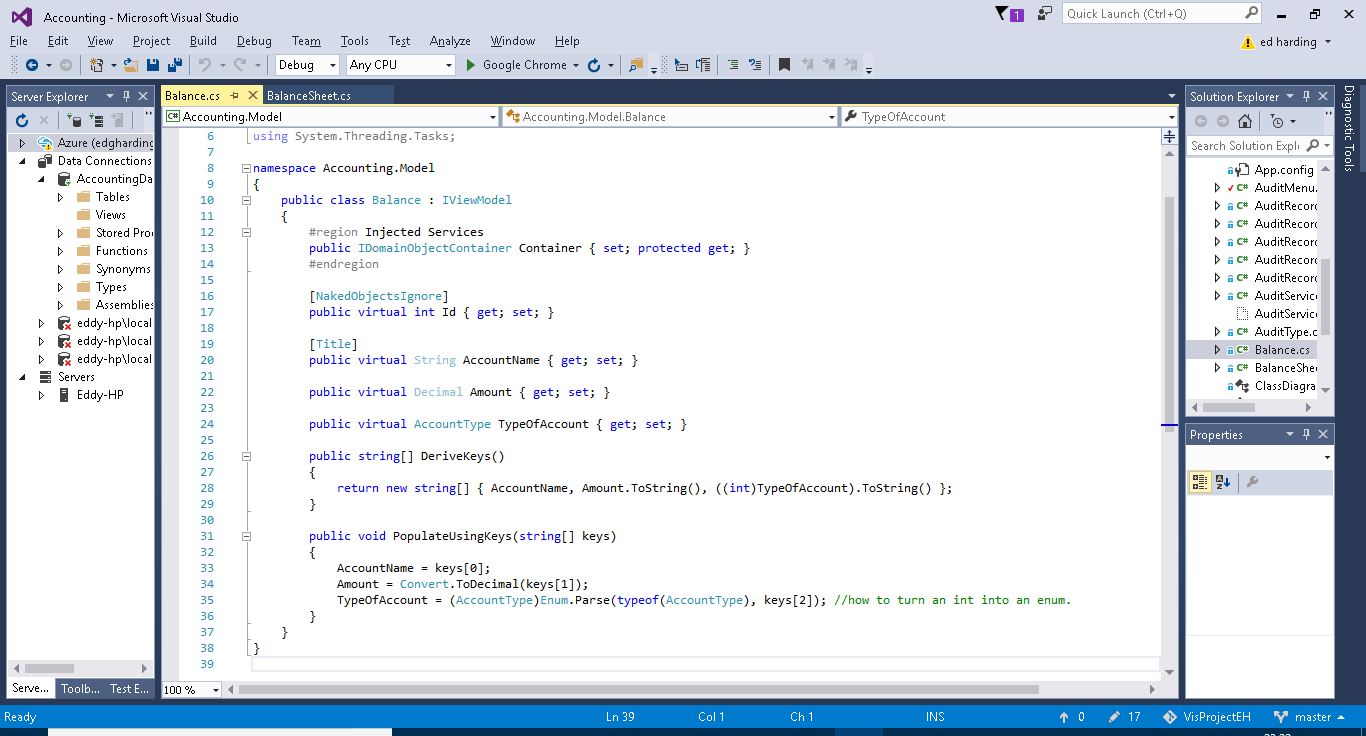
For creating the balance sheet, I needed to get the balance for each account. The balance of an account depends on the different credit and debit transactions performed on it. I could simply have accessed the accounts balance directly from the balance sheet class, however I saw 2 main advantages to using a view model class instead.

First, I will explain what a view model is and then I will move on to the 2 reasons I decided to use one. A view model is a class which appears to the user through the GUI as if it was a persisted object but is not.

