# A-LEVEL COMPUTER SCIENCE PROJECT

## Inventory management system

OCR A-Level Computer Science

Townley Grammar School
Arun Patel

# Table of Contents

Analysis ..... 5
1 Problem identification ..... 5
1.1 Description of the organisation ..... 5
1.2 Description of the problem ..... 5
2 Stakeholders ..... 6
2.1 Stakeholders who will have an interest in the solution ..... 6
3 Research the problem ..... 6
3.1 Analysing the problem ..... 6
3.2 Existing solutions ..... 8
3.3 Advantages and disadvantages of research ..... 10
3.4 Forms of data ..... 11
4 Data collection ..... 13
4.1 Questionnaire ..... 13
4.2 Interview ..... 16
5 Diagrams ..... 17
5.1 The current system ..... 17
5.2 Capabilities and Limitations of the current system ..... 19
6 The proposed solution ..... 20
6.1 Solution requirements ..... 20
6.2 Summary of the solution to be created ..... 23
7 Signatures ..... 24
Design ..... 25
1 System objectives ..... 25
1.1 User interface ..... 25
1.2 Input ..... 25
1.3 Processing ..... 25
1.4 Output ..... 25
1.5 Internal Objectives ..... 26
2 Scheduling ..... 26
2.1 Design plan ..... 26
2.2 Development plan ..... 27
3 New System ..... 27
3.1 Use-Case diagram of new system features ..... 27
3.2 Input Output diagrams ..... 32
3.3 Inventory Database ERD ..... 32

3.4 Data Dictionary ..... 34
3.5 Database normalisation ..... 36
4 User Interface ..... 37
4.1 Splash screen design ..... 37
4.2 Login screen design ..... 37
4.3 Main menu ..... 38
4.4 Inventory page ..... 39
4.5 For sale page ..... 40
4.6 Performance page ..... 43
5 Data capture ..... 45
5.1 Validation ..... 45
5.2 CRUD permissions ..... 47
6 Algorithms ..... 49
6.1 Structure diagram ..... 49
6.2 Class diagram ..... 51
6.3 Algorithm design ..... 52
6.4 Database Queries ..... 59
7 Test Strategy ..... 61
7.1 Test plan ..... 63
8 Signatures ..... 69
Development ..... 70
1 Interface ..... 70
1.1 Splash Screen ..... 70
1.2 Login window ..... 72
1.3 Main window ..... 75
1.4 Login window user controls ..... 79
1.5 Main window user controls ..... 92
2 Interface changes ..... 98
2.1 Adding the filter section to the inventory user control ..... 98
2.2 Adding the filter section to the for sale user control ..... 99
2.3 Adding a maximise and minimise button ..... 99
2.4 Adding a background to Inventory and for sale table ..... 100
2.5 Making the interface resizable ..... 102
3 Interface prototype ..... 112
4 Database configuration ..... 116
4.1 SQL Server ..... 116

4.2 Table relationships ..... 118
4.3 Auto-numbering fields and testing relationships ..... 119
4.4 Connecting the database ..... 121
4.5 Setting up CRUD operations ..... 124
5 Creating user accounts ..... 146
6 Transferring items from inventory to the for sale table ..... 150
7 Action logging ..... 156
8 Implementing CRUD permissions ..... 160
8.1 Manager Permissions ..... 160
8.2 Senior Staff Permissions ..... 162
8.3 Staff Permissions ..... 163
9 Performance analysis ..... 163
9.1 Performance page user control ..... 163
9.2 Creating performance entries ..... 166
9.3 Top five best performing items ..... 168
9.4 Top five worst performing items ..... 171
9.5 Determining if an item will sell well in the current month ..... 171
9.6 Individual item performance analysis ..... 174
9.7 Performance graphs ..... 177
Evaluation ..... 181
1 Testing the robustness of the solution ..... 181
1.1 testing against test plan ..... 181
1.2 End to End test ..... 190
1.3 Usability testing ..... 195
2 Success criteria evaluation ..... 199
3 Unmet success criteria ..... 201
4 Resolving usability issues ..... 201
4.1 Account creation message ..... 201
4.2 Transferring some but not all of an item ..... 202
5 Maintenance issues and limitations of the solution ..... 202
5.1 Maintenance issues ..... 202
5.2 Solution limitations ..... 203
5.3 Resolving solution limitations ..... 203
Code listing ..... 205
LoginAbout.xaml.cs ..... 205
LoginCreateAccount.xaml.cs ..... 205

LoginForgotPassword.xaml.cs ..... 207
LoginHelp.xaml.cs ..... 207
LoginMainCtrl.xaml ..... 207
MainForSale.xaml.cs ..... 208
MainInventory.xaml ..... 211
MainPage.xaml.cs ..... 215
MainPerformance.xaml.cs ..... 216
AddNewItemWindow.xaml.cs ..... 217
AddNewItemWindowForSale.xaml.cs ..... 219
GraphDisplayWindow.xaml.cs ..... 220
Login.xaml.cs ..... 221
logWindow.xaml.cs ..... 222
MainWindow.xaml.cs ..... 223
PeformanceAnalysis.cs ..... 224
SplashScreen.xaml.cs ..... 235
SQLServerAccessor.cs ..... 236
References ..... 246
Table of figures ..... 247

# Analysis

# 1 Problem identification

## 1.1 Description of the organisation

Best-One New Cross, is a convenience store selling everyday items such as food and drink to locals in the area. Having been established for more than 35 years, it has a well-established and loyal customer base all with different needs. When the shop was first opened, it featured a very small range of products, suited for a small audience with similar needs, however, as more and more customers, who had different needs, started to visit the shop, it needed to expand its product range to satisfy customers.

Best-One is a chain of convenience stores based in the UK and Jersey. The company has over 600 stores throughout the UK, with most of its stock being sourced by Best way cash and carry. Best-One New-cross was opened and ran by the current owner's father as a franchisee of the Best-One franchise. Once the current owner, Purnesh Patel, was of a suitable age and had enough experience, the shop was then passed on to him. As the shop grew and grew, the decision was made to buy the building next to it, allowing for more space for expansion

## 1.2 Description of the problem

When new stock is brought into the shop, a member of staff writes down the name of the item onto a piece of paper, which is stored in a filing cabinet. When stock is taken out of the storage room and put on display, the member of staff must find the item on the list and cross it out. This can be immensely difficult as there is often hundreds of items on the list all with similar names. It has also led to errors in the past when staff members cross out the wrong item. It is almost near impossible to work out who made the mistake since there is currently no way of tracking who made the changes. This can lead to blame being passed around to the wrong employees which can severely de-motivate the staff member. The prices of some items can change often, when this happens it can cause confusion as the prices of items stored in the system don't match others.

These problems can be solved by making use of computational methods. For example, abstraction can be used during the design of the interface, by not including unnecessary information. Instead of asking the user where the item came from, it can be suitable to ask the user the name of the item and the quantity of that item.

Furthermore, decomposition can be used to solve problem of working out who made an error when managing stock. For example, by decomposing the problem, I can understand that everyone who alters the inventory has a name. They will also do this at a specific time. From this, I can come up with a solution to this problem. By implementing a security system, the solution can detect who is making changes to the inventory. In addition, the solution can also log the time of day that the staff member altered the inventory. By doing this, I can record who has made changes to the inventory and at what time.

As searching for items is a huge issue for the organisation and often leads to errors. Queries can be implemented into the application that allows the program to search the database holding all the items for the desired record. Methods of computational thinking such as thinking ahead can be used here in the form of caching. Frequently searched for items can be cached, so that the user can quickly go to the items details without having to waste time. Indexing can also be implemented into data stored in the system, this allows the system to search for data quicker, making queries on large

pieces of data take a shorter amount of time. Indexing involves putting markers into sections of the list of data to indicate the data underneath are all of a same category.

Automation can be used when determining which food products have gone out of date. Rather than the user have to analyse a list of products and find which products have gone out of data , which can lead to errors such as missing an out of date item, the solution could feature an automated alert to the user , making them aware of products that have gone past their expiry date.

Data collection allows me to take in information from the client. This could be in the form of questionnaires and interviews. I could question my client on what the solution to their problem should include and be able to do, ensuring the final solution is correct. This can be used together with data analysis, to make sense of the data I have collected, this could be achieved by making use of graphs and charts.

# 2 Stakeholders

## 2.1 Stakeholders who will have an interest in the solution

The main stakeholder of this solution is my client, Purnesh Patel. This is the person who has asked to create a solution to a problem that they have identified. This person will make the most use solution, therefor the solution must be suitable for this person. Since this stakeholder has limited experience in computer systems and using software, it is essential that the software is as user friendly as possible, which means that I can make use of a Graphical User Interface (GUI). A GUI has features such as windows, icons, menus and pointers, which can make software much easier for the user to use, especially since they have limited experience. The solution will also require the user to input a lot of data, therefor it is essential for the system to make use of validation. Validation is an automatic computer check to ensure that the data entered is sensible and reasonable. It does not check the accuracy of data.

Another stakeholder of the solution, is the members of staff at the shop. They could potentially make use of the system as much as the main client. This stakeholder will act as the audience of the solution, the audience is the person who will look at the finished product. Since some members of staff have a lot of experience in using Computer systems and others have little experience, I will have to design the solution in such a way that can be suitable for both kinds of audiences. For example, creating an interface with a lot of help dialogs and tooltips can make it very frustrating for skilled users to use the solution, whilst removing all help features from the solution can make it very hard and even impossible for inexperienced users to use the solution.

# 3 Research the problem

## 3.1 Analysing the problem

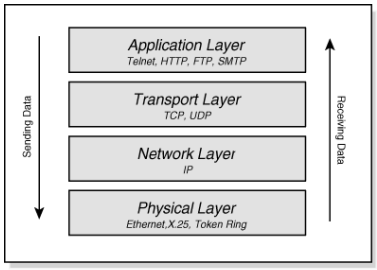
The problem which I am creating a solution to must satisfy the end user of the system. For example my client wanted the solution to be useable on different computers, where the data accessed and saved on one computer is available on another. To do this, I could make use of networking, saving the data in a small server. I could also achieve this by making use of real-time databases.

Google's Firebase is an example of a service that offers real time databases. It is made for web based solutions but has third party wrappers for desktop programming languages such as C#. Real time databases allows each instance of the solution to be able to create, read, update and delete records in real time, so changes made by one instance of the solution will be seen by another. Initially Firebase can be used at no additional cost, however, in order to expand storage capacity and speeds you must pay a fee in order to upgrade to a higher package. This would be necessary if the solution

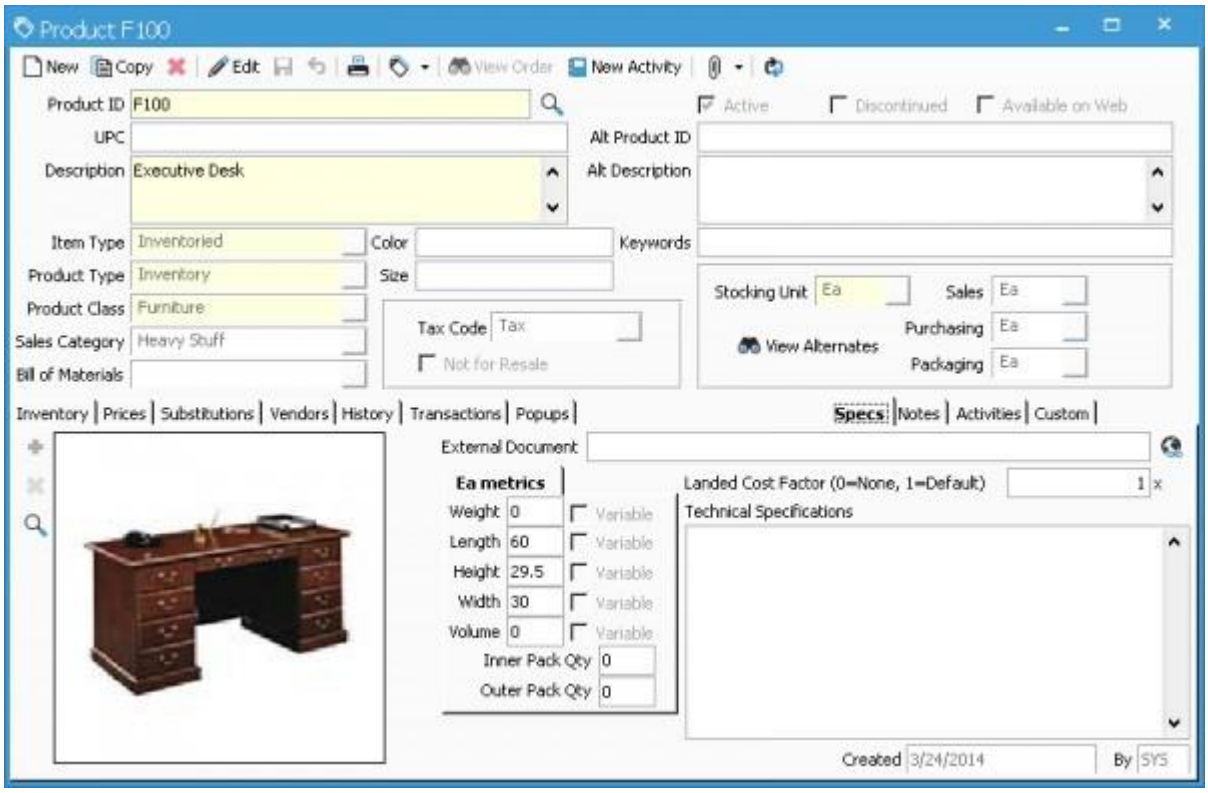
is going to experience high traffic loads. This is unlikely since the solution is only going to be used by the client and staff members, which would not cause any delays due to traffic. If the store was to expand, offering more and more products, the Firebase package may have to be upgraded in order to expand the capacity of the database.

In order to access all the data of the system, the user must be using a computer connected to the internet, this creates a software and hardware requirement – access to the internet. This also increases the cost of the overall solution as fees would needed to be paid in order to keep the server running without stopping. To implement this feature I can make use of a computational technique, layering. "Layering is the organization of programming into separate functional components that interact in some sequential and hierarchical way, with each layer usually having an interface only to the layer above it and the layer below it". In order to achieve layering, computational thinking methods such as decomposing the problem and abstraction must be used. Hardware such as a router are needed in order to access the internet. Furthermore, a subscription with an Internet Service Provider is needed, this can be very costly. If the internet has slow upload and download speeds the user could experience lag whilst using the solution.

(Fletcher, 2016)



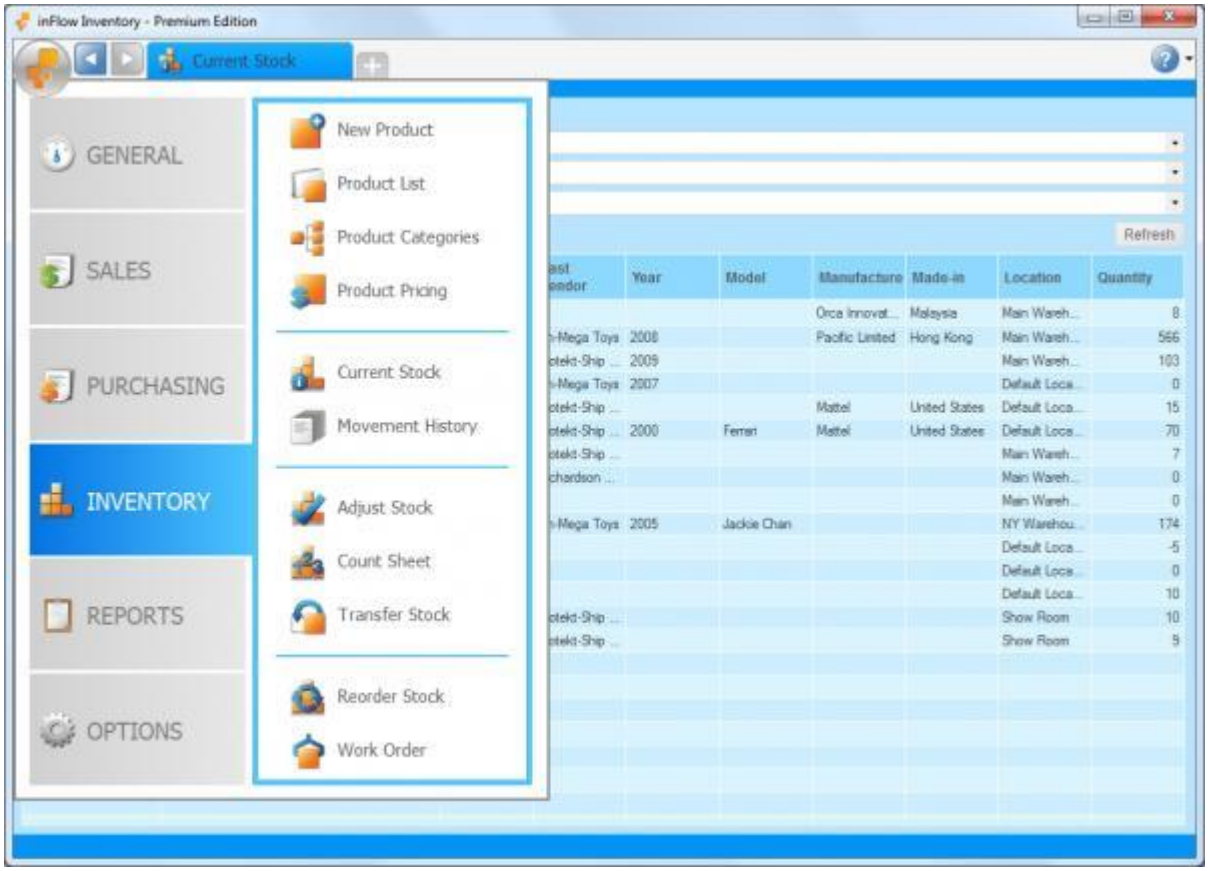
## 3.2 Existing solutions




(Anonymous, 2016)

Acctivate inventory management software is a potential solution to the proposed problem. It is an off the shelf product and does not offer bespoke features, tailored to the client. Based on the features Acctivate has to offer, I could include some of fields shown above when storing the information of a product. For example, I could also allow the user to assign a product to a category. The advantage of doing this, is faster searching time, as indexing can be used. Rather than the solution searching the entire list of products it can search for products in the chosen category, therefore reducing the searching time. Also, I can see that Acctivate gives each product an ID, this could be a feature of the final solution, since all the products will be stored in a database I can make use of IDs to give each product a unique identifier, making duplicates unique from each other.

On the other hand, the product shown above has a cluttered user interface, which can come across as overwhelming for inexperienced users. For example it features many tabs near the bottom of the page, this can come across as overwhelming. It also allows the user to classify the product into different categories such as item type, product type and product class. This allows data to be stored in a more organised fashion allowing for quicker searching times but can be very time consuming for the user.

