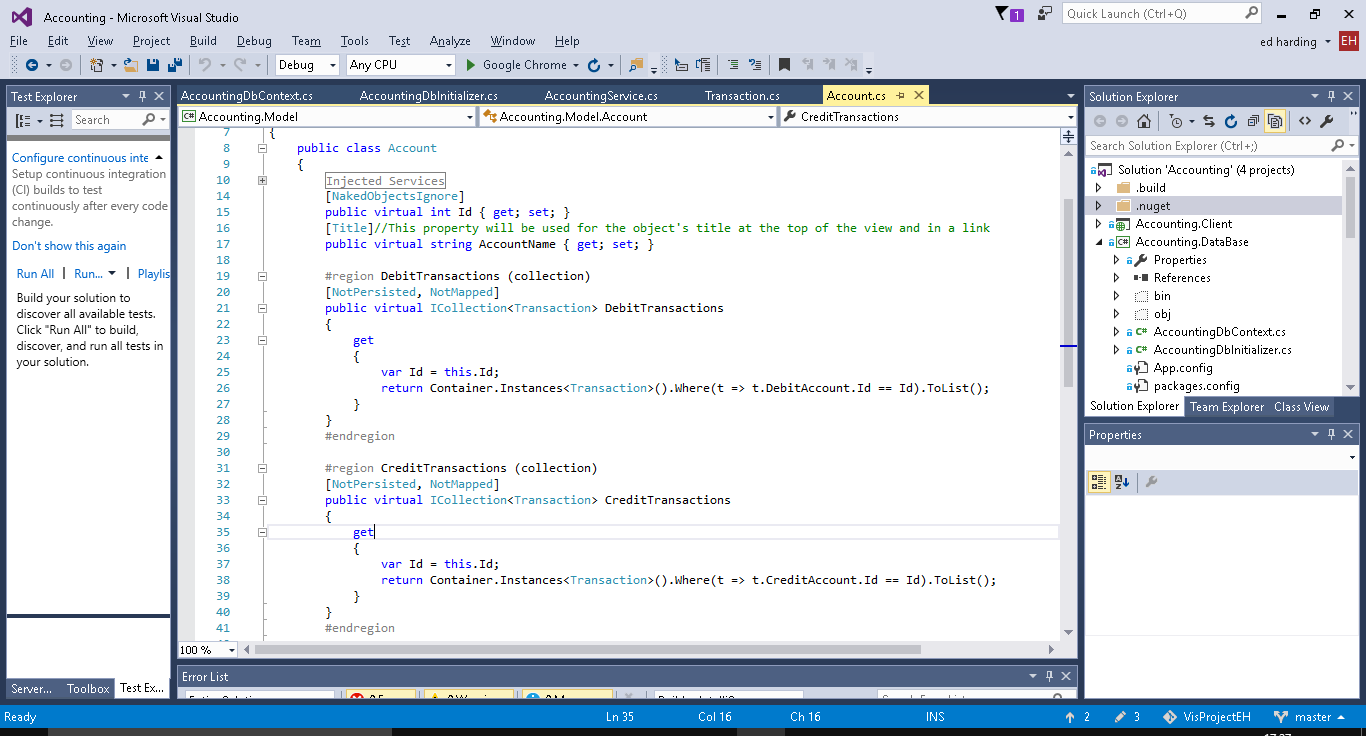
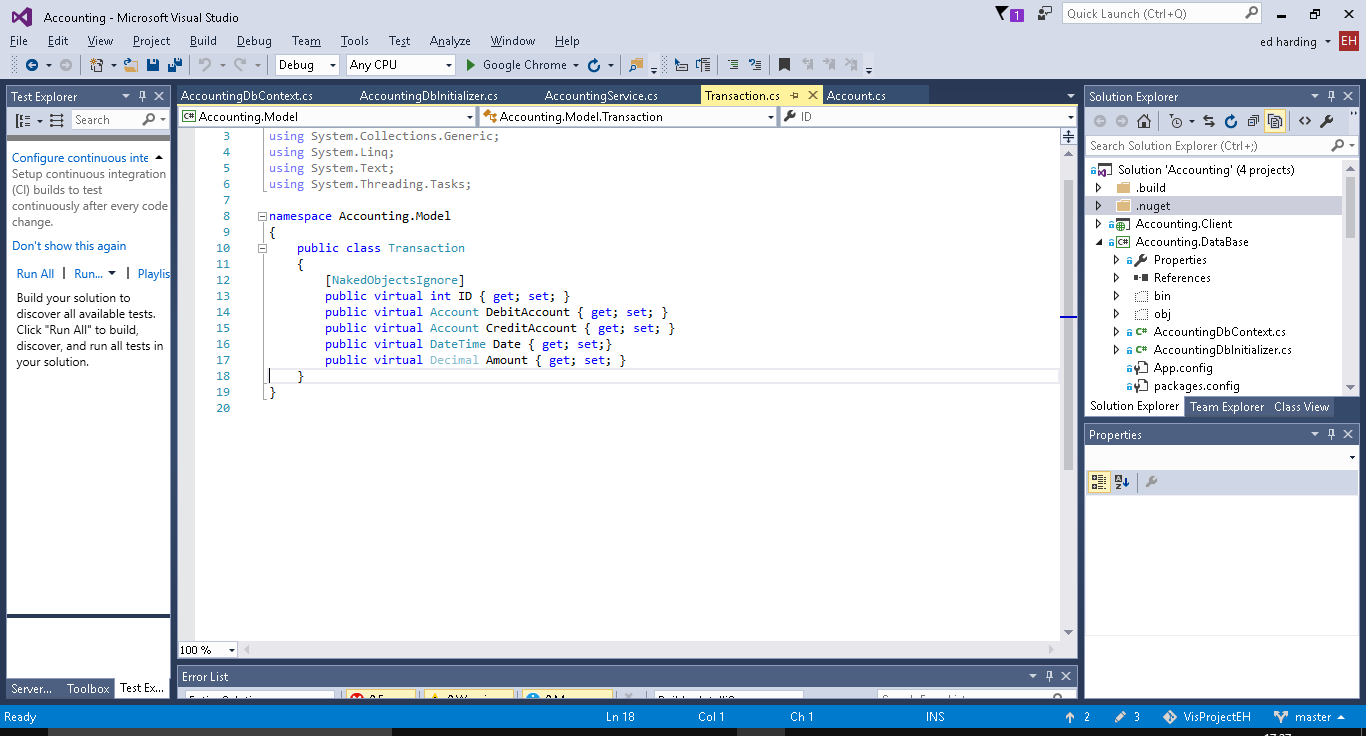
## 1

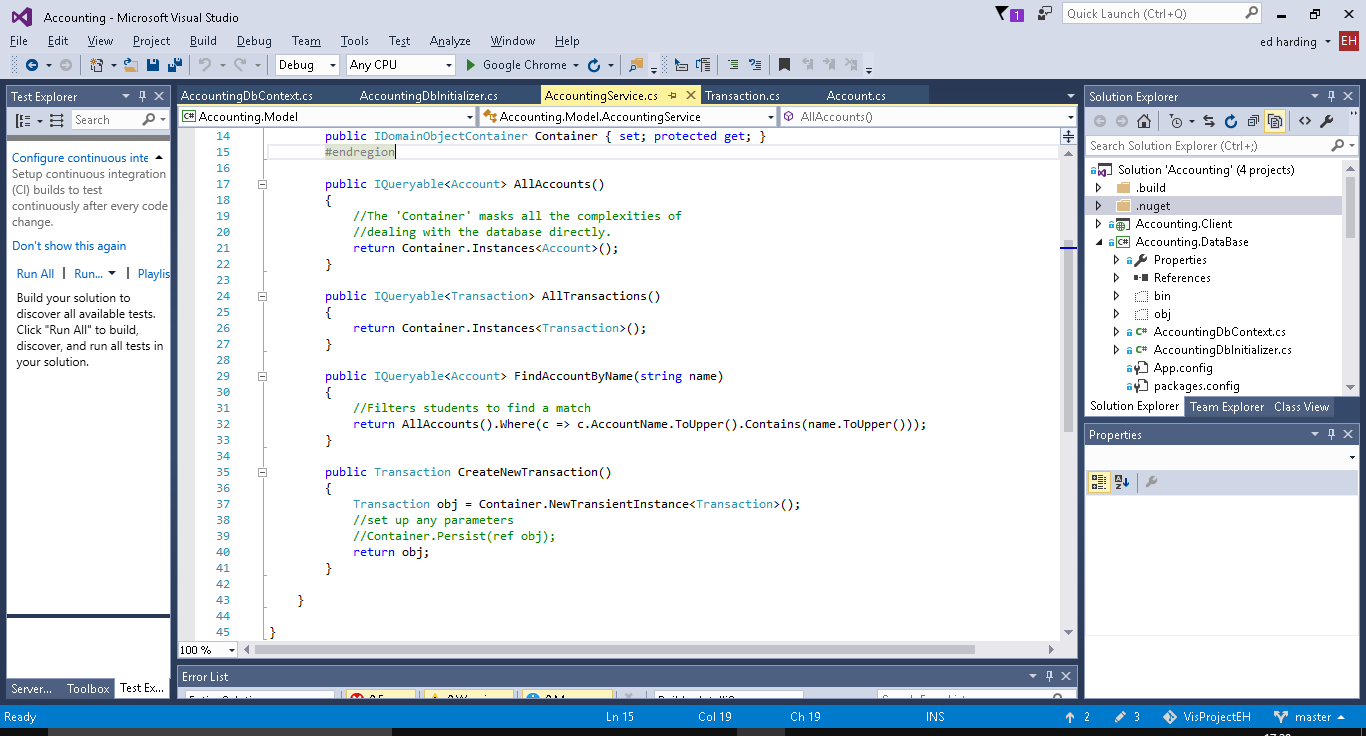
For iteration 1 of my program I would like to create two classes to be used in the application, one being an Account class and one being a transaction class. The transaction class will need to contain information about the date of the transaction, the amount of money, and the debit account and the credit account it is affecting. The Account class will need to have a name and a list of the individual debit and credit transactions enacting onto it. This is my first goal as it will allow for transactions and accounts to be created, and the transactions will record both a debit and credit alteration to separate accounts which is the basis of double entry book keeping.



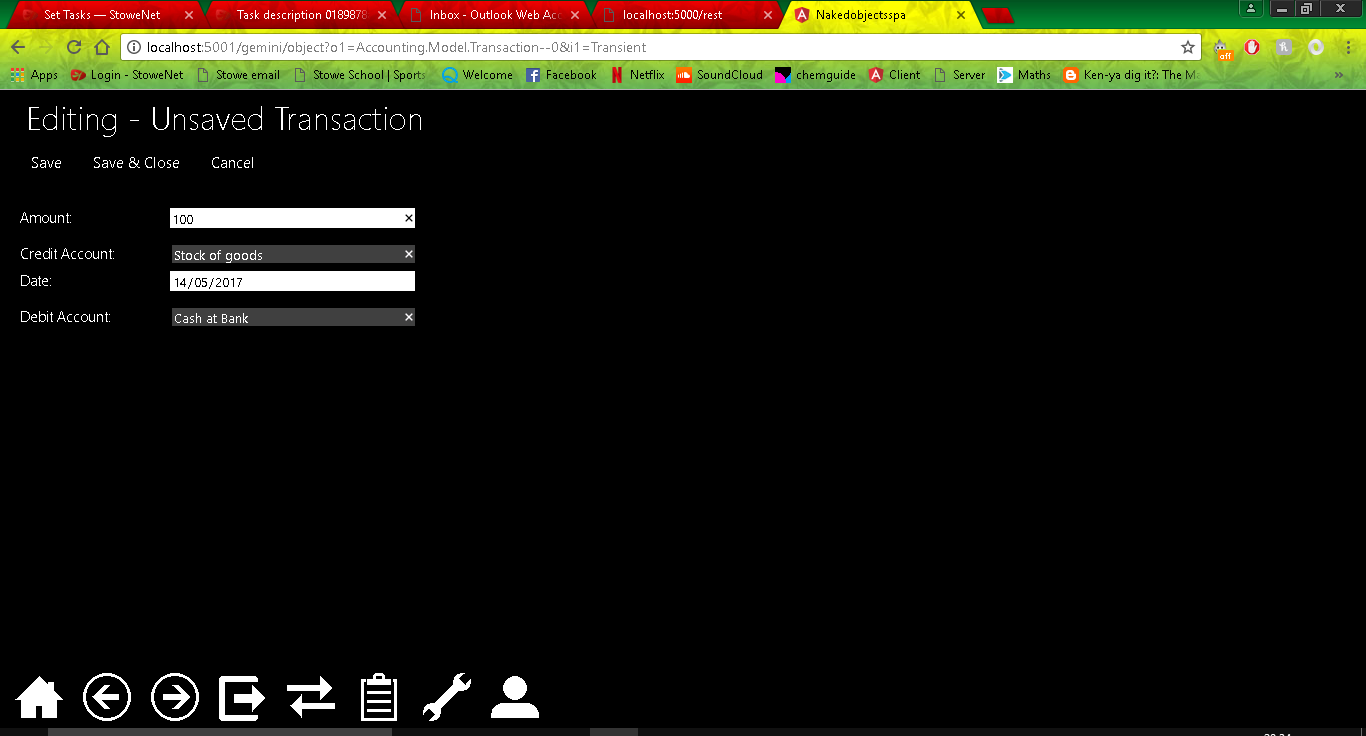
Above is the Account class, as well as the Id property which all classes have to have in Naked Objects it has an AccountName property and 2 collect properties which hold a list of debit and/or credit transactions (which are instances of the Transaction class below).