1. The fact that a view model is not persisted means that it must instead get the information from somewhere else, in my case this is from the Accounts class, the advantage of this is that the balance sheet is based between two dates, therefore the balances need to be based on the balance of the accounts between the two dates. However, it is of course possible to add transactions (which effect an accounts balance) with past dates. Therefore, if a balance sheet is created and then a transaction is added which has a date between the time frame of the balance sheet, if the balances r persisted objects the balance sheet will not update and will therefore be inaccurate. Therefore, using a view model causes the balances to be created live every time the balance sheet is opened (how this works will be explained later on).
2. The other advantage of using a view model is that it allows for an increased level of customisation of how the balance sheet is displayed to the user.

These advantages are demonstrated in the code break down which follows.

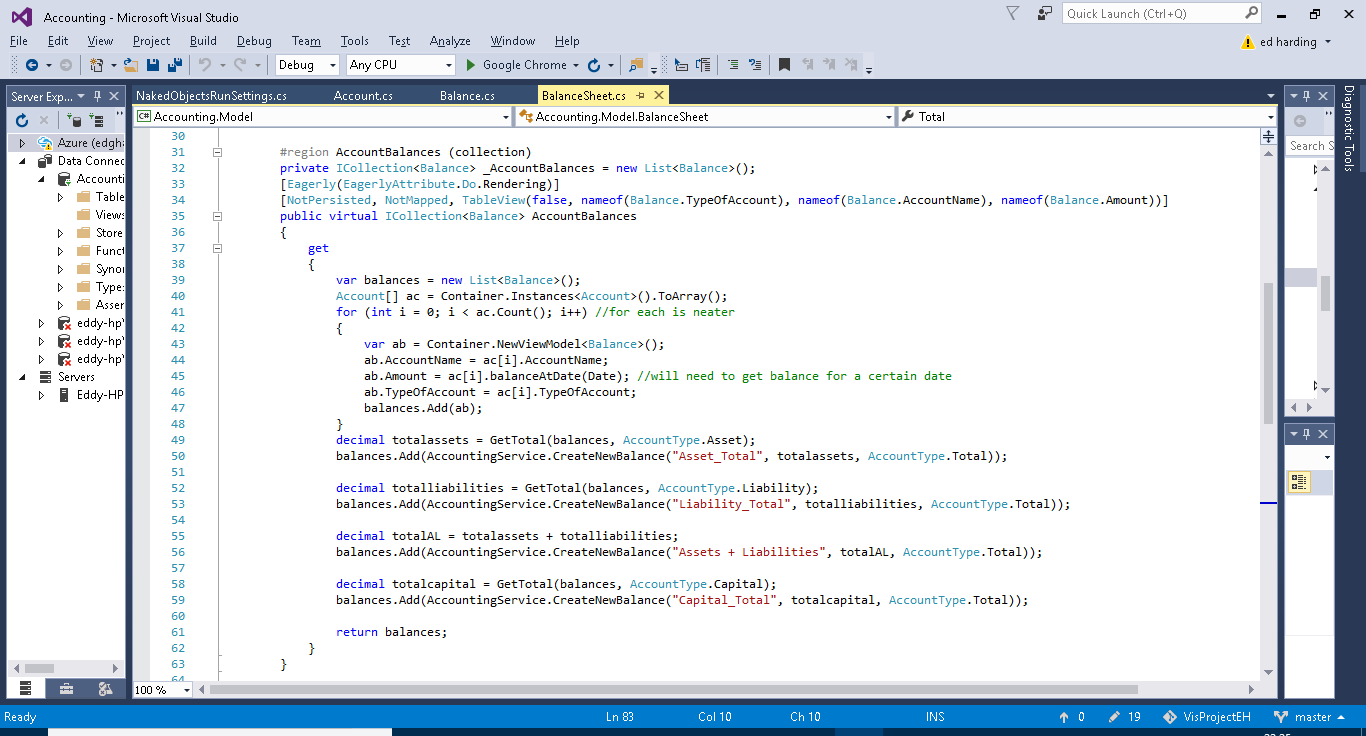
Below is the code for the Balance View model, as you can see it ‘copies’ some of the data held in the accounts class. It achieves this by using two method DeriveKeys and PopulateKeys, I will go into more detail of what they do below.