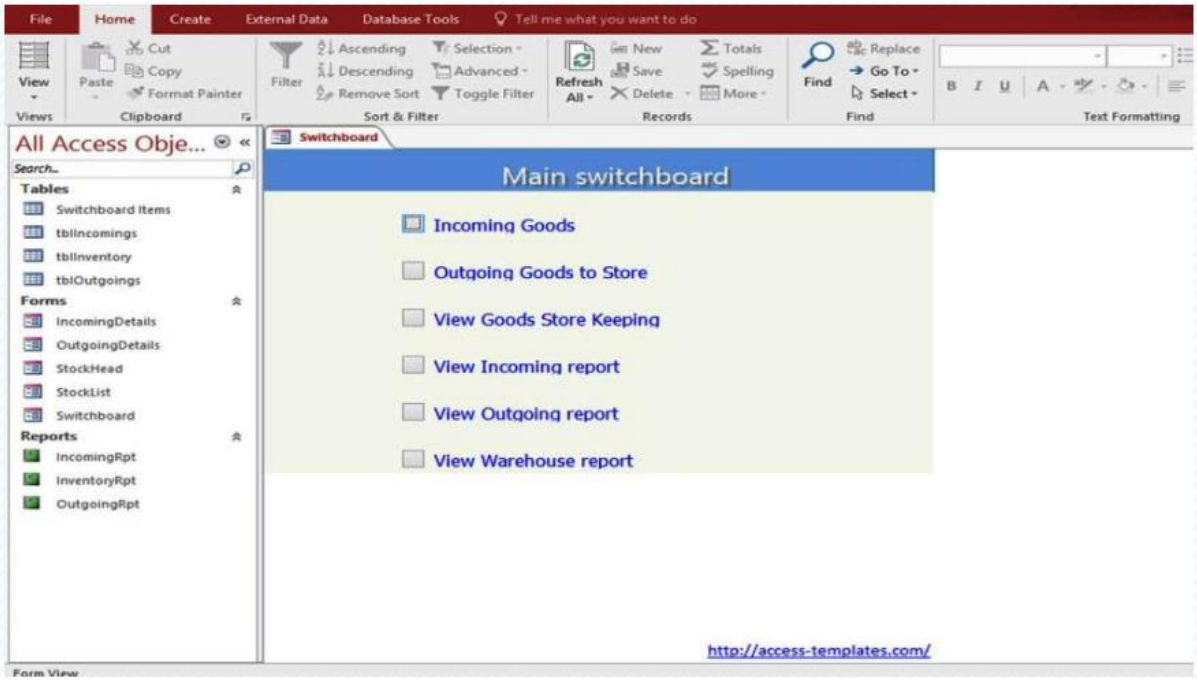



(Anonymous, 2016)

InFlow inventory is another potential solution to the problem. This program has a more de-cluttered interface, making it more suitable for inexperienced users. Features such as tabs, tables and menus are all features that I could include in my solution. The use of large menus on the left hand side could be beneficial to inexperienced as the icons and text are large and easy to view.

Rather than presenting all of the data entry fields on one interface like Acctivate inventory does, InFlow makes use of large icons and text, shown on the left hand side of the image. When the user clicks on one of these icons, more options appear. This is known as a menu driven interface, which is capable of displaying many options to the user whilst keeping the main user interface tidy, this can lead to the user feeling less amounts of stress due to being overwhelmed.

In the top right hand side corner, the interface contains a question mark. If the user clicked this button when using Inflow Inventory, they would be taken to a help dialogue, where they can find information on how to solve a problem they might have. This could be a very useful feature to include in the solution, since the users of my solution are inexperienced, they are likely to need additional help to use the solution, and this could be one way in which the solution could provide this.




(Templates, 2017)

Another solution to the problem would be to use Microsoft Access database software. Despite the solution being much easier to develop, it would be a very basic inventory management solution. It would not be able to incorporate bespoke features such as Artificial Intelligence and machine learning.

Shown in the image above is an example of an inventory management solution created in Microsoft Access. The main menu form is shown in the image it is clear to see that the user interface is very basic, which can be beneficial is inexperienced users. However, this means that there is not much room for customization, which means that developing a bespoke product in excel can be very difficult. For example, some user interface features that the client may want, may be unavailable in Microsoft Access.

Furthermore by making user Microsoft Access, the user is required to have the software installed on their machine, which can further increase costs. In addition, it also means that a mobile based solution cannot be created, if Access is only being used.

## 3.3 Advantages and disadvantages of research

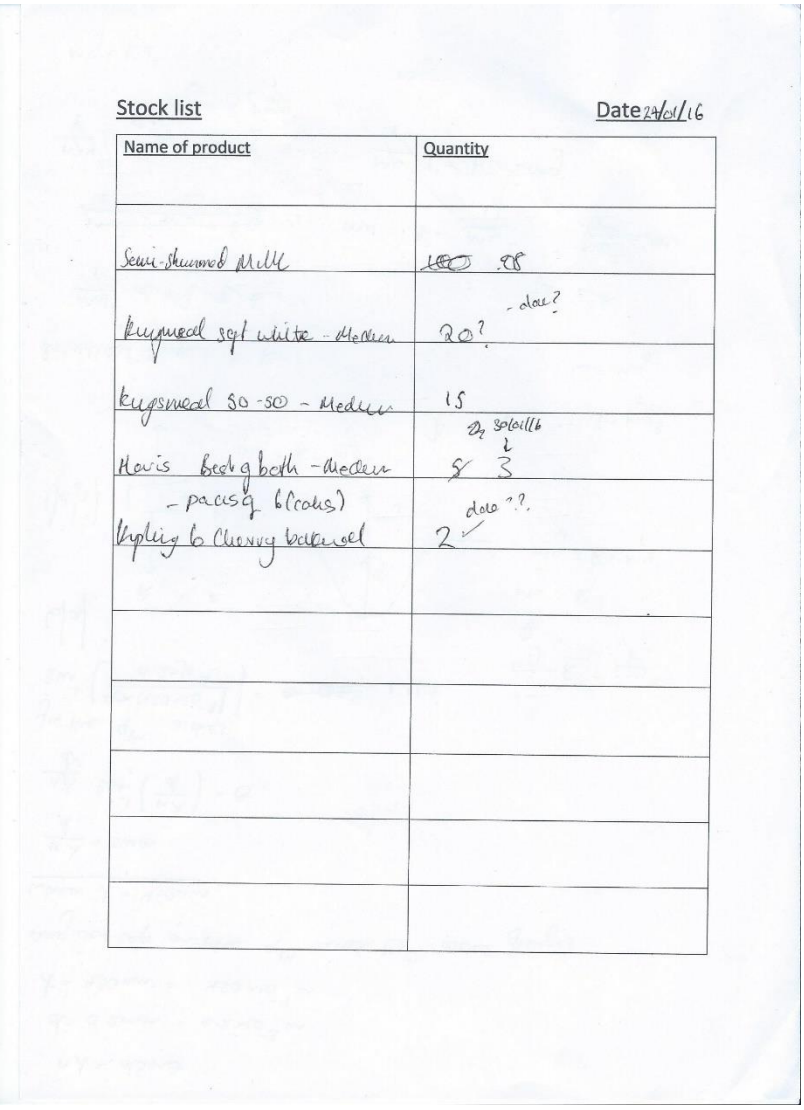
The research I have made on existing solutions has allowed me to gain a better understanding of what is required of an inventory management system such as it being able to perform CRUD (Create Read, Update and Delete) operations on the database. It has also showed me how to create an effective User interface, by showing me poor and good designs. I now have a better chance of creating an interface more suitable for my client.

However, my research has not showed me how an inventory works exactly for a small business such as a convenience store. They have all been based on large businesses with a large inventory. Although the principle will be the same, there will be subtle differences between the system I create and the systems I have researched.

## 3.4 Forms of data

### *3.4.1 Paper based*

Currently, there are two paper documents in use. One of the documents, called the stock list, stores a list of the products currently stored in the shop, that isn't available for sale. It also stores the quantity of that product in the stock room. Furthermore, the data the list was created is also stored at the top of documents.



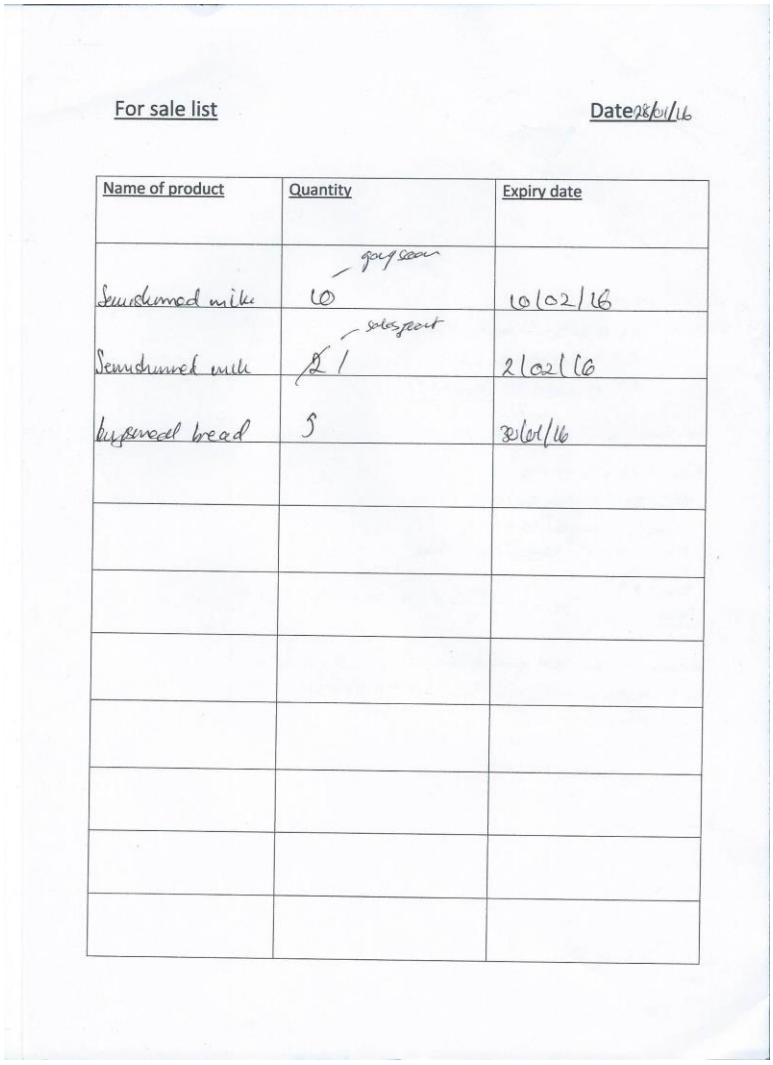
Stock list Date 24/01/16

| Name of product | Quantity |
|---|---|
| | |
| Semi-skimmed Milk | ~~100~~ 98 |
| Kingsmead soft white - Medium | 20? - date? |
| Kingsmead 50-50 - Medium | 15 |
| Mavis Beckg bath - Medium - packs of 6 (packs) | 2 solo 11/6 ~~5~~ 3 |
| Kipling 6 Chunky bakewell | 2 date ?? |
| | |
| | |
| | |
| | |
| | |
| | |

*Figure 1*

Shown above (Figure 1) is a scan of a stock list sheet used earlier on this year.

In addition to the stock list paper document, the shop also makes use of another paper document, called the for sale list. This document stores the list of products being taken out of storage and on to 'the shelf', making it available for purchase. The list stores information such as the name of the product, the quantity of that product and the expiry date of that product, should it have one. It also stores the date the list was created.



For sale list

Date 28/01/16

| Name of product | Quantity | Expiry date |
|---|---|---|
| | | |
| Semi skimmed milk | 10 (pay scan) | 10/02/16 |
| Semi skimmed milk | ~~2~~ 1 (sales point) | 2/02/16 |
| Unsliced bread | 5 | 30/01/16 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

*Figure 2*

Shown above (figure 2) is a scan of an empty for sale list sheet.

# 4 Data collection

At this stage, I am going to get as much information from the client as I can regarding what the client wants from the solution. This will ensure that I produce a solution that does exactly what the client wants it to do. This will also allow for strong communication between myself and the client, meaning less time will be wasted in the future and the chances for mistakes to be made are reduced.

## 4.1 Questionnaire

The following questionnaire was given to my client and staff members to complete.

**Inventory management questionnaire**

*The purpose of this questionnaire is to help me understand what your expectation of the solution is.*
*Please answer questions marked with a '*' in full sentences.*

**1)** What would you rate the current system in place out of 10?
☐

**2)** Do you need to record the amount of products out on shelves?
Yes ☐ No ☐ Unsure ☐

**3)** Which statistical representation of stock do you prefer most? *(Please order 1(best) -3)*
Graphs ☐ Pie charts ☐ Bar charts ☐

**4)** Are computers easily accessible in your workplace?
Yes ☐ No ☐ Unsure ☐

**5)** Do you own an Android or Apple based smartphone?
Apple ☐ Android ☐ Other ☐

**5.1)*** if you answered other, please describe what type of phone you have.

**6)*** Please describe how stock is logged when made available for sale.

**7)** Do you agree that an electronic based system is needed?
Strongly agree ☐ Agree ☐ Disagree ☐ Strongly Disagree ☐

*Thank you for taking the time to complete this questionnaire. All questionnaires will be Collected by Arun Patel*

My client's answers were as follows:

**1)** What would you rate the current system out of 10?
Answer: 4
**2)** Do you need to record the amount of products out on shelves?
Answer: Yes
**3)** Which statistical representation of stock do you prefer most?
Answer: Graphs (1), Bar charts (2) and Pie charts (3)
**4)** Are computers easily accessible in your workplace?
Answer: No
**5)** Do you own an Android or Apple based smartphone?
Answer: Android
**6)*** Please describe how stock is logged when made available for sale.
Answer: When stock is put out on the shelf, it is initially taken off of the stock list. It is then Added to a for sale list. If the product is an item that has an expiry date , that date is then recorded on to the list as well , so we know when stop selling a certain product. We also record the amount of that certain product taken out, for sales analysis purposes.
**7)** Do you agree that an electronic based system is needed?
Answer: Agree.

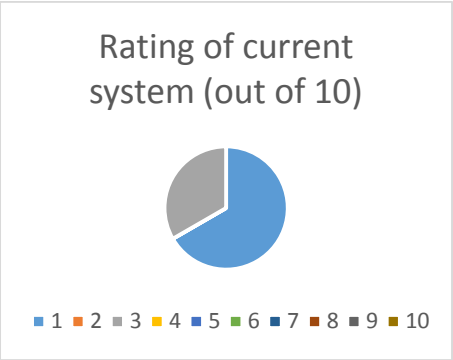
From the answers my client has given me for this questionnaire I can deduce that my solution should present statistics in the form of graphs and bar charts. I also understand that my client wants the new solution to be created.

I have also identified that my client does not have easy access to computers, this means I will have to find a way of allowing my client to be in use with the system whilst not a computer. One possible solution to this is to make use of a smartphone. My client has an android based smartphone, this means I could build an app to go alongside the main computer based system. This will make it easier to record the number of products out on the shelves.

I have also gained an understanding for how the stock is logged when made available for sale, this is an essential part of the system that would have to be accurately modelled in the final solution.

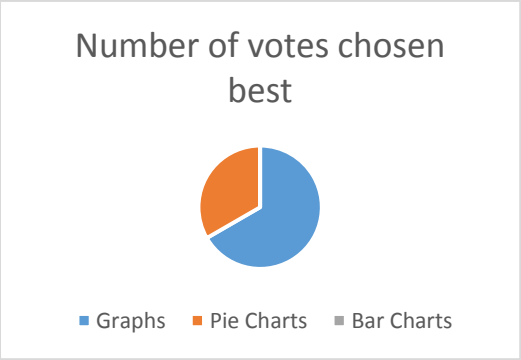
The questionnaire was also given to 3 other staff members at the business. The results of those questionnaires have been summarised below (excluding question 6). I processed my results using the charts feature in Microsoft Office Word (Microsoft, n.d.)

Question 1 -

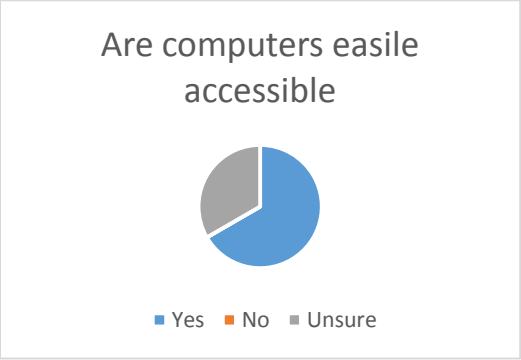



Question 2 – All staff members ticked the yes box

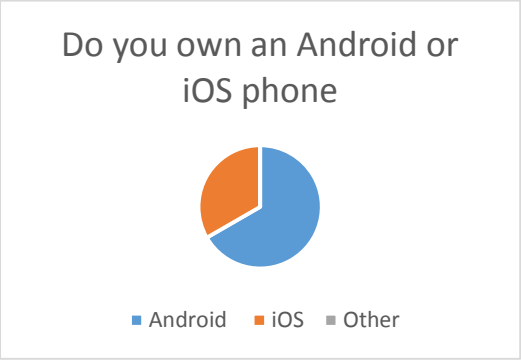
Question 3 –



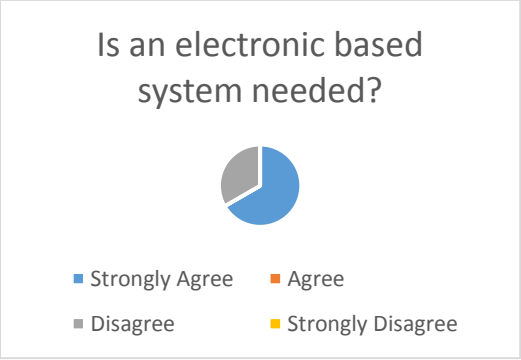

Question 4 –




Question 5 –




Question 7 –



## 4.2 Interview

### *4.2.1 Face to face interview with client summary*

I had arranged to have a face to face interview with my client at the workplace. I arranged for this interview by emailing my client and asking them if it was ok for me to conduct the interview and at what time and day would suit them best. The aim of the interview was to gain as much information as possible, mainly about the client themselves, such as their computer preferences, workload and computer skills. I also wanted to see, in person, exactly how some of the processes such as receiving new stock are actually conducted.

After agreeing on a time and date for the interview, I made my way to the business and conducted the interview. Initially, my client showed me the current system and how it works, he showed me the process of how new products are delivered to the store, added to the logs and stored in the stock room. After, he showed me the process of taking out stock from the stock room and making it available for sale.

The main issue with these processed seemed to be the amount of time it took to complete the tasks. The client seemed to get frustrated when trying to find items on the stock list as there was at least 100 items on the list.

After showing me a demonstration of the current processes the client undertakes, he then went on to give some ideas that could be added to the system. He suggested that the main system be able to search for products stored in the system. He also suggested that the new system maintain two separate lists, for products in the stock room and products for sale.

The client also raised an issue about the new solution, the fact that it wouldn't be practical to use whilst performing everyday tasks. He then went on to explain that with the current system, he is able to walk around with the pen and list and make changes anywhere, whilst with a computerised solution, he would have to be at a computer which he cannot easily carry out with him.