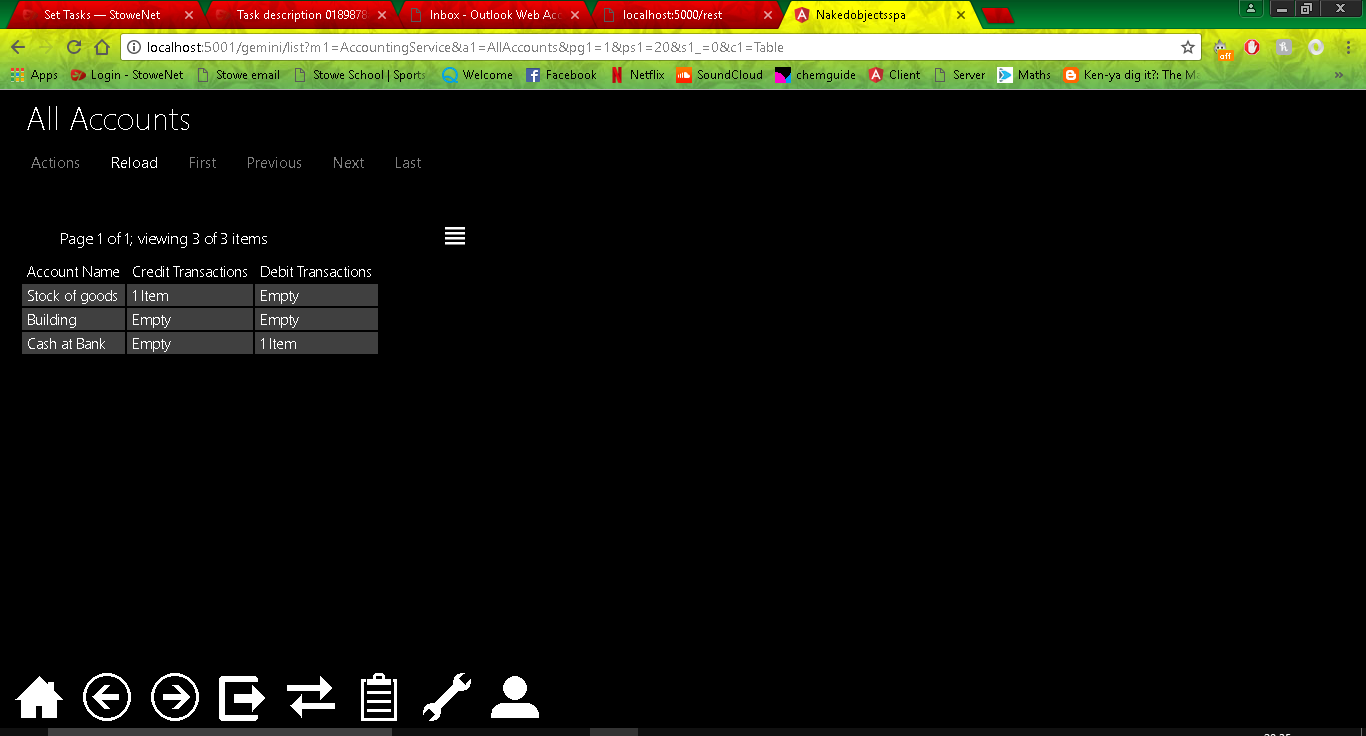


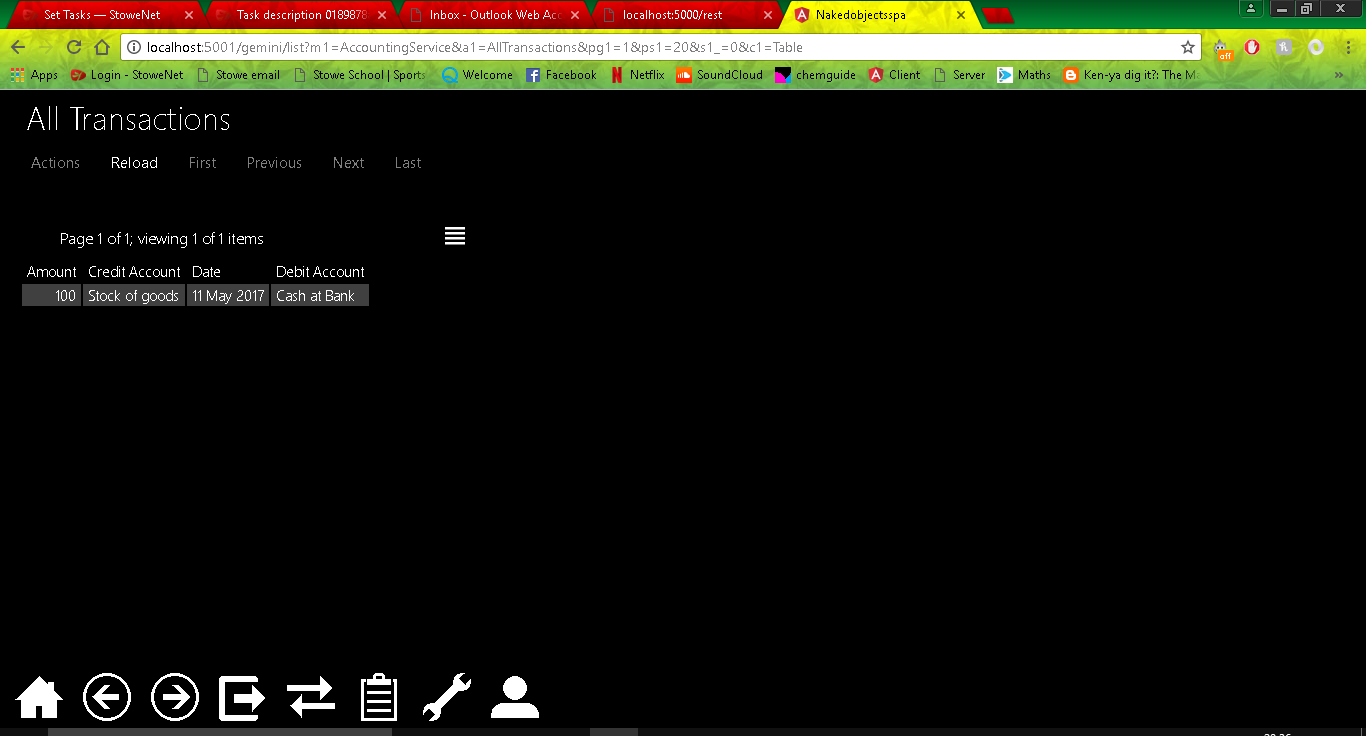
Above is the Transaction class which contains properties for the amount of money, the Date, and then the accounts it is affecting.

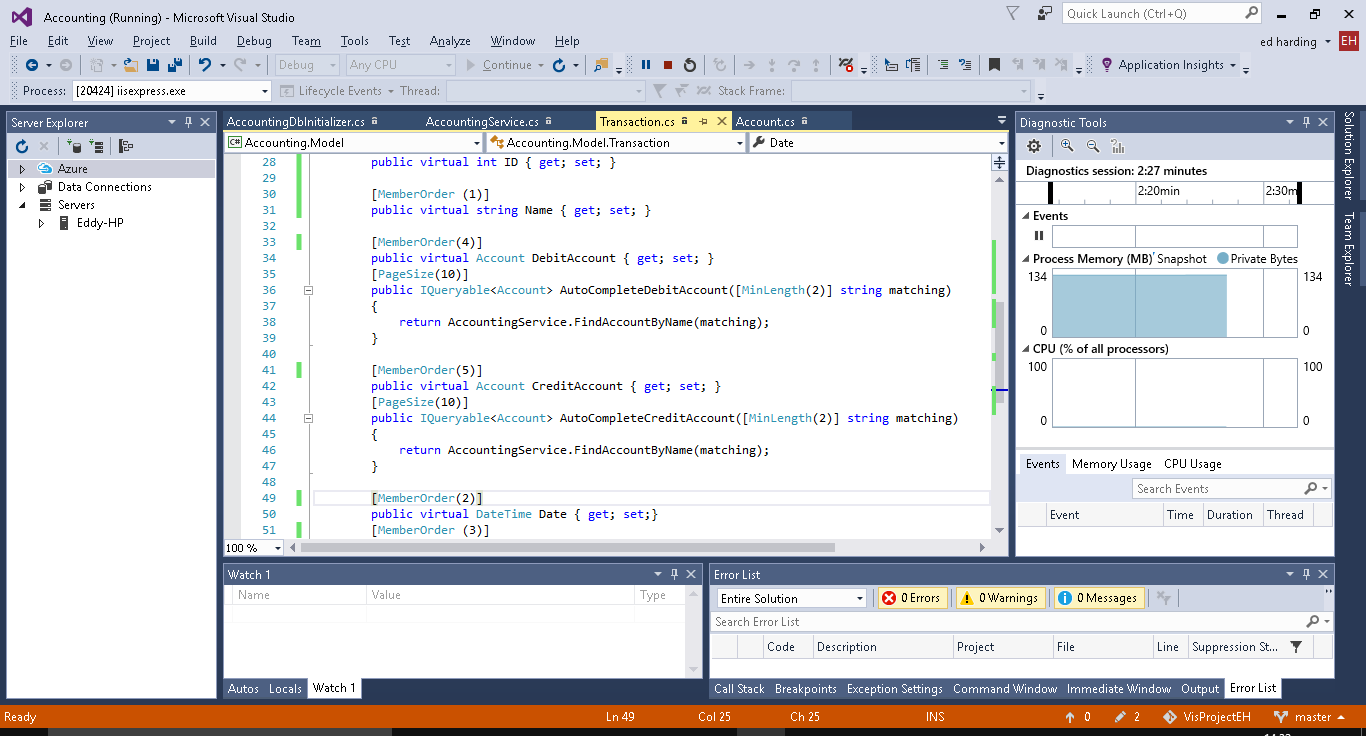


Above are the different methods (services) which can be used on both/either the transaction and/or account classes. Examples of them being used in the User interface will be shown below, they all do what you would expect based on their names.