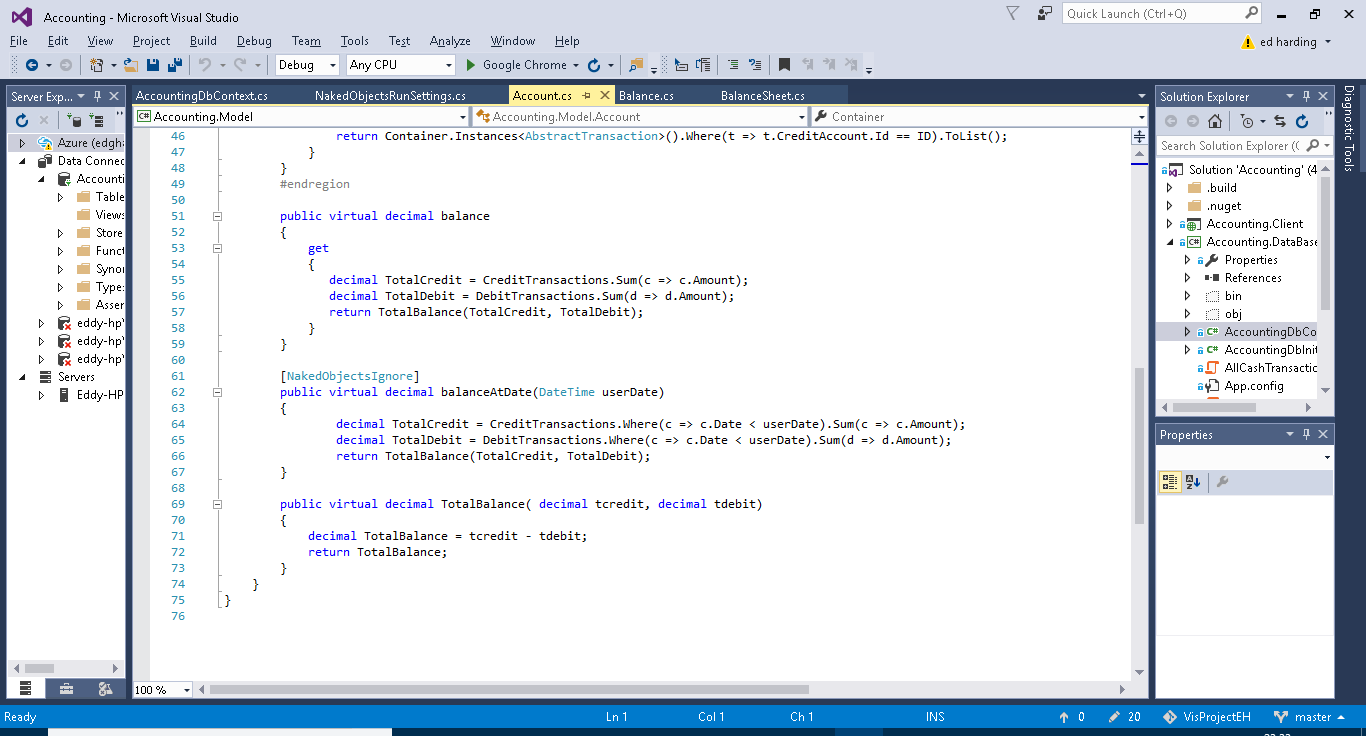
DeriveKeys: This method returns a string array containing the keys derived from the object which the view model represents, so in this case it contains three of the properties from the Account class; AccountName, Amount and TypeOfAccount.

Populate Keys: This method populates the properties of the View Model with the keys which are retrieved by the DeriveKeys method. As you can see the Balance AccountName is being set to keys[0] which in the string of keys returned by DeriveKeys is the AccountName gotten from the Account Object. This is how the View Model is able to be created live every time it is called upon.

The below screenshots are of the balance sheet class and show the use of the balances view model inside the balance sheet class.