### *4.2.2 Face to face interview with client transcript*

The following is a transcript of the interview I had with my client

*Me:* The first question I would like to ask, is what happens to new items when they have been delivered.

*Purnesh:* When we receive a delivery, the first thing that happens is that the person who delivered the items and myself count the quantity of items that have arrived, to ensure that we have received everything we paid for. After, I then sign a form to confirm receipt of the items. I then pass over receipts to other staff members who will then put away and store the items. This is done by writing the name of the item on the stock list, as well as its quantity.

*Me: Ok, so all items in the shop are recorded the stock list sheet?*

*Purnesh:* All items in the stock are logged on the stock list sheet but items out on shelves are logged on another sheet, the for sale list sheet. Items are transferred to this sheet when moving stock from storage to the shelves.

*Me:* Could you please describe the process of transferring stock from storage to the shelves

*Purnesh:* Yes, when an item on the shelf needs to be replenished or an item needs to be made available to customers, a staff member will firstly need to find the item in the stock list remove it

from the list or amend its quantity , based on the amount of that item to be taken out. The item and quantity of that item is then added to the for sale list.  If the item has an expiry data, it is added to the list. The actual item is then moved to the relevant location in the shop.

*Me:* Have there been any issues with this process?

*Purnesh:* There have been many issues such as staff members making mistakes, changing the details of the wrong item,  not being able to tell which staff member has made changes and not being able to find certain items since the stock or for sale paper containing that item had been lost. The main issue with doing this, is that it is very time consuming and inefficient. It can sometimes take hours to work through hundreds of items at a time in order to find a particular item.

*Me:*  Ah ok, the new system will mainly be based around the use of a computer, what type of computer do you have?

*Purnesh:* I have a relatively modern desktop computer over near the counter area. It's not very fast but good enough to get jobs done.

(My client then went to show me his desktop computer)

*Me:* Do you have much computer experience, how confident would you say you in getting to grips with new software?

*Purnesh:* I've been using computers for many years however, I would not consider myself 'good' with computers. I am able to use software such as the office word and excel but I've never really been exposed to new software. I think I would need a lot of help to get started with it.

*Me:* The new system can be made in a way that is user friendly to new users. Do you have any concerns about moving from a paper based system to a computerised system?

*Purnesh:* Yes, with a computerised system, I feel as if I would be stuck at the computer when using it and I would have to move around the shop and back to the computer multiple times. For example, at the end of the day the actual stock on the shelves are compared with the stock on the 'for sale list' and changes are made. If I had to do this with a computer, I would have to check the stock on the shelves and move back to the computer over and over again.

*Me:* Ok, that could potentially be resolved by making use of a smartphone. Do you have any suggestions as to what you would like to see in the new system?

*Purnesh:* I think the new system should be able to easily search for items, it shouldn't take up to an hour to find an item in the list, like it currently does. I think maintaining two separate lists for items in the stock room and products for sale would be suitable.

*Me*: Ok, thank you for your ideas and your time today.

*End of interview.*

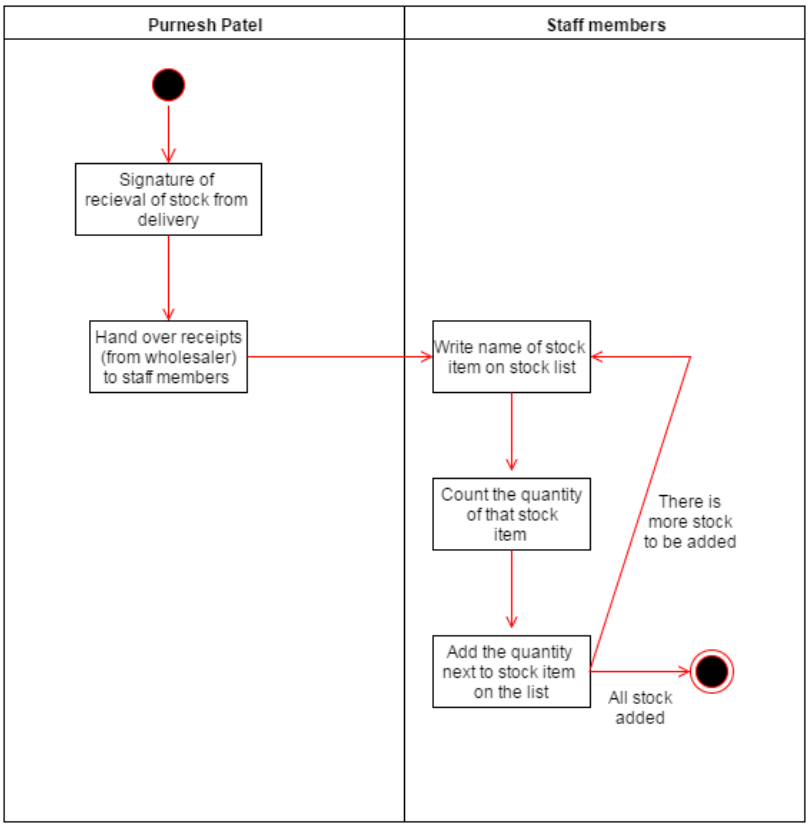
# 5 Diagrams

## 5.1 The current system

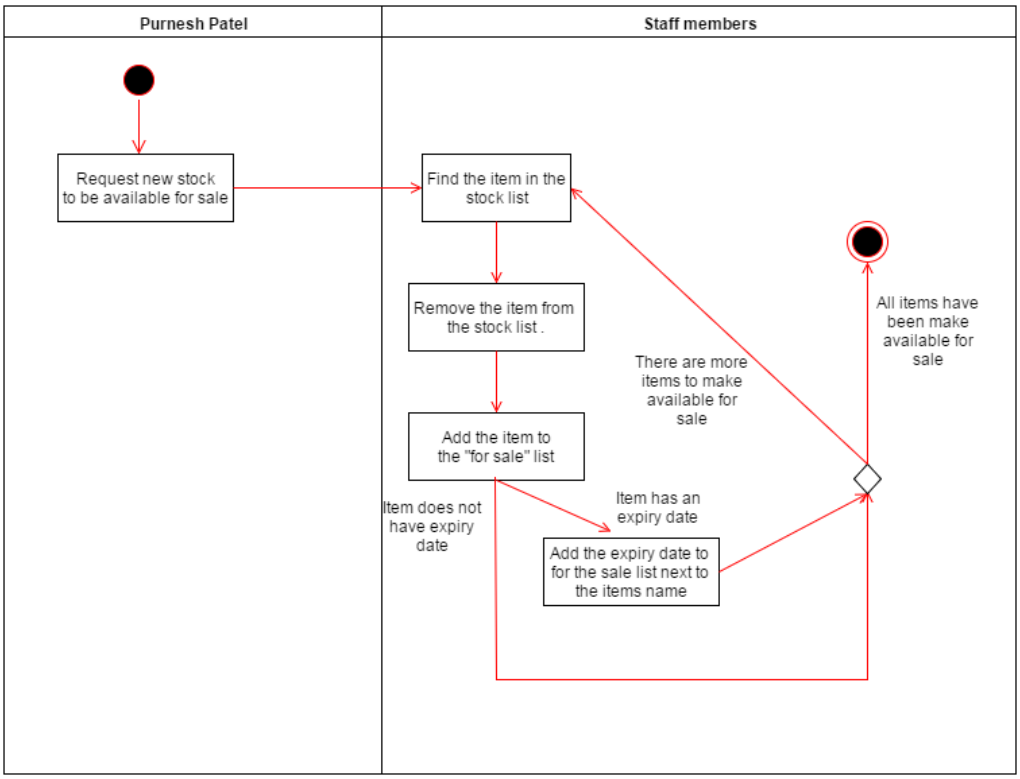
Here is a summary of the processes undertaken in the current system in the form of an Activity diagram.

*5.1.1 Receiving new stock*

The signature of revival is done by the person receiving the stock, this would be Manager, Purnesh Patel, in almost all cases.



*5.1.2 Making stock available for sale*




## 5.2 Capabilities and Limitations of the current system

*Capabilities*

The current system is able to store a list of items currently held in inventory as well a list of items currently for sale.

It is able to remove items from both lists, change details about those items as well as allow staff members to read the data held about the items.

*Limitations*

Transferring items from the inventory list to the for sale list can take a long time and can lead to errors being made.

Changing the details of each item can only be done a finite times as it would be amended on a paper based system.

It can take staff members a lot of time to find and read the data about items. The system is not able to track who makes changes to the data being held in the system, for example, if a staff member adds a new item to the inventory list , there is no way of determining which staff member added that item.

It is not able to analyse trends about the data being held, it would have to be done manually by a user. It not able to predict which items will sell better and which won't.

The data about the items is only accessible in one place. If a user is another location to the system, they will not be able to access the data, which will prevent them from finishing their work.

# 6 The proposed solution

## 6.1 Solution requirements

### *6.1.1 Hardware requirements*

- A computer with an internet connection
  - This will allow the user to make use of the solution I have created.
- A smartphone
  - Since there is not easy access to computers, a smartphone can be used instead when not at a computer to complete simpler tasks.
- A Keyboard
  - This is a basic input device used to input characters into the computer. It is essential for both the users of the system and myself, to use the computer.
- A Mouse
  - This is another basic input device used to allow the user to move a mouse pointer around the screen. This is necessary for the basic operation of the computer.

#### Required computer specification

The minimum hardware requirements that a computer needs to run this software is the minimum hardware requirements to be able to run Microsoft's Windows 10 64-bit (Microsoft, 2015), however, for an optical experience, the user should have a computer whose specification are in line with the recommended hardware specification.

#### Minimum hardware specification

Processor – any processor with a clock speed of 1GHz or more
RAM – at least 2 GB
Hard disk space – at least 21GB
Graphics – Direct X 9 or later graphics card.
Display – a minimum of 800x600

#### Recommended hardware specification

Processor – any processor with a clock speed of 2GHz or more
RAM – at least 2 GB
Hard disk space – 30GB or more
Graphics – Direct X 9 or later graphics card.
Display – 1920x1080 or higher.

### *6.1.2 Software requirements*

#### *Developer Software requirements*

- A desktop operating system – Windows 10/8/7 OS
  - The solution will be created using the C# programming language and will therefor only run on a Microsoft Windows Operating System. This means that the users of the solution must have access to either a Windows 7 , 8 or 10
- An Integrated Development Environment (IDE) - Visual Studio
  - In order to create the solution efficiently and effectively, I will need a programming IDE to help me write code.

- Photo editing package – Adobe Photoshop
  - The solution may require graphics to be created, or to edit existing graphics to suit the solution.
- Microsoft Office suite
  - The office suite is used to create documents, slide shows and spreadsheets, which all necessary for the development of this project.

*User Software requirements*

- A desktop operating system – Windows 10/8/7 OS
  - The solution will be created using the C# programming language and will therefor only run on a Microsoft Windows Operating System. This means that the users of the solution must have access to either a Windows 7 , 8 or 10
- .NET Framework
  - The .NET framework is required to be installed on the user's computer in order to execute .NET applications, including the solution being built.

*6.1.3 Solution requirements / success criteria*

Functional requirements

- The solution must be able to input new stock items.
  - This is essential for entering new items into the system when a delivery arrives.
  - Entering the new details about an item into the system should cause the system to retain this information in the database connected to it.
- The solution must be able to take out stock items
  - This allows the user to make stock available for purchase.
  - Choosing an item , its quantity and making the request to make it available for sale should cause the system to move the item from the inventory list to the for sale list.
- The solution must be accessible from different computers
  - This allows the user to access the database from different computers. This is needed as the client will be using different devices to make use of the solution.
  - If a user uses the solution on a computer at the shop and then at a computer in their home, the solution should still be fully functional.
- The solution should be useable by inexperienced users
  - The client is not very experienced with computers therefore the solution must be user friendly enough for inexperienced users.
  - The user as well as all staff members should be able to use the solution without any additional help from anyone else and by only using help provided in the solution.
- It must be able to find popular products
  - In order to continue selling products that are generating the most profit, the system must be able to identify these products and tell the user.
  - If a product is moved from inventory to for sale and the amount of that item on the shelf everyday decrements frequently then the system must detect this and notify the user.
- It must be able to find unpopular products
  - Products that are not selling well should be made to the attention of the user so that they can stop buying more of it.
  - If a product is rarely moved from inventory to for sale and the amount of that item on the shelf everyday doesn't change frequently then the system must detect this and notify the user.
- It must be able to predict when products will sell well

    - Products will sell be better at certain times of the year, the system should learn this trend for certain products and notify the user.
    - If a product such as ice cream sells more during the summer and worse during the winter, the system can detect this and notify the user.
- Data stored in the solution must be valid
    - This will ensure that the data stored in the system is free of errors , reducing the chance of an output being incorrect as a result of incorrect processing
    - Entering data into a validated field should present the user with an error.
- The solution must keep a log of all actions that have taken place by users
    - For example if a user adds a new item to inventory, this action should be added to a log.
    - This will ensure there will be no confusion as to which staff member altered an item or added an item for example. This issue was highlighted to me by my client during an interview with them.

Non-functional requirements

- The solution must be able to load in under 20 seconds
    - This will ensure the user is not left waiting for the program to load when they initially open the program. This is dependent on the size of the database (the number of stock items stored), speed of the computer and speed of the internet connection.
    - The system should load and be fully useable within 20 seconds.
- The solution must have a security login system
    - This will ensure that unauthorised access does not occur. Since the solution will store data about the business, the solution must be secure enough to prevent anyone from accessing this data. Providing a login system will also allow the system to track who has been using the solution.
    - When the system starts up, it should prompt the user for a login ID and password. It should only give users access to the system if the credentials they entered matches the details in the database.
- The solution should include help tools
    - During the interview with my client, it was made clear he had not much experience with new software. He mentioned that he had experience with Microsoft office Word and Excel packages. Therefore, the final solution should be as complex or less as these software packages.

### *6.1.4 Success criteria*

*Inputting new stock items*

- When a new is added to inventory, the new item should be added to the systems database. This will cause the new item to be retained in the system. When any user uses the system at a later date, the item should be shown to the user.
- If the item is shown to any user after the item has been added then it would have been successfully added and retained by the solution.

*Taking out stock items*

- Items should be able to move from inventory to the shops shelves. In the solution, this involves moving it from one table to another. When user transfers an item from inventory to the shops shelves, the item should move from one table to another.
- If this happens, it means that the solution understands that the particular item is not available for sale and isn't being held in storage anymore.

*Accessing the solution from different computers*

- Data held about items should be accessible on one machine and another, in different locations and on another network.
- If the solution is able to do this, then any other data stored by the solution will be accessible on that other computer and so any other computer.

*Finding popular products*

- When an empty database used by the solution is filled with the random items with random sales data, the solution should be able to find the five most popular items in the data being held.
- This would show that the solution can find the most popular items for any items ever hold in the solution.

*Finding unpopular products*

- When an empty database used by the solution is filled with the random items with random sales data, the solution should be able to find the five most unpopular items in the data being held.
- This would show that the solution can find the most unpopular items for any items ever hold in the solution.

*Predicting when items will sell well*

- When a database used by the solution is holding the sales data about of a single item, the solution should be able to predict when the item will sell best, giving a particular date range.
- If the solution can find the bestselling time for this particular item then it should be able to do this for any item.

*Logging actions*

- When the user completes an action such as creating a new account, an action event should be added to a log stored by the solution. It would include the time of the action and the username of user completing the action.
- This would show that the user could log the events of actions taken by the user, with useful information stored with the event.

*User requirements*

Due to the nature of the solution and the hardware available an existing manual process of physically counting the number of items out on the shelves at the end of the day must remain. A solution to this would be to use a handheld stock taking bar code terminal such as the "ERS-STOCKCOUNT-KIT1 ERS STOCKCOUNT Barcode Stock Taking Solution" (ERS, 2016) . The cost of such equipment is approximately £700 per unit, which is not always viable for a small business. However this process could be simplified with the addition of a mobile based solution.

## 6.2 Summary of the solution to be created

The solution I am going to create is an inventory management system for my clients shop. It will be capable of adding new items that has been delivered to the shop into the system, changing any details about that item and removing that item from the system. It will be capable of differentiating between items that are currently for sale and items that are in the stock room (not currently for sale).

Due to hardware limitations (see above), there will also be a smartphone based solution for the android operation system. This will be provide the basic feature of being able to log the quantity of products that are actually "on the shelf" at the end of the day. This will make it much easier to do

this since the user can simply walk around the shop with their smartphone, logging the quantity of each product.

This leads to the main Unique Selling Point of the solution. The solution will make use of Artificial intelligence, Machine learning and statistical analysis to make predictions and analyse the sales performance of products. For example, Artificial intelligence can be used to make decisions on which products should be stocked more or less. If a new product does not sell well, a decision can be made by the solution to not order any more of that product. Similarly an example of were machine learning can be used in the solution is to predict the time of year a certain product will sell more. For example, the solution can learn that ice creams sell well during the summer and not as well in the winter. This means that the solution will understand that some products are seasonal. Statistical analysis will be used on the data collected.

### *6.2.1 Changes to the solution*

Due to the programming languages and platform to be used in the final solution, I will no longer be able to use Google's Firebase. This means the solution will no longer have a real time aspect to it, however, this does not mean changes will not be seen across instances of the solution open at the same time but there may be a small delay. Google's Firebase does not have support for SQL, which is needed in order to create an effective solution.

Instead of using Google's Firebase, I will be using Microsoft Azure SQL Database (Azure, 2016). Microsoft Azure SQL Database offers more support for C# developers and integrates well with the IDE I will be using, Visual Studio. In addition, it also has support for Java developers, the language I will be using to create the smartphone solution. Microsoft Azure SQL Database is also a pay as expand product, which means the business will only need to pay more for their services as the business grows. This will be due to the amount of inventory being held by the business expanding and so the amount of data being held in the database expanding meaning it requires more resources to hold the data leading to having to pay a greater fee to use the service. Despite this, initially there will be no cost to use their service.

# 7 Signatures

Confirmation of hardware requirements: _____ Date of signature-

Confirmation of functional requirements: _____ Date of signature-

Confirmation of non-functional requirements: _____ Date of signature-

Confirmation of software requirements: _____ Date of signature-

Confirmation user requirements: _____ Date of signature-

# Design

## 1 System objectives

**Due to the time constraints of the project I have decided that I will not create a mobile based solution to alongside the main solution.** This feature could be implemented at a later date.

Although I have already defined solution requirements above, by defining system objectives I am able to expand on the solution requirements in order to deduce what type of interface the system should have based on the solution requirements. Each design created will follow the applicable functional requirements listed above. Some of the objectives outlined here can also be classed as primary objectives; required features that must be added to the system.

### 1.1 User interface

- Background colour will be plain white.
- It must use fields for data entry.
    - To allow the entry of inputs described in the system requirements
    - These fields must have a black border to show contrast against a white background.
- A navigation bar will be present on every page on the left hand side of the interface
- An exit and minimise button must be displayed on the top right hand side of the interface
- The interface should be of suitable dimensions to view on a screen with a resolution of 1920x1080 pixels.
    - This is not required of the mobile based application.