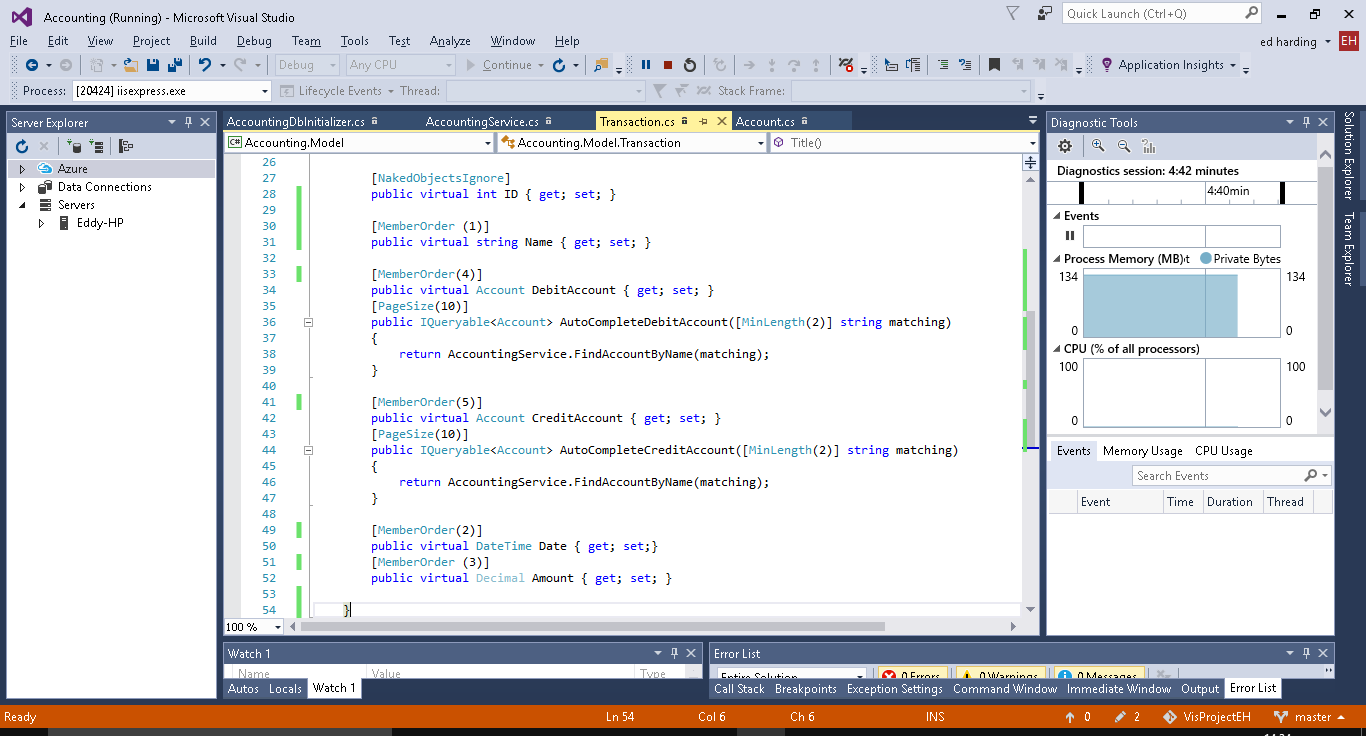




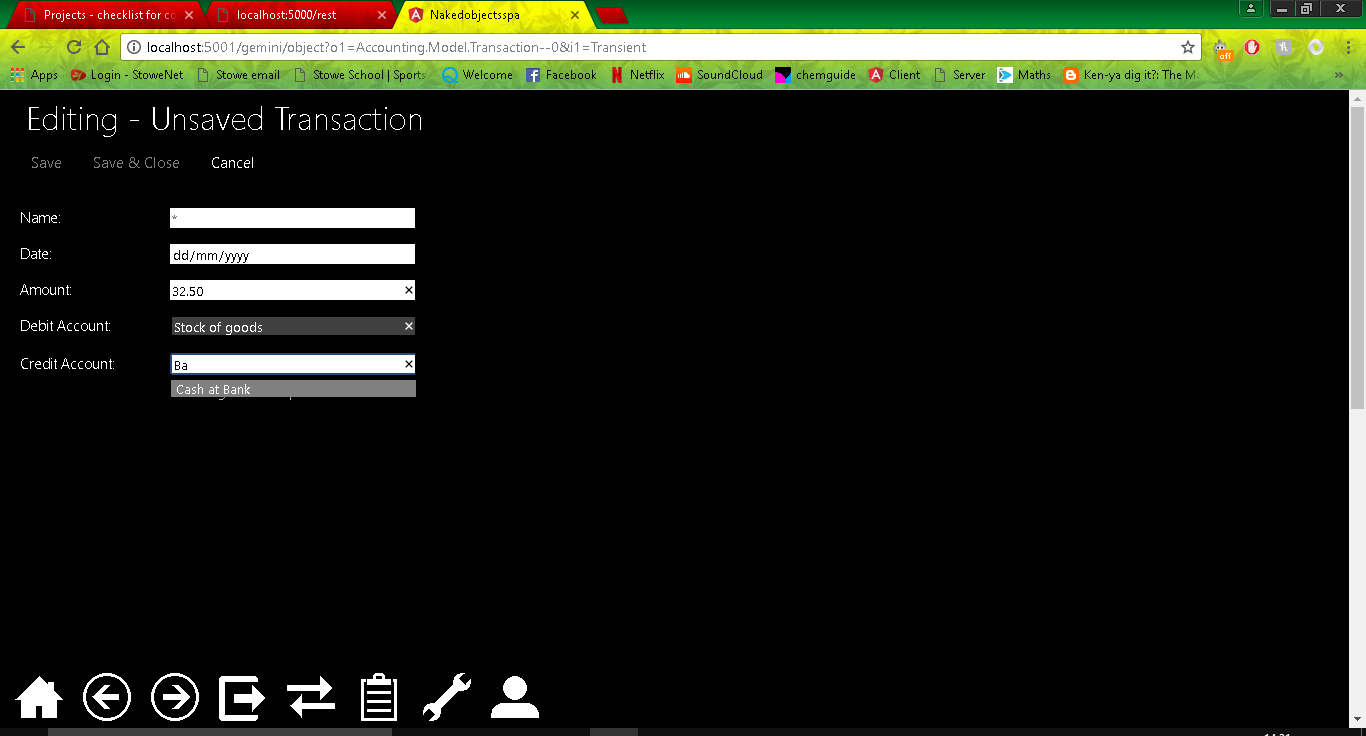


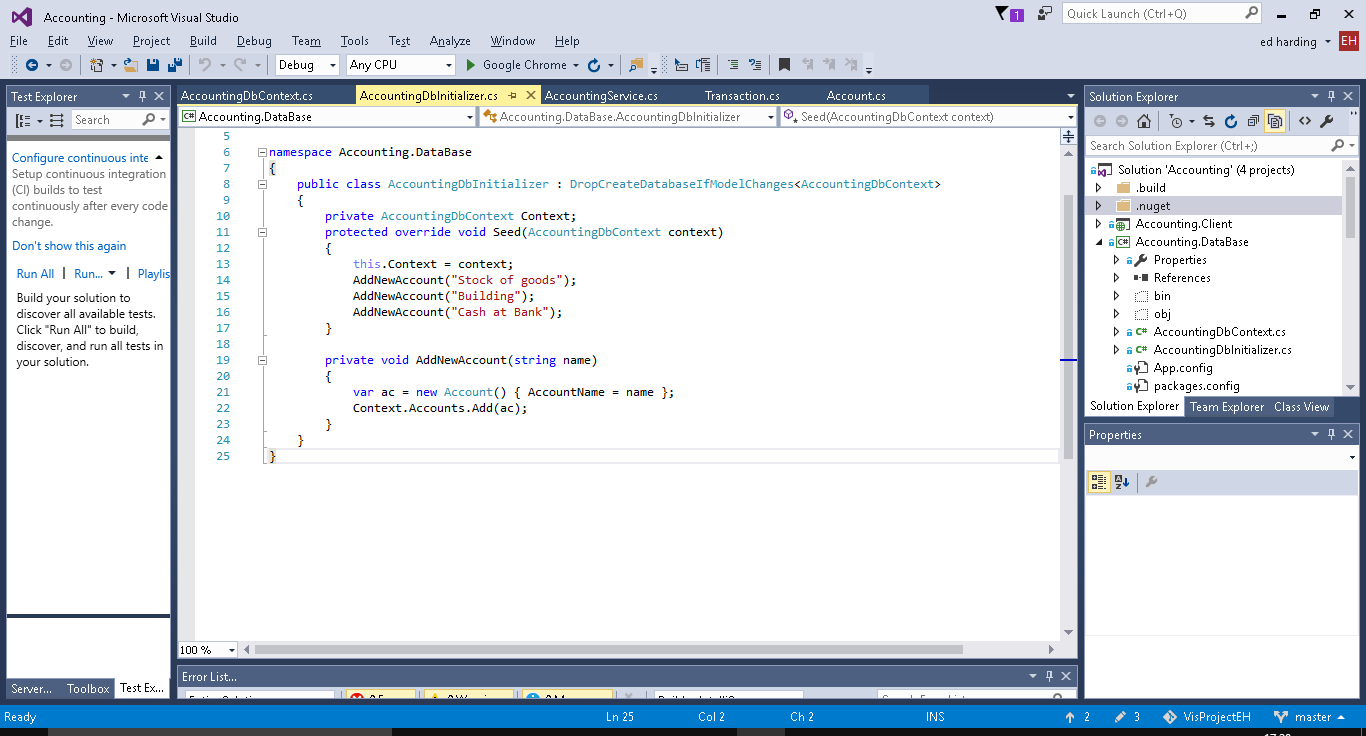
I decided to add in auto complete methods to the Debit and Credit Account fields when creating a transaction this was just to increase usability of the program as it means that you will not have to drag and drop, this will also help if there are a lot of accounts. 

I also decided to add member order to the properties of the Transaction class this will order the CreateNewtransaction fields as numbered in the code above, as with the previous change this is not necessary but it is for purely presentation purposes as before this the Account fields were separated by the Date field. I also added a new property called Name to be used as a title for the transactions as when in AllTransactions and not in table view each transaction was called Untitled.



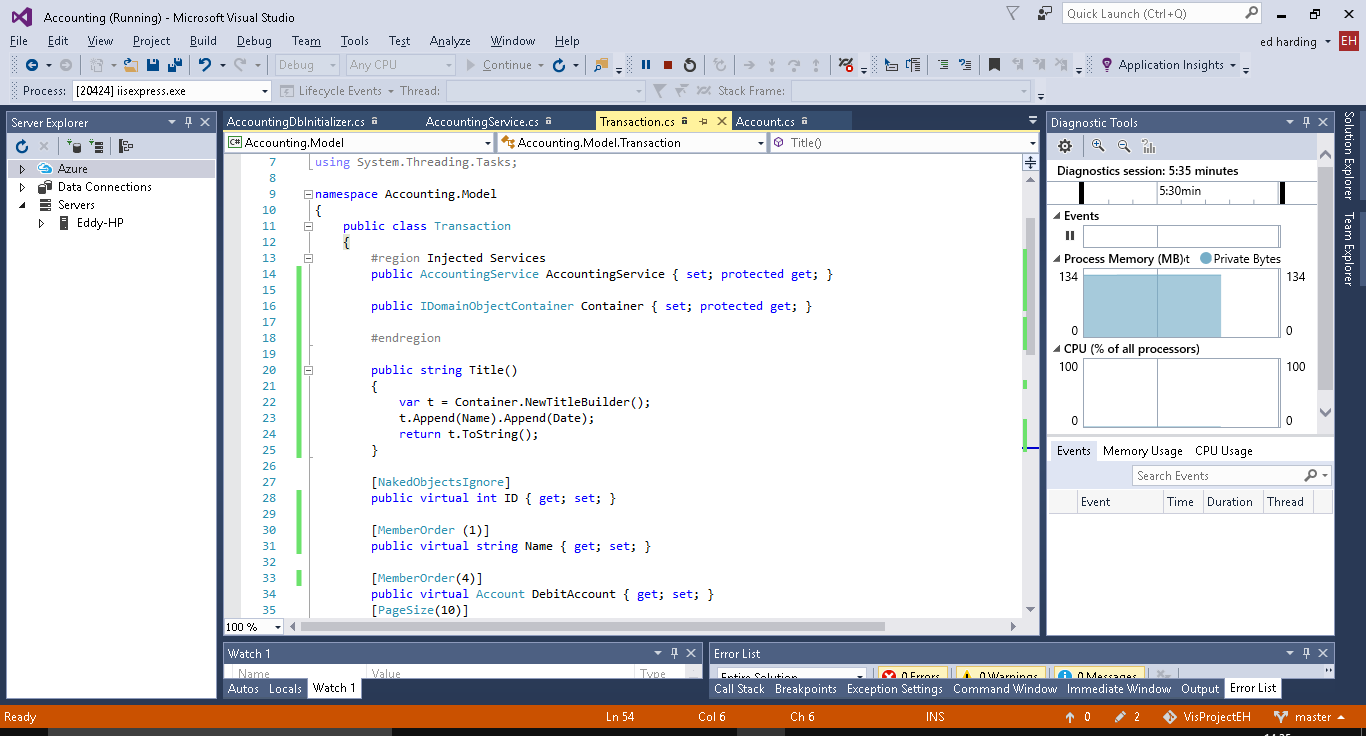
In the screenshot below you can see that that there is a new property called Name, the properties are ordered as to the member order and the autocomplete methods for the Debit and credit accounts is working.

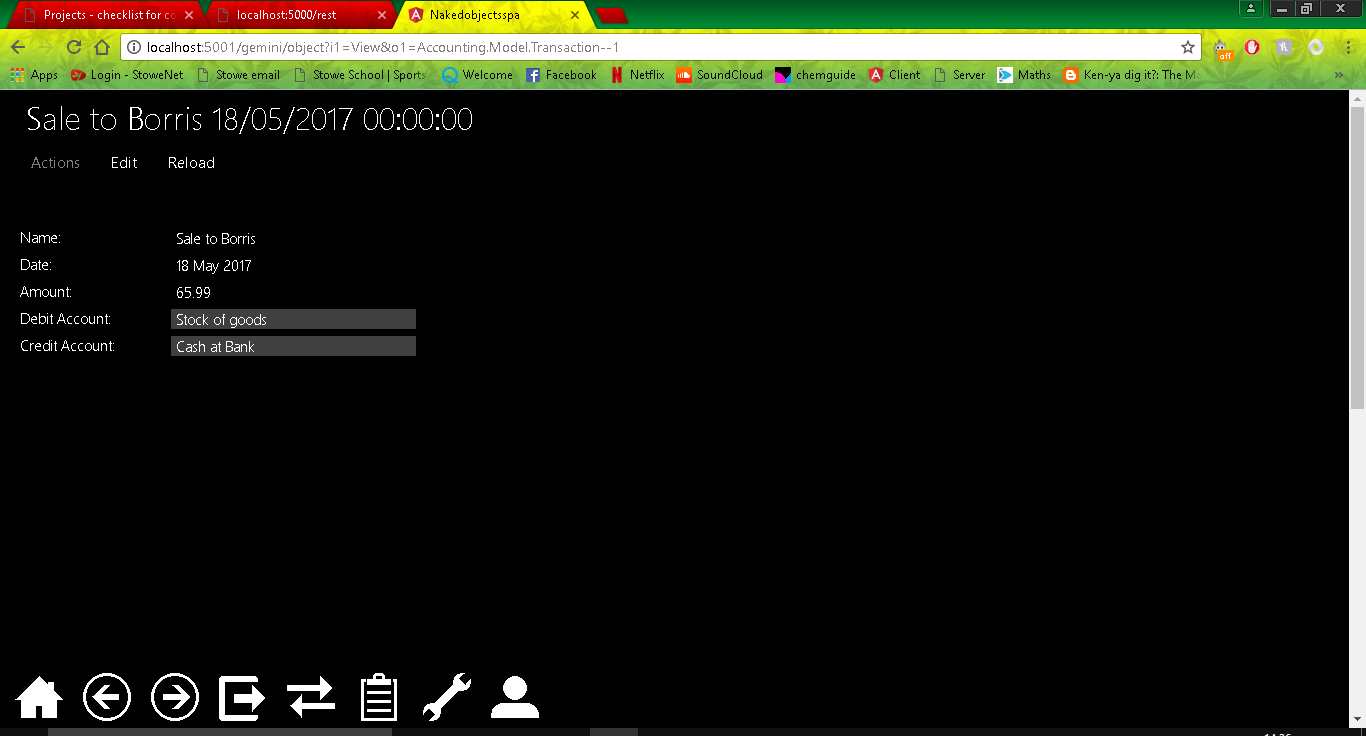




Above is some seed data for some Accounts and the method used to create them, this is in the DB initializer.

I then decided to create a better title to be used for the Transactions, I used a Title Builder method inside the transaction class to create a title by combining the Name and Date properties, again this is just to add additional usability to the software.



