# Analysis

Need to talk about the need to audit and why it is important.

## **The problem background**

Utopia Handmade and Vintage is a retail and clothing company. It is run by my brother Mr. T. Harding and his partner. The business is quite small and they tend to make most of their sales through online third-party websites such as etsy.com and also at stalls at multiple different music festivals. As the business is starting to grow and the company is transitioning from something more of a hobby over to an actual business, my brother has come to the realization that taxes and bookkeeping are legal responsibilities which will become very prevalent in his future. A lot of small business owners tend to use Microsoft Excel or bookkeeping software which can be bought online such as ‘Quickbooks’ and ‘Sage’ however I intend to make a program which can be used (most likely by him) to help assist and record the bookkeeping for the company.

As I personally had a very limited understanding of bookkeeping and accounting I decided that step one would be to do some basic preliminary research into this field. I came to the understanding that double entry bookkeeping is the standard method used in the modern world. Some other things I noted is the need and usefulness of balance sheets and profit and loss statements.

The second step I decided to take was to arrange a meeting with my brother as he would be the end user of the product I intend to build. The two things I was hoping to address with him were his current options for bookkeeping software which he could purchase and use, and the requirements he would need in a bookkeeping software for his business. After some quick research we concluded that although excel was the most cost-efficient option not only did my brother not know anything about using it but it also only allowed cashbook accounting where as he was looking for a double entry bookkeeping system. The other requirements he noted will be mentioned further on.

To conclude I decided that I would start of by creating a very standard double entry bookkeeping system as this would act as a good foundation to add improvements to and tailor it towards the end users more specific requirements. This also helps as it allows for me to make sure the basics are all up and running before getting into a greater level of depth and detail.

## **Research into Accounting**

I came to the realization that I will need to learn a substantial amount more about accounting and bookkeeping to be able to fully take on and produce a suitable product for my client. To do this I have decided to use a very well acclaimed textbook which is used by most schools and colleges in the UK to teach accounting. It is called “Business Accounting 1” by Frank Woods and Alan Sangster. The book starts from the very basics of accounting and provides multiple practice questions and examples which I will be able to use in the testing and development of my project. I am hoping to progress my program as I progress through the textbook so I can develop the program hand in hand with my learning of the subject. This is how I plan to create the foundation for my program as stated earlier.

## **The current system**

The company is currently quite small as I stated earlier and is therefore under the minimum tax bracket. This means that currently they do not have any system in place for bookkeeping as there has been no need for it so far.

## **Clarification of The End User**

Although my client is Utopia Handmade and Vintage I would say that Mr. T Harding will be my soul end user for the system as he is one of the owner’s and the most likely out of the two to be doing the bookkeeping. He will be my first point of contact for information on what he will require from me. Therefore, the system will be catered towards his requirements and he will also be the person I get to test the program once it is in its later stages and past the book level testing.

## **User requirements**

1. The program should be able to perform the basic functions of a double entry bookkeeping system, this includes but is not limited to:

* Allowing the user to create multiple different accounts.
* Allow the user to add sub accounts.
* Allow the user to record transactions between the accounts.
* Record these transactions and use them to form a balance sheet.
* Use transactions to make up a profit and loss statement for a requested time slot.

1. The system must be easy to use, simple to understand and navigate, and display data and information in a way which is tidy, efficient and easy to understand.
2. The system will NOT require any authentication process as It will be a locally run program therefore it will only be accessed from the clients own computing device.
3. The client would like the system to be as aesthetically pleasing as possible; as it will be used for long hours therefore it should not be dull.

Of course, the example I mentioned earlier such as Quickbooks and Sage can do all of these things and more however I believe that as Mr. Harding the end user is new to bookkeeping it would in fact be more beneficial to keep the accounting side to the simplest level possible. One other important thing is that my code will all be free and open source (being shared via GitHub) this allows for other users to further extend to its functionality and add complexity to it if needs be.

# Design

Add more class diagrams showing relations etc. not just inheritance.

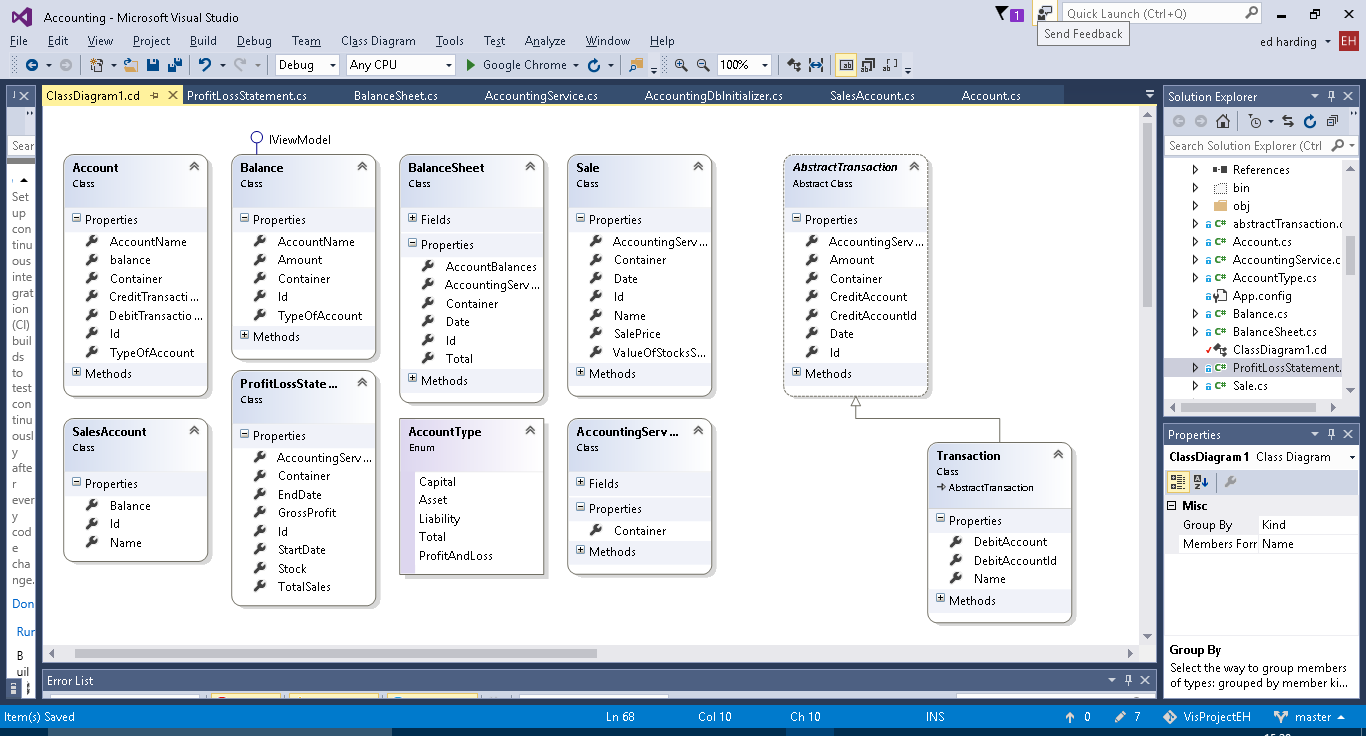
## **Why I chose to do an Object-Oriented Project**

The greatest reason I had for choosing object oriented programming was due to the ability to create user defined objects to be used in my program. After I decided on my project idea I sat down and thought about some of the basic functionality it would need to do and the easiest way to implement it. I had already had some experience with OOP and after looking at the basics of an accounting system I noticed that there were lots of clear objects which could be created. The ability to create your own object set properties and give functionality to the objects really stood out as a necessity to me in designing my project.

A more trivial reason for my decision to go with OOP was just for organisation of the code in visual studio. OOP allows for your code to be split up over multiple segments which allows for easy navigation and organisation of your program.

Another big selling point to use polymorphism is inheritance, having the ability to create sub/super classes of your use defined objects is incredibly useful to reduce the repetition of code and functionality, which in turn reduces the likelihood of errors and usually makes them easier to fix if they effect one part of a class hierarchy rather than finding an error in a part of code in an object which is in other objects and then having to individually fix them all.

## **Class Diagram and Objects Used**

****Account

The account class is – as it is named – meant to represent an account in a running business. Every business has a list of different accounts which are used to record the movement of money and value of objects owned by the company. For example, a small retail company may have an account for the current cash held by them in store.

The account contains a collection of transactions and a calculated balance from these transactions. As this balance is used in the balance sheets and balance sheets can be made for past dates a function to calculate the balance at a specific date is needed. For both of the above a function to calculate a total balance is needed therefore a function can be made to do this to reduce repetition.

Balance

The Balance class doesn’t really have a business application behind it, it was just necessary to have for functionality purposes to be used as to retrieve and format the account data into the balance sheet.

Balance Sheet

As it is named the balance sheet object is made to create a balance sheet and calculate the balances of all the accounts and make sure that the capital equals the company assets minus liabilities.

A title builder method is implemented this is so that an apt title can be created automatically for the user using the date for which the balance sheet is for. This allows for them to be ordered in the user interface by date for user convenience.

Abstract transaction

An Abstract super class for transactions.

Contains an auto complete method for the account to be credited for user convenience.

Transaction

The most common type of transaction made in a double entry bookkeeping system. It debits and credits individual accounts as it should. As you can see from the class diagram it is a sub class of Abstract Transaction.

Transaction also contains a title builder which creates a title consisting of the inputted transaction name and the date of which the transaction occurred, again this is just for user convenience and for data recording purposes. As Transaction is a sub class of abstract transaction it auto inherits the auto complete method for the account to be credited. However, it also contains an additional auto complete for the account to be debited (works the same as the credit one).

Account Type

The accounts in a double entry book keeping system fall into three categories capital, assets and liabilities. I need a way to represent them, so that total balances of each category can be calculated so that the balance can be checked as already mentioned in the Balance Sheet. The easiest way to do this was to create an enum.

Accounting Services

Accounting services doesn’t really represent anything specific in business terms it just adds functions which are useful to multiple different objects and to the user in the user interface, such as being able to add new instances of objects like Account and transactions. It also contains methods for the user such as listing all the accounts, transactions, balance sheets etc.

**Sale**

Is used in the profit and loss side of the program. Represents a sale of goods to a customer, and records the profit or loss made from that sale.

Sale also has a Title builder method which like transaction creates a title from the date of the sale and the name given to the sale.

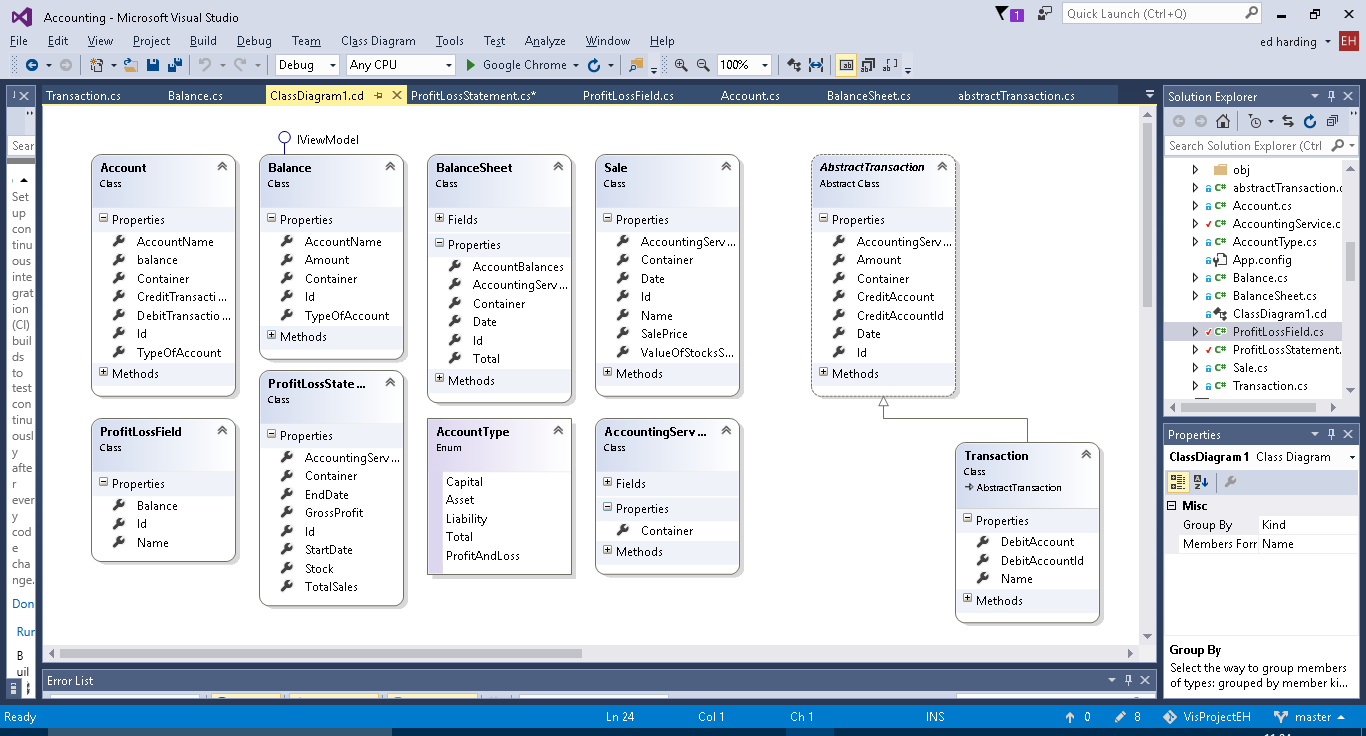
**Sales Account**

Is used to create the fields used in the profit and loss statement, those being the total value of the stock and the total value for sales made.

Needs renaming to ProfitLossField.

**Profit Loss Statement**

As the name suggests this class creates a profit and loss statement for a given time frame. It does this by getting the Sales (from Sale) made in that time frame and the value of the stock (at the latest point of the time frame) and then calculating the amount of stock sold and for what total price it was all sold at.