### 1.2 Input

- The user will be able to add new items into the system.
    - These changes will be seen across any device logged into the system
- The user will be able to make certain items available for sale.
    - The user will be able to search for items
    - The user will be able to add an expiry date to an item if needed.
- The user will be able to record who has made changes to the inventory.

### 1.3 Processing

- A database will be created on initial use.
- Items are added to this database in real-time
    - The database will be able to store the name of an item, the quantity, its expiry date (if applicable) and whether or not it's for sale.
    - The user will be able to order the table (e.g. in alphabetical order)
- A query is ran on the database for the item being searched, the user can then make changes on the returned record.
    - Changes include making an item available for sale, adding an expiry date, removing the item and adding more of the same item (CRUD).
- The items moved to for sale the most during the week are recorded summarised.

### 1.4 Output

- A real-time view of the database of items presented on the interface.
    - This will include, item name, quantity, and whether or not it's for sale.
    - This will also be presented in a mixed table – containing all items , regardless of whether or not it's for sale , a for sales table – containing all items that are currently

for sale and an inventory table – containing all items that are currently in storage , not for sale.

- A weekly performance report. Products that are taken out of storage and made for sale are considered successful products (as they are in demand). The most successful products can be summarised on the report.
    - Graphical analysis will also be utilised here.

### 1.5 Internal Objectives

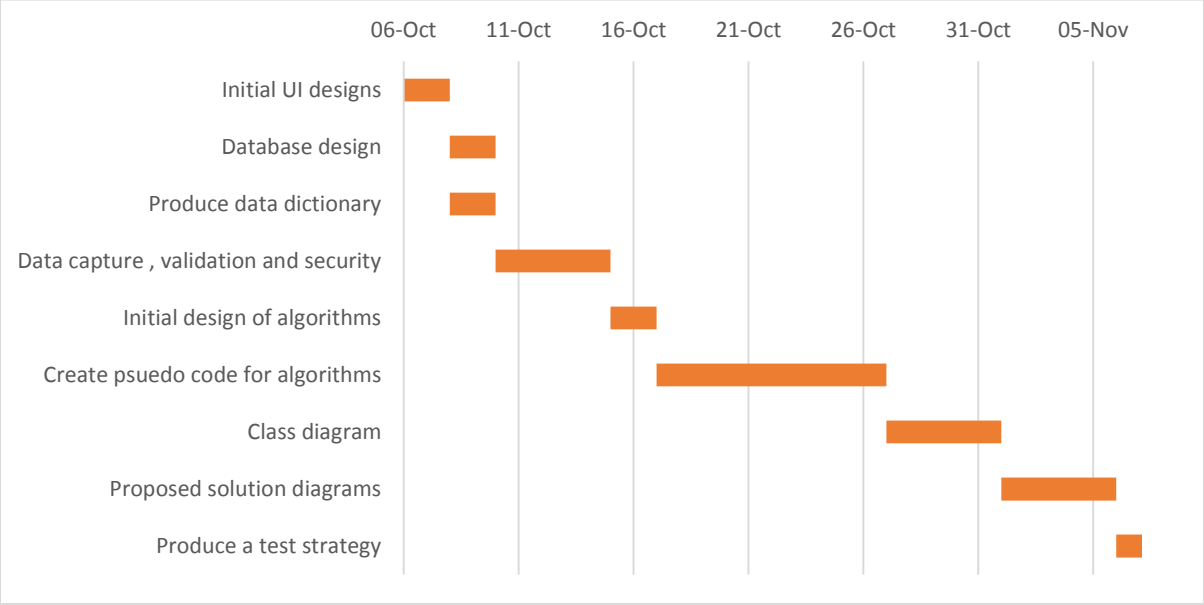
These objectives are objectives that I need to achieve in order to develop the solution.

- Learn SQL
    - Structured Query Language (SQL), is the language that is used to manipulate and access databases. It is therefore essential that I am comfortable with the language in order to create an effective solution
- Learn how to setup and use Microsoft Azure Database.
    - Microsoft Azure Database is the tool I will be using to store data being stored in the system. Without it, the solution would not be able to store any data.

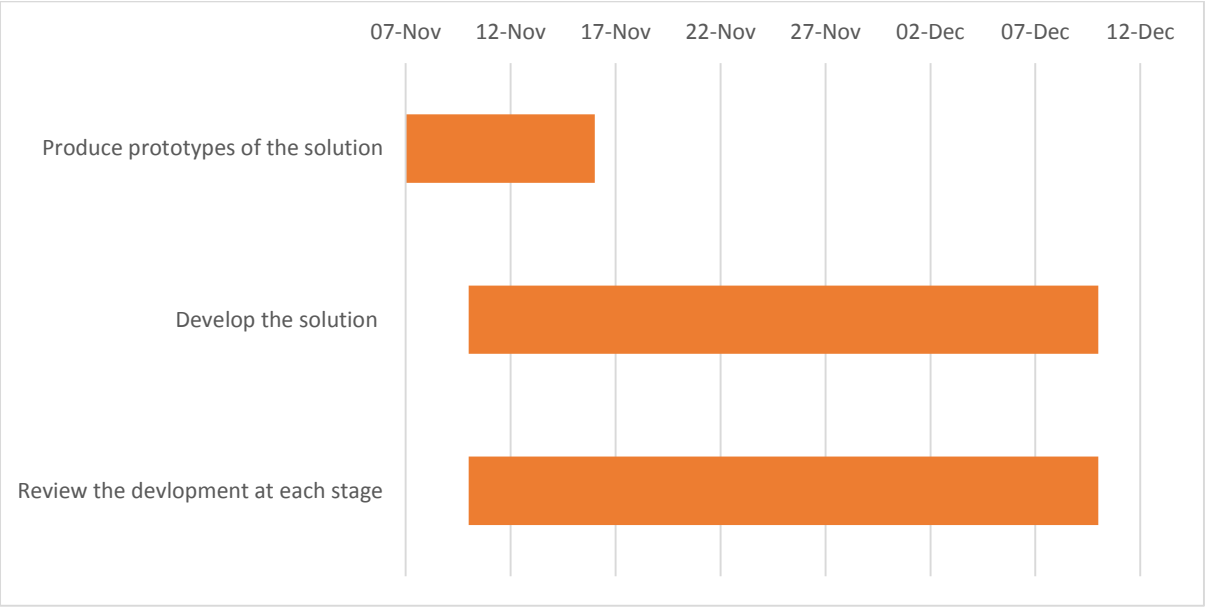
# 2 Scheduling

In order to produce an effective solution, I have made the following Gantt chart to illustrate my plan of how I will develop the solution.

### 2.1 Design plan



## 2.2 Development plan




# 3 New System

## 3.1 Use-Case diagram of new system features

The purpose of this use case diagram is to show how each user can utilise the system.

*3.1.1 Use-Case descriptions*

Add items to stock list

*Summary*: Add new items to the system that are being stored in inventory.

*Precondition*: New items have been delivered.

*Actors*: Manager, Senior Staff, Staff

*Description*:

***Best case***

1- User enters the name and quantity of the item(s) being moved to inventory.
2- The new items are added into the system
3- The new items are shown in the inventory table

***Worst case***

1a- The user enters an invalid item name or quantity
The system takes the user back to the data entry section of the system

1b- The user enters an incorrect item name or quantity
1- The new item is added to inventory
2- The user updates the item in inventory with the correct details

2a- The item fails to be uploaded to the system
1- The system presents an error message to the user
2- The system prompts the user to try again

Remove items to stock list

*Summary*: Remove items from inventory

*Precondition*: There are items in inventory to remove

*Actors*: Manager, Senior Staff, Staff

*Description:*

***Best case***

1- User enters the name and quantity of the item(s) being removed
2- The items are then deleted inventory table
3- The items are no longer shown in the inventory page

***Worst case***

1a- The user enters an invalid item name or quantity
The system takes the user back to the data entry section of the system

1b- The user enters the wrong quantity

*Case 1 – removed quantity is more than intended*

1- The item is removed from inventory
2- The user adds the items back as if they were a new item being added

*Case 2 – removed quantity is less than intended*

1- The item is removed from inventory
2- The user then removes the remaining items they wish to remove

2a- The item fails to be deleted from the system

- 1- The system presents an error message to the user
- 2- The system prompts the user to try again

Update items in stock list

*Summary*: Allows the user to amend the data held about items

*Precondition*: There are items stored in inventory

*Actors*: Manager, Senior Staff

*Description*:

***Best case***

1- User enters the name of the item to be amended.
2- The system searches and returns items related to the searched for item
3- The user selects the item the wish to amend
4- The user makes the changes on the item details
5- The system updates the data on that item

***Worst case***

2a- The system returns items but none are what the user wanted
The system tells the user to make their search more precise or that the item doesn't exist

4a- The user makes invalid changes (such as entering a new invalid quantity)
1- The system identifies which changes were invalid
2- The system tells the user which change was invalid
3- The system reverts the change and prompts the user to try again

5a- The item fails to be updated to the system
1- The system presents an error message to the user
2- The system prompts the user to try again

Tracking the actions of users including stock list changes

*Summary*: Allows the manager to track who has made any actions on the system, such as creating an account or adding a new item.

*Precondition*: Senior staff and staff members have valid accounts with the system.

*Actors*: Manager

*Description*:

***Best case***

1- The user requests to see who has made changes to the inventory tables
2- The system searches for and returns a log of changes
3- The user can then view the log.

***Worst case***

1a- The user does not have permission to view the log
1- The system should present an error message to the user, stating that the user does not have permission to view the log.
2- The system should then not show the log.

2a- The log contains no data
1- The system tells the user that no actions have been made by staff members

Creating and viewing reports

*Summary*: Allows the manager to view and create reports on items, based primarily on the performance of a particular item.

*Precondition*: The item the report is being created for has valid sales date being held about the item

*Actors*: Manager

*Description*:

***Best case***

1- The user enters the name of the item to generate a report about
2- The system searches for and returns a list of items similar or exactly the same as the searched item
3- The user selects the item
4- The system fetches sales data about the item
5- The system generates a performance report
6- The system shows the report to the user

***Worst case***

1a- The user enters the name of an item that doesn't exist
  1- The system returns a list of no items
  2- The user then enters the name of the next item

2a- No sales data for the input item exists
  1- The system returns no items in the list of items
  2- The user then enters the name of the next item

4 - The system fails to fetch any data from the database
  1- The system should present an error message to the user
  2- The user is then prompted to try again

Making an item available for sale

*Summary*: Allows the user to make an item available for sale, indicating that an item is no longer being held in inventory but it is now on the shops shelves.

*Precondition*: The item already exists in inventory

*Description*:

***Best case***

1- The user selects the item to make for sale
2- The quantity of the of item selected is moved to the for sale table

*Case 1 – The item requires a sell by date*

- The user selects and enters the sell by date
- The item is moved.

*Case 1 – The item does require a sell by date*

- The item is moved without a sell by date added

3- The item is no longer shown in the inventory table (if all of it was moved).
4- The item is shown in the for sale table with the sell by date added (if it was required)

***Worst case***

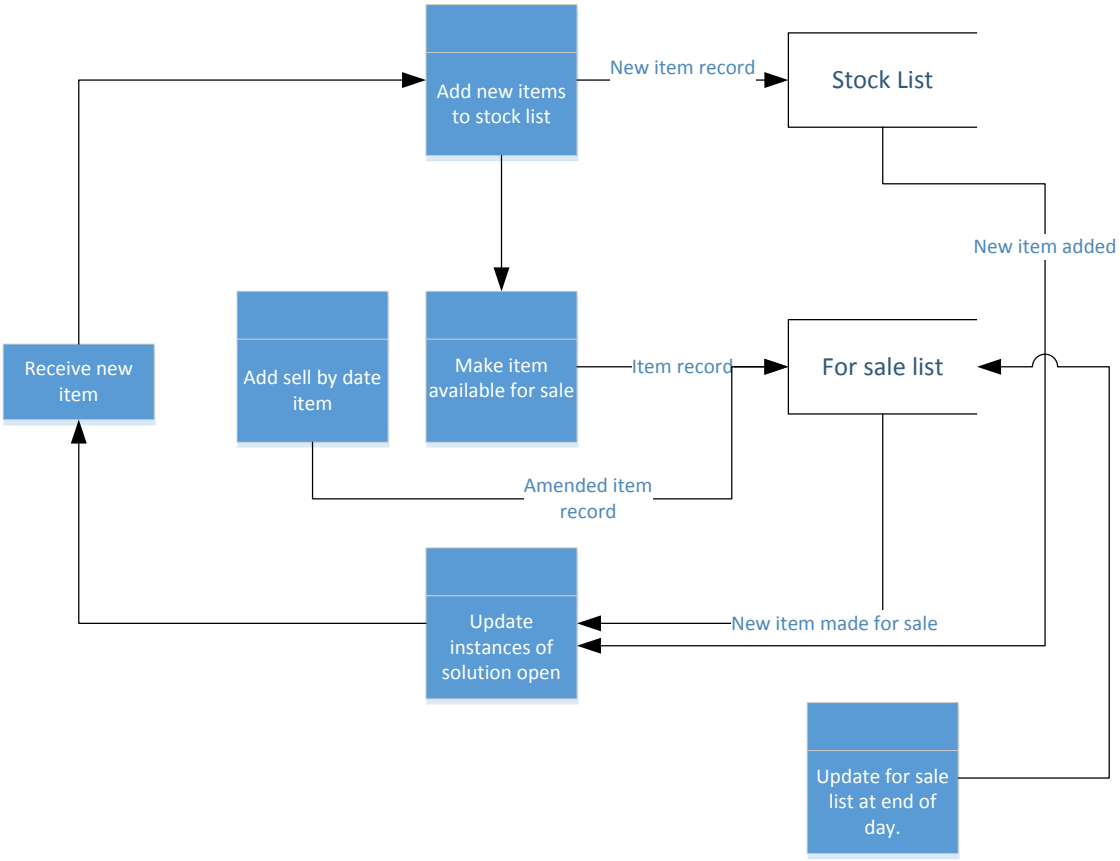
2a- The user enters an invalid quantity amount
  1- The system tells the user the quantity is invalid
  2- The system prompts the user to try again

3a-- The system fails to move the item from inventory to the for sale table
  1- The system should present an error message to the user
  2- The user is then prompted to try again

As shown on the Use Case diagram above, the manager, who is my client - Purnesh Patel, has access to everything the solution is able do. Senior Staff members have access to everything apart from tracking and reports.  Staff members, on the other hand, only have access to the adding and removing items feature , as well as being able access the for sale list , allowing them to make items available for sale, as well as changing or adding sell by dates on items stored in the for sale list.

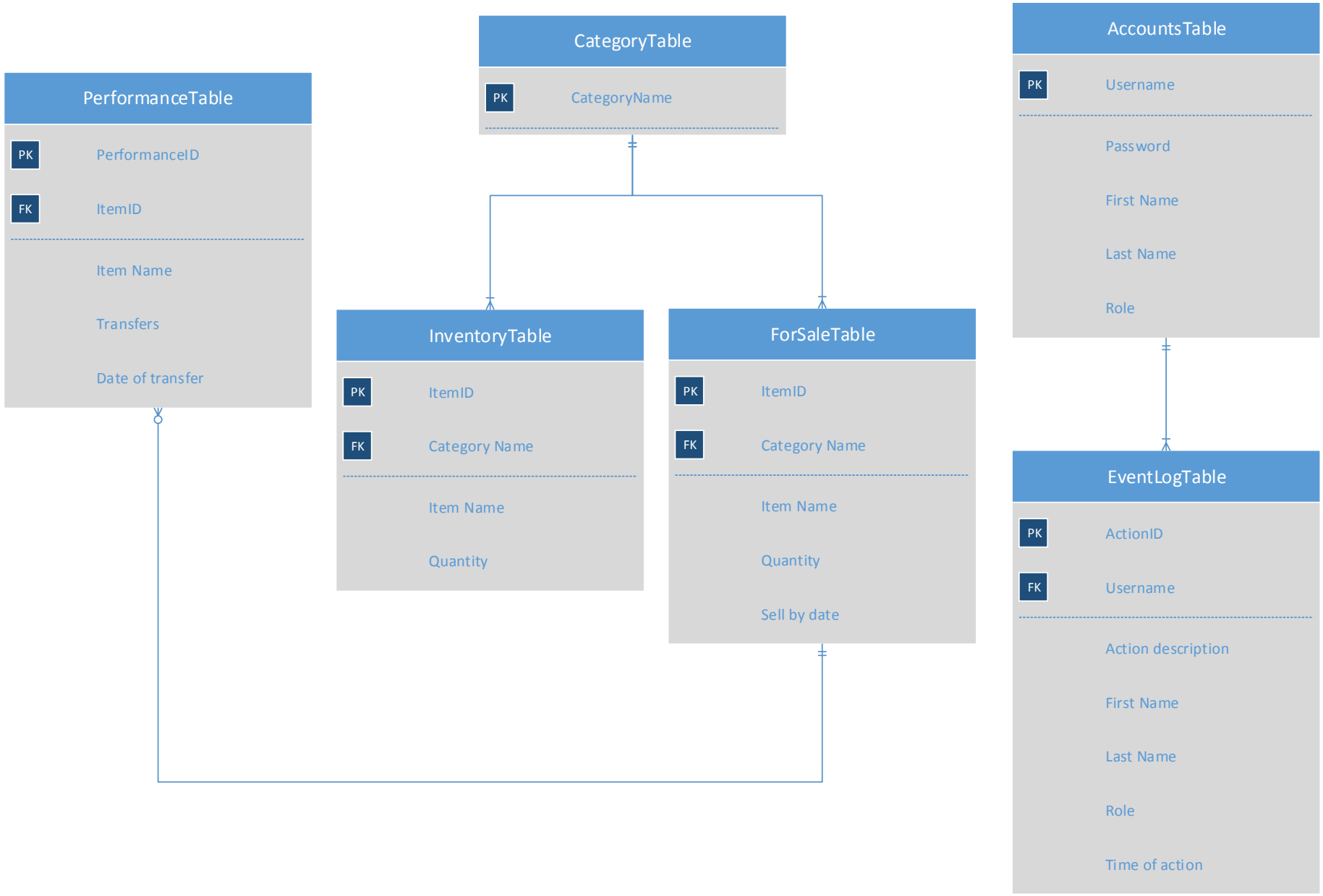
## 3.2 Input Output diagrams

The following Data Flow Diagram (DFD) is an extension of the system objectives outlined in the previous section. It will represent the different processes and data involved in the system objectives.




## 3.3 Inventory Database ERD

All of the data stored in the solution be held in a real time database. It will contain more than one table, making it a relational database. The following Entity Relationship diagram (ERD) summarises the structure of the database to be used with the solution.