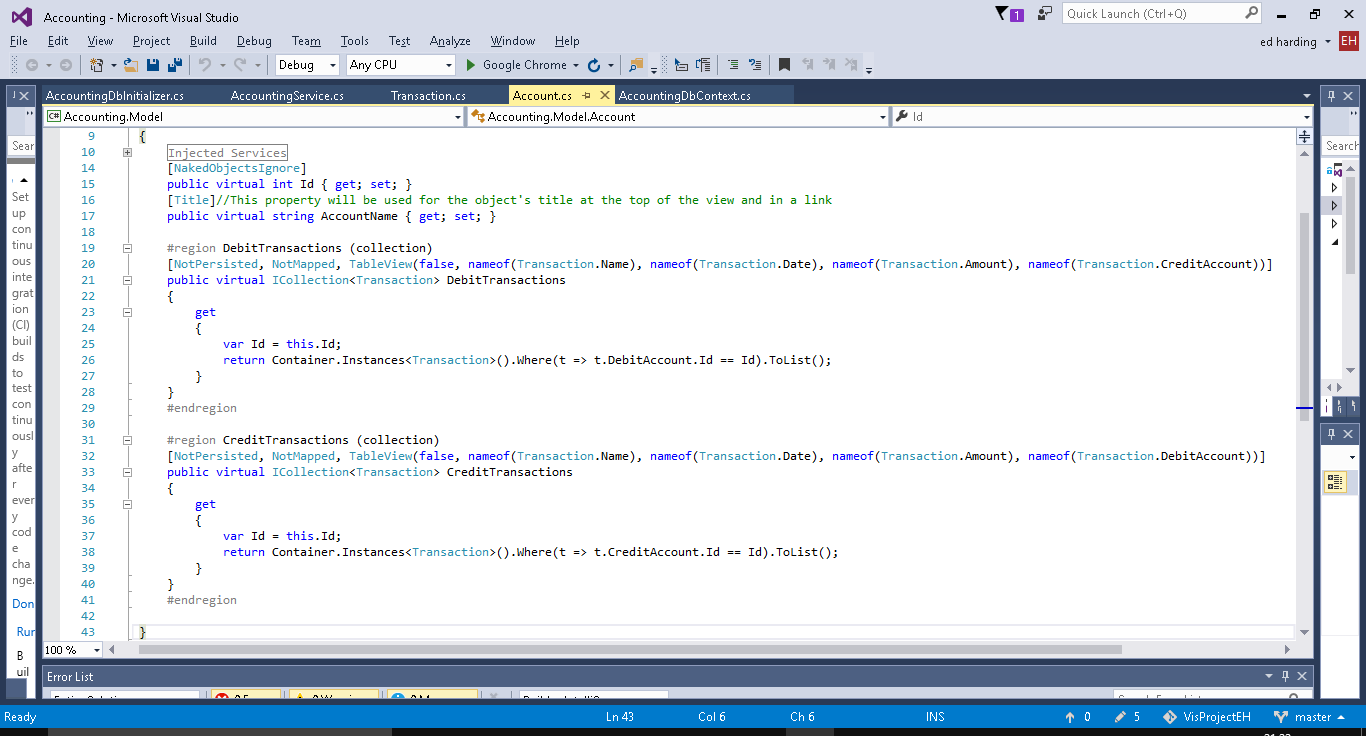


## 2

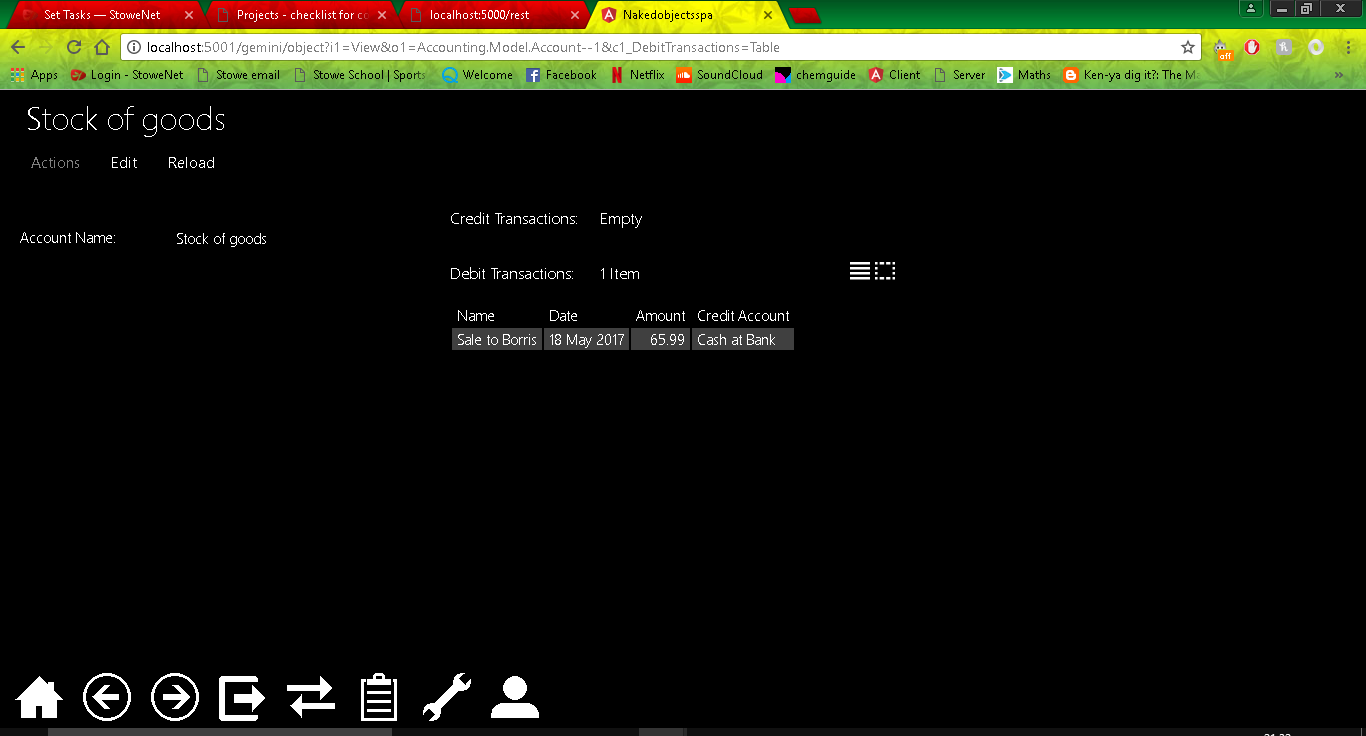
For Iteration 2 my brother (the primary user of the product) has said that he would like to be able to see the debit and credit changes in a balance sheet format. He would like to be able to see the total value of the credit and debit transactions made to an account, and then be able to open a table which will show these all combined together and which shows (by doing a total) that the debit and credit balance is equal to one another.

As the developer of the program I have decided that I would like to add a Balance table as a derived property from the Account Class, the balance table will act like a normal accounting balance table as I have learned from the book I am using. I plan on it to display the credit and debit of the Accounts and for it to show that they are equal to one another as they should be. I have also decided to add some minor tweaks to the program to help accommodate for these changes in to make the program more stream-lined.

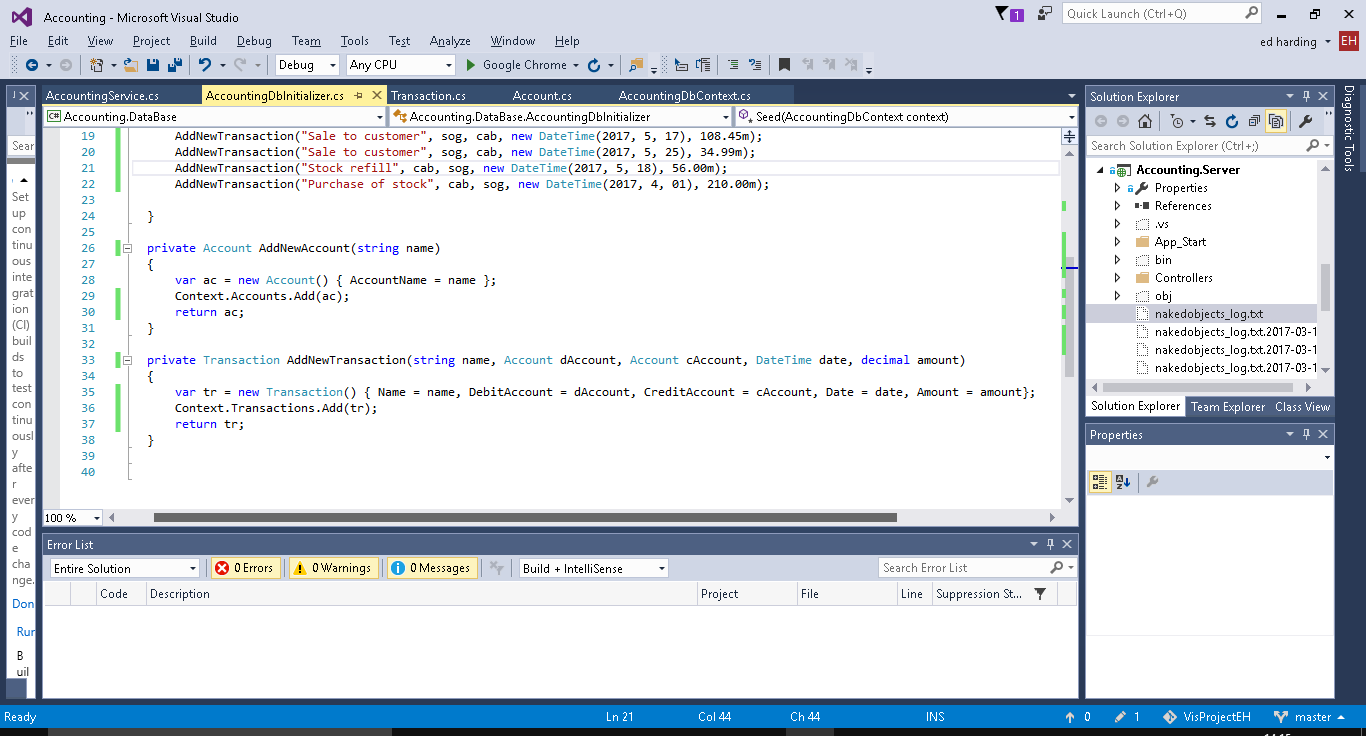
The first thing I have decided to change is how the list of transactions is displayed when looking at an Account. When the debit or credit transactions were displayed in table form you would be able to see the accounts it effects however one field of that table whether its debit or credit account would be the name of the account you are looking at, this is useless information which makes the program look unsophisticated. This change will also help as I plan to add a total to each table of debit and credit transactions for each account to be used in the balance sheet, therefore doing this early provides an organized basis to start from.

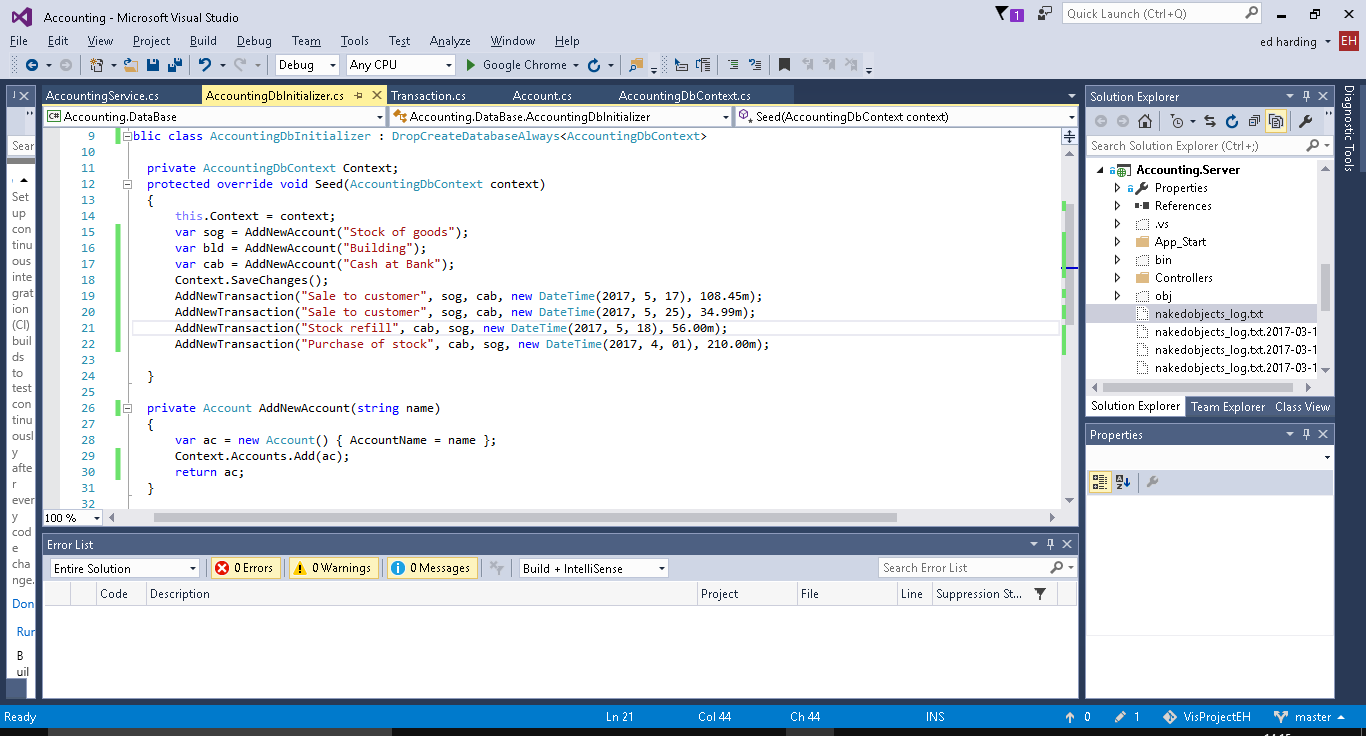


To do this (as you can see above) I used the ‘TableView’ attribute which allows me to designate which properties are visible when put into table view.