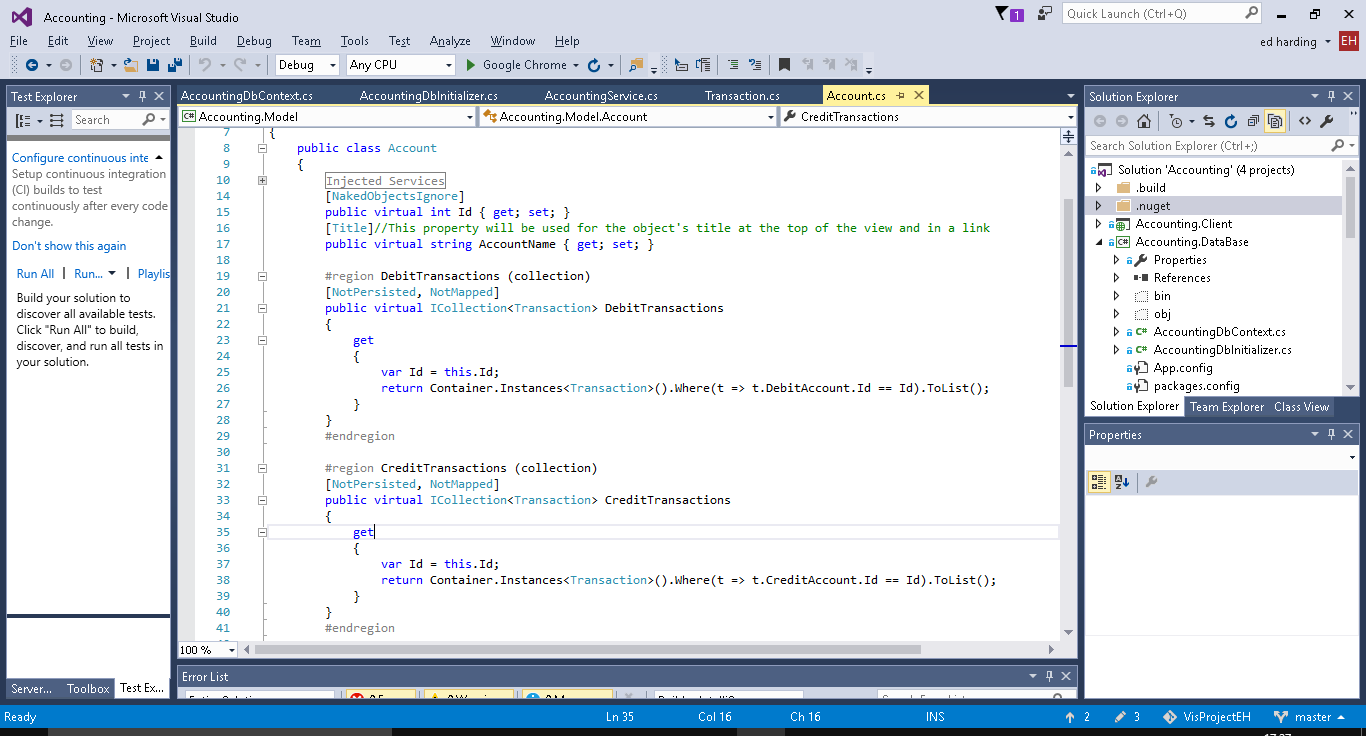
**Updated Class Diagram**

# Technical Solution

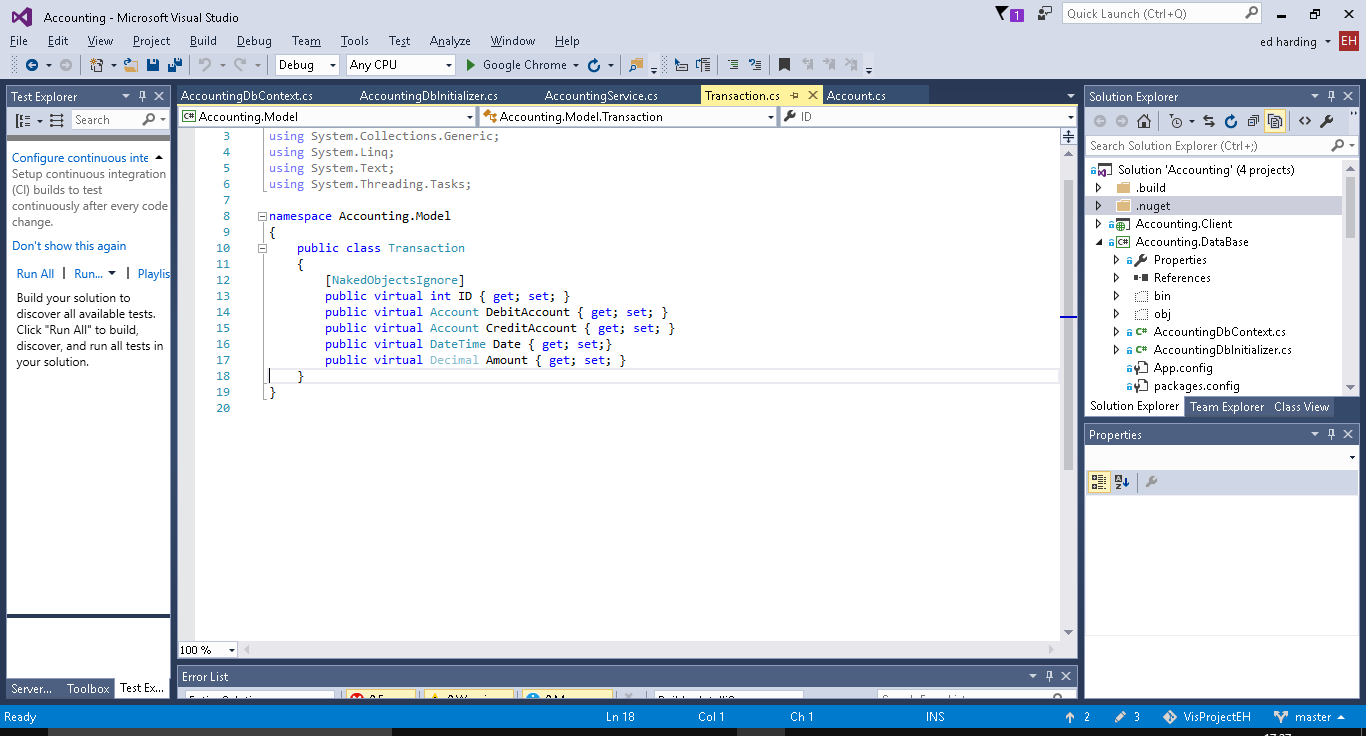
Added iterations, numbered to keep them separate for now. Once they are all fleshed out will combine them together/re-format. Are iterations design or technical solution?

## 1

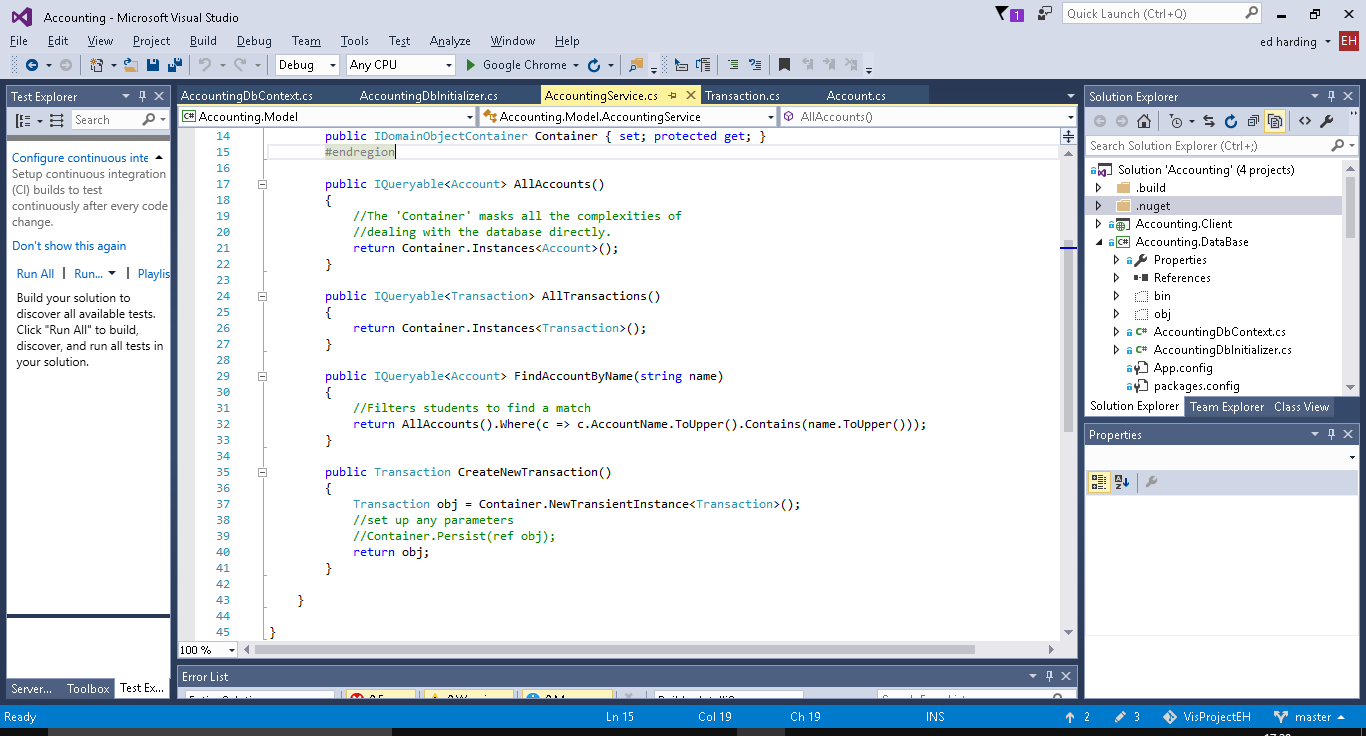
For iteration 1 of my program I would like to create two classes to be used in the application, one being an Account class and one being a transaction class. The transaction class will need to contain information about the date of the transaction, the amount of money, and the debit account and the credit 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.



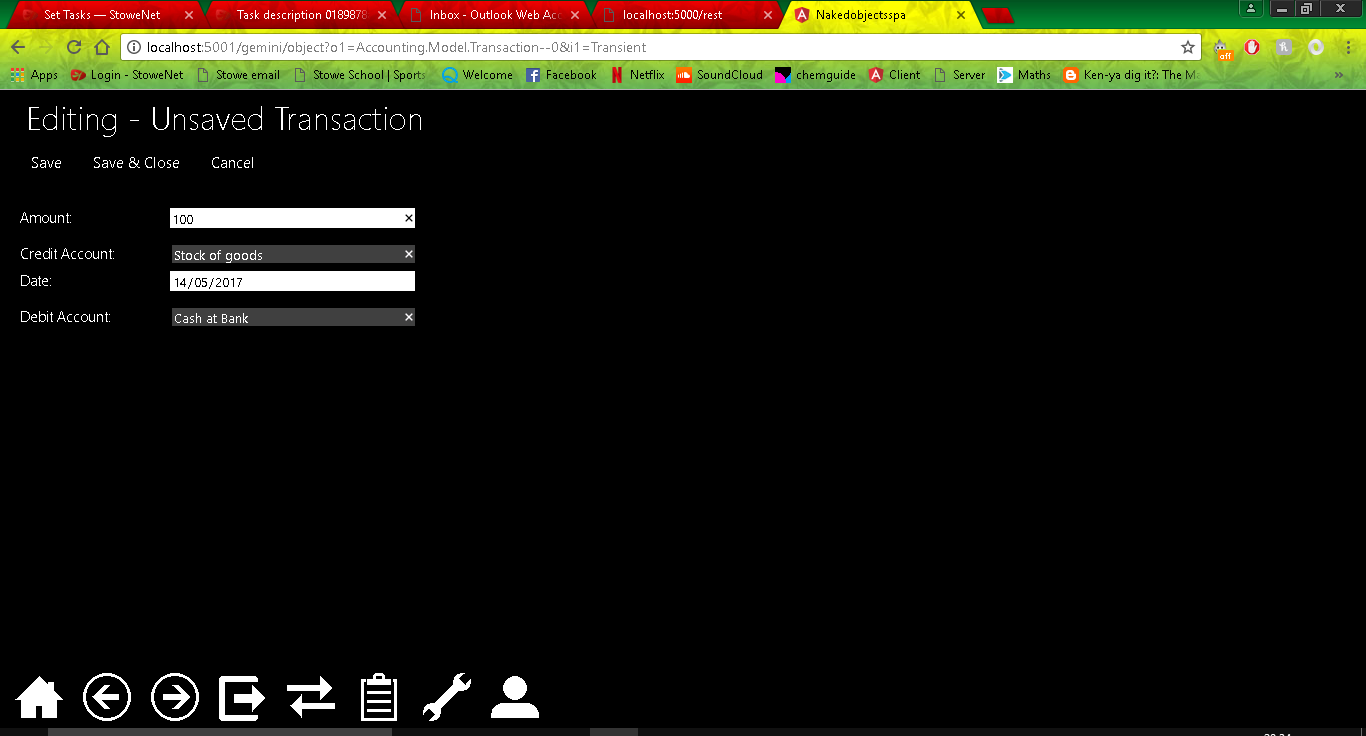
Above is the Account class, as well as the Id property which all classes have to have in Naked Objects it has an AccountName property and 2 collect properties which hold a list of debit and/or credit transactions (which are instances of the Transaction class below).

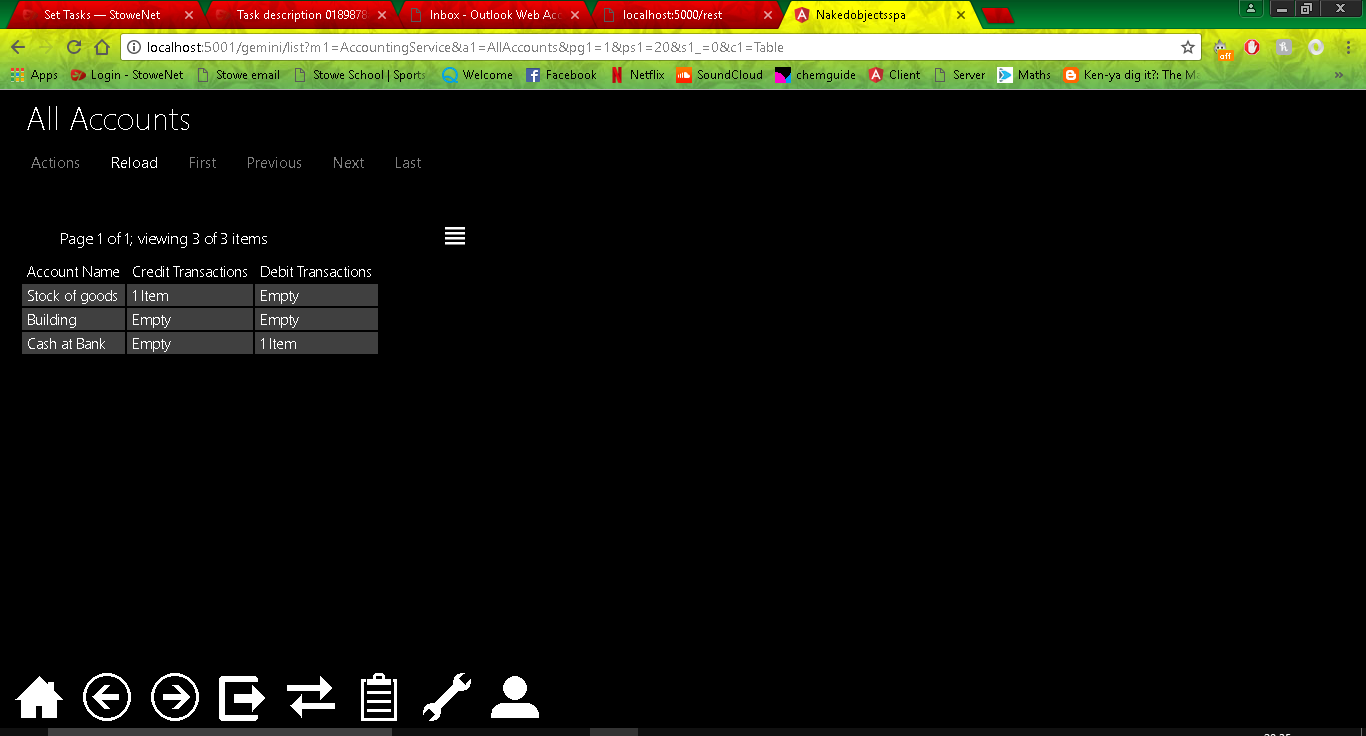


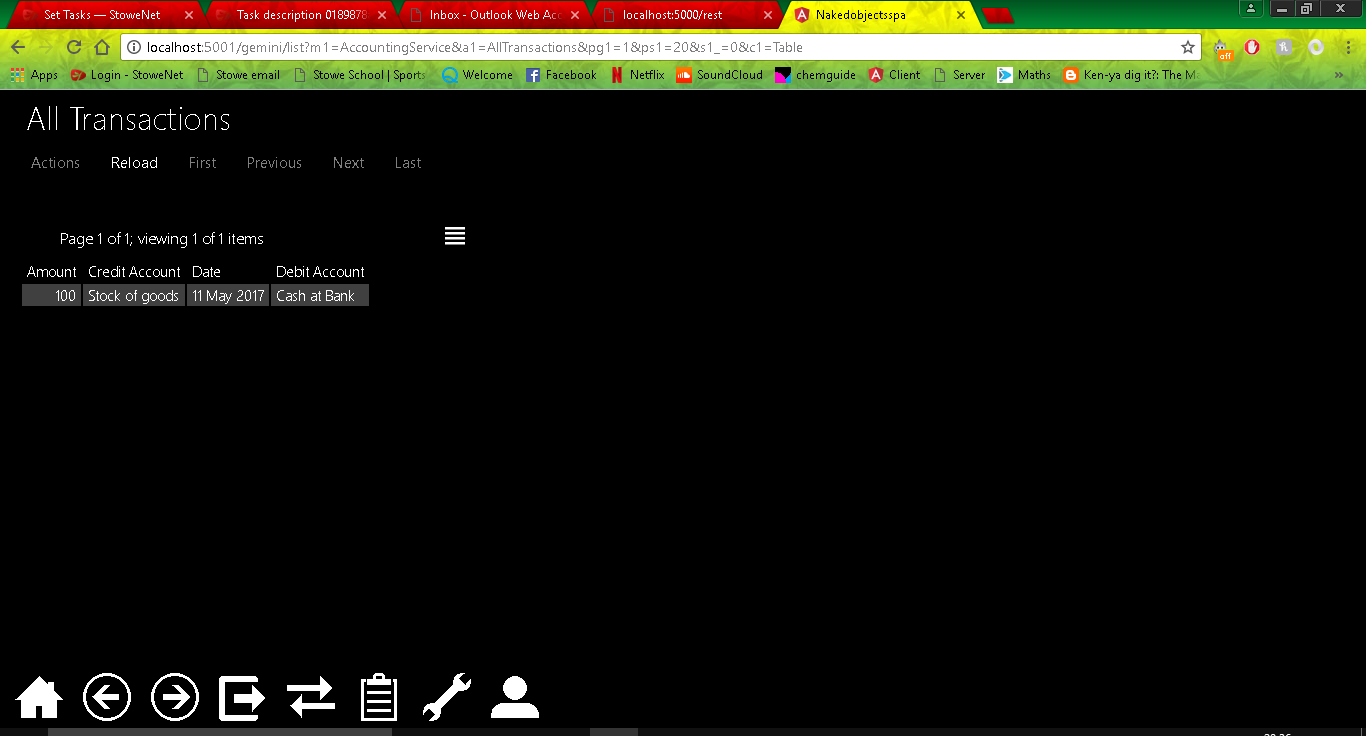
Above is the Transaction class which contains properties for the amount of money, the Date, and then the accounts it is affecting.