CategoryTable
PK
CategoryName
AccountsTable
PK
Username
Password
First Name
Last Name
Role
PerformanceTable
PK
PerformanceID
FK
ItemID
Item Name
Transfers
Date of transfer
InventoryTable
PK
ItemID
FK
Category Name
Item Name
Quantity
ForSaleTable
PK
ItemID
FK
Category Name
Item Name
Quantity
Sell by date
EventLogTable
PK
ActionID
FK
Username
Action description
First Name
Last Name
Role
Time of action

## 3.4 Data Dictionary

A data dictionary defines the structure of the database itself. It records what data is stored, the name, description and characteristics of each data element.

| Field Name | Field Size | Data Type | Data Format | Description | Example |
|---|---|---|---|---|---|
| User ID | 10 | String | | User ID for each staff member , used to log in. | PPatel |
| Password | 10 | String | | Password for authenticating logins. | password |
| Name | 20 | String | | Used to store the name of the person corresponding to the User ID. | Purnesh |
| Job Title | 10 | String | | This is used to identify the privileges of each user. | Manager |
| Item ID | 7 | String | LLLLNNN | Primary key of items stored in the database. | ITEM001 |
| Item Name | 20 | String | | Gives the Item ID an actual product name. | Kingsmill bread |
| Quantity | 3 | Integer | | This value is the number of an item in the table | 5 |
| Price | 8 | Currency | £NNNN.NN | The selling price of a particular item. | £3.20 |
| Category | 10 | String | | The category a particular item falls under. | Food items |
| For Sale ID | 8 | String | LLLLLNNN | The primary key for items that are for sale. | FSALE001 |
| Sell by date | 10 | Date | DD/MM/YYYY | The date in which an item that is for sale will no longer be able to be kept for sale. | 01/01/2017 |
| Performance ID | 5 | String | LLLLNNN | The primary key for items who's performance are being tracked | PMCE001 |
| All time quantity of item made available for sale | 3 | Integer | | This is the total number of times the item has been made for sale since the first use of the solution | 5 |
| Quantity of item made available for | 3 | Integer | | This is the total number of times the item has been made | 2 |

| | | | | | |
|---|---|---|---|---|---|
| sale past week | | | | for sale for the current week. | |
| Items | | Multi-dimensional string array | Dimension 1 : LLLLLLLLLL…L (L characters) Dimension 2: NNNN | A multi-dimensional array used to hold the contents of items retrieved from the database. The index 0 of the first dimension represents the name of the item. The second dimension represents the quantity of that item. | Index 0, 0 could have the value Kings meal white bread. Index 0, 1 could have the value 10. |
| Selection | | String array | LLLLLLLLLLLL | A string array used to hold a list of items that the user has selected from either the inventory list or the for sale list. | The value at index 0 could be Kings meal white bread. The value at index 1 could be Semi skimmed milk. |
| Best-Performed-Items | Bounds: maximum 5. | String array | LLLLLLLLLLLL | A string array used to hold the best performed items for that week (or any other given range). | The value at index 0 could be Kings meal white bread. The value at index 1 could be Semi skimmed milk. |
| Worst-Performed-Items | Bounds: maximum 5. | String array | LLLLLLLLLLLL | A string array used to hold the worst performed items for that week (or any other given range). | The value at index 0 could be Kings meal white bread. The value at index 1 could be Semi skimmed milk. |
| Authenticate | 1 | Boolean | True/False | A Boolean flag used in the logging in process. The flag is false if the inputted credentials have not yet been authenticated. The flag is changed to true once a successful, authenticated, login has been made. | An incorrect password has been entered, so the value is false. The authentication was successful so the value is set to True. |

## 3.5 Database normalisation

Database normalisation is the optimisation of a database to make it organised in such a way that there is no data duplication , data is consistent throughout the database , the structure of each table is flexible enough to enter as many or as few items needed and it should enable a user to make complex queries relating data from different tables.

This will help make the solution more efficient, duplicated data will removed, reducing the size of data stored in the database. This will reduce the cost of resources for the client, they will not have to pay for expansion costs for the database hosting provider to store the same amount of items without database normalisation.

### *3.5.1 Zero Normal Form (0NF)*

This describes the state database without any normalisation. The database is most likely to run into errors and is least efficient and effective at this point. After initially creating the inventory database, it will be in this form.

### *3.5.2 First Normal Form (1NF)*

In this form, all the tables in the database contain no repeating attributes or groups of attributes. Each data item cannot be broken down any further, each row must contain a primary key and field must have a unique name. An example of how the database used in solution could be normalised to the First Normal Form is by deleting repeated data is making sure each row contains a primary key, for example, in the items table, giving each item an item ID will give each row a primary key, the database is now in the First Normal Form, if the other tables meet the conditions for First Normal Form.

### *3.5.3 Second Normal Form (2NF)*

For a database to be in the Second Normal Form, It must first be in the First Normal Form. It also must ensure that non-key attributes depend on every part of the primary key. This means that any database in the First Normal Form with simple primary keys (not compound primary keys) will automatically be in the Second Normal Form. Normalising to this form ensures no redundant data is being stored. An example of normalising to this form in the solutions database would be changing a table with primary keys Item ID and Performance ID. It also contains the attribute Price. This attribute is relevant to the Performance ID key but not the Item ID key, therefore preventing the database from being in the Second Normal Form. In order to normalise this database, another table must be created, which will hold the price of the items. It could be given the primary key Price ID.

### *3.5.4 Third Normal Form (3NF)*

To normalise to the Third Normal Form, the database must already be in the Second Normal Form. In this form, there are no non-key attributes that depend on another non-key attribute. Similarly to 2NF, 3NF tries to remove another source of redundant data. It does this by making sure the value of one attribute cannot be worked out looking at another attribute. For example in the solutions database, it may contain the attributes quantity, price and revenue, however, since the price and revenue are given , the quantity can of the items could be worked out , by dividing revenue by the items price.  This could database could be normalised by creating an additional table, Quantities, and moving the quantity of each item to that table.

# 4 User Interface

In this section I will be designing the user interface for the different sections of the system.

## 4.1 Splash screen design

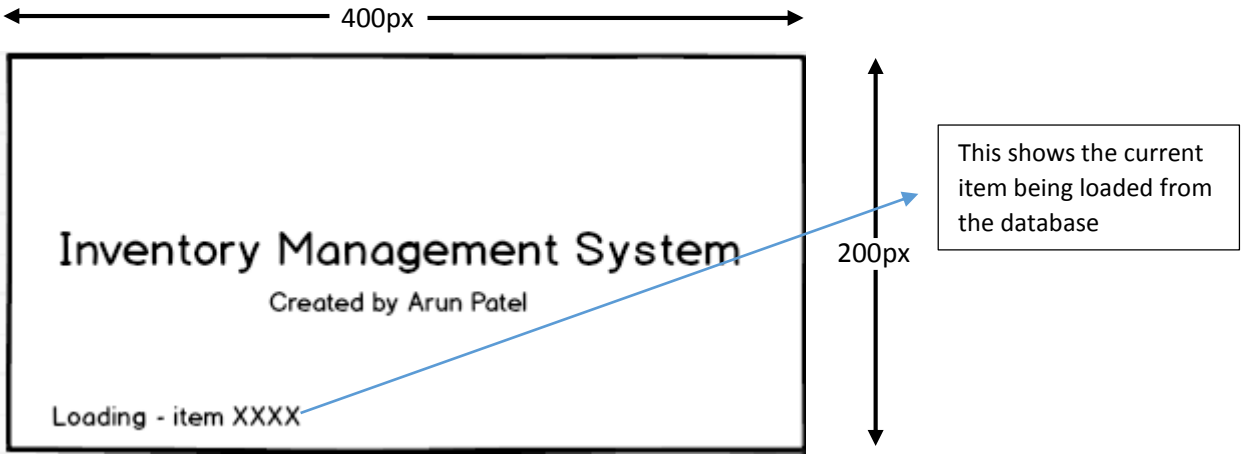



*Figure 3*

Figure 3 is a design of the systems splash screen, the first window to be displayed. It will remain on the screen until all of the items in the database has been loaded. This could take varying amounts of time based on internet connection speed.

## 4.2 Login screen design

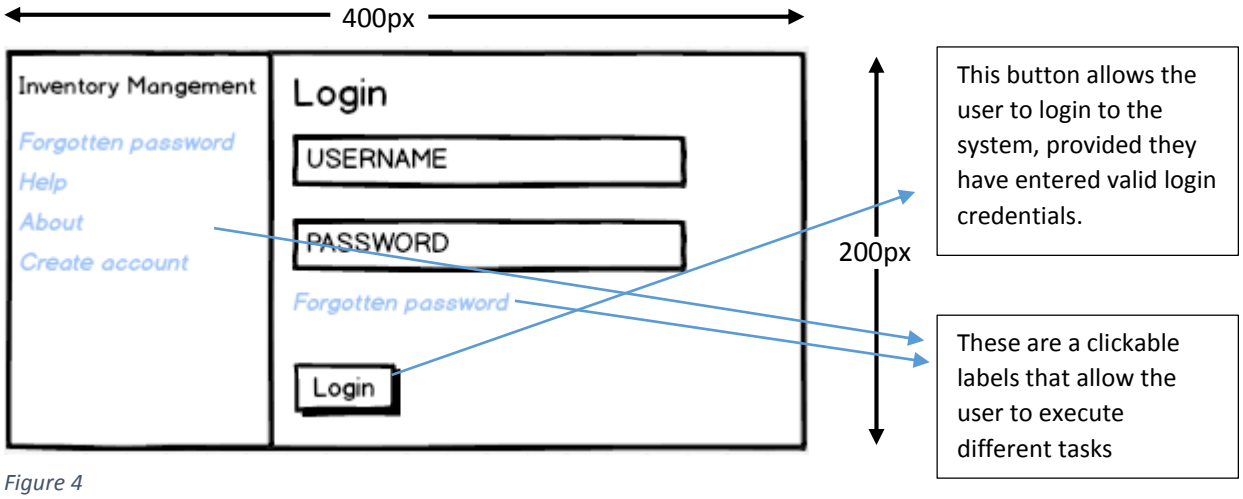



*Figure 4*

Figure 4 shows the design of the second screen to be displayed to the user. This screen will be displayed to the user without rendering a new windows but instead will overwrite the contents of

## 4.3 Main menu

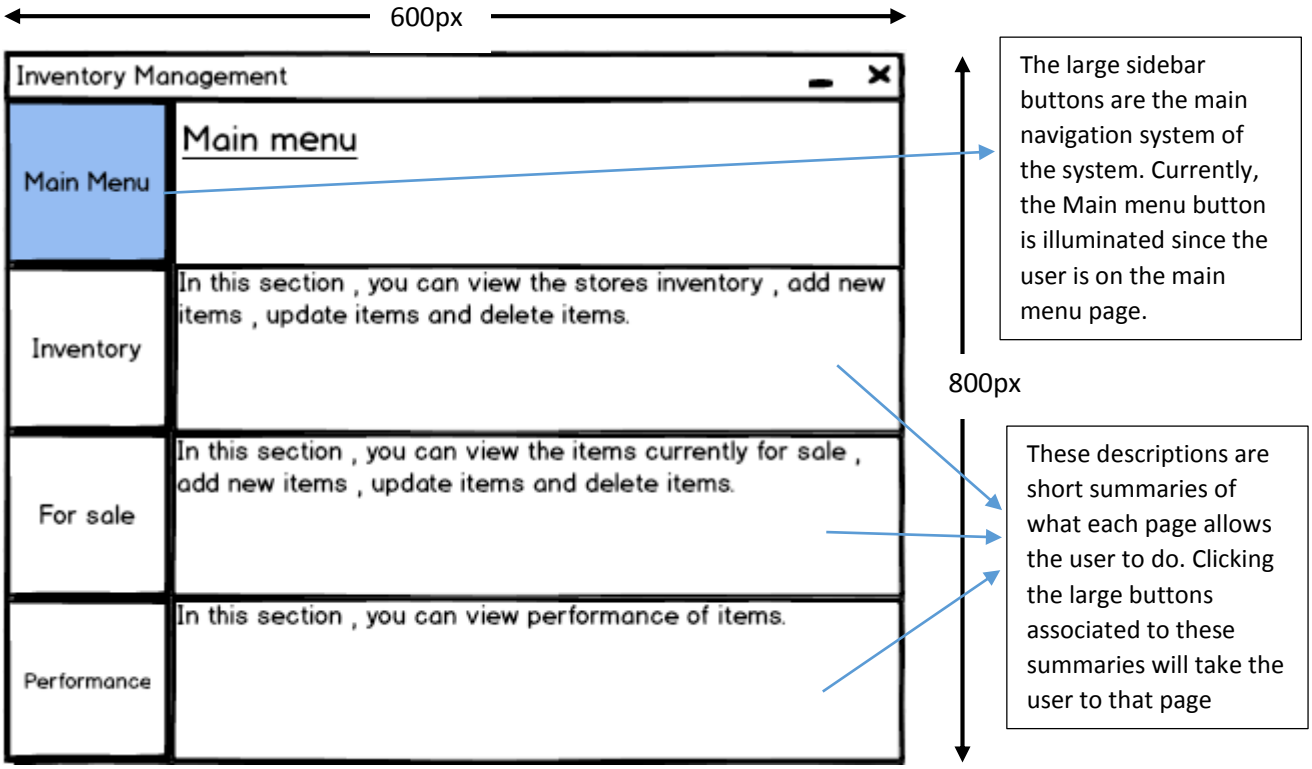



*Figure 5*

Figure 5 shows a design of the main menu of the solution. Shown on the left hand side of the interface is the navigation system that will be present on every page of the solution. Clicking the large buttons will result in the user being taken to the relevant page of the interface.

## 4.4 Inventory page

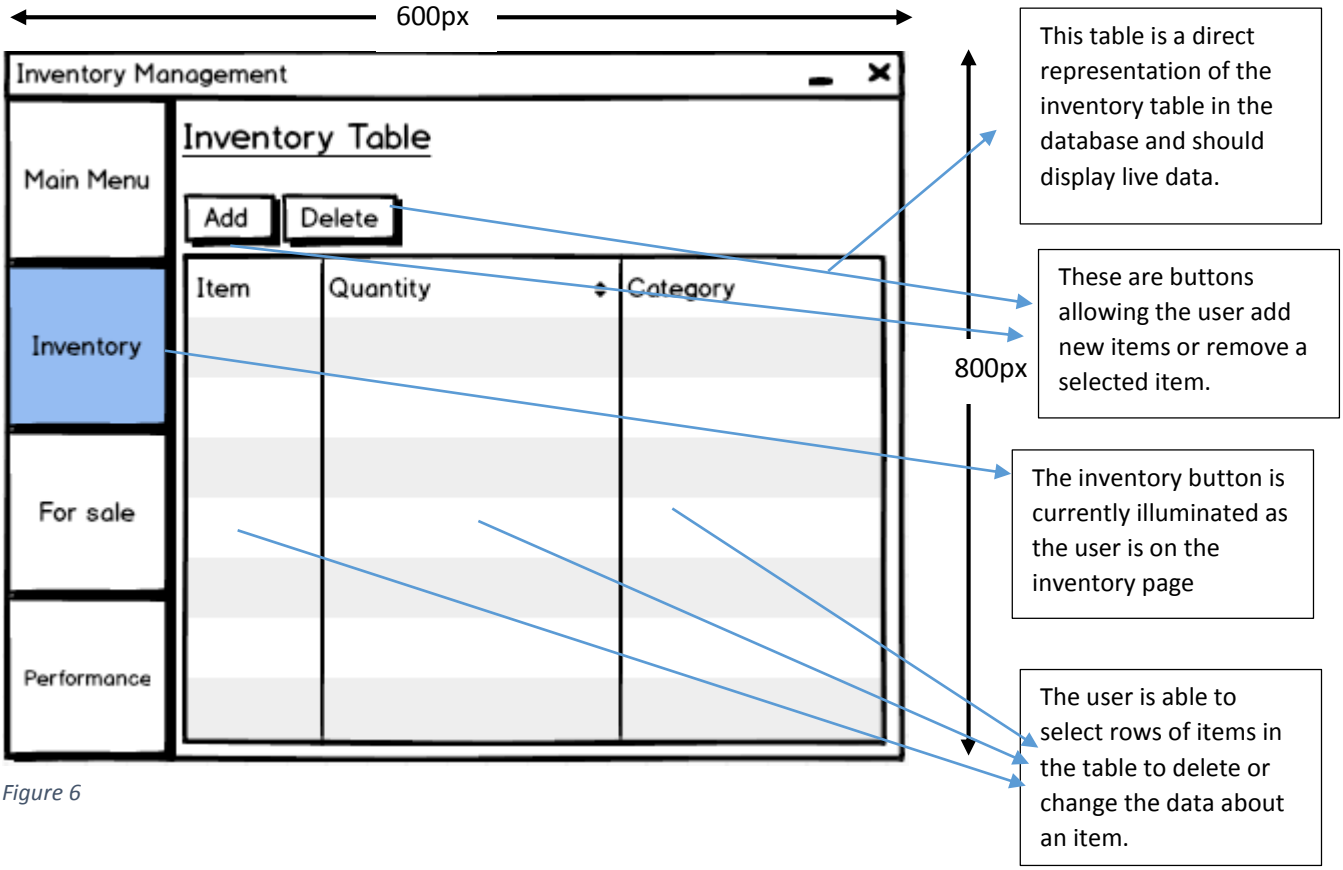



*Figure 6*

Figure 6 shows a design of the inventory screen. On this screen, the user is able to view a list of items stored in inventory, loaded and updated in real time from the database. The user is able to select items (or a single item) from the list and delete the item. The user is also able to edit details of any items by simply clicking on an item in the list and changing its field's details.

Furthermore, the user is able to click on the add button, allowing the user to add a new item into the inventory. Clicking the add button causes the following window to be opened (fig 4.4.1).

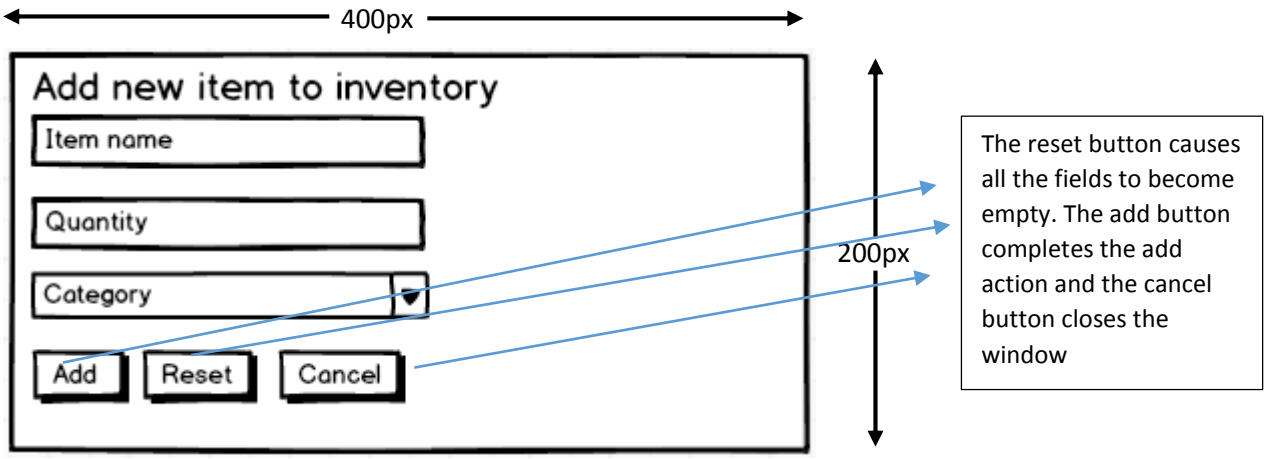



*Figure 7*

When this screen has been presented to the user (figure 7) is able to enter the details of a new item. These details include the name of the item, the amount of that item (quantity) and its category (an example of a category is food). The user then clicks on the add button, causing the new item to be added to the database and to be shown on the inventory table. Clicking the reset button causes the input fields to be reset to their default state. The cancel button causes the window to be closed and no changes will be made.