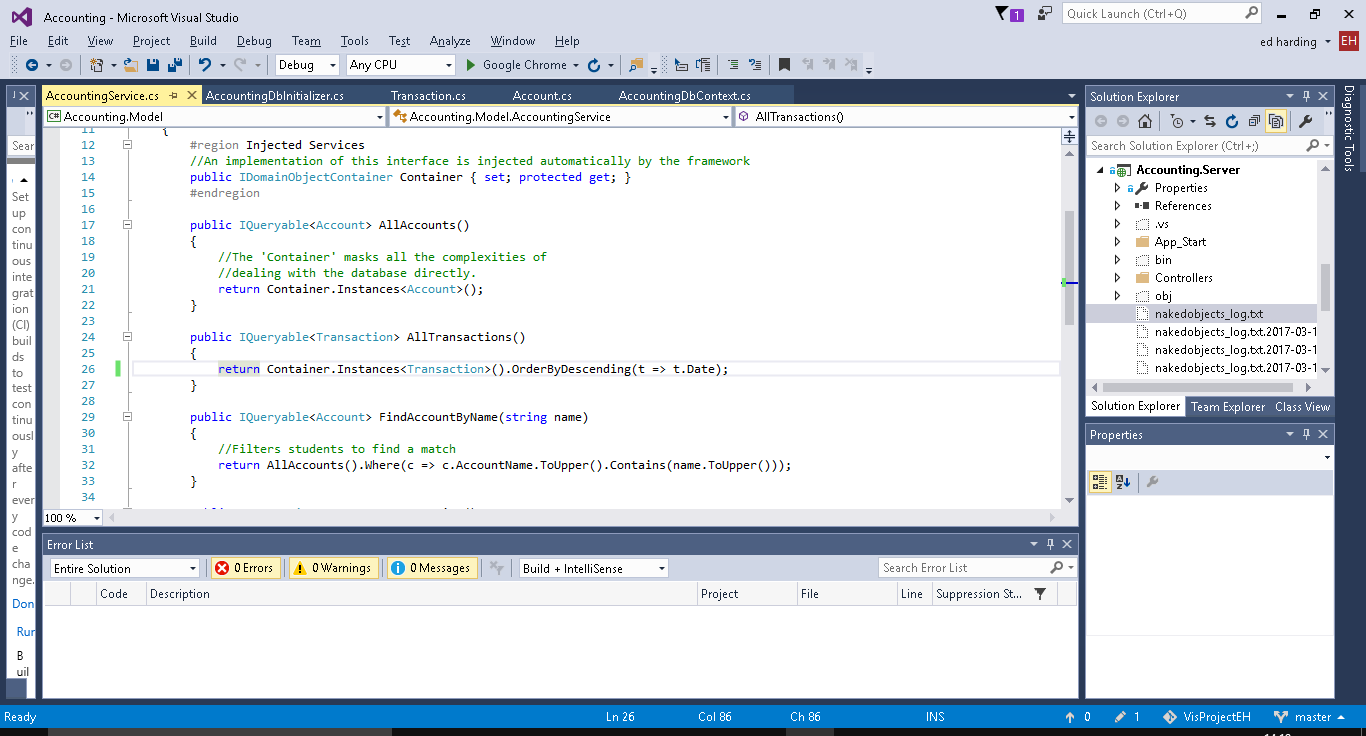


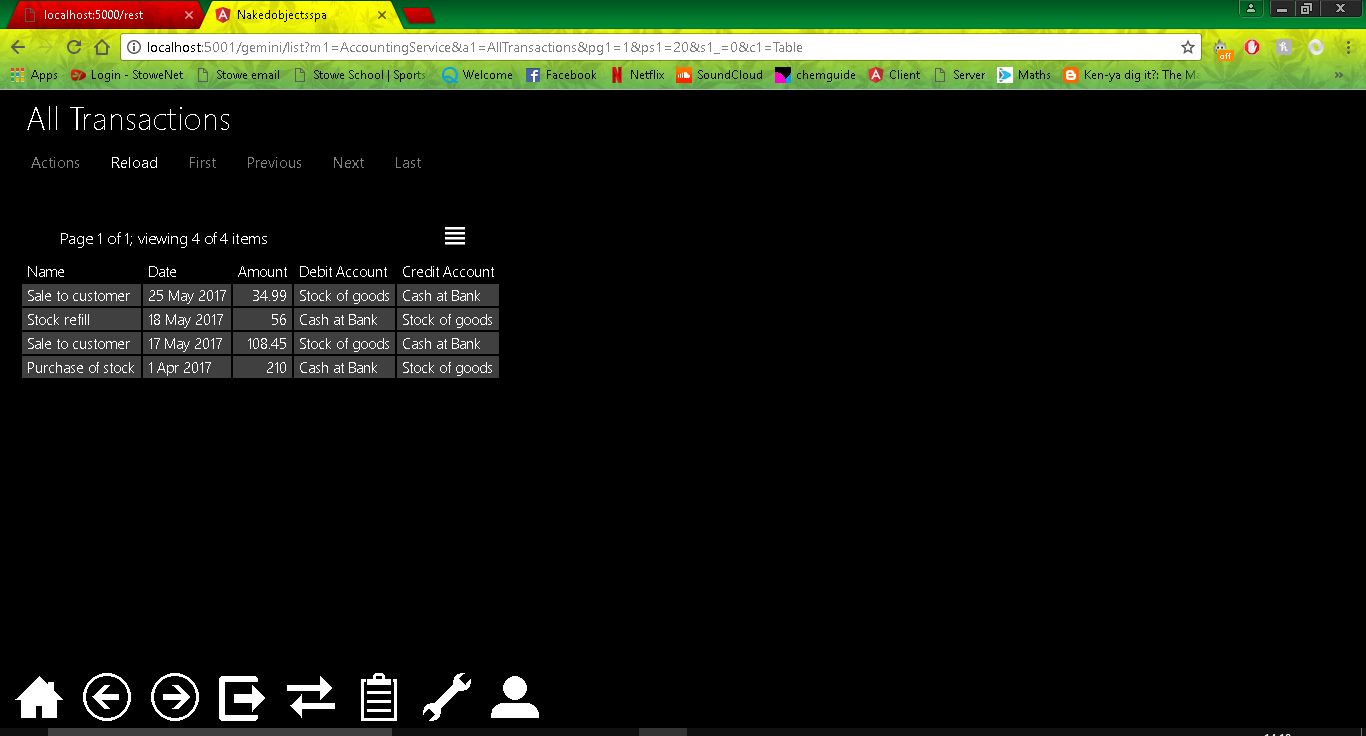
Now that I have done this I decided I should add in some seed data for some past transactions. Currently I have to add all the transactions for when I want to test something therefore having some premade will not only make the system look more like a real life system therefore improving testing but will also assist in the planning and testing of the balance sheet as it will be very heavily focused on the transactions.





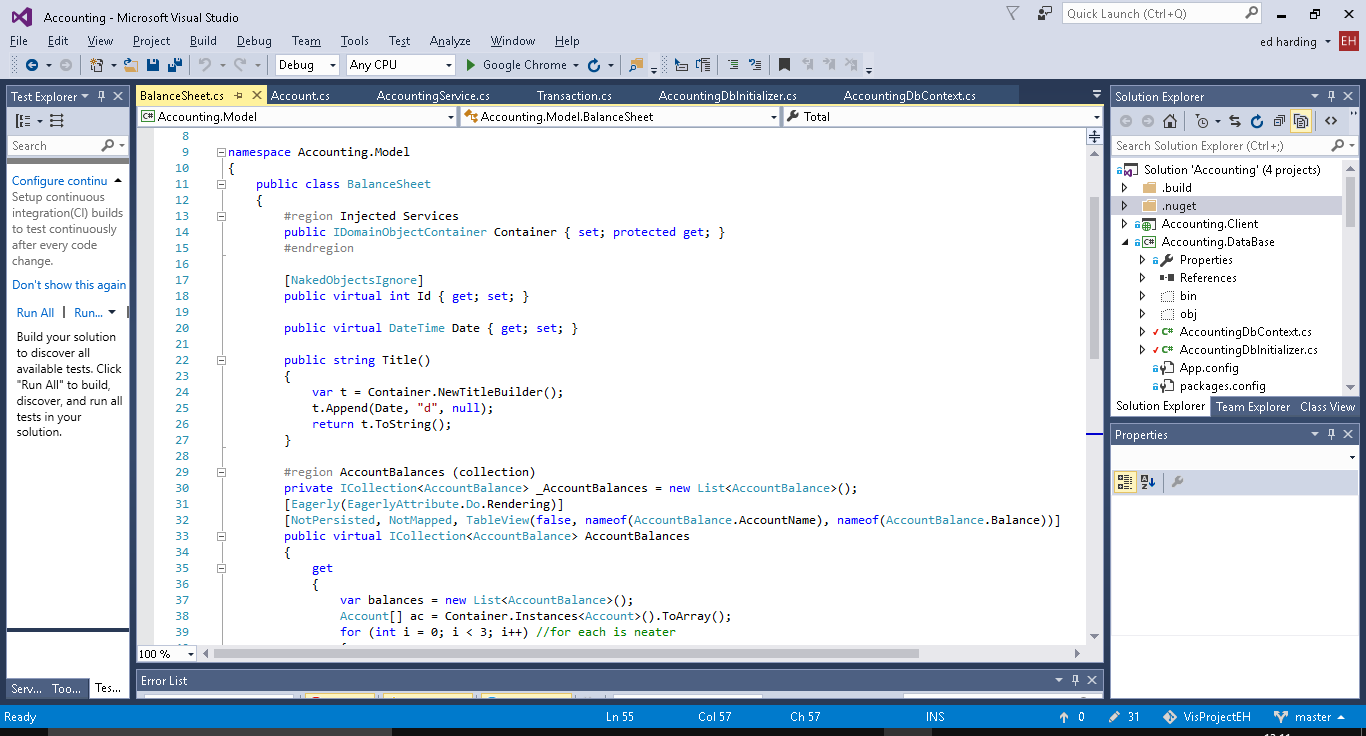
I also added linq code to the AllTransactions method so that all of the transactions are ordered from newest to oldest.

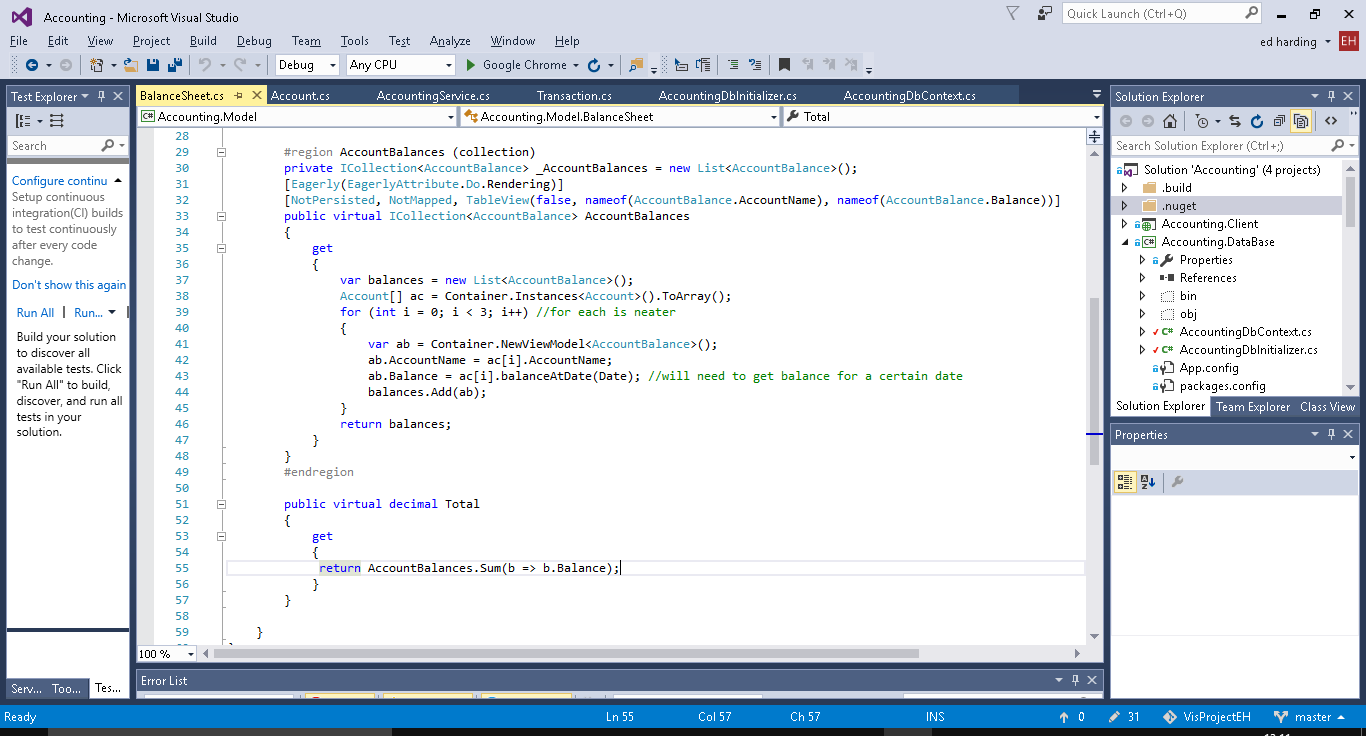




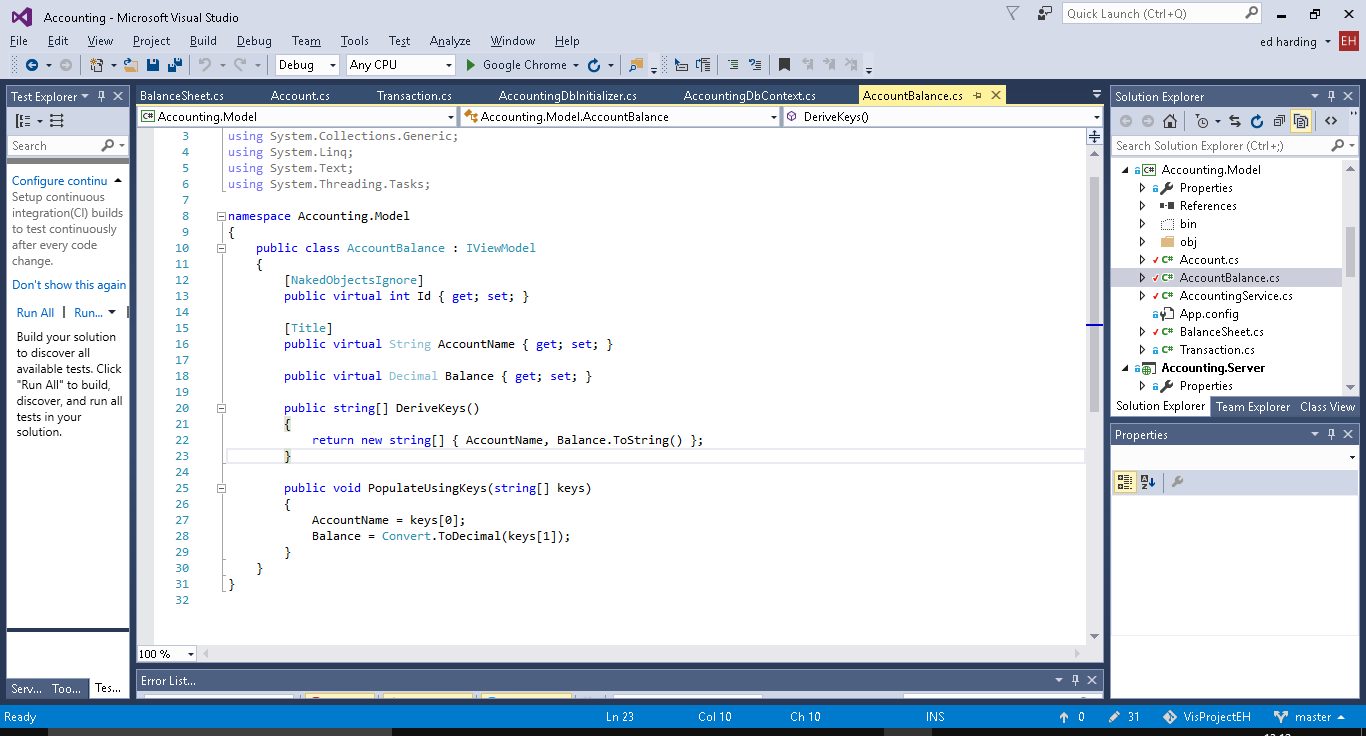
Below are some screenshots of the code for the balance sheets and examples of them in action.

Balance Sheet Class.