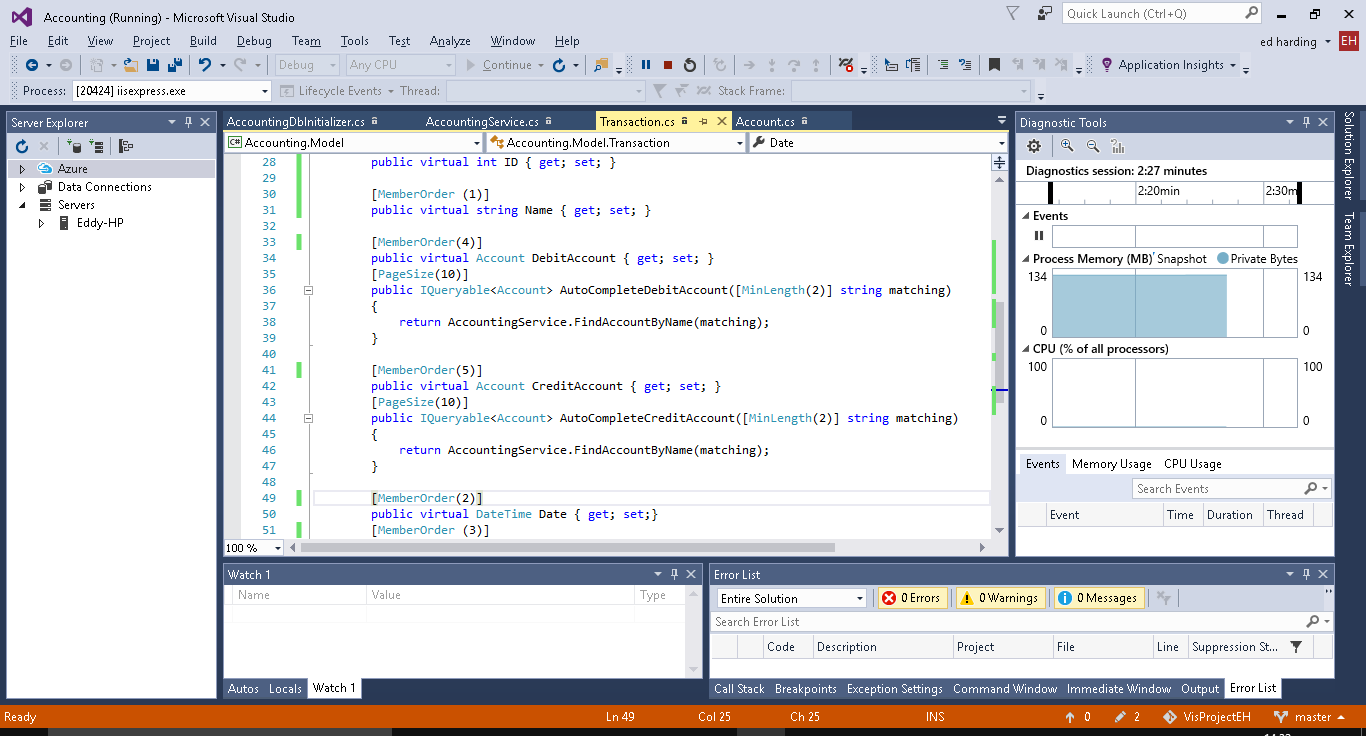


Above are the different methods (services) which can be used on both/either the transaction and/or account classes. Examples of them being used in the User interface will be shown below, they all do what you would expect based on their names.

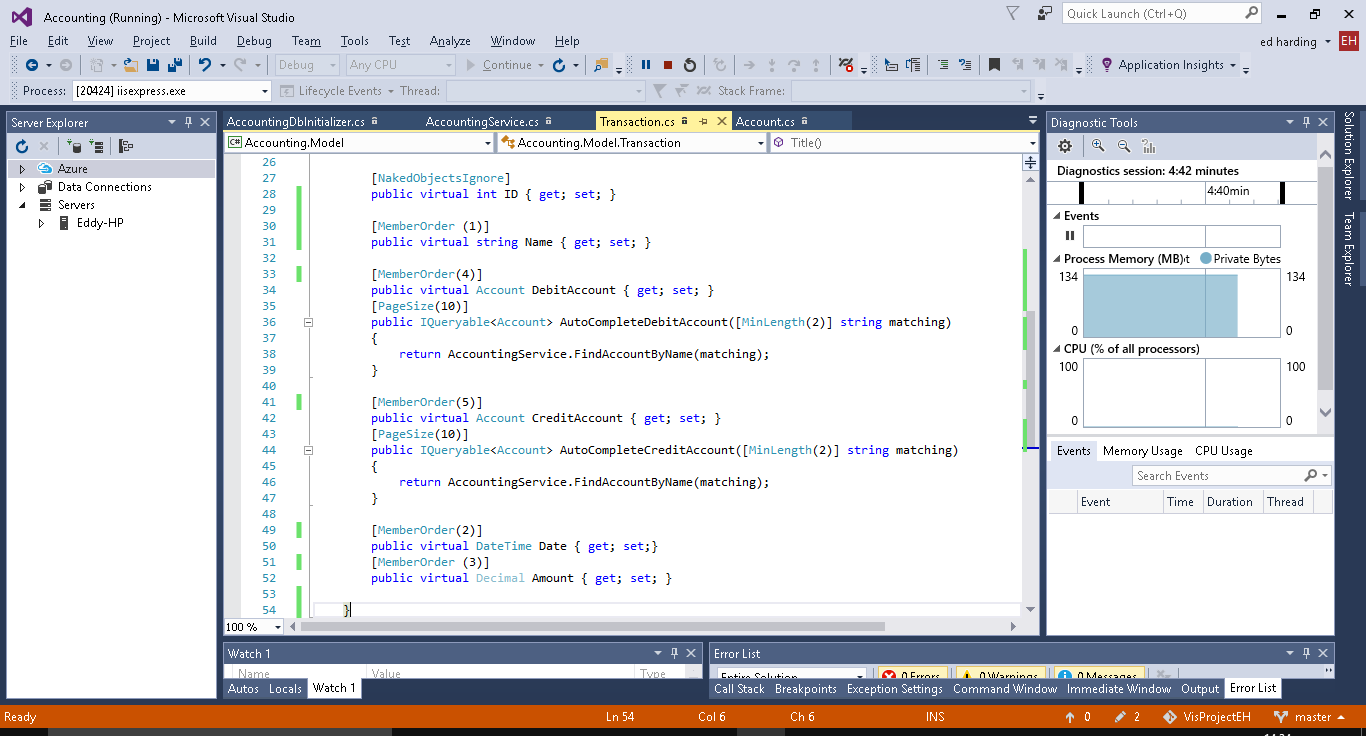




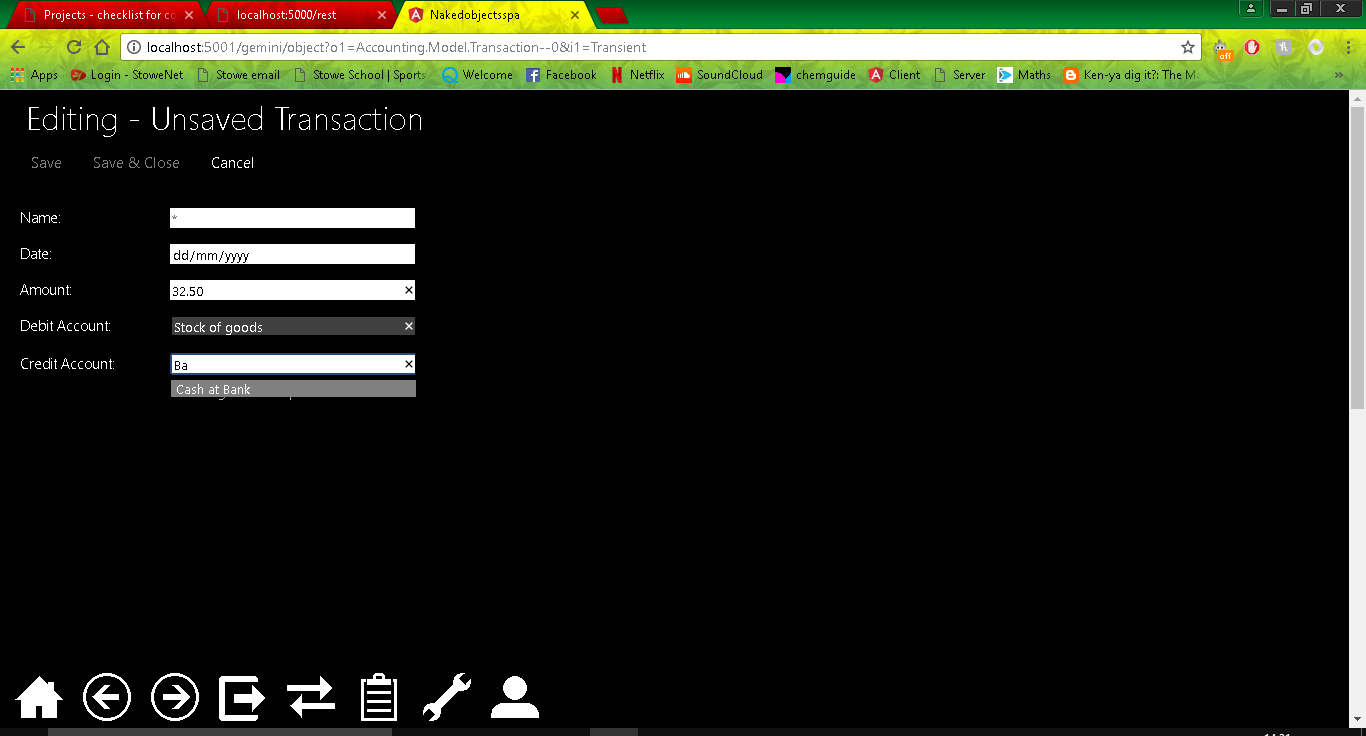


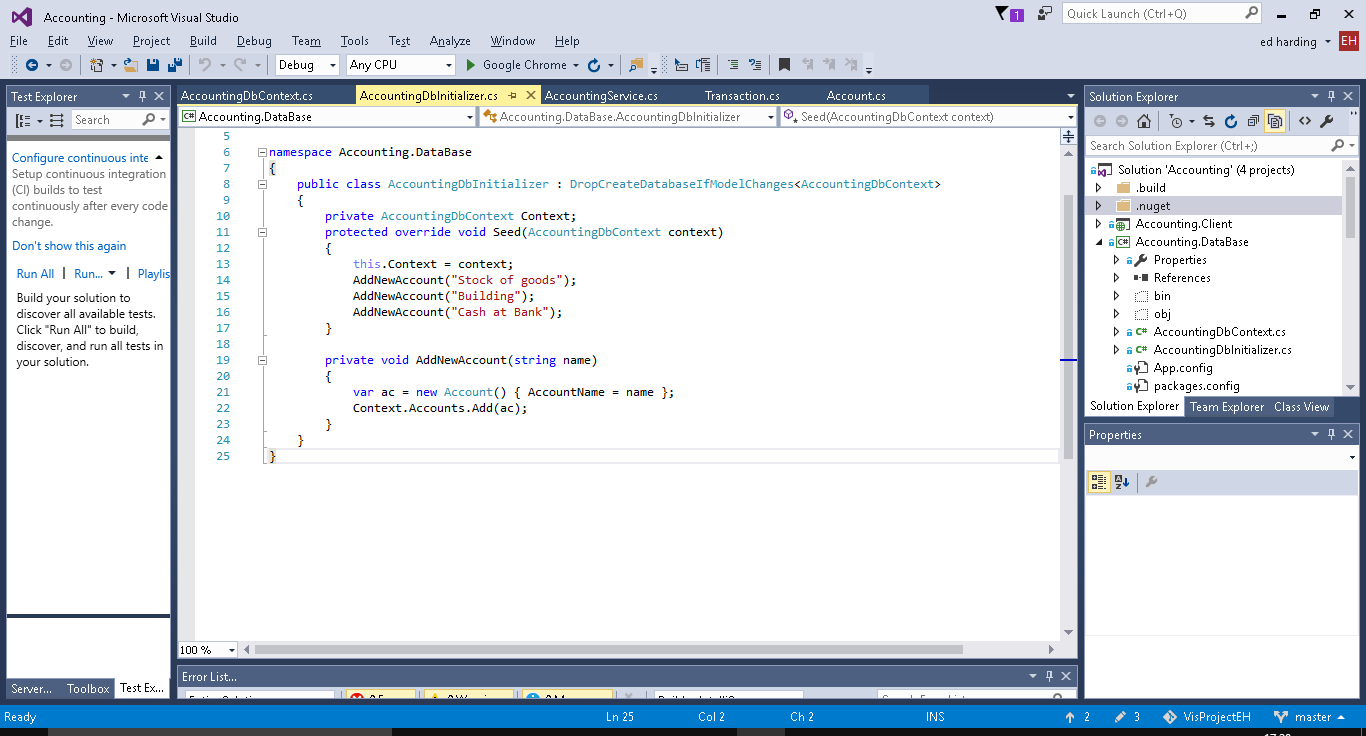
I decided to add in auto complete methods to the Debit and Credit Account fields when creating a transaction this was just to increase usability of the program as it means that you will not have to drag and drop, this will also help if there are a lot of accounts. 

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.