## 4.5 For sale page

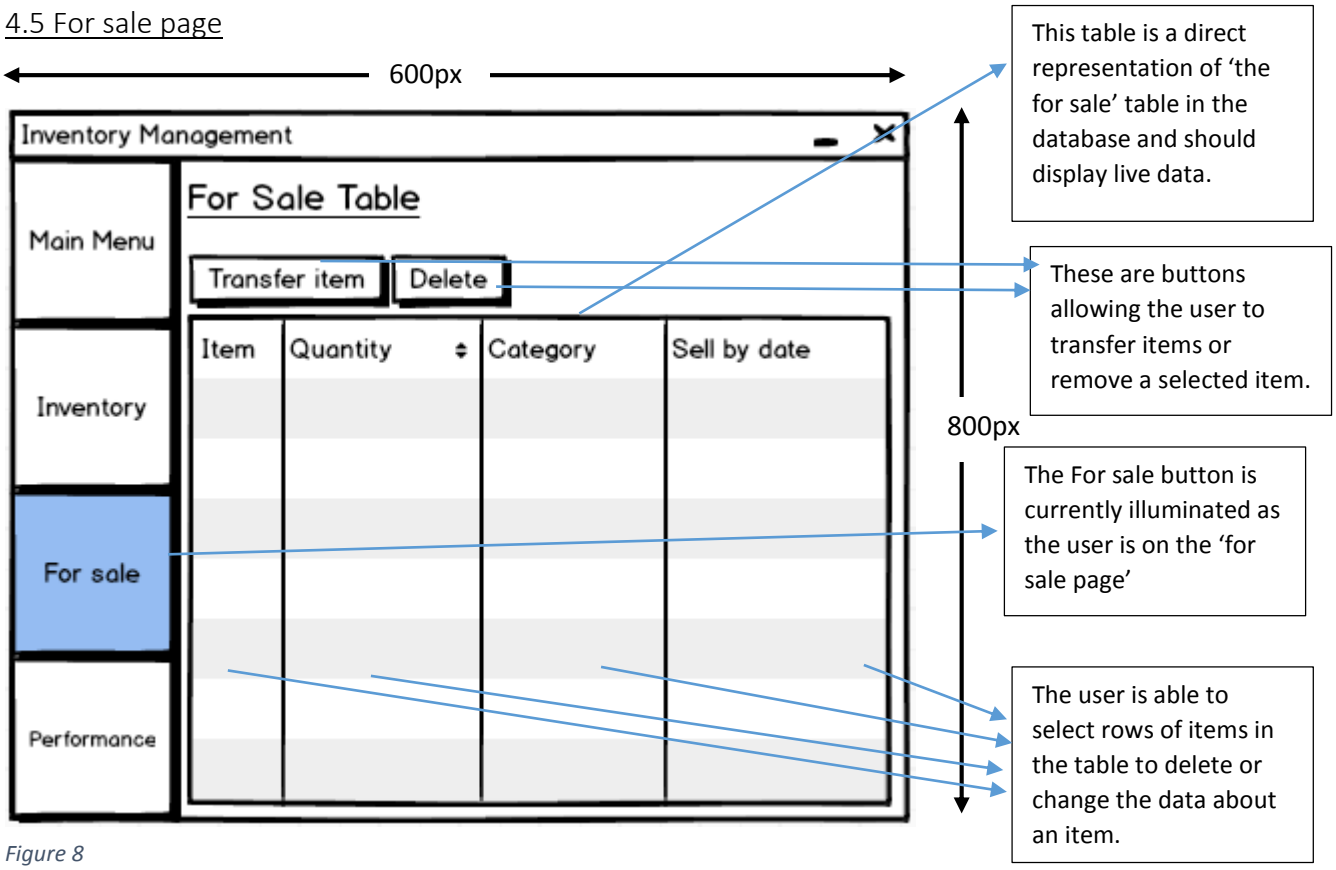



*Figure 8*

Figure 8 shows the design of the for sale page. Here, the user can view a list of all the items currently available for sale, loaded and updated in real time from the database. As with the inventory table, the user is able to change details of any fields by simply clicking on the fields and changing their values. If the user wishes to remove a selected item from the table, they can simply select the item(s) from the list and click delete, causing them to be removed from the database and the shown list.

Transferring items from inventory to the 'for sale' table requires the user clicking transfer item button. They will then be presented with the following window (figure 9).

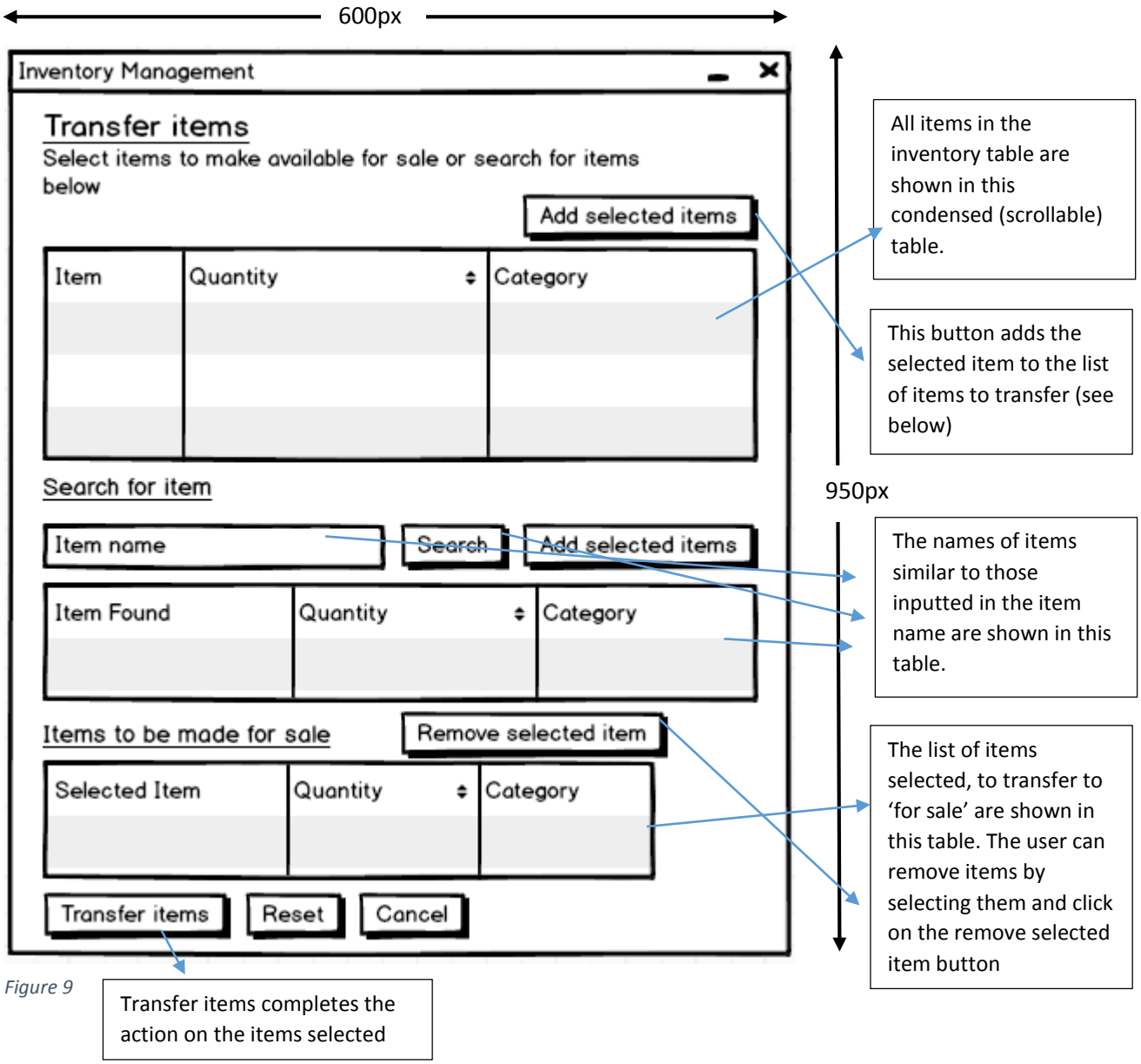



*Figure 9*

On this screen (figure 9) the user must select the items from the inventory list to transfer to the for sale list. There are two ways the user can do this, the first being manually selecting the items from a copy of the inventory list and determining the quantity of that to add. Once the items have been found, the user can click the "add selected items" button, causing the selected items to show in the table at the bottom of the page, labelled "Items to be made for sale ".

The second method involves searching for the item to be added. The user can input the name of the item in the Item name field and click search, causing a query to be ran. The results of that query are shown in the table underneath the field. The user can then select the item needed, specifying the quantity to add. They can click on the "Add selected items "button, causing the items to be added to the list at the bottom of the page.

The user is also able to remove items from the list of items to be made for sale table by selecting records in the table and clicking the "Remove selected item" button.

The Reset button at the bottom of the page causes all the fields and tables on the screen to be reset back their original state, allowing the user to start over again. The Cancel button causes the window to be closed resulting in no changes being made.

Once the user is ready to transfer the items, they can click on the Transfer items button. However, if some of the items are in the food category (or any that requires a sell by date) another window is presented to the user (fig 4.5.2). If not, then the items are added to the list and database.

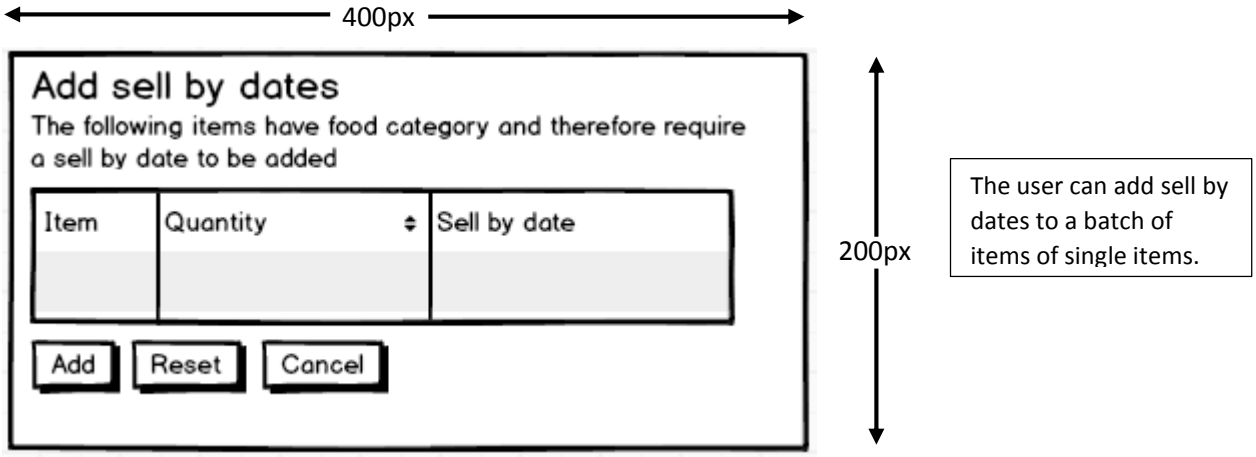



*Figure 10*

On this page, the user is prompted to add sell by dates to the required items trying to be transferred. The affected items are shown in the table on the window. The user selects the item in the list and adds the sell by date into the relevant field.

The Reset button reverts the page back to its original state and the cancel button takes the user back to the previous page (figure 10). Clicking the Add button results in the items (along with the items with sell by dates) being added to the list and database.

## 4.6 Performance page

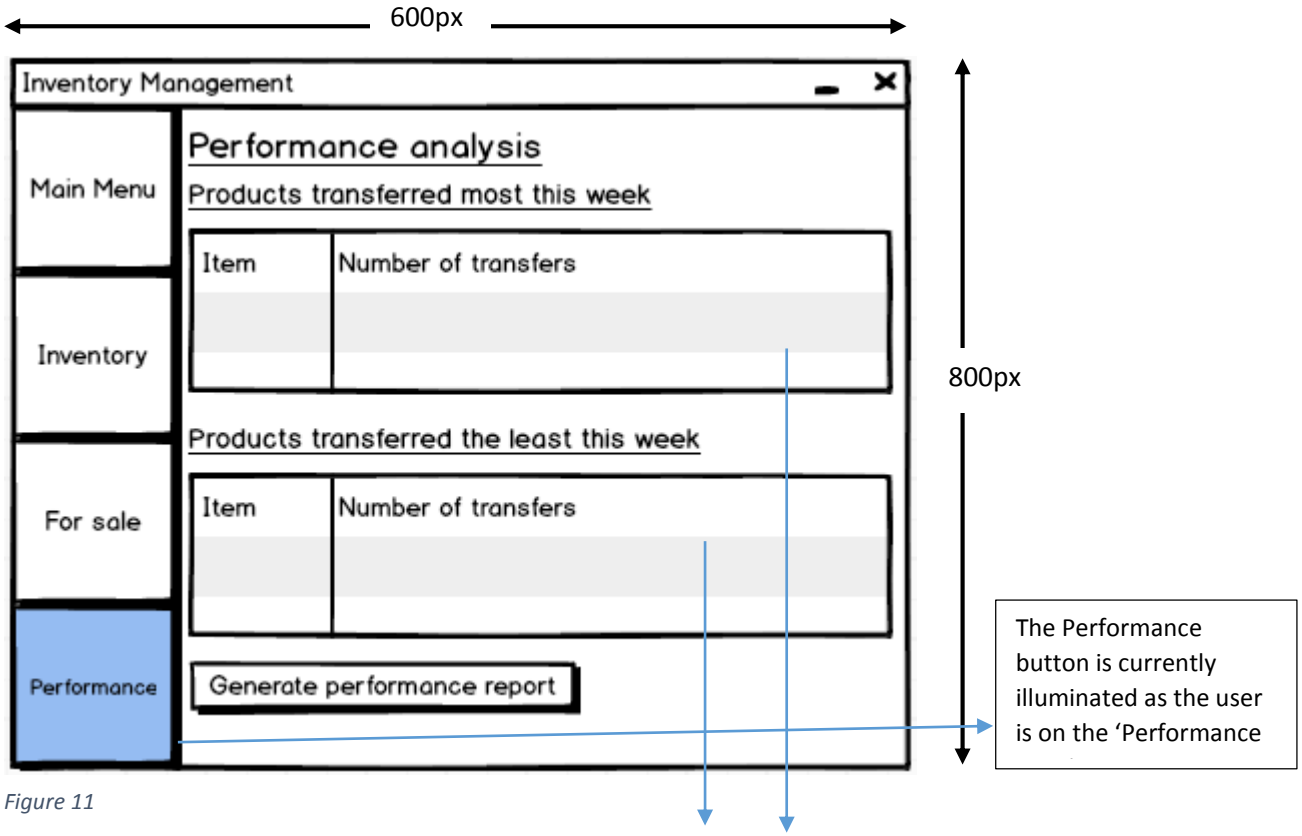



*Figure 11*

Figure 11 shows the design of the performance page. On this page, the best performed and worst performed products of the current week are shown on the page. The user can also generate reports by clicking on the "Generate performance report button". The user is then presented with the following window (figure 12).

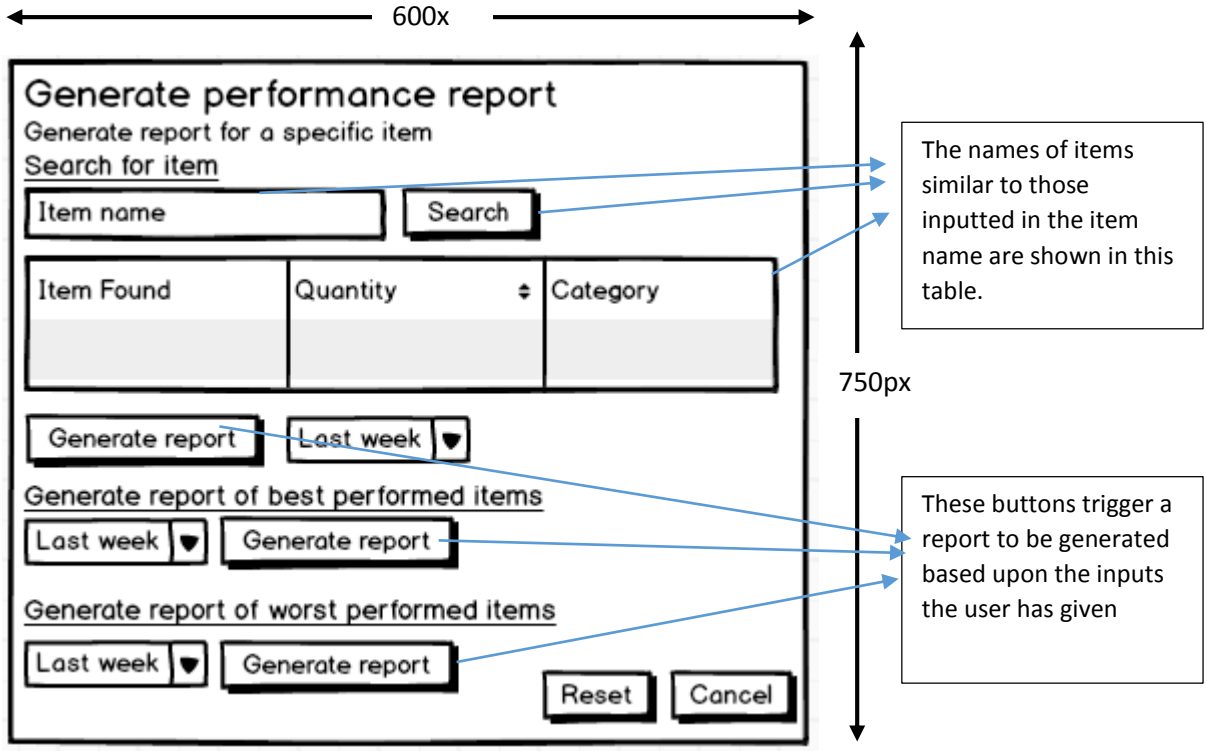



*Figure 12*

On this window (figure 12), the user is able to generate 3 types of report; a performance report for a specific item, a performance report the best performed items and the performance report for the worst performed products. In all of these reports, the user can specify the range of data to use , for example , from the last week , month , year or from a specific range ( e.g. 10/12/16 – 1/12/16 ).

The user can search for a specific item by typing in the name of the item to find in the item name field and clicking search. The results of the search are shown in the table below the field. The user can then desired from the table and then select a range for the data to be used. Once they have done this they can click generate report.