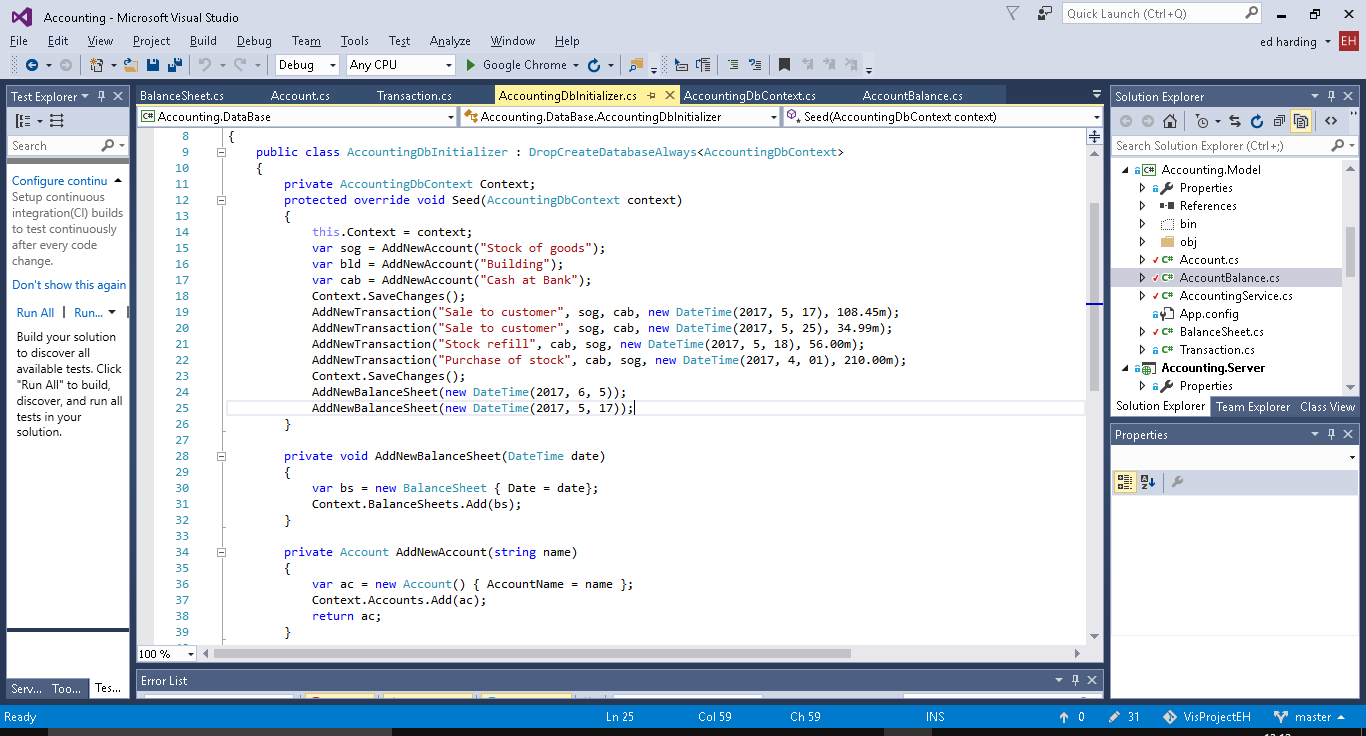




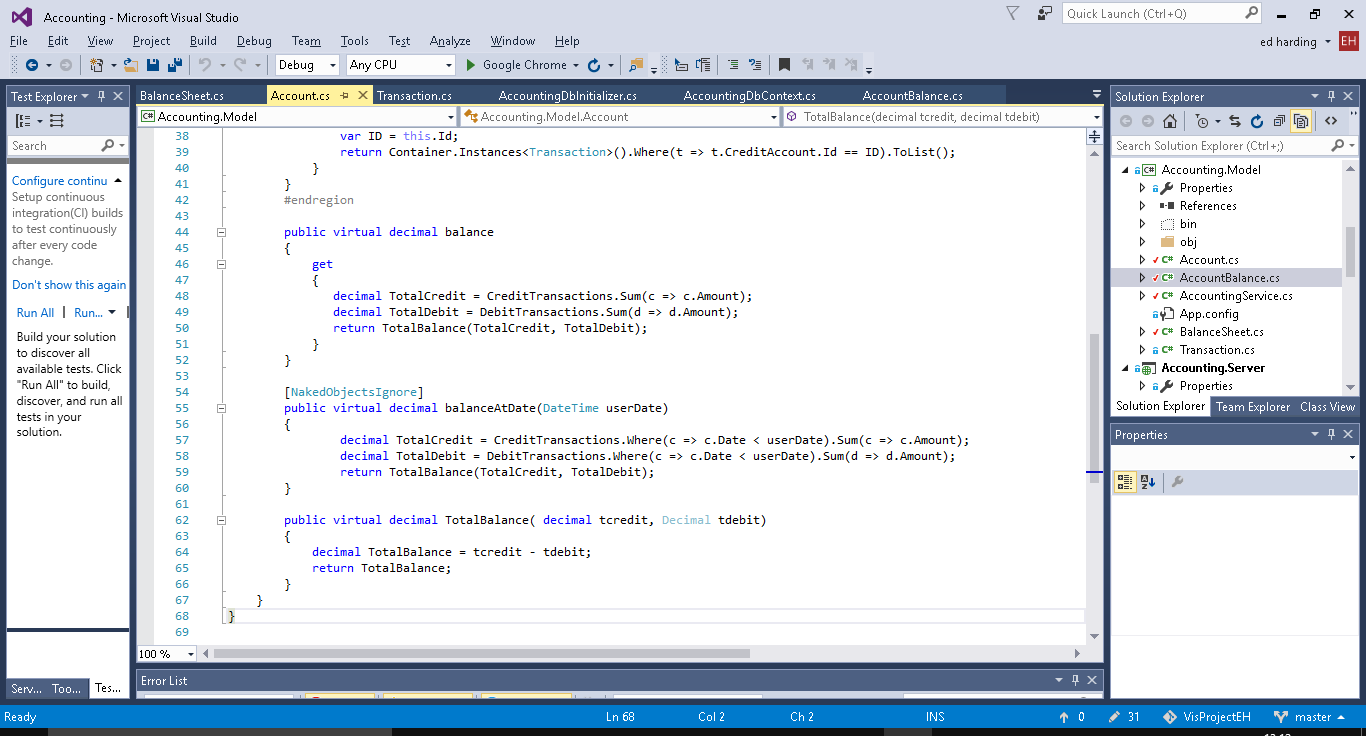
Account Balance Class



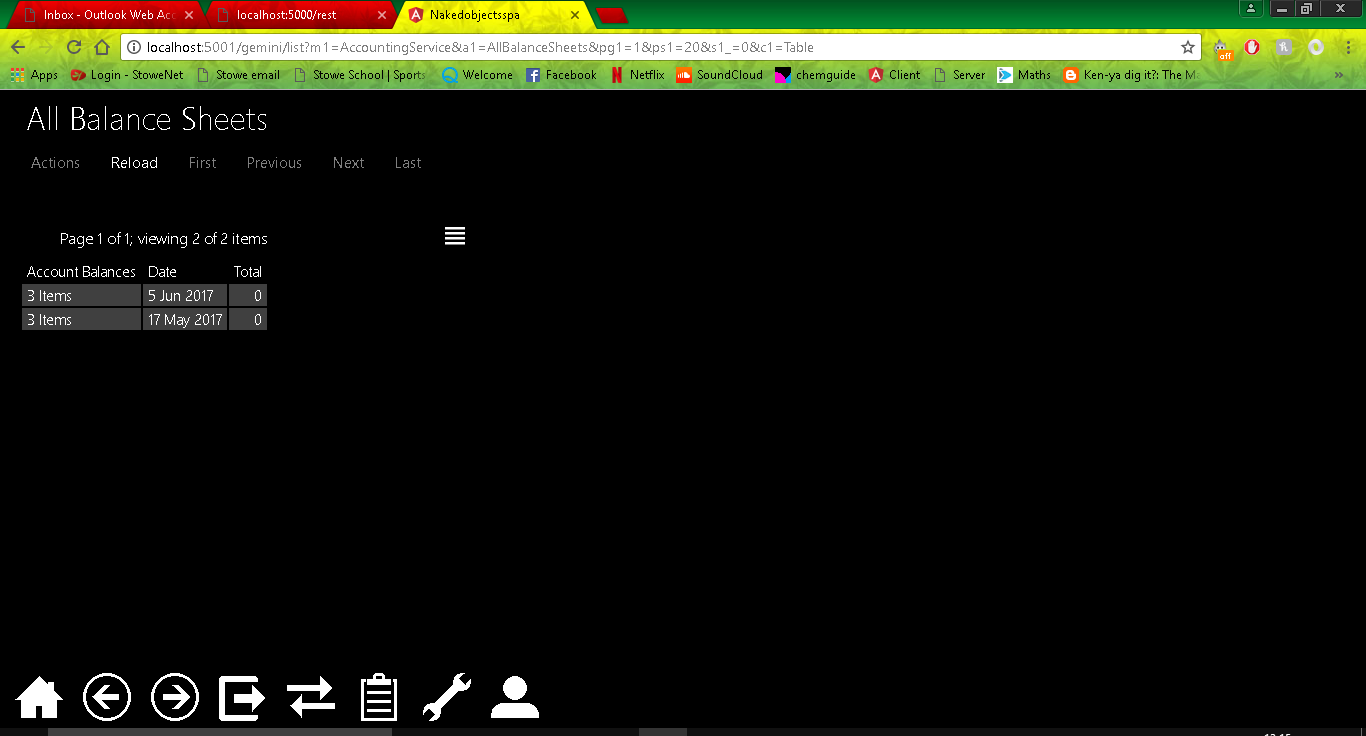
Balance Sheet Seed Data

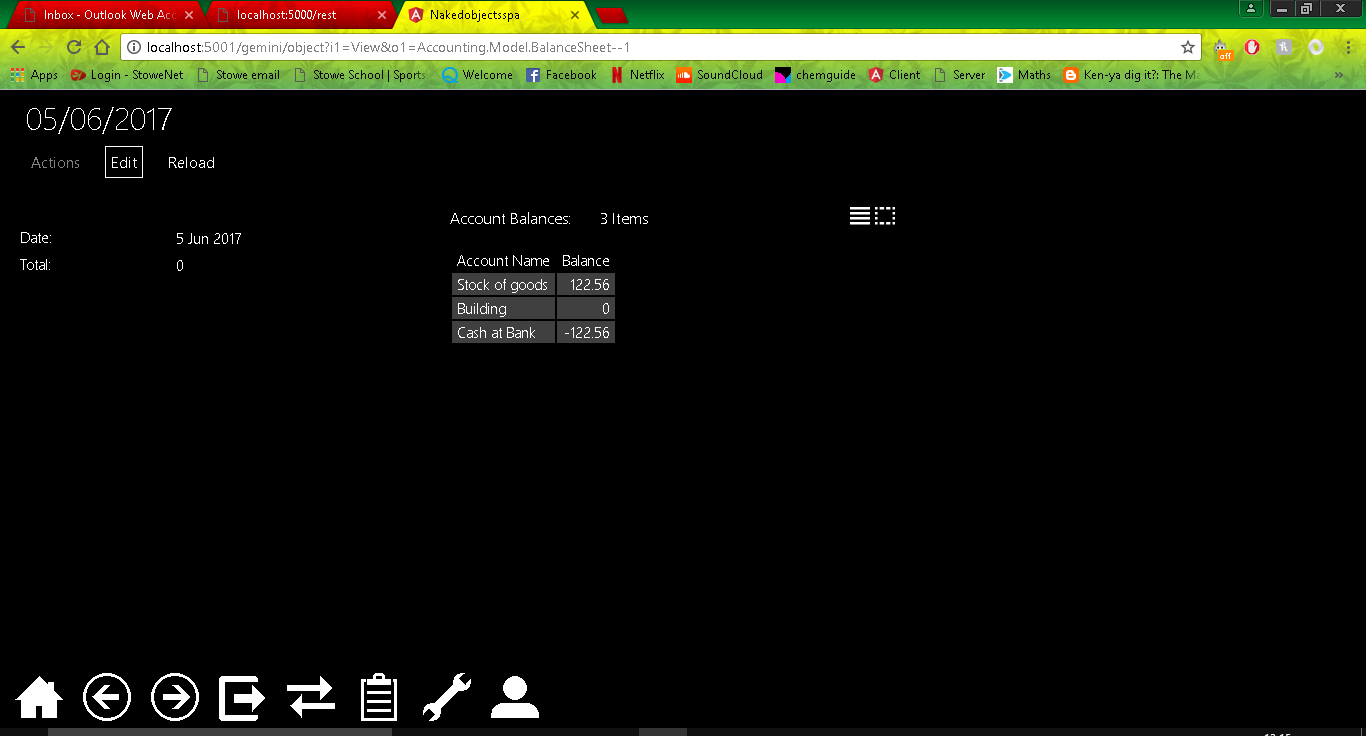


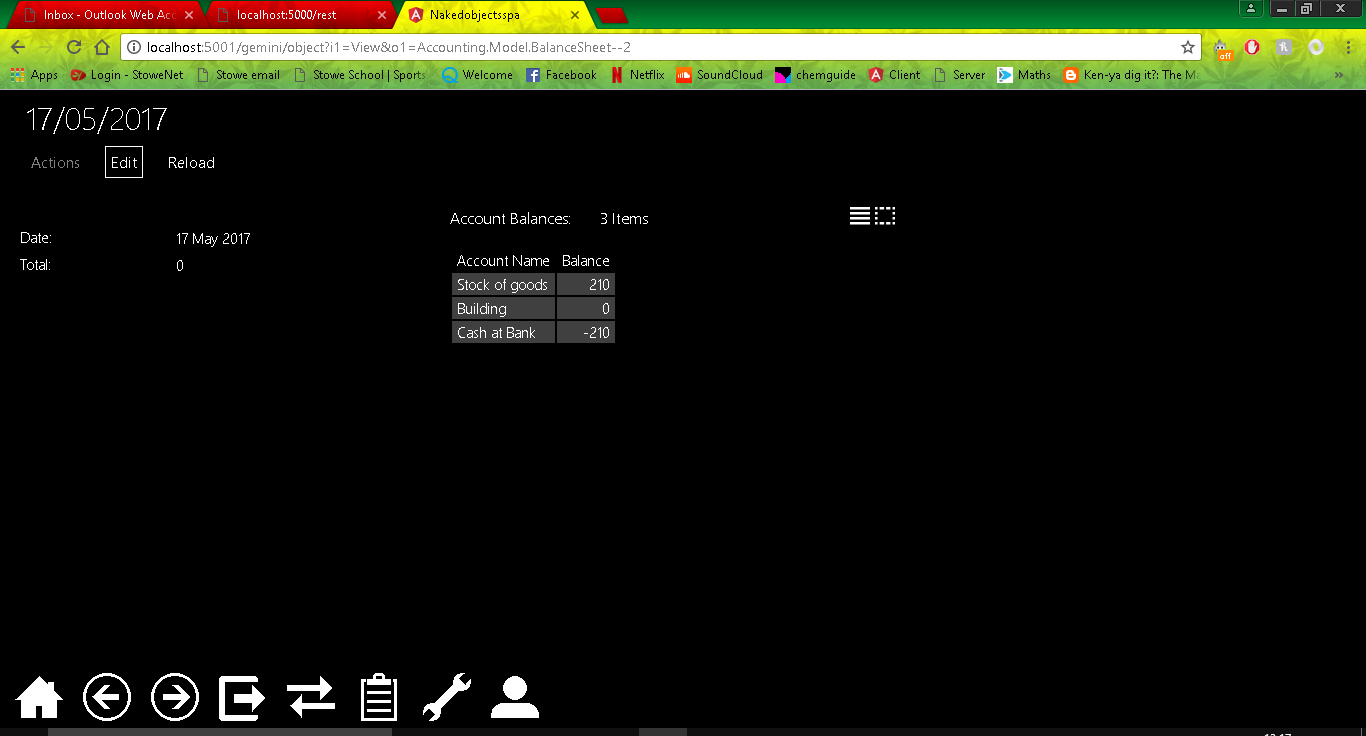
New methods on Account for balance calculations



Tests/proof





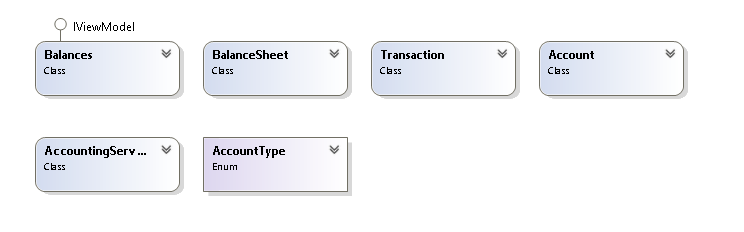


## 3

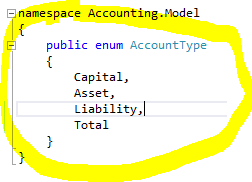
The objective for iteration 3 is to transform the current balance sheet into a more realistic one which will calculate the value of all of the assets minus the liabilities which should equal the sum of the capital accounts.