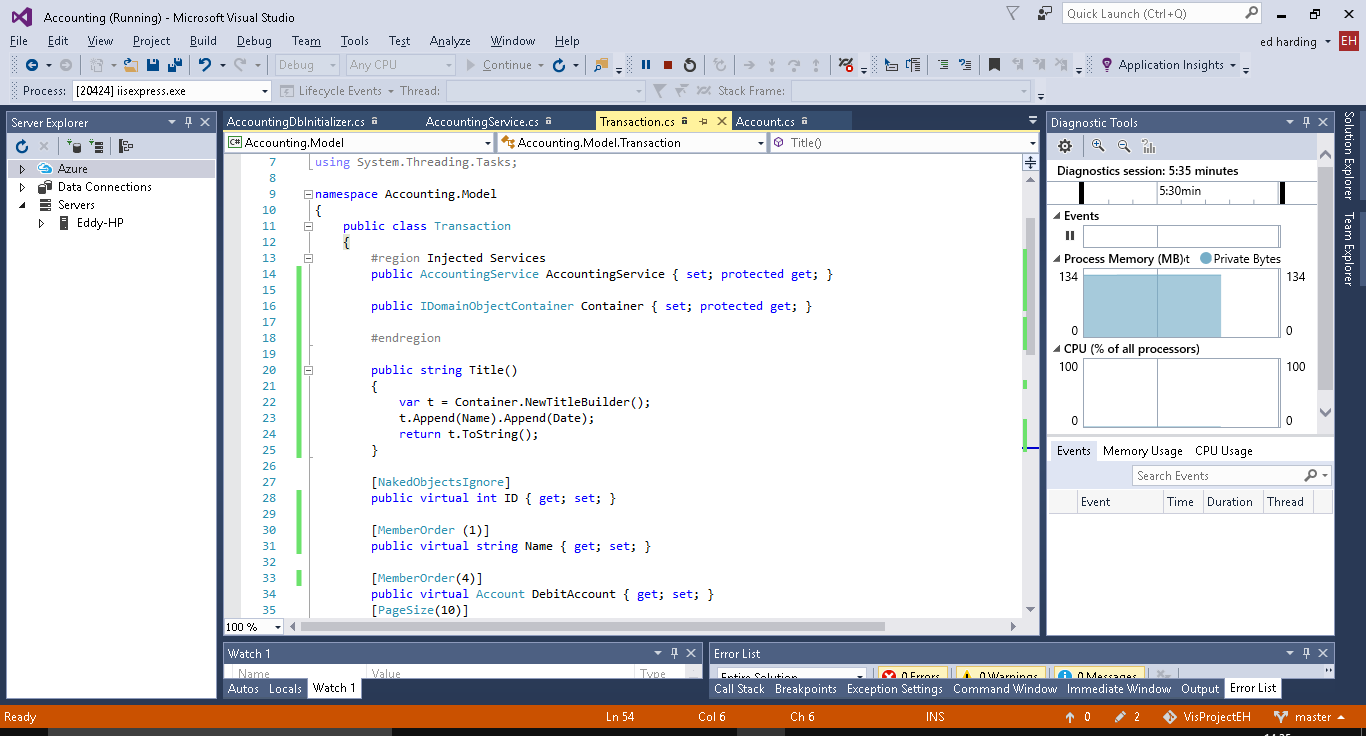
In the screenshot below you can see that that there is a new property called Name, the properties are ordered as to the member order and the autocomplete methods for the Debit and credit accounts is working.

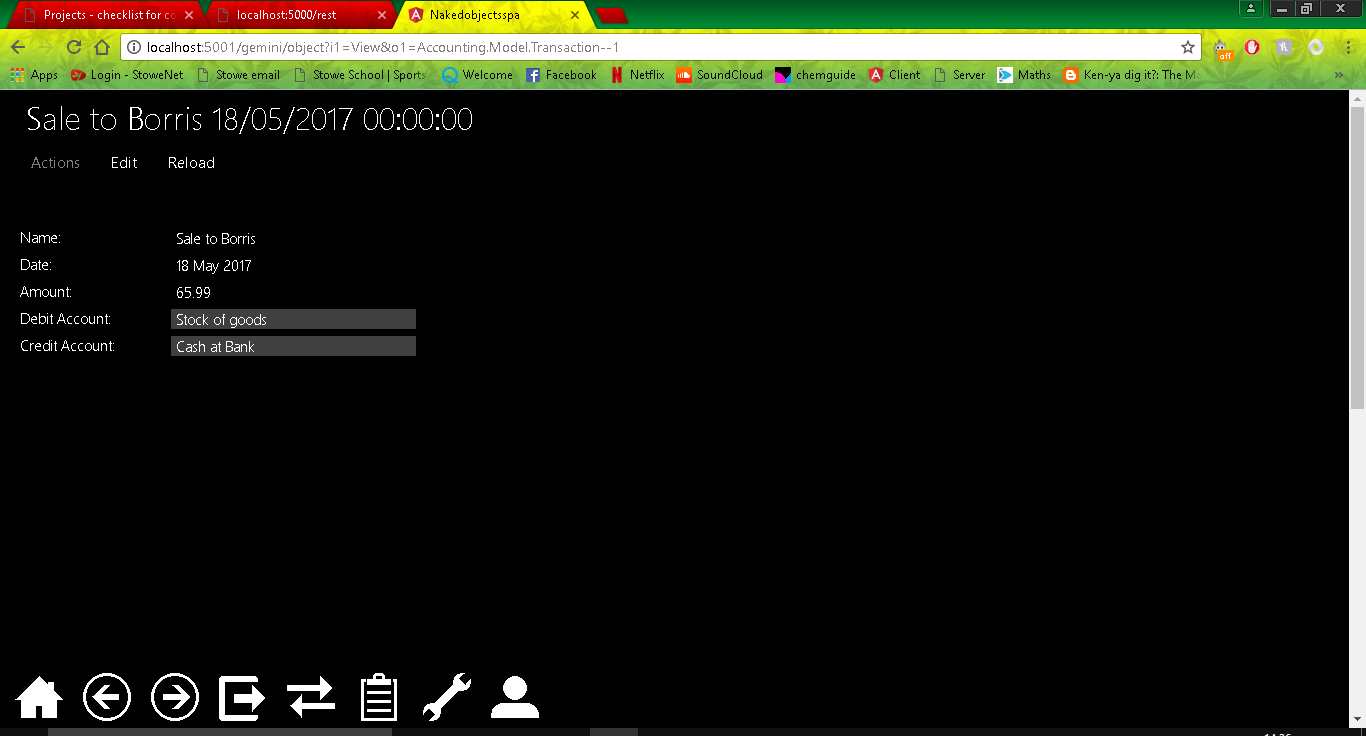




Above is some seed data for some Accounts and the method used to create them, this is in the DB initializer.

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.



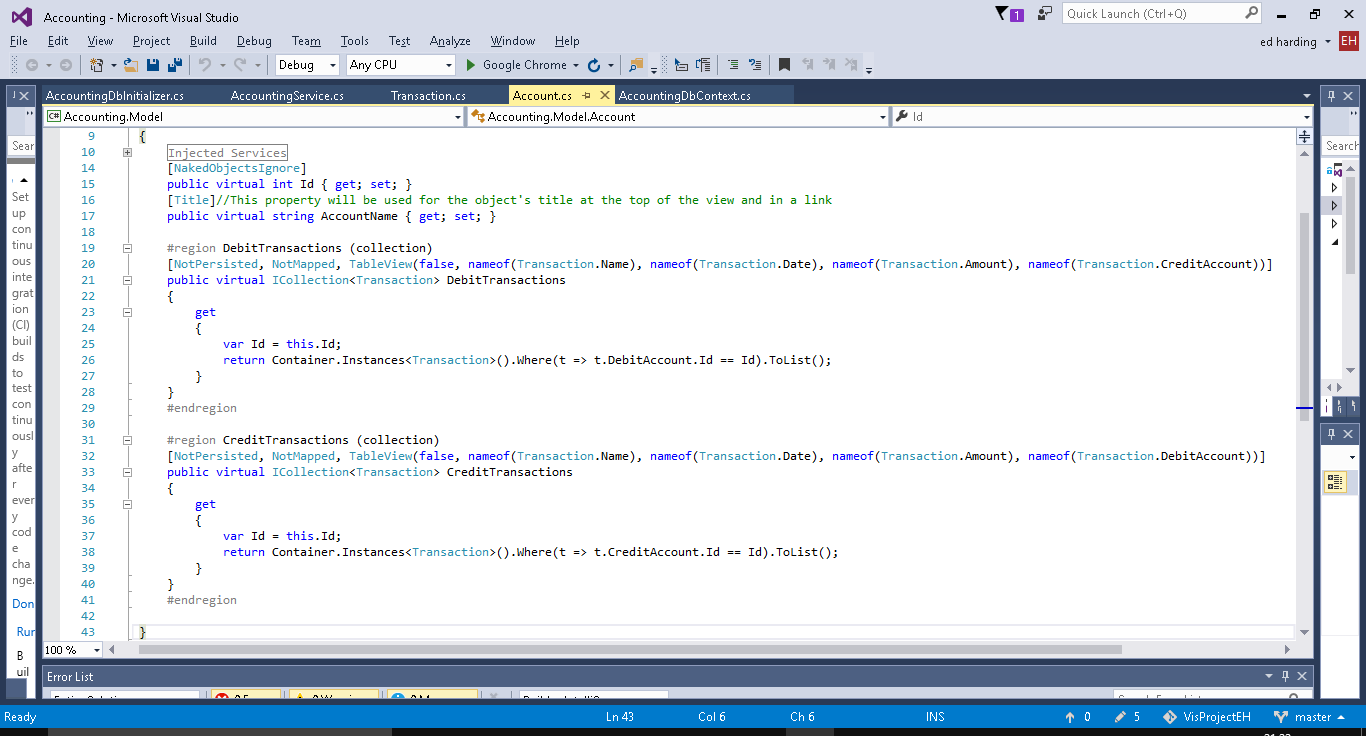


## 2

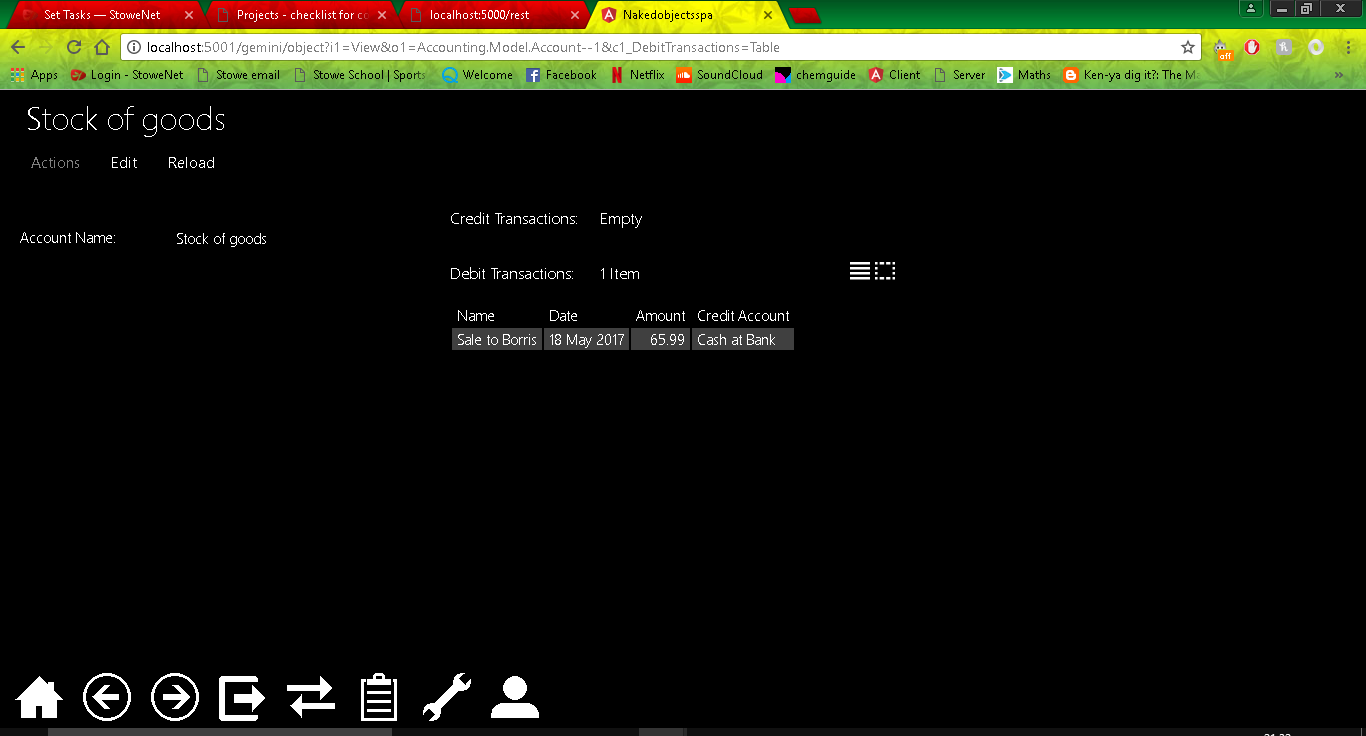
For Iteration 2 my brother (the primary user of the product) has said that he would like to be able to see the debit and credit changes in a balance sheet format. He would like to be able to see the total value of the credit and debit transactions made to an account, and then be able to open a table 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.

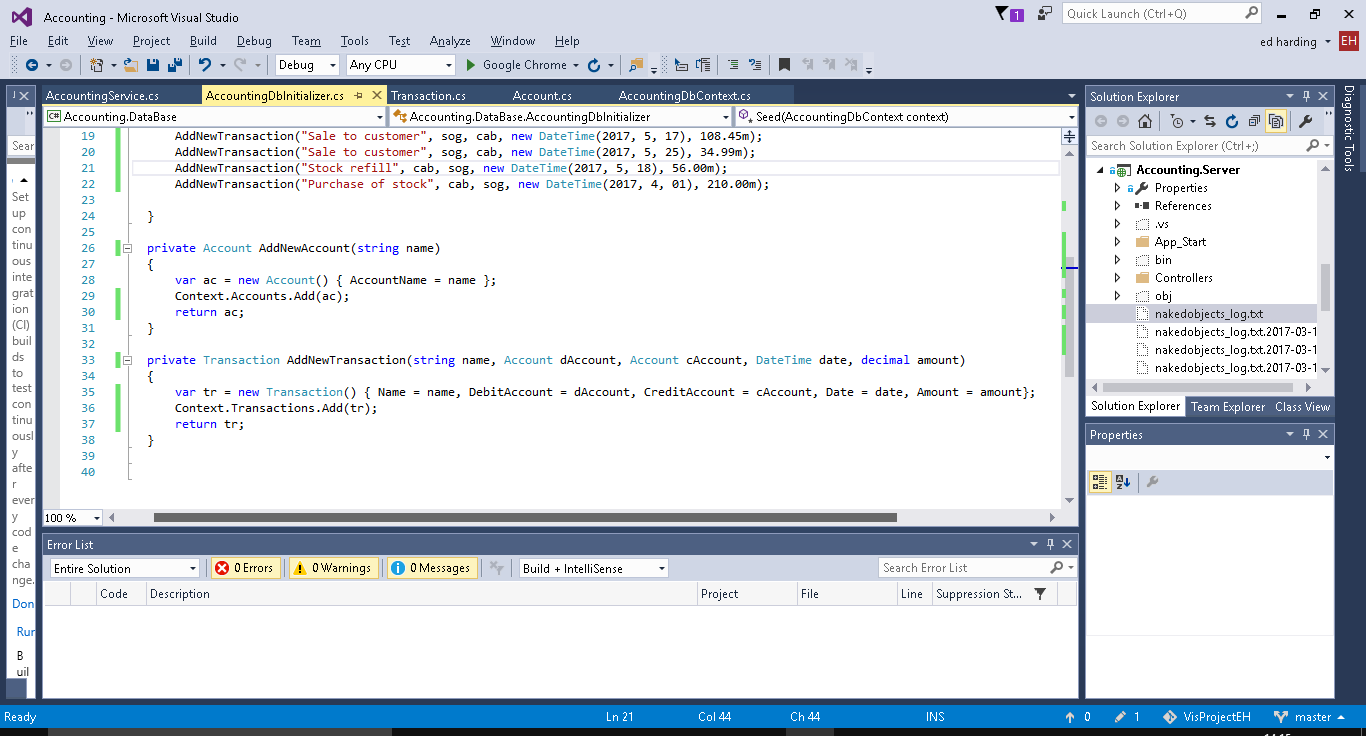
The first thing I have decided to change is how the list of transactions is displayed when looking at an Account. When the debit or credit transactions were displayed in table form you would be able to see the accounts it effects however one field of that table whether its debit or credit account would 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.