The best and worst performed reports can be generated in a similar way. The user simply specifies the range of data to use and clicks generate report. The reset button causes the page to revert back to its original state, the cancel button causes the window to be closed, no changes or reports will be made.

# 5 Data capture

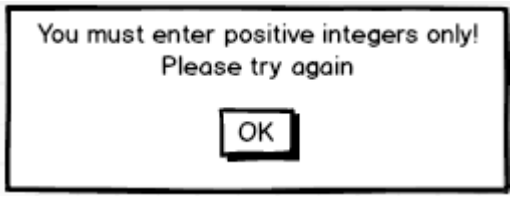
Data is inputted into fields located on the main user interface. The user is also able to input data whilst selecting specific records on the tables displayed in some sections of the interface.

## 5.1 Validation

Validation is an automatic computer check to ensure that the data entered is sensible and reasonable. However, it does not check the accuracy of data. Validation will be used wherever possible in the solution to ensure that data is as correct as possible.

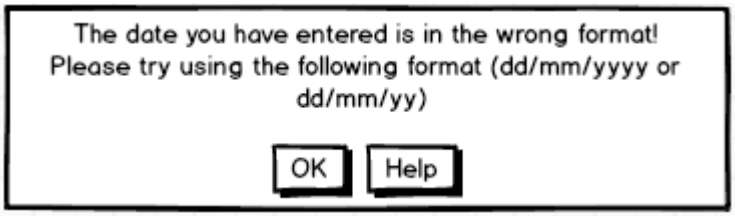
### *5.1.1 Datatype checks*

A Datatype involves checking the type of data that a user has inputted. For example, if the user inputs the following, “534”, the datatype check would identify the datatype of the input as integers. This type of check can be used to validate the input of the user when entering the quantity of an item to be added to the inventory database. By restricting the datatype to be of only positive integers I can prevent the user from entering incorrect data. For example, with the validation rule setup on the field, if the user tries to enter an input such as “w34”, the user will be presented with the following error. A suitable input would be “10”.



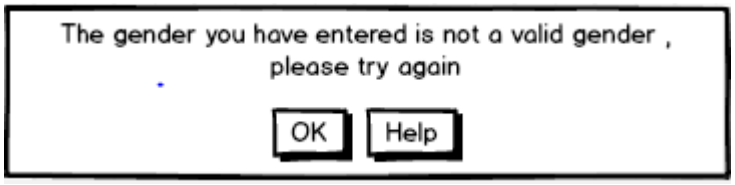

### *5.1.2 Format checks*

This validation check involves checking to see if the data is in the correct format. This can be used when the user is inputting the sell by date of an item when transferring an item. For example a valid format may be “dd/mm/yyyy” or “dd/mm/yy”, were d m and y are positive integers. A suitable input would be “01/01/2015” or “01/01/15”. However, if the user entered the following input “se/02/x4”, the validation check would detect an incorrect input and the following error message could be presented to the user.




### *5.1.3 Table Lookup checks*

A table lookup check takes the entered data item and compares it to a list of valid entries that are stored in a database table. If the entry is in the list of valid entries then it is allowed. If not, then the user is presented with an error. For example, if the user is asked to their gender when creating a staff account with the system, the solution can detect if an error was made as there are only two genders stored in the database, Male or Female. Any other inputs would not match the data stored in the table and so would cause the user to be presented with an error. The following error message could be presented the user if the validation check catches an error.


The gender you have entered is not a valid gender ,
please try again
OK
Help

## 5.2 CRUD permissions

The solution will have permissions for CRUD (Create, read, update and delete) operations. The following user groups will be created, which will determine the permissions each user will have. Users will be placed into the group depending on their role in the business.

A summary of user groups are shown below in order of most power to least. User groups share the permissions of user groups below them.

### *5.2.1 Admin group*

This user group will have full CRUD access on both tables in the database as well as direct access to the raw database. No users of the system should be in this user group, its purpose is for debugging only.

### *5.2.2 Manager group*

This user group will have full CRUD access on both tables in the database but will not have direct access to the raw database. They also have access to reports and user tracking. This user group is for the main client of the solution – Purnesh Patel.

### *5.2.3 Senior Staff group*

This user group will have Create, Read and update access to both tables only.

### *5.2.4 Staff group*

This group will have CRUD access to the inventory (stock) list table but only have Read access to the for sale table.

The permissions are explained in further detail in the following table

| **Permission group** | **Permissions** |
|---|---|
| Admin | Full access to any features of the solution. It is only to be used for debugging purposes. It shouldn't be used by the end users, they shouldn't have access to this user group either. End users can use this permission group at their own risk |
| Manager group | This user group will have the most functionality. They will have access to all features of the solution intended for the end user. Whilst they won't have direct access to the actual database, storing the data, they will have access to the database through the solution.<br>As well as this, they will have access to administrative tools such as tracking who has made changes to the inventory list or for sale list. The actual tracking log will be available to the user in this permissions group.<br>They will also have access to the item analysis section of the solution. Here they can view the performance of products.<br>They will not have access to the developer console. |
| Senior Staff group | This permission group offers most functionality of the solution but misses out some features reserved only for the manager group.<br>Users in this permission group will not be able to update items in the inventory table and will not be able to track who has made changes to any of the tables.<br>In regards to the for sale table, users in this group will be able to perform CRUD operations. In addition, they will be able to move items from inventory to the for sale table. |
| Staff group | This permission group is the most basic group the solution has to offer and offers limited functionality. It Is aimed towards staff members who are new |

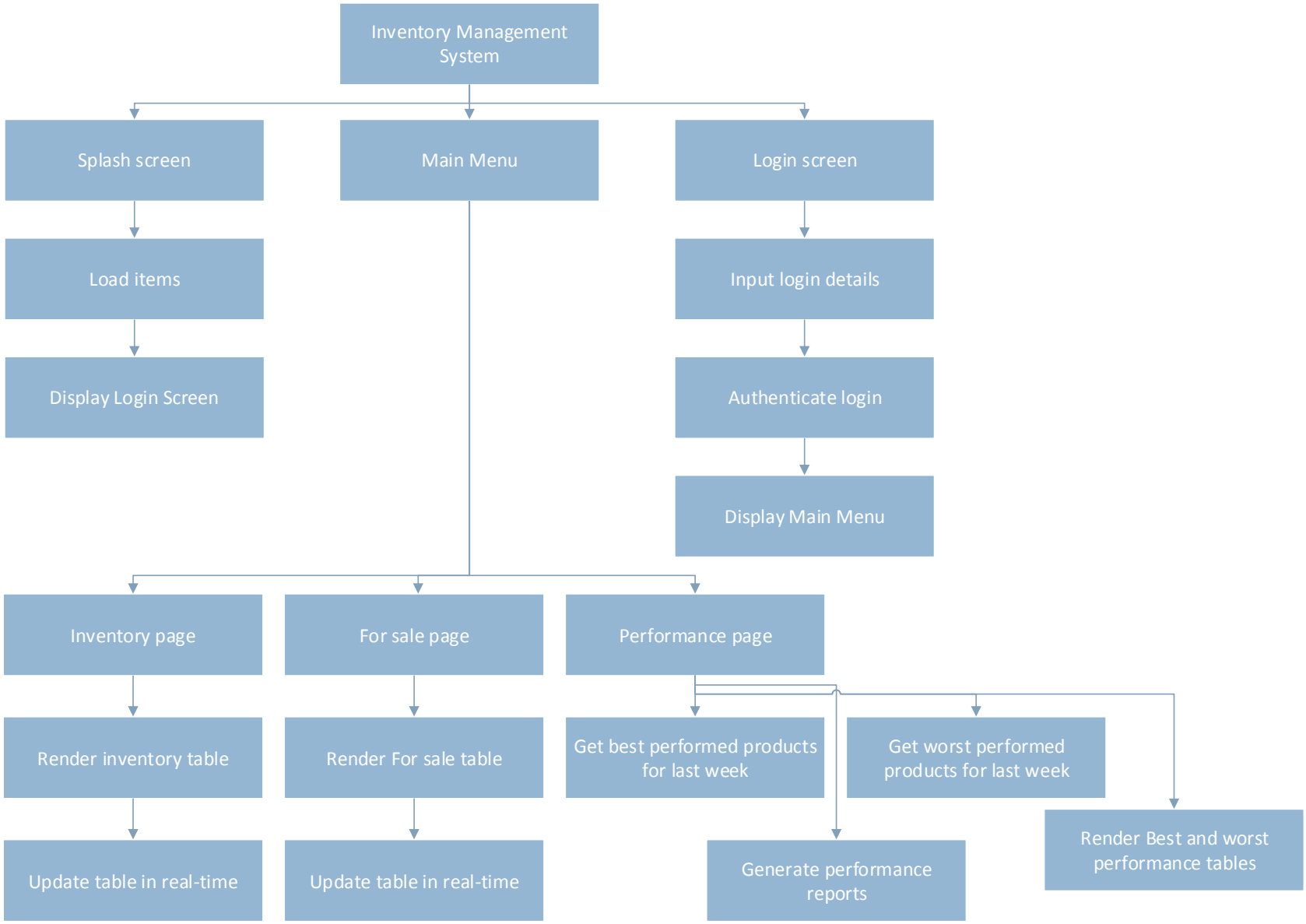
| | |
|---|---|
| | to the shop or staff members who have little experience with computers. They will be able to add or remove items from inventory table and for sale table. |

# 6 Algorithms

The purpose of designing algorithms is to make it easier for the programmer to begin programming the system.

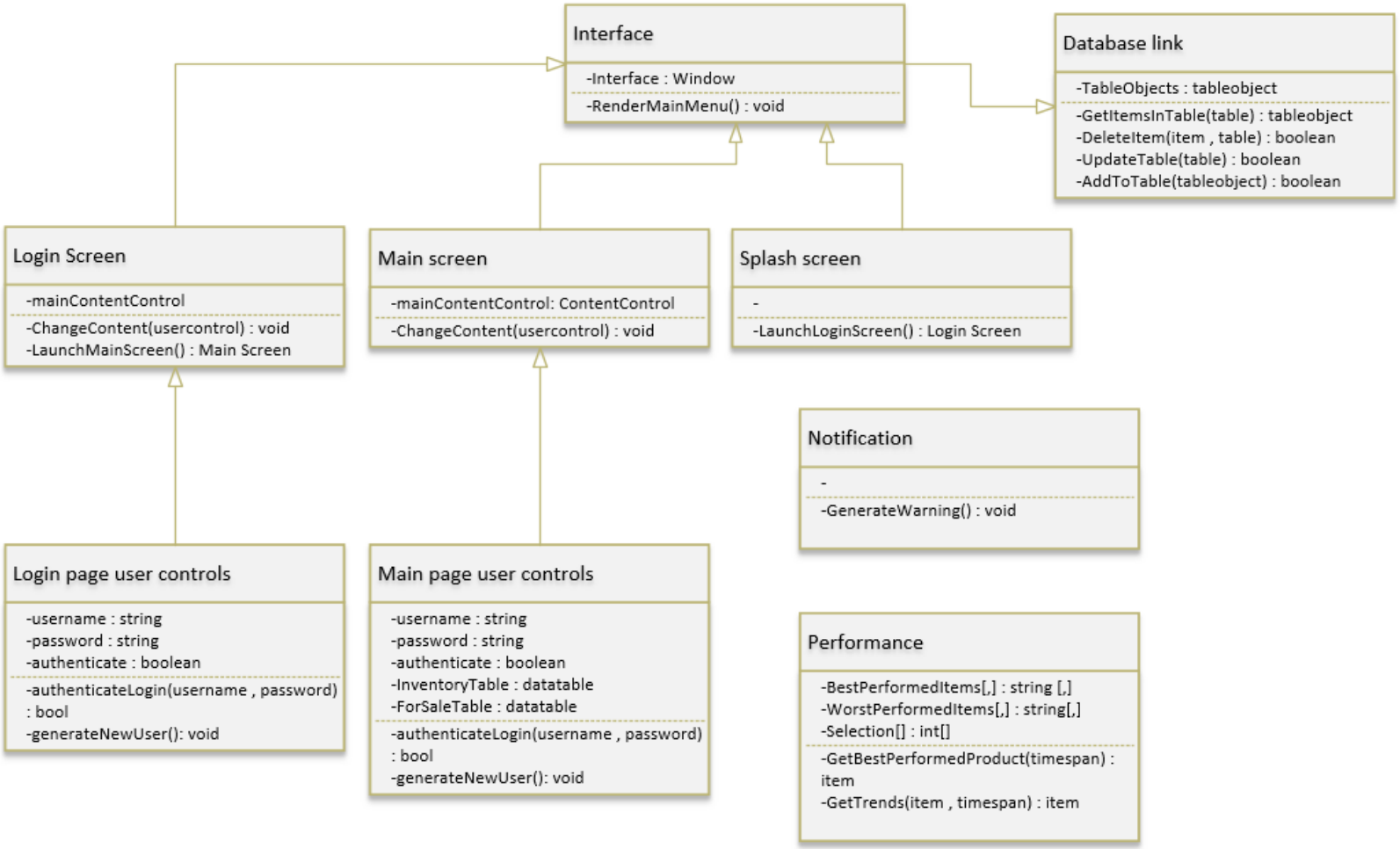
## 6.1 Structure diagram

I have created the following structure chart to show all of the sections of the system.


Inventory Management System
Splash screen
Main Menu
Login screen
Load items
Input login details
Display Login Screen
Authenticate login
Display Main Menu
Inventory page
For sale page
Performance page
Render inventory table
Render For sale table
Get best performed products for last week
Get worst performed products for last week
Update table in real-time
Update table in real-time
Generate performance reports
Render Best and worst performance tables

## 6.2 Class diagram

The class diagram below shows the classes, their subroutines and variables used in the system



## 6.3 Algorithm design

In order to design algorithms for the different parts of the system, I will be making use of the structure diagram created above (6.1). The algorithms will be represented in pseudo code or flowcharts.

*Splash Screen Loading*

The following algorithm in pseudo code represents steps involved in initialising the system whilst the splash screen is open.

```
procedure SplashScreen()
    ItemsArray[]
    counter = 0
    while(ThereAreItemsToDownload)
        ItemsArray[x] = dbase.get()
        counter++
    endwhile
    populatetables(itemsArray)
endprocedure
```

*Login authentication*

```
procedure login()
    authenticate = false
    while( authenticate == false )
        username = UserNameField.getText()
        password = PasswordField.getText()
        result[] = dbase.get(username , password) //returns an array of the username and its associated password in the database
        //result[0] = username , result[1] password
        if( (result[0] == null) OR (result[1] != password )  ) then
            authenticate = false
        else
            authenticate = true
        endif
    endwhile
endprocedure
```

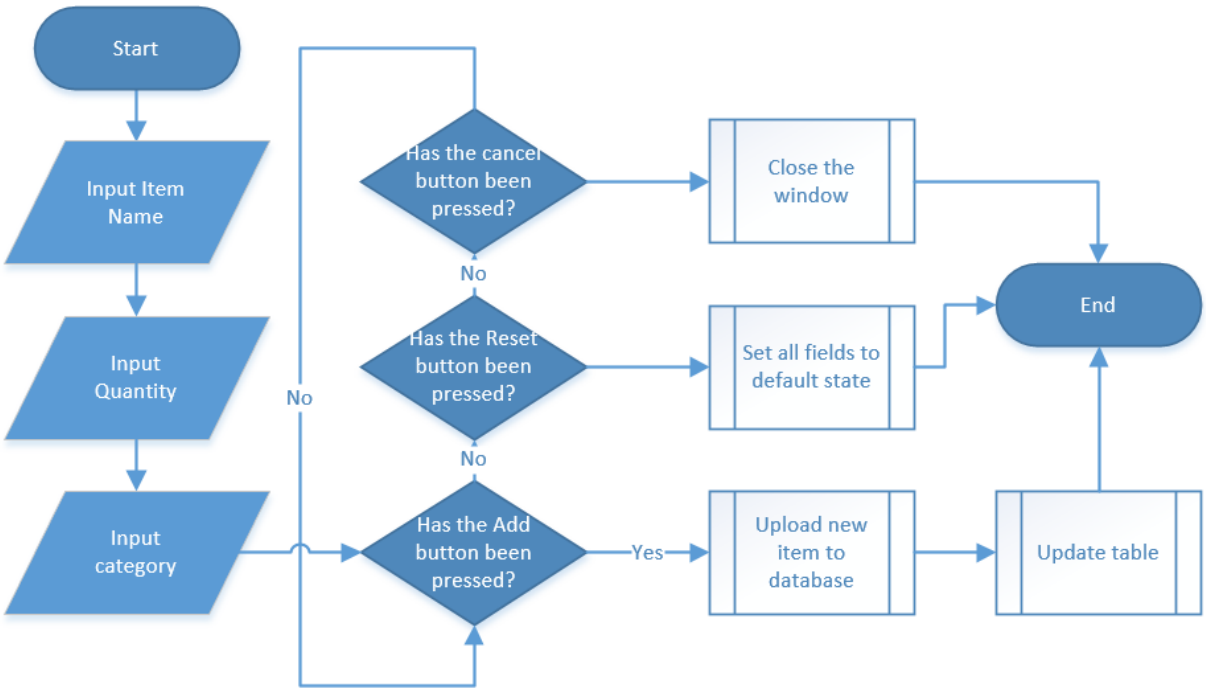
*Populate tables*

This algorithm is called when the splash screen has loaded the “itemsArray” array.

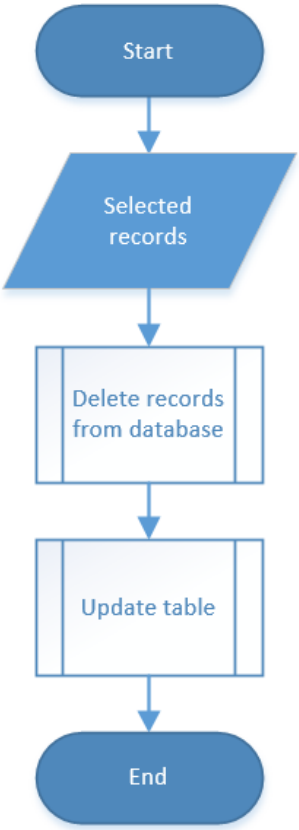
```
procedure populatetables(itemsArray[,] , table[,])
    for(y=0 to itemsArray[].getLength(0))
        for(x=0 to itemsArray[].getLength(1))
            table[y,x] = itemsArray[y,x]
        endfor
    endfor
endprocedure
```

*Adding new items to inventory*

This algorithm assumes the system is fully loaded and describes the steps involved in adding new items to the inventory.