To do this I will need to separate the accounts into three account types; capital, assets and liabilities. I will do this by creating a new field in Accounts, and an Enum for the account types. I will then need to modify the Balances class so that it also takes the Account Type from the accounts class, and modify the BalanceSheet Class do to the calculations as described above.

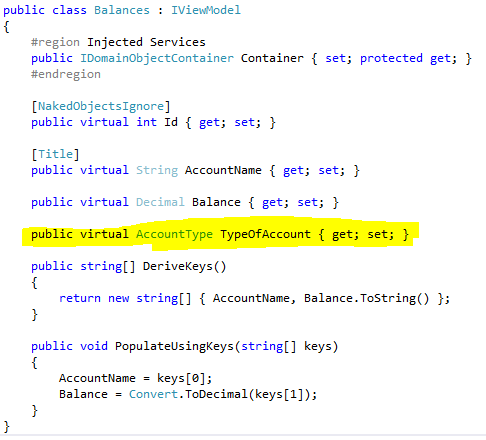
Current Class Diagram:



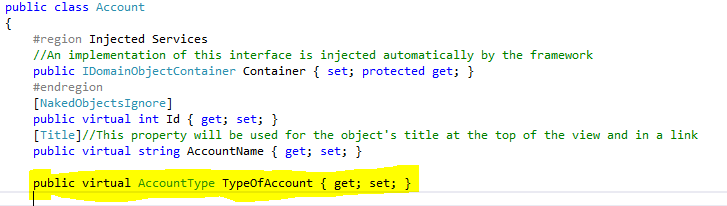
Enum for the Account Type:



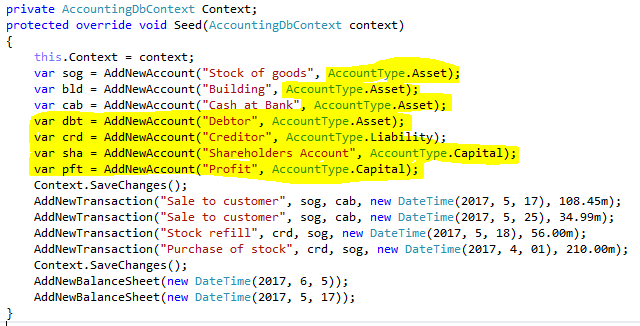
Addition of the Account Type field to Balances class.



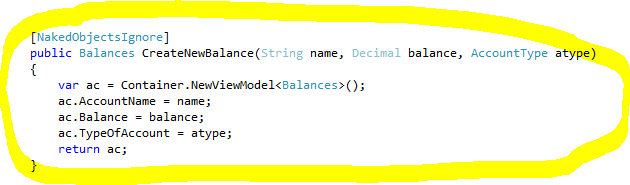
Addition of the Account Type field to Account class.



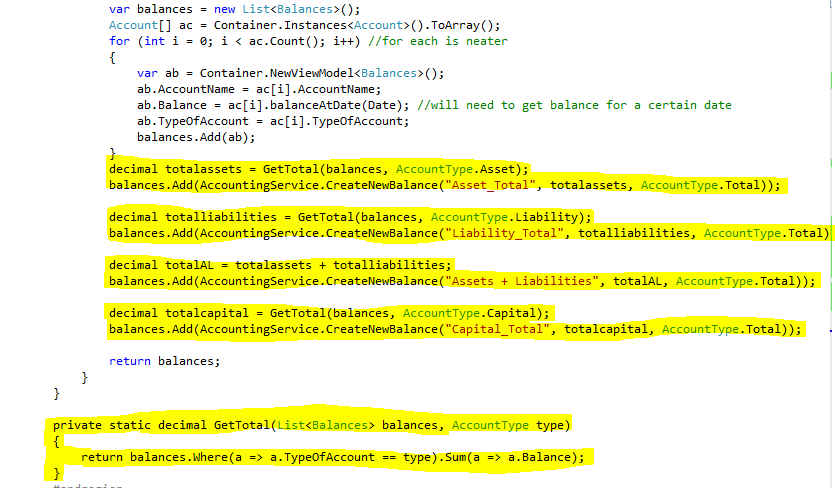
Addition and modification of seed data to include new fields.



New function to create a new instance of the Balnces Class, function found in the Accounting services.



Code to calculate the totals and then add them to the balances collection in the BalanceSheet Class.



A screenshot which shows the New collection with totals added in etc.



I believe this Iteration went rather well according to what I had planned. As expected the Total Assets and Liabilities equals the total Assets, However currently stock can’t be sold at a profit using my system, therefore everything adds up to 0, this is something I will change in a future iteration, probably the next one.

## 4

For iteration 4 I would like to correct on a flaw in the program which I mentioned at the end of iteration 3. Currently the balance sheet is working however stock can’t be sold at a profit as the debit and credit values stay the same. This means that the company’s overall capital will stay the same no matter how much is bought or sold.

To resolve this I will need to add profit and loss accounts, the profit being of the capital account type and loss being of the liability account type. I will need to make a separate method for creating sales transactions (when a profit or loss is made). The user will need to enter the price the stock was bought at and the price the stock was sold at, the usual accounts will then debited and credited and a profit or loss should be calculated by taking the amount the stock was sold at and subtracting the buying price.

## 5

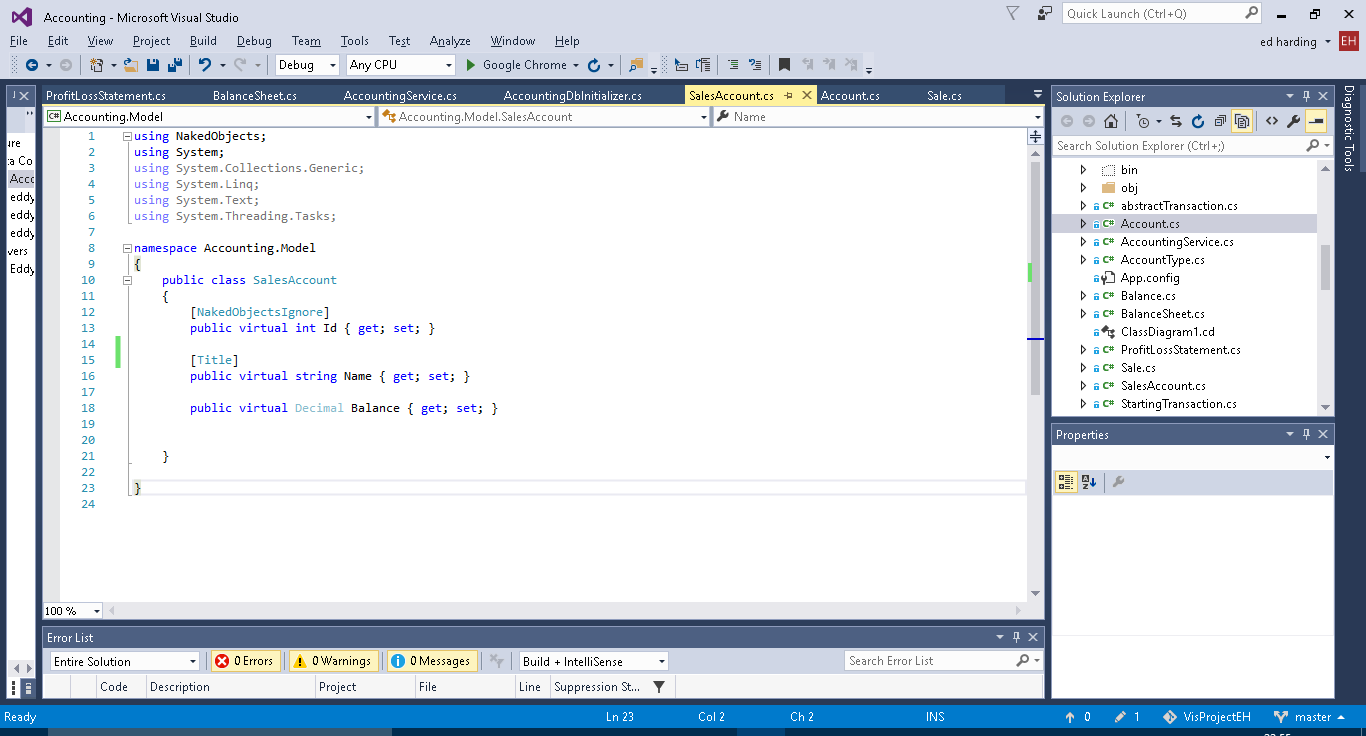
Profit and Loss Statement.

I plan on creating three new classes, one which will be similar to the balance sheet class and will be called profit and loss statement, the other will be a sales class to allow for specific sales to be entered by the user. The profit and loss statement will contain the value of stock of a given month and a field for the sales amount. The sale class will allow a user to enter the price of goods sold and what they were sold for, this will then affect the profit and loss statement accordingly. I final field will calculate the gross profit for that month of sales and display it to the user. The third class will be used as a sort of account class it will just have a name i.e. stock and will hold the values.

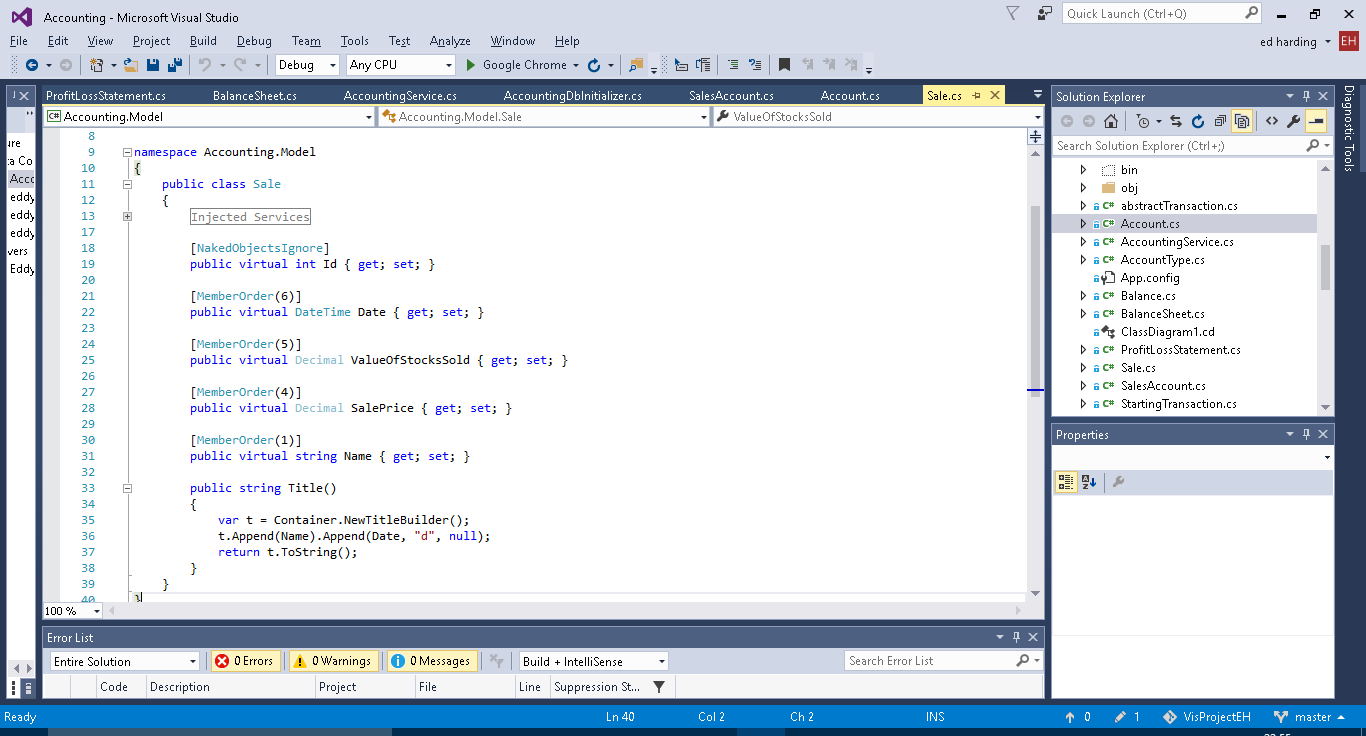
I will have to it this way because if I want the user to be able to check profit and loss for specific stretches of time then the accounts will need to contain a collection of all the previous sales transactions. The planned system will be very similar to the way the account, balance and balance sheet classes all interact with one another.

This will currently only calculate the gross profit.

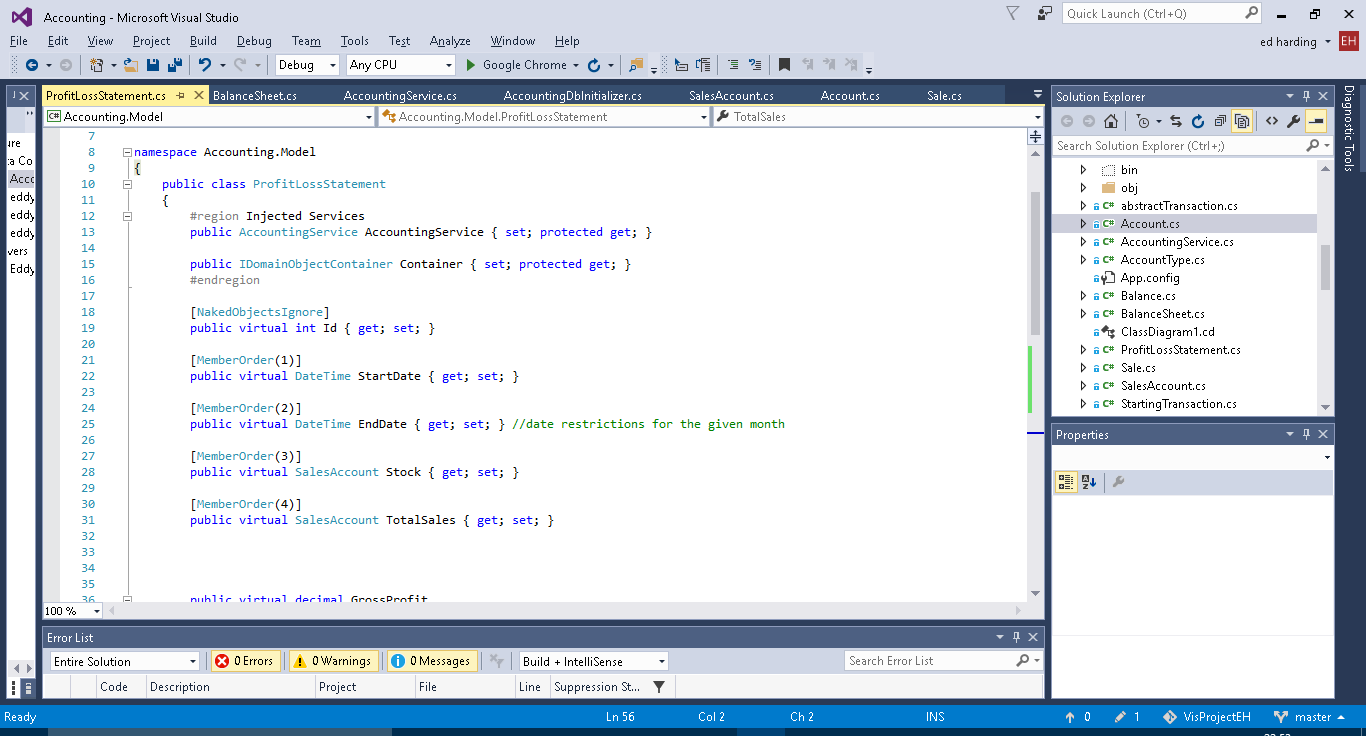
Screenshots below of new class called SalesAccount. It is used to hold the value of the stock from the stock account and the prices goods were sold for when a sale transaction is made using the Sale class below.