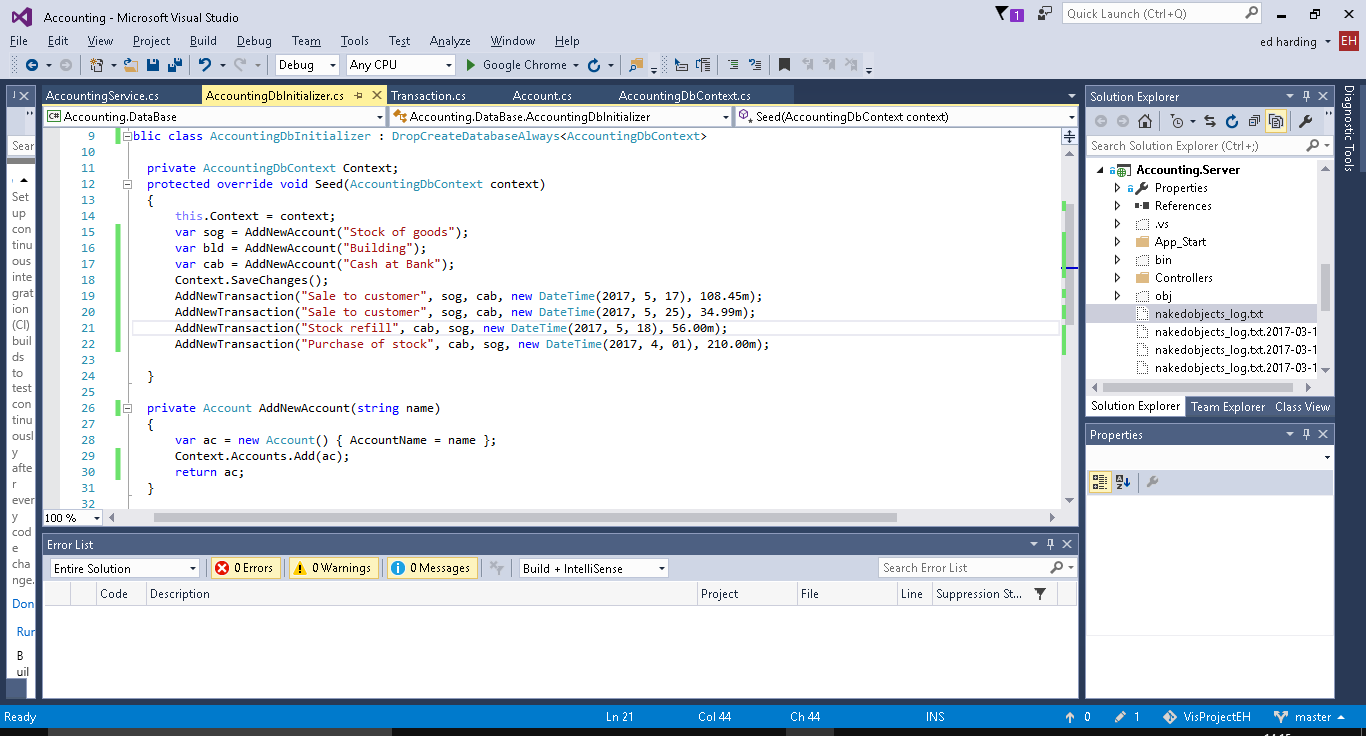


To do this (as you can see above) I used the ‘TableView’ attribute which allows me to designate which properties are visible when put into table view.

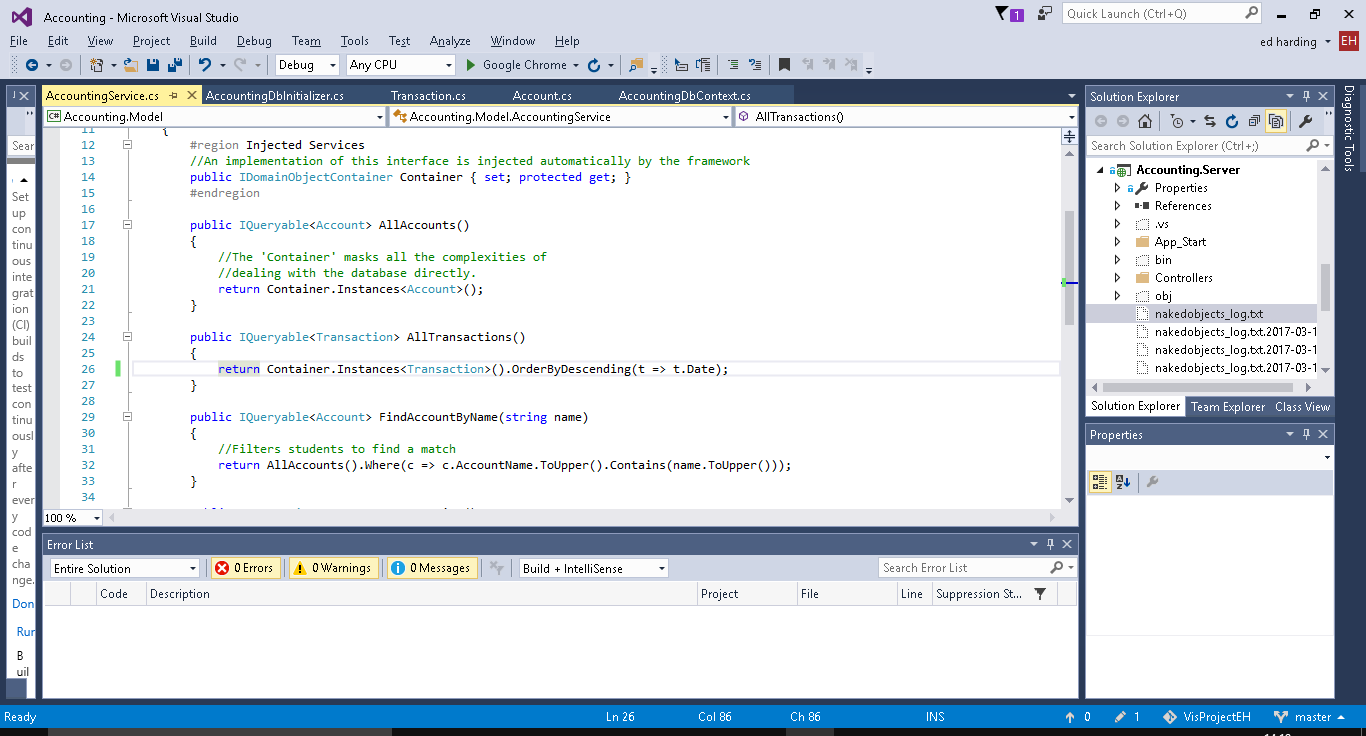


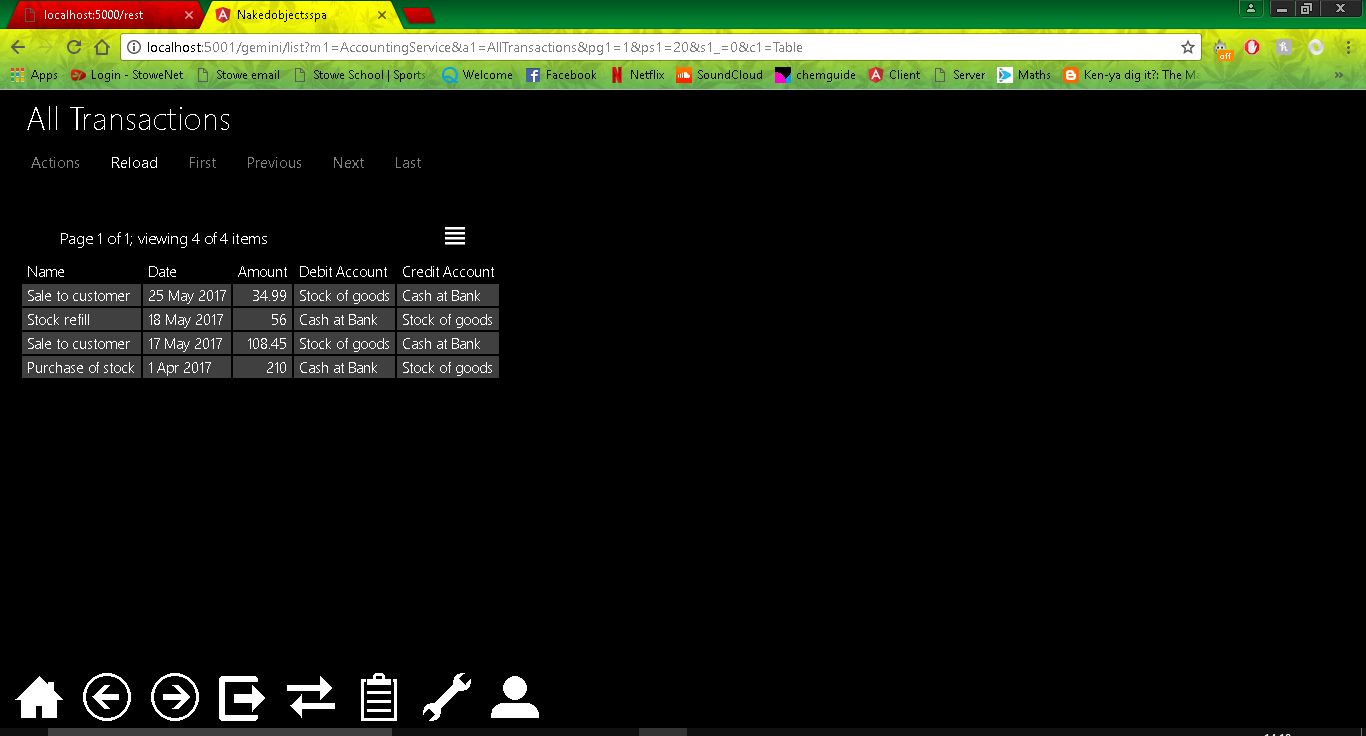
Now that I have done this I decided I should add in some seed data for some past transactions. Currently I have to add all the transactions for when I want to test something therefore having some premade will not only make the system look more like a real life system therefore improving testing but will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





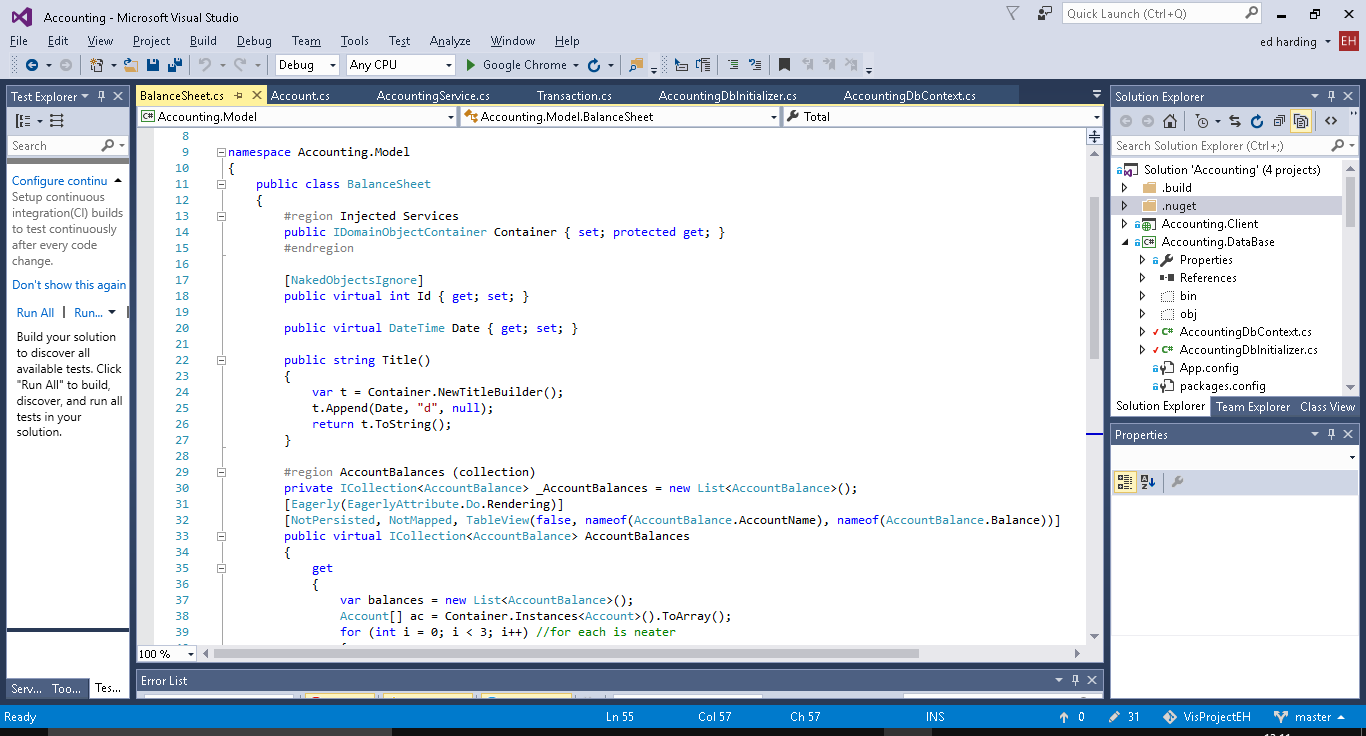
I also added linq code to the AllTransactions method so that all of the transactions are ordered from newest to oldest.