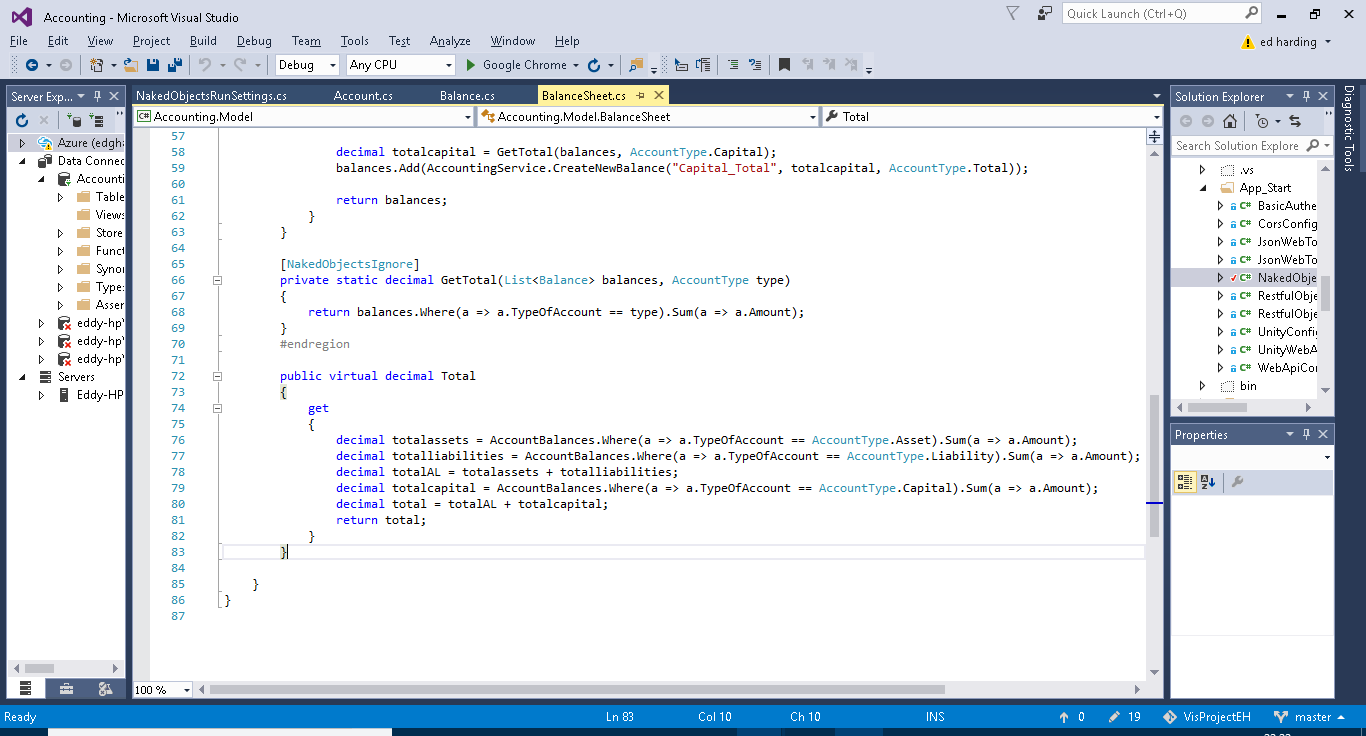


Here you can see a collection of balnces being created, I chose to use a collection as it would be able to create a nice ordered table of the accounts (as the balnces class represents the accounts class), this provides and easily readable and follows the typical format of a balance sheet. As you can see above (line 39 to 48) a new list of balances is being created and the balnce of each individual account is being assigned to a new instance of balance every time until this has been done for all of the accounts. The balnceAtDate method comes from Accounts and allows for the balnance of the account to be calculated at a specific date, the method is shown below, this like a lot of my methods is done using LINQ queries to find the data required.(back to the above screenshot) I am then creating 4 custom balances and adding them to the list of balnces. These custom balances are each totals which are used for calculating the overall balance of the balance sheet (Assets +liabilites – Capital), this is done using the GetTotal method which is showna dne xlained below. Again thanks to the View model and that overall the whole thing is a collection, these will be included in the table of accounts (balances view model).



Below (lines 65 to 69) is a method for getting the total balances of all of the accounts of a certain type, it is pretty straight forward and uses a LINQ query to do this.