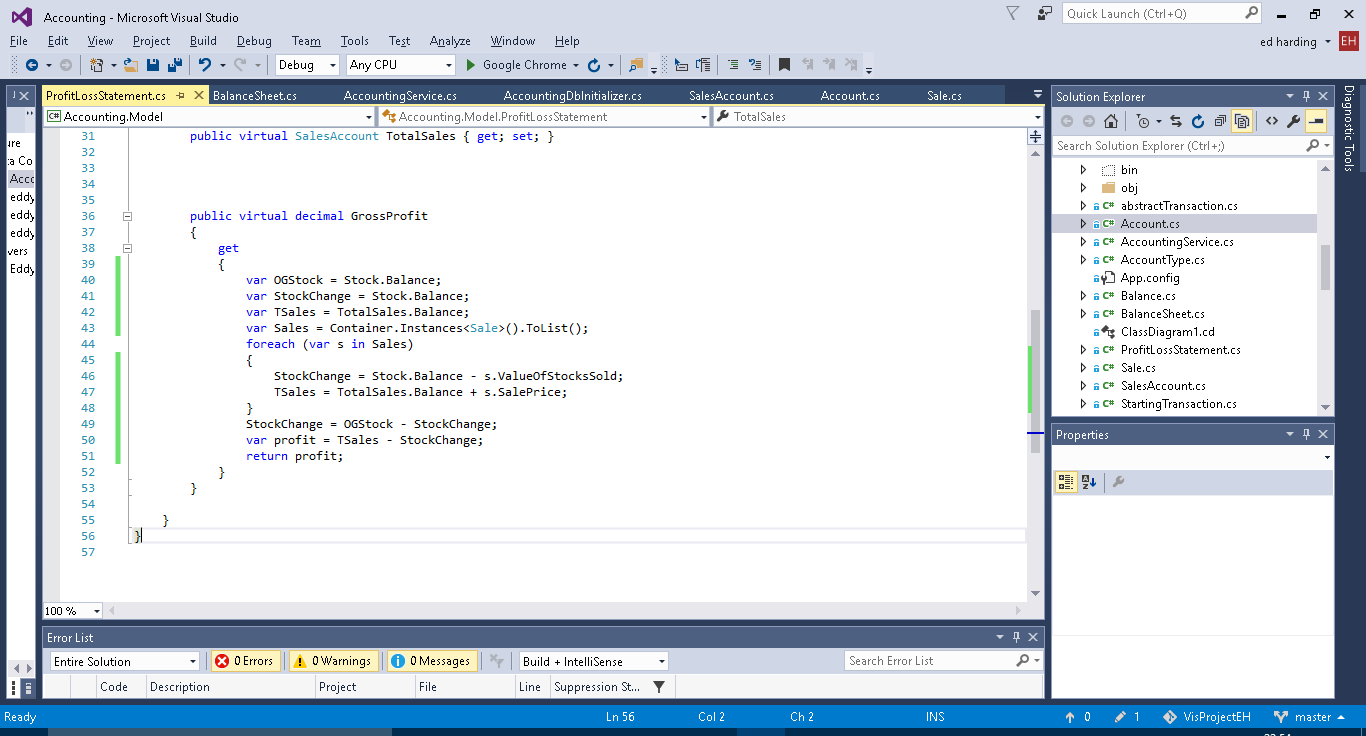


Screenshots of Sale class. Used in a similar way to transaction however it contains the date of the sale and information on the price of the stock sold and the price it was sold for. This is used to calculate gross profit.

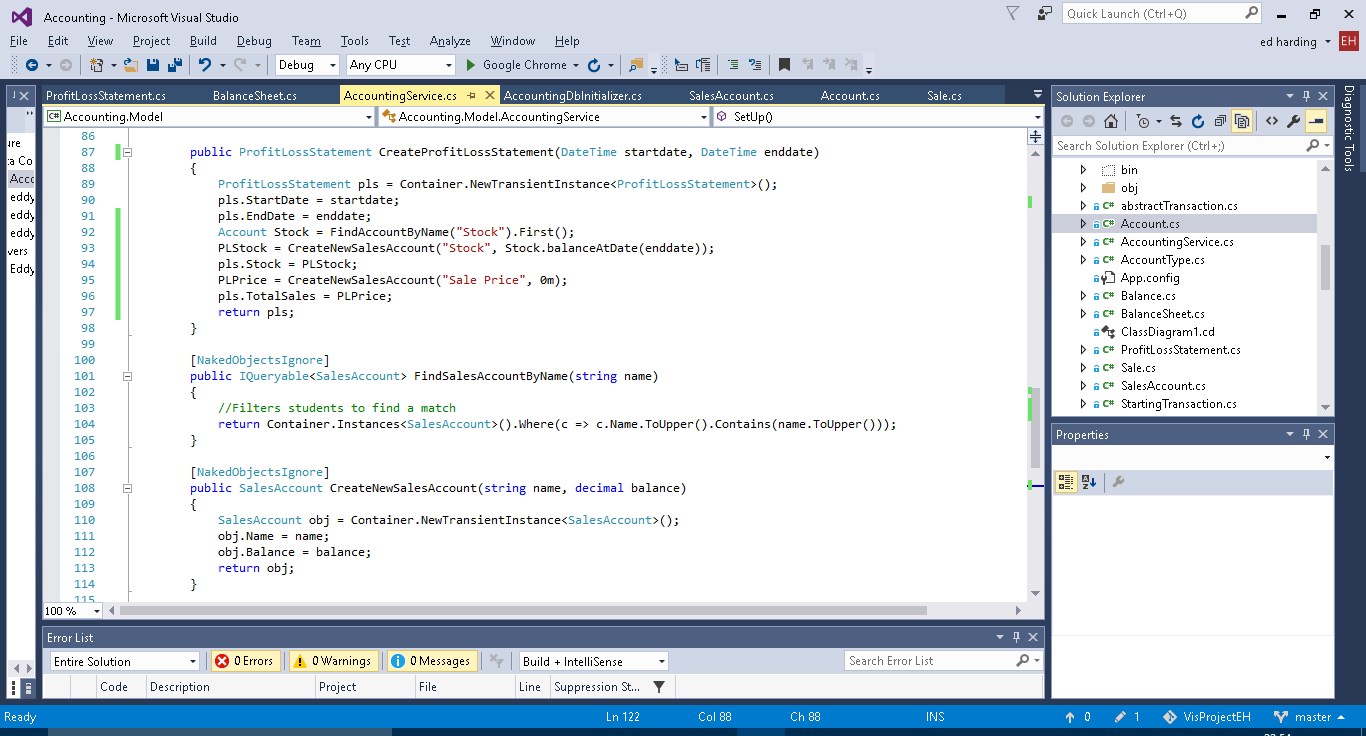


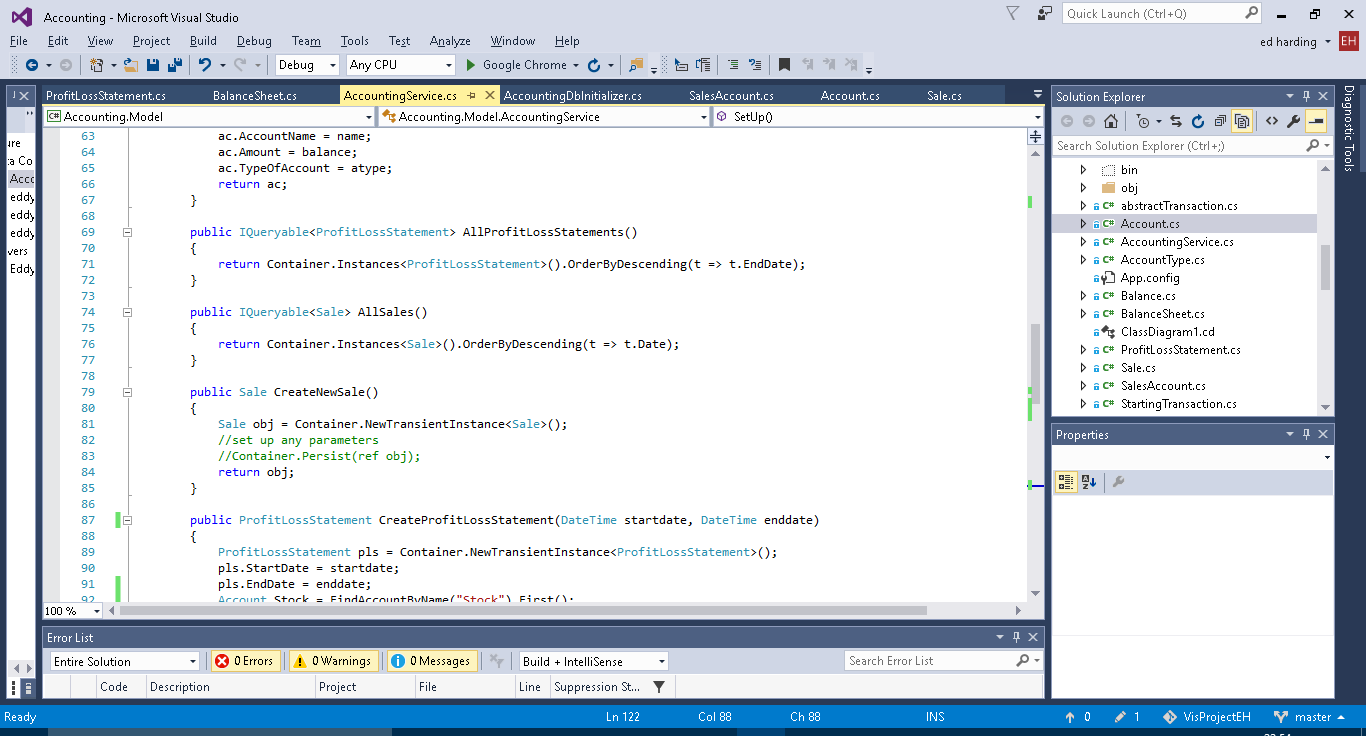
Screenshots of the new Profit and Loss Statement class. It pulls together the above to classes. The get statement is used to calculate the gross profit and loss from the sales transactions.



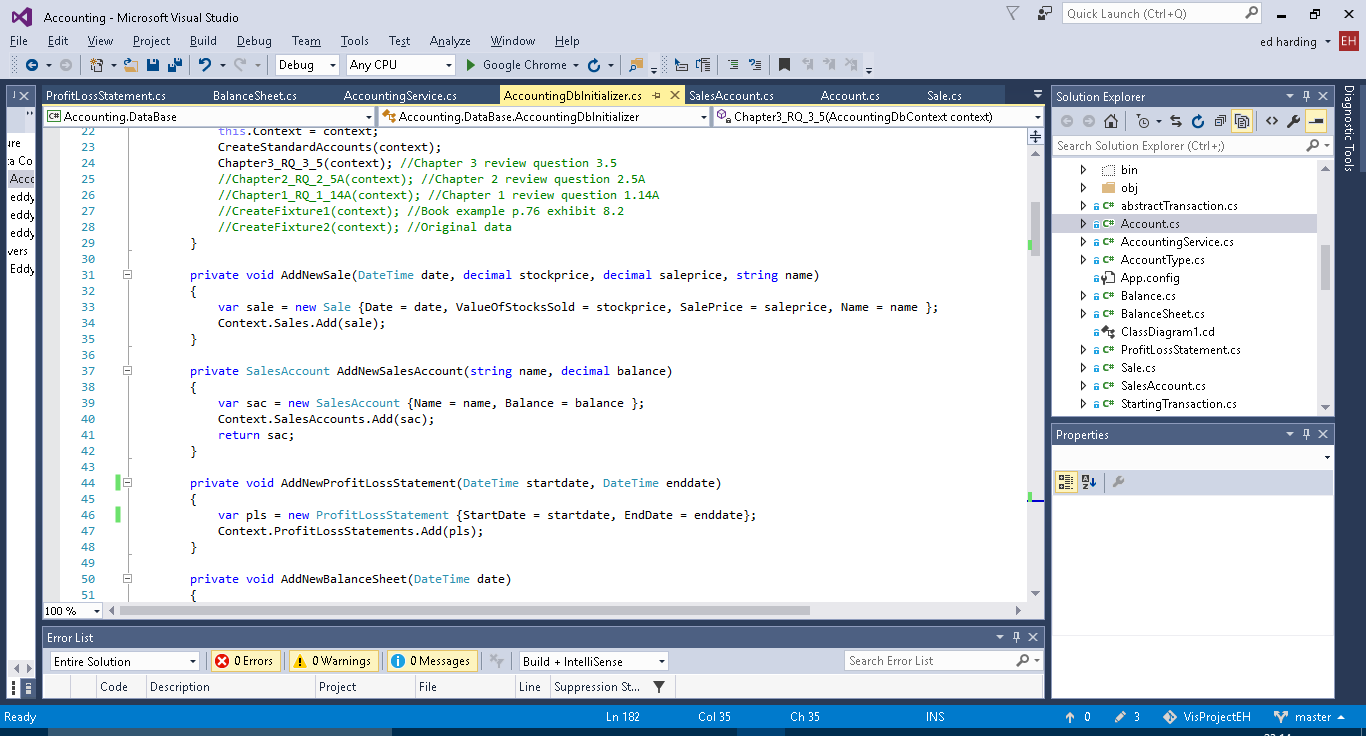


Screenshots of the new functions added to accounting services (i.e. the menu).





New functions added to the DB initializer. Only the AddNewSale is currently being used.



## 6

Auditing

When I first implemented the audit service functions, when an any of them were called an infinite loop would occur due to the fact that I was persisting the audit record object which would call the object persisted audit function which would persist a new audit record object and then call itself and keep looping until a stack overflow error occurred. I fixed this with the if statement.