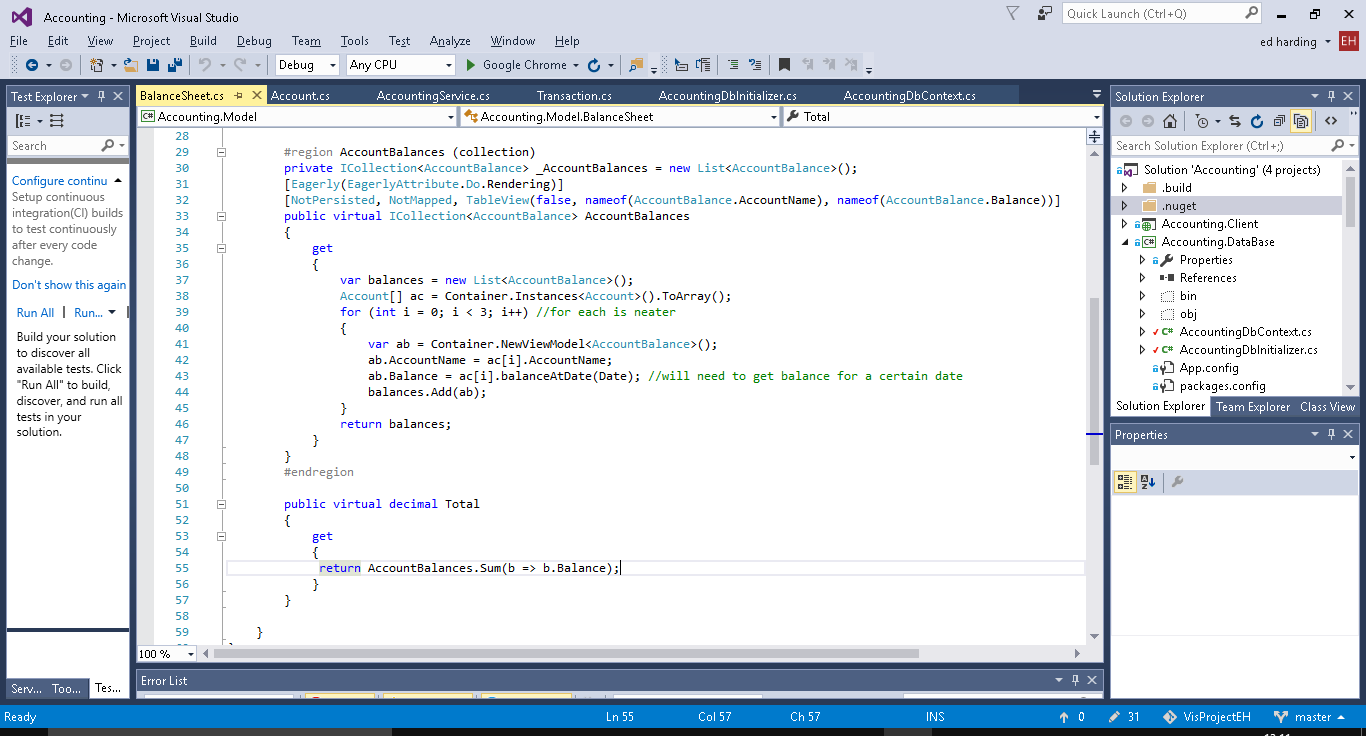




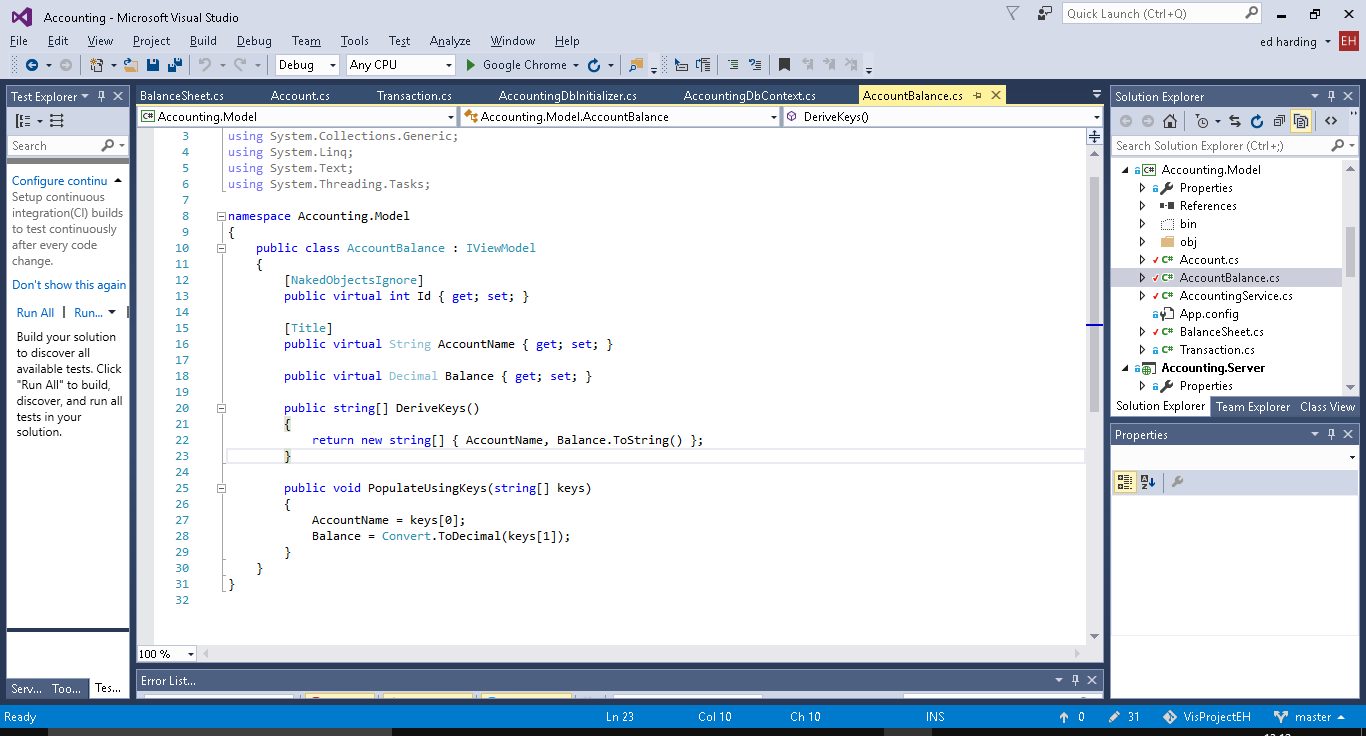
Below are some screenshots of the code for the balance sheets and examples of them in action.

Balance Sheet Class.