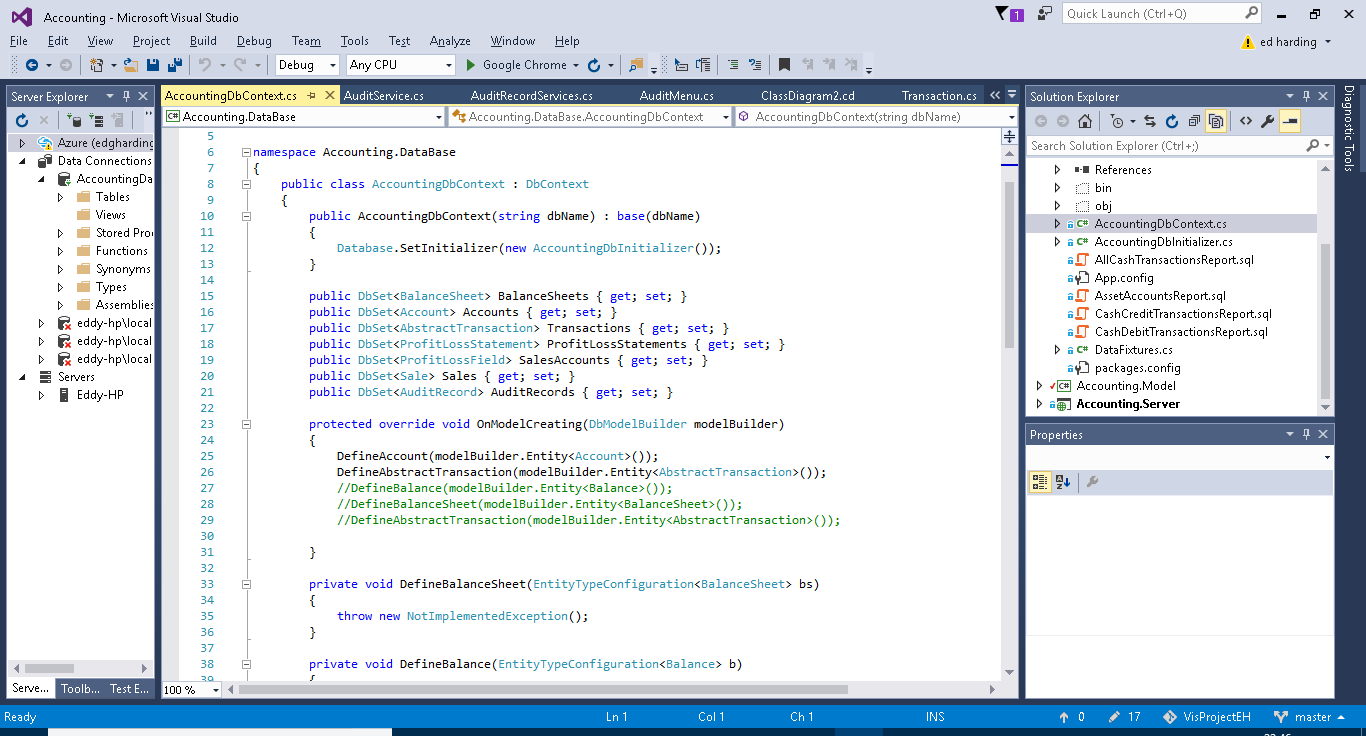
Below (lines 72 to 83) is the final part of the balance sheet the total. If a double entry bookkeeping system has been programmed and used correctly the total of a balance sheet (which uses the equation total assets + total liabalities + total capital) should always = 0. Below is simply the method I created which calculates this from the totals balances which I mentioned earlier.