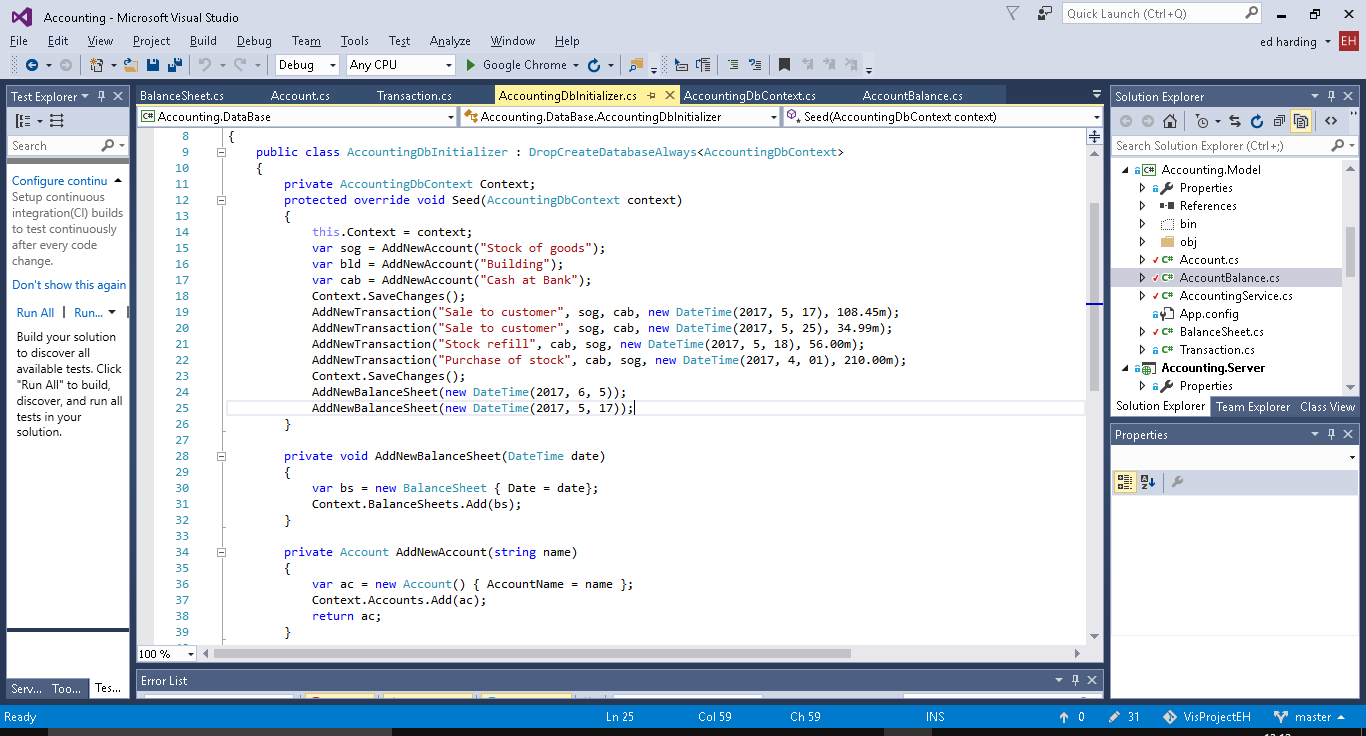




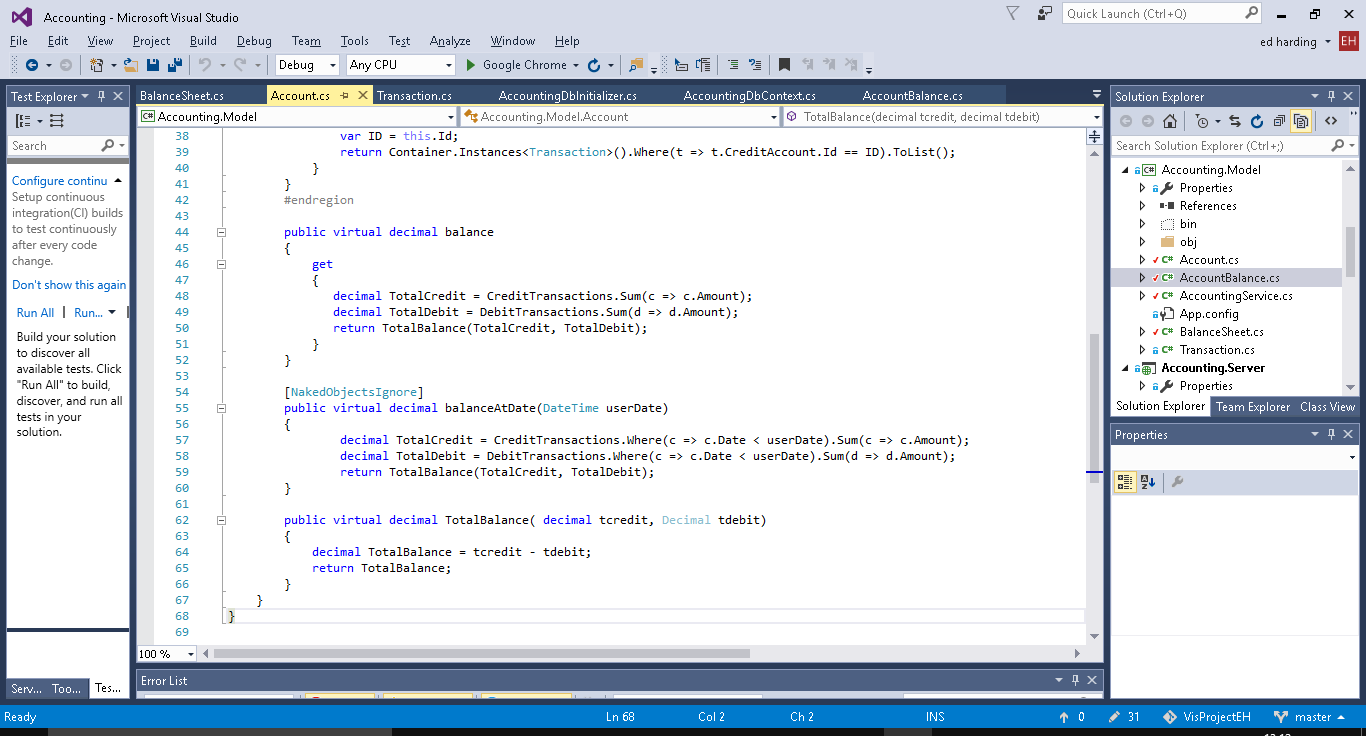
Account Balance Class



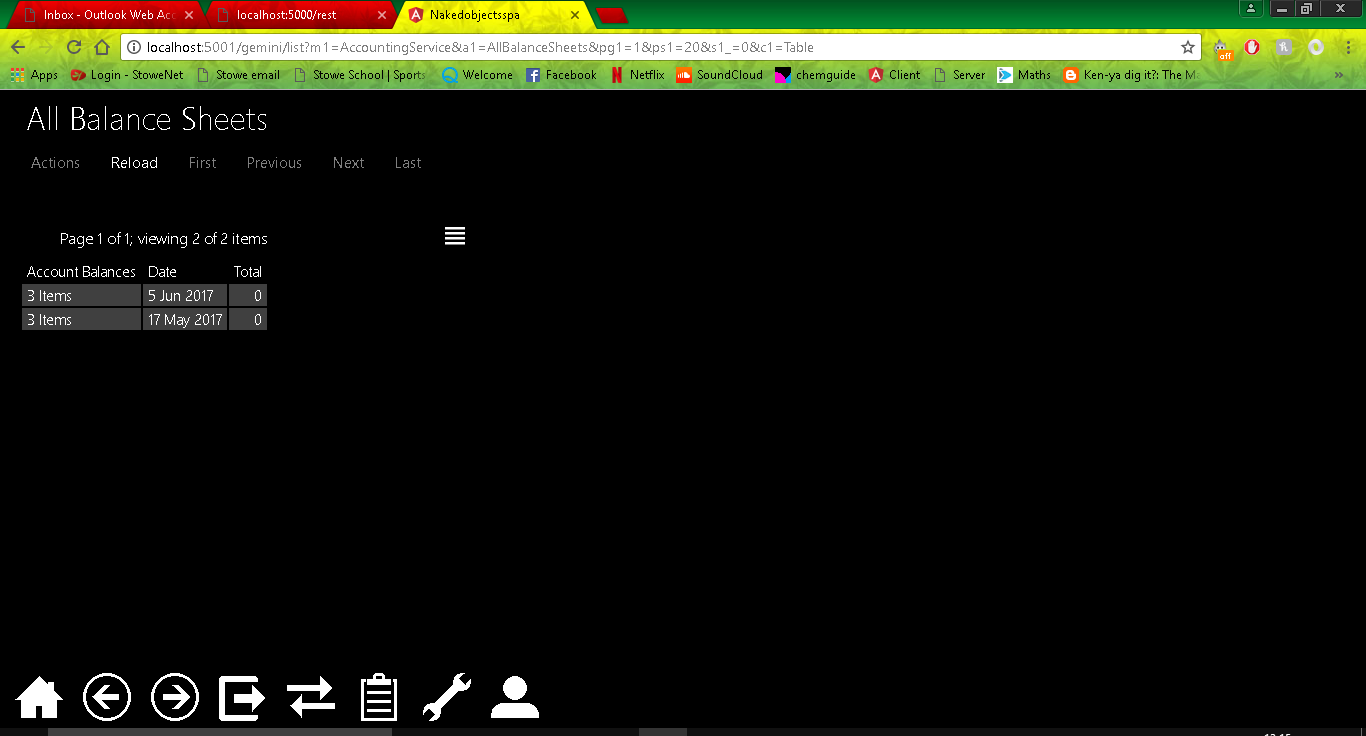
Balance Sheet Seed Data

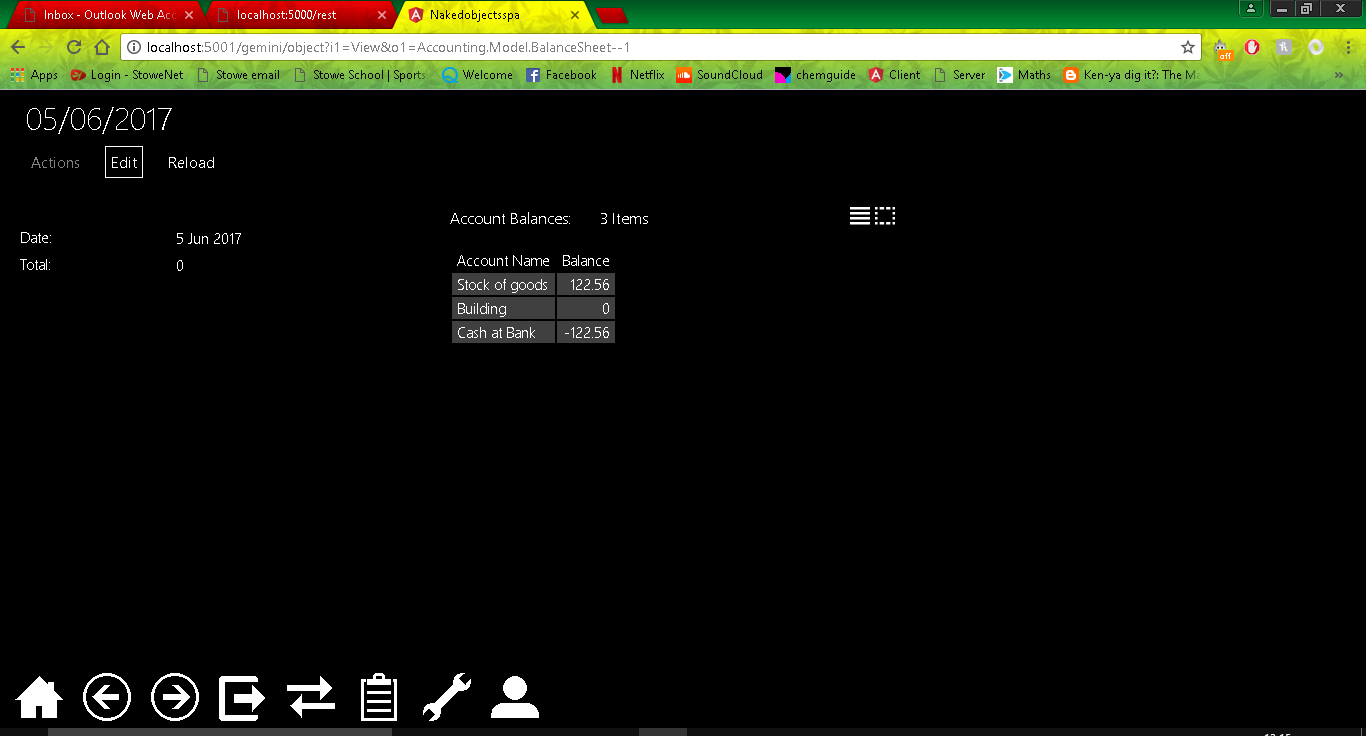


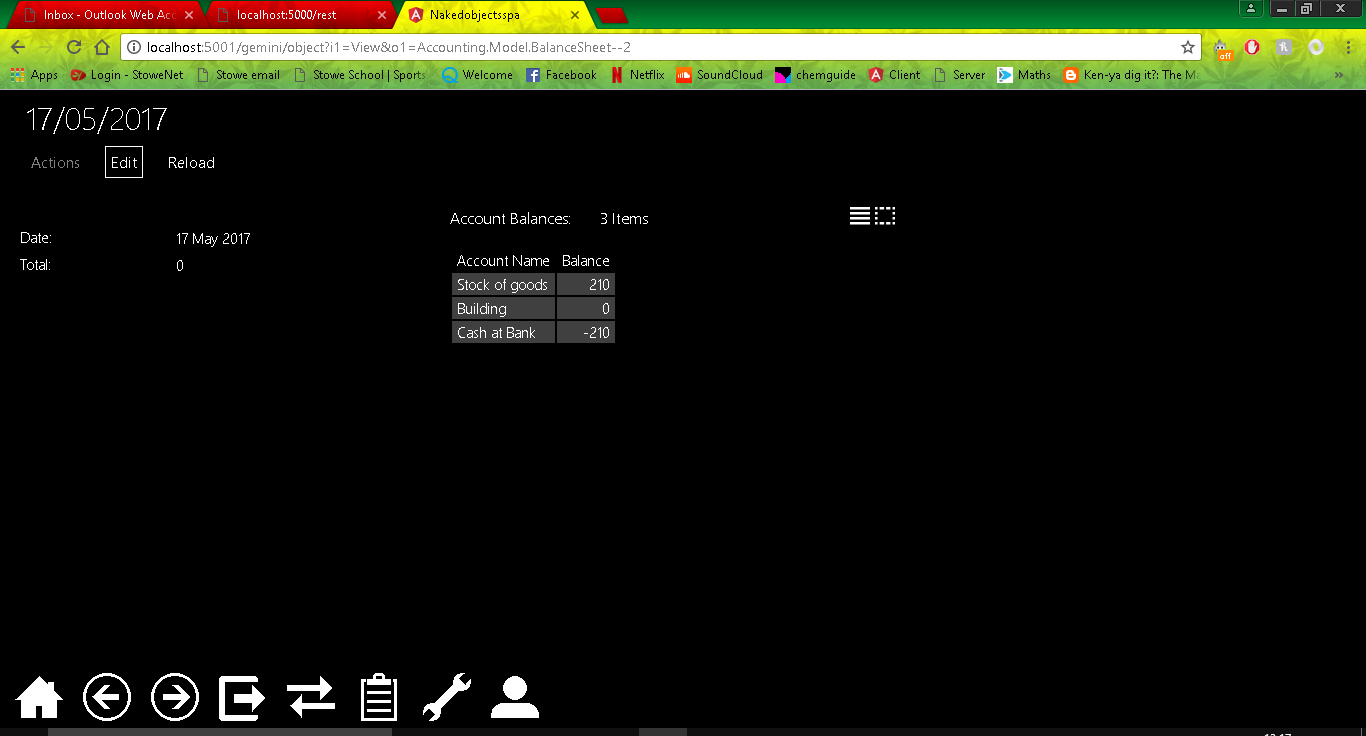
New methods on Account for balance calculations



Tests/proof





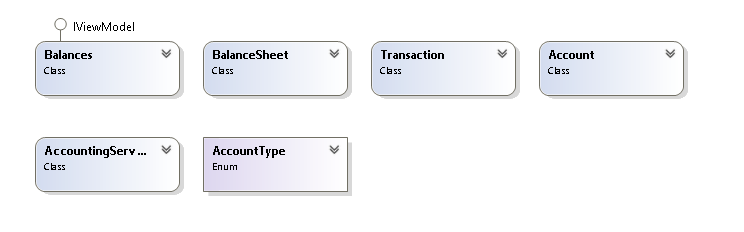


## 3

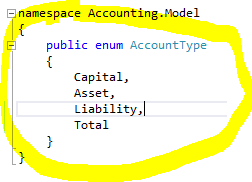
The objective for iteration 3 is to transform the current balance sheet into a more realistic one which will calculate the value of all of the assets minus the liabilities which should equal the sum of the capital accounts.

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.

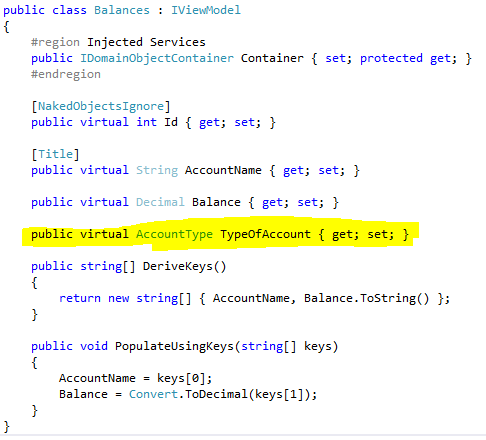
Current Class Diagram:



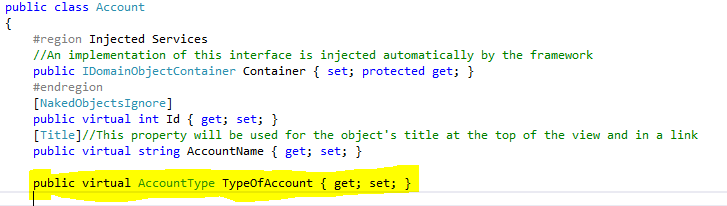
Enum for the Account Type:



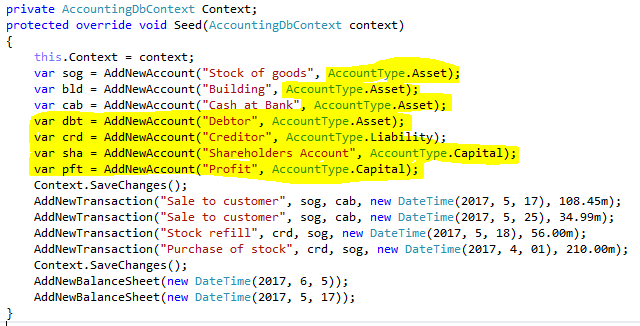
Addition of the Account Type field to Balances class.



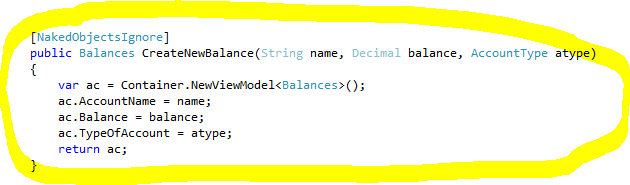
Addition of the Account Type field to Account class.



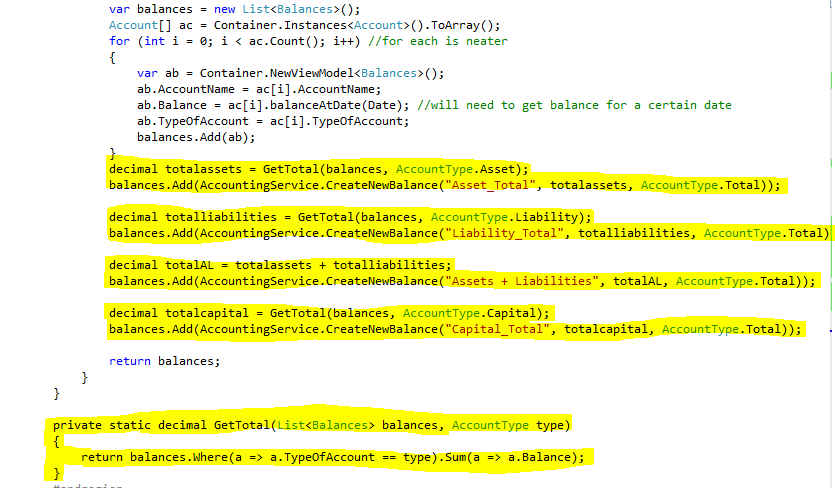
Addition and modification of seed data to include new fields.



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

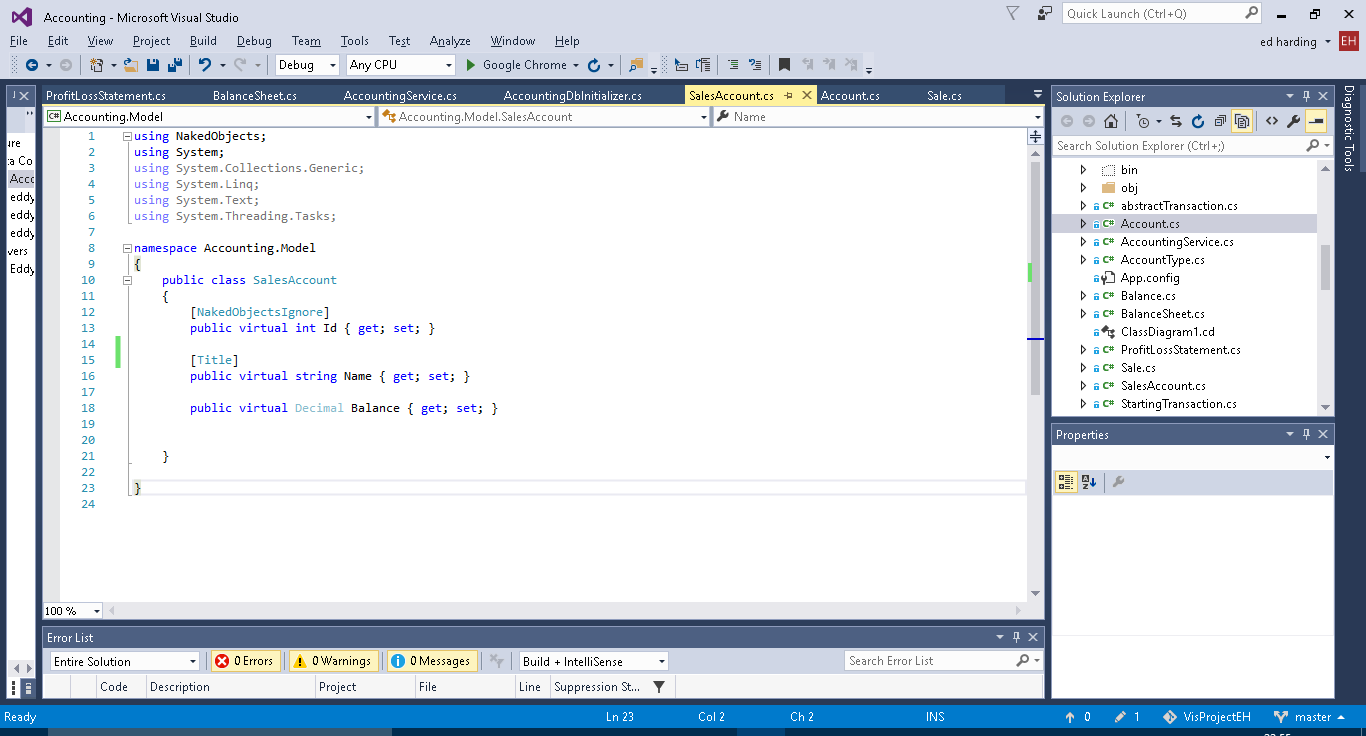
Profit and Loss Statement.

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.

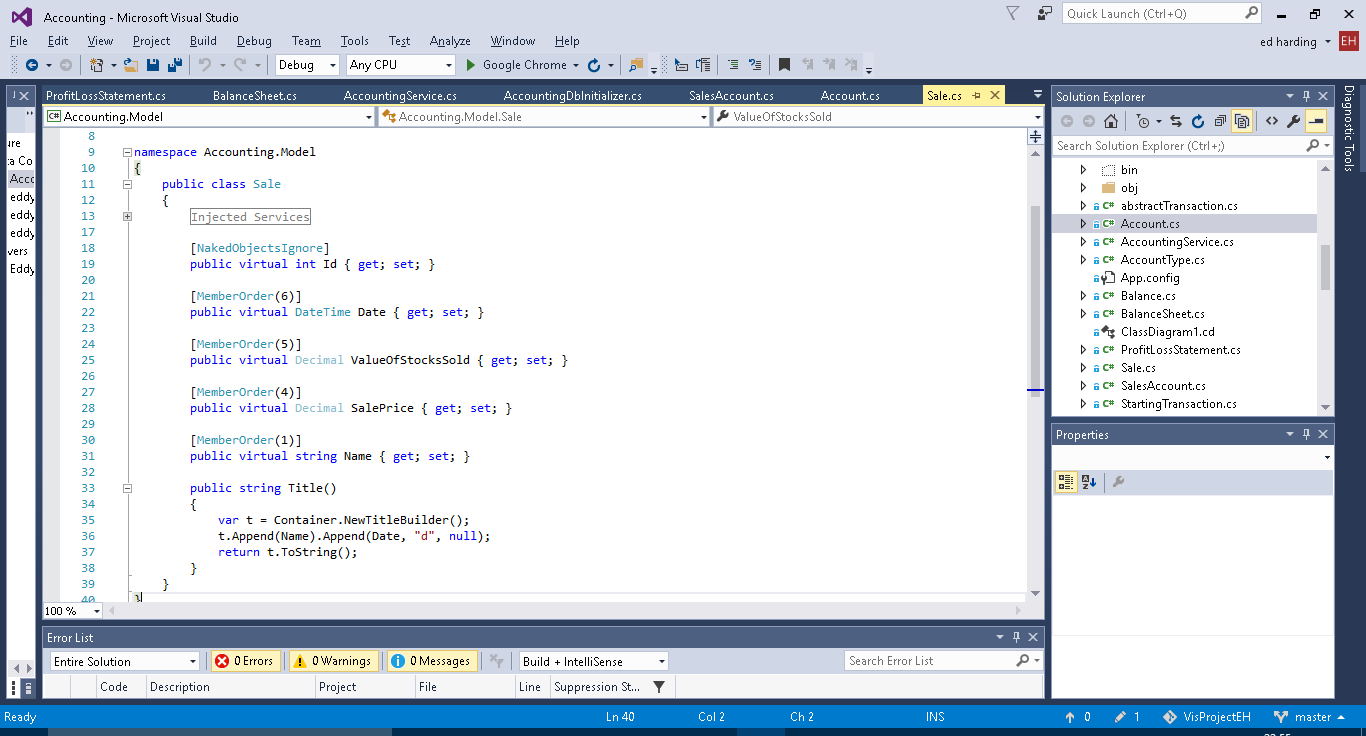
I will have to it this way because if I want the user to be able to check profit and loss for specific stretches of time then the accounts will need to contain a collection of all the previous sales transactions. The planned system will be very similar to the way the account, balance and balance sheet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

This will currently only calculate the gross profit.

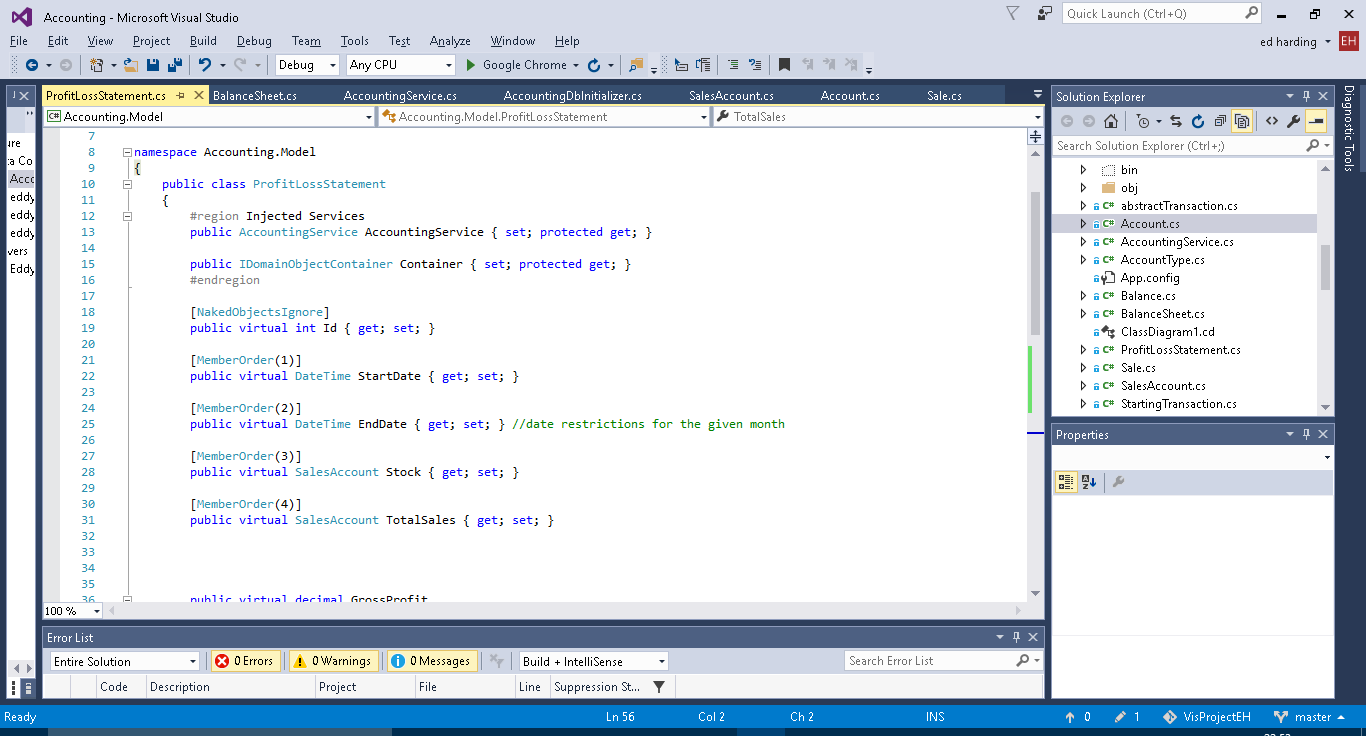
Screenshots below of new class called SalesAccount. It is used to hold the value of the stock from the stock account and the prices goods were sold for when a sale transaction is made using the Sale class below.

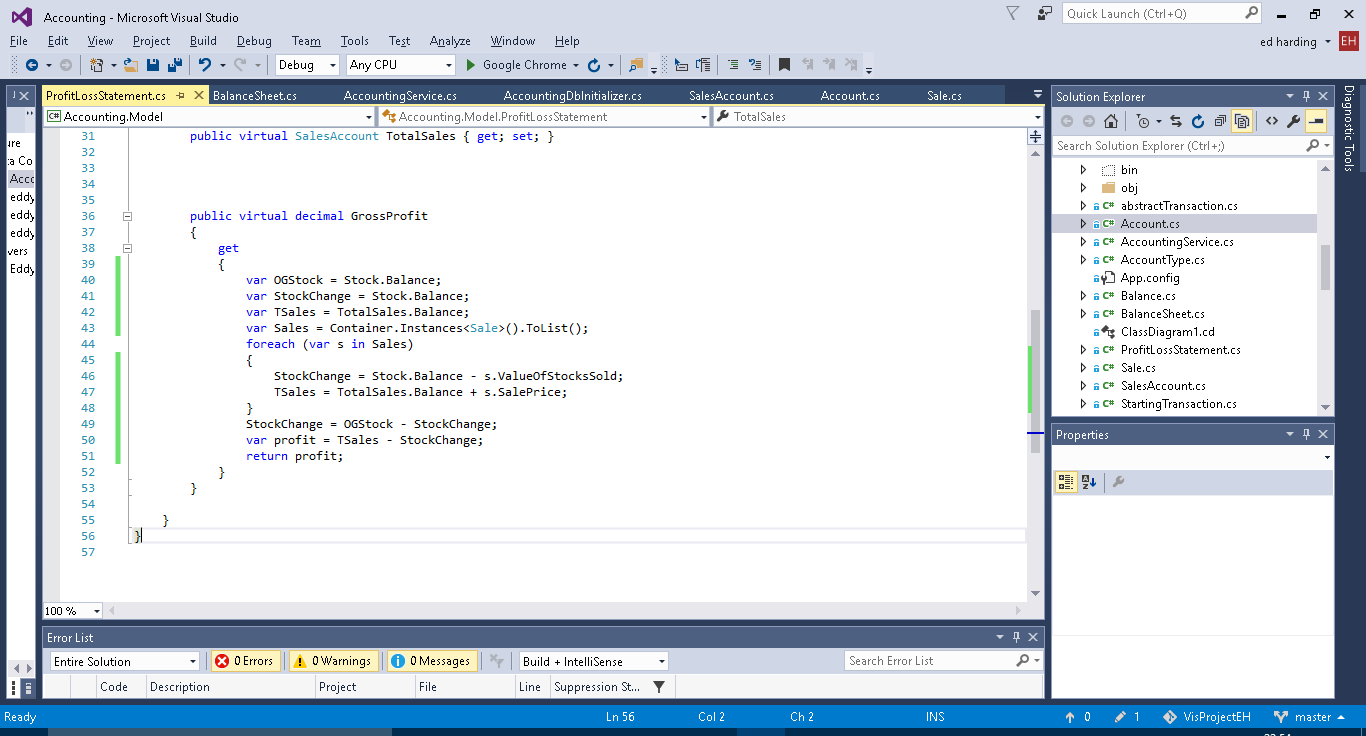


Screenshots of Sale class. Used in a similar way to transaction however it contains the date of the sale and information on the price of the stock sold and the price it was sold for. This is used to calculate gross profit.

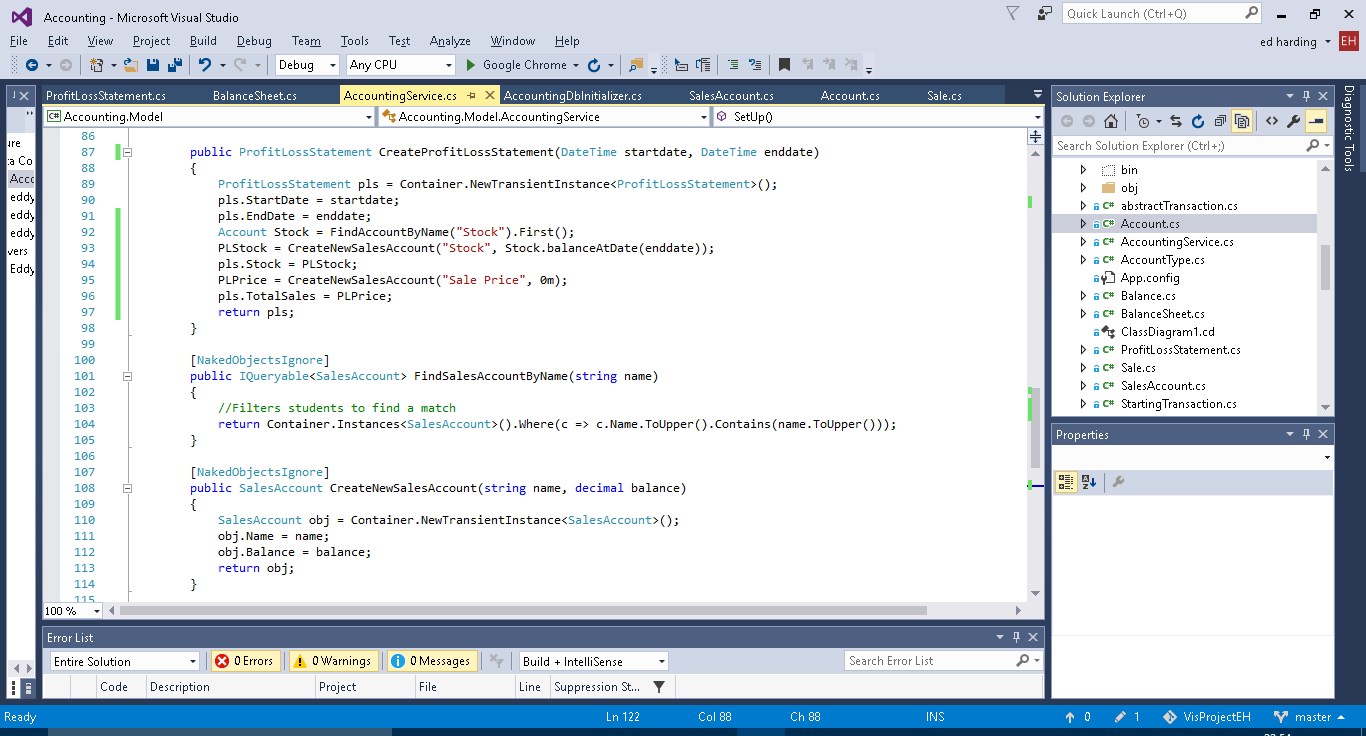


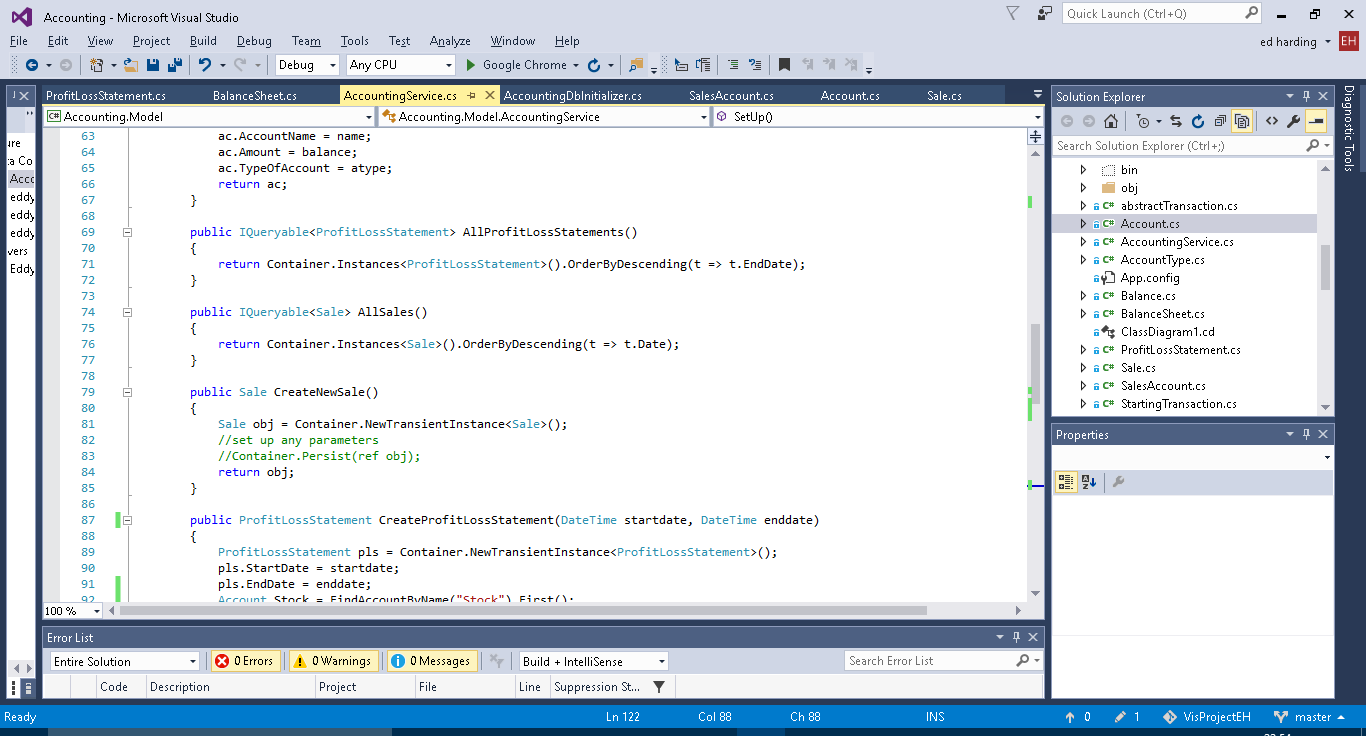
Screenshots of the new Profit and Loss Statement class. It pulls together the above to classes. The get statement is used to calculate the gross profit and loss from the sales transactions.



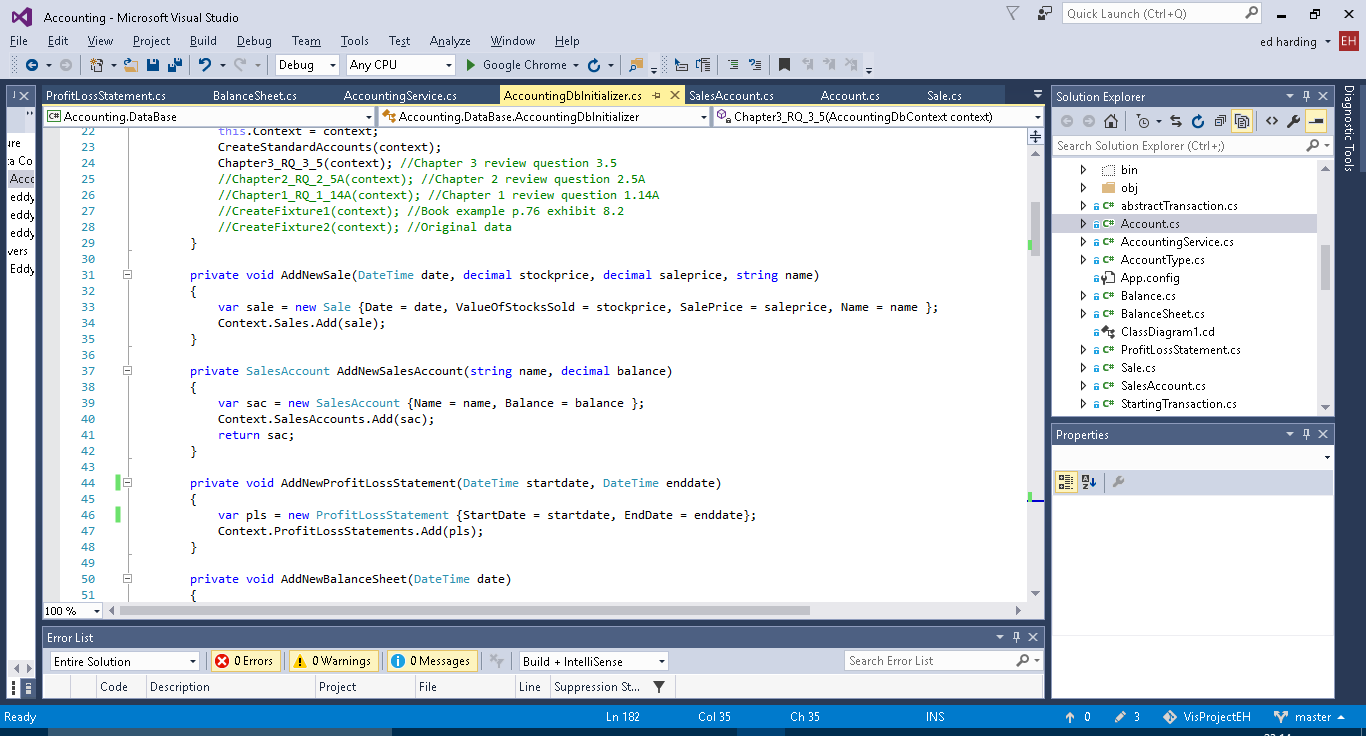


Screenshots of the new functions added to accounting services (i.e. the menu).





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.

# Testing

* Test Table and results, explain results especially if not expected result.
* Test/demonstration video, should start scripting soon, will help pick up on anything I need to fix or add to the program.

# Evaluation

* Did it meet end user requirements.
* What I could do better/improve with more time.
* What it should have if it was a complete business level program.