#### DB context

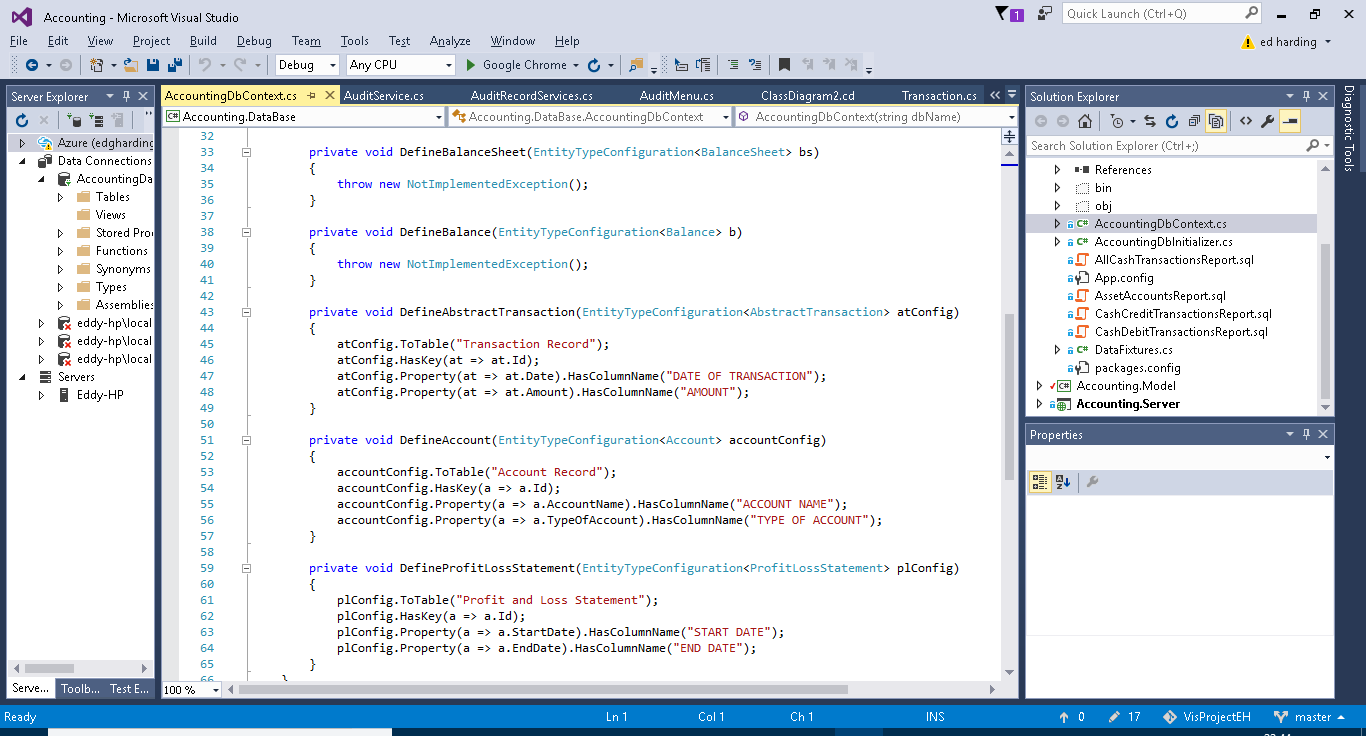
The DB Context class is used for specifying which classes need to be persisted and it is used for adding methods which allow for the customisation of the tables created by entity framework.

On lines 10 to 13 below there is a method which just tells the server which Db initializer is being used, this is because you could theoreitcally write lots of different ones and then call the one you want by changing these lines of code.



Above from line 15 to 21 are a list of the different classes which need to be persisted inside the database, this basically just points out the classes to entity framework so it knows which ones to model. It also allows for instances of these classes to be created from inside the DB initializer.

Above from line 23 to 31 is the method which overides the entity framework tables with the methods listed inside it, these methods are shown and explained below. The commented out lines are methods show below on lines 33 to 41,as you can see I did not impliment anything here and therefore have commented them out in the overide method.



Lines 43 to 65 contains 3 methods, each do the same thing just for different objects. As mentioned previously these methods are for customization the tables which entity framework creates. These methods simply help to format the tables created, as you can see the key for the tables is being set to the ID of the class, and then the statements following that are just giving the columns more appropriate names. I mainly did this for the use of the SQL statements, it means that when a table is returned from a SQL report it will have more user-friendly column names etc.