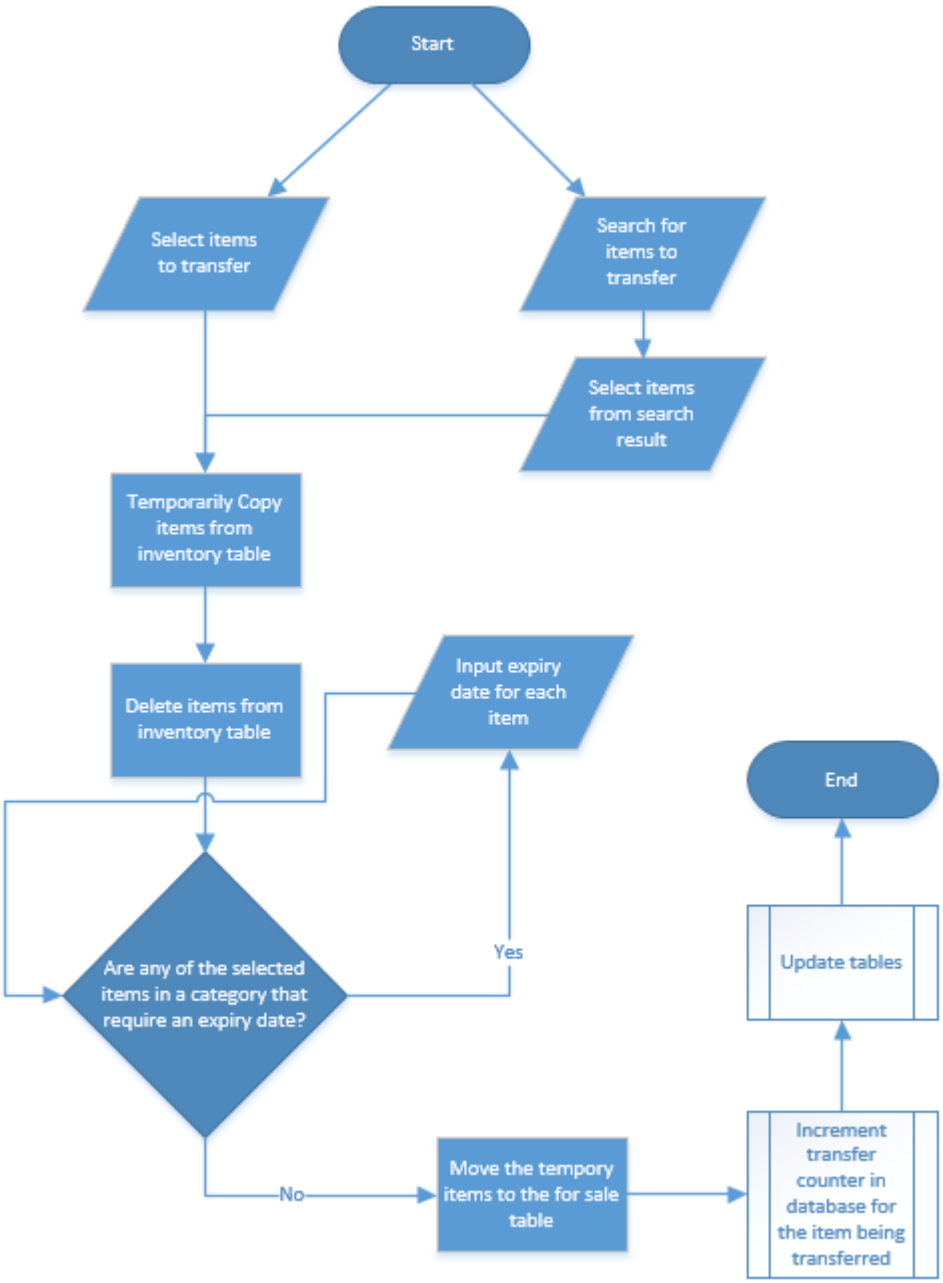



*Removing items from inventory*




*Transfer items*

The user is able to select items in two different ways, both producing the same result


Start
Select items to transfer
Search for items to transfer
Select items from search result
Temporarily Copy items from inventory table
Delete items from inventory table
Input expiry date for each item
Are any of the selected items in a category that require an expiry date?
Yes
No
Move the tempory items to the for sale table
Increment transfer counter in database for the item being transferred
Update tables
End

*Get best performed items*

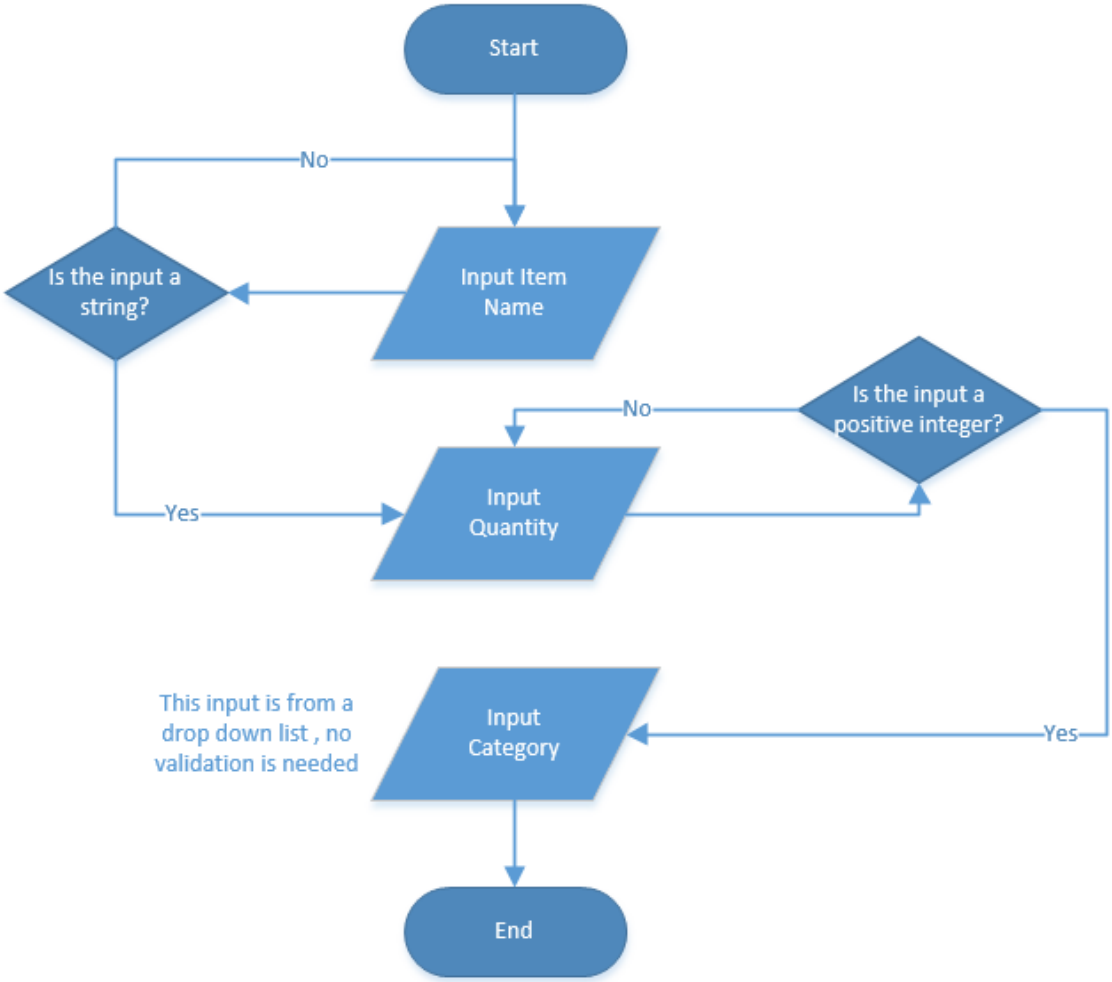
This algorithm finds the 5 best performed items. The performance of an item is determined by the amount of times an item is taken out of inventory and made available for sale. The process of logging transfers is shown in the Transfer items algorithm.

```
function GetBestPerformedItems()
    highestTransfers = 0
    highestItem[]
    counter = 0
    for ( y = 0 to dbase.getlength( inventorytable, records)  )  //loop for the amount of records in the inventory table
        if( (dbase.get(inventorytable , y , transfercounter) > highestTransfers)  AND (counter < 5)  then
            highestTransfers = dbase.get(inventorytable , y , transfercounter)
            highestItem[counter] = dbase.get(inventorytable , y , itemname)
            counter++
        endif
    endfor
    return highestItem
endfunction
```

*Get worst performed items*

```
function GetWorstPerformedItems()
    lowestTransfers = 0
    lowestItem[]
    counter = 0
    for ( y = 0 to dbase.getlength( inventorytable, records)  )  //loop for the amount of records in the inventory table
        if( (dbase.get(inventorytable , y , transfercounter) < highestTransfers) AND (counter < 5)  then
            lowestTransfers = dbase.get(inventorytable , y , transfercounter)
            lowestItem[counter] = dbase.get(inventorytable , y , itemname)
            counter++
        endif
    endfor
    return lowersItem
endfunction
```

*Validating input when adding new items to inventory*

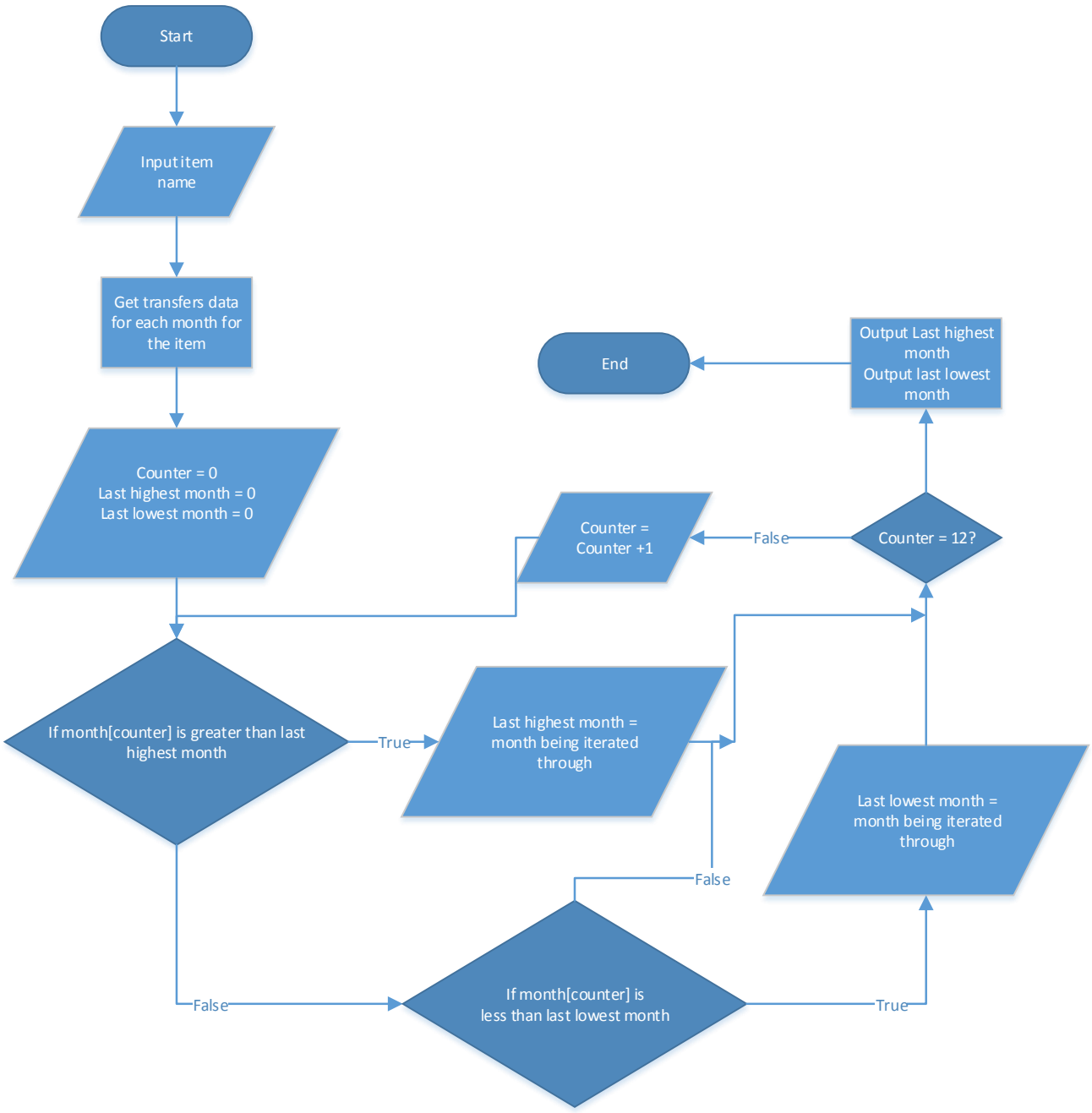


*Predicting when an item will sell best*

This function returns the month of when an item is likely to sell best.

```
function SellBest(string itemName)
    PerfData  = database.GetPerformanceItems(itemName)
    //returns the performance data (number of transfers each month) held about the inputted item
    string month = GetCurrentMonth();// returns the current month , e.g. March
    int[] TransfersMonths = new int[12]
    TransfersMonths[0] = PerfData.January.Value
    TransfersMonths[1] = PerfData.February.Value
    TransfersMonths[2] = PerfData.March.Value
    TransfersMonths[3] = PerfData.April.Value
    TransfersMonths[4] = PerfData.May.Value
    TransfersMonths[5] = PerfData.June.Value
    TransfersMonths[6] = PerfData.July.Value
    TransfersMonths[7] = PerfData.August.Value
    TransfersMonths[8] = PerfData.September.Value
    TransfersMonths[9] = PerfData.October.Value
    TransfersMonths[10] = PerfData.November.Value
    TransfersMonths[11] = PerfData.December.Value
    int highestIndex;
    int highestValue;
    for( pos = 0 to TransfersMonths.length )
        if(pos = 0)
            highestIndex = pos
            highestValue = TransfersMonths[pos]
        endif
        if(TransfersMonths[pos] > highestValue)
            highestIndex = pos
            highestValue = TransfersMonths[pos]
        endif
    endfor
    if(highestIndex = 0)
        return "January"
    elseif(highestIndex = 1)
        return "February"
    elseif(highestIndex = 2)
        return "March"
    elseif(highestIndex = 3)
        return "April"
    elseif(highestIndex = 4)
        return "May"
    elseif(highestIndex = 5)
        return "June"
    elseif(highestIndex = 6)
        return "July"
    elseif(highestIndex = 7)
        return "August"
    elseif(highestIndex = 8)
        return "September"
    elseif(highestIndex = 9)
        return "October"
    elseif(highestIndex = 10)
        return "November"
    elseif(highestIndex = 11)
        return "December"
endfunction
```

*Generating performance reports*



# 6.4 Database Queries

In this section I all will plan out the queries that will be ran on the systems database during operation.

The following list contains the types of queries the system will be able to run on the database.

- Select Query
  - This type of query is used to retrieve data from one or more tables and display the results
- Action Query
  - The database undergoes a specific action depending on what was specified in the query itself. This can include such things as creating new tables, deleting rows from existing ones and updating records or creating entirely new ones.
- Parameter Query
  - This type of query allows the same query to re used but with different criteria (a different parameter).

| **Query Name** | **Description** | **Query Type** | **Input fields** | **Output fields** |
|---|---|---|---|---|
| Search for an item in the inventory table. | This query will search for an inputted item stored in the inventory table. | Select query. | The name of the item to search for | The closest matching records in the database to the input field. |
| Search for an items quantity in the inventory table. | This query will search for an item in the inventory table and return its quantity. | Select query. | The name of the item to search for. | The closest matching records in the database but only including the item name and quantity. |
| Search for an item in the for sale table. | This query will search for an inputted item stored in the for sale table. | Select query. | The name of the item to search for. | The closest matching records in the database to the input field. |
| Search for an items quantity in the for sale table. | This query will search for an item in the 'for sale table' and return its quantity. | Select query. | The name of the item to search for. | The closest matching records in the database but only including the item name and quantity. |
| General item search | Searches all tables in the database to find an inputted item. | Parameter query. | Item name , Table (this can be all tables). | The closest matching records in the database to the input field. |

| | | | | |
|---|---|---|---|---|
| User Authenticate | Searches the database for a username and its password. | Select query. | Username and password | Returns the record of the username if the password and username match the input exactly. |
| New user | Searches the database to check if the username already exists | Action query. | Username -Account details | If there isn't an existing record with the same username then create a new record with the inputted details. |

# 7 Test Strategy

The purpose of a test plan is to ensure that the client requirements are met, also, to ensure that there is valid inputs and accurate outputs. I have created a test plan to include the purpose of a test the test data and the expected result of that test. I will be testing the system myself, to ensure the functions of the solution works. Another way of testing, is to get feedback from the user on the system. I will show the result of the test (evidence) to show if it was successful or not. Should a test be unsuccessful, I will have to try and solve the issue and test it again to see if it's working or not.

I will be carrying out functional testing in the testing phase. This will involve testing the functionality of the program as it is developed using normal, boundary and erroneous data sets. This will ensure that the solution works for whatever input it is given. Invalid inputs should be detected and brought to the attention of the user.

*Data sets*

Normal data is data within the range and data type you would normally expect. For example entering the quantity of an item should be a positive integer such as 5.

Boundary data is data at the ends of the expected range. In terms of the quantity of an item input, the boundary data set could be 1 to 1000. Although there is theoretically no limit on the upper bound of this data set, 1000 is likely the maximum quantity for a specific item in the business I am developing for.

Erroneous data is data that is either outside the expected range or of the wrong data type. For the quantity input, an erroneous data set could be "a" or -1. Here, "a" is of the wrong data type, an integer is required and -1 is outside the expected range, it is impossible to have negative one of something or 0 of something.

*Dry-running*

For some tests, it may be suitable to trial a dry run of the program. This can be helpful in locating a logic error in a program. In a dry run, I can follow through the logic of a particular part of the program in the same sequence the computer does with the aid of a trace table. In a trace table, I can note down when each variable changes and what its value is.

*Black box testing*

This will be main testing strategy. It involves carrying out tests independently of the code. It involves creating test data, based on the solutions specification that covers all the inputs, outputs and program functions.

*White box testing*

In white box testing I will be considering the solutions code, focusing on its logic. It focuses more the programs structure rather than its functionality. Tests are created which are designed to test each possible path at least once.

*Alpha testing*

This testing is carried out by the developer's in house testing team. In my case I will be handing over my solution to family members who are experienced with computers to test the functionality of the system. The user may discover errors and omissions in the systems requirements definitions. Due to the scale of my solution, I will not be carrying out Beta testing,

*Acceptance testing*

This testing is used to determine whether or not the solution has met the requirement specifications. This testing will be carried out at the end of the development stage prior to release. Depending on whether my client will have time to do this, I plan to go through every solution requirement with my client and demonstrate them in the solution I have created. However, this can take a very long time and so my client may not want to do this.

*Testing strategy*

Overall, I will be making use of Black-box, White-box and Acceptance testing. I have decided not to make use of alpha testing due to scale of my development. Since I am the only developer working on this solution, I feel as though there will be no benefit in conducting Alpha testing.

## 7.1 Test plan

| **Test No.** | **Purpose of test and Solution Requirement** | **Data set** | **Action** | **Expected result** |
|---|---|---|---|---|
| 1 | Demonstrate the requirement : The solution must be able to input new stock items. | *Normal*<br>Item name – Kings meal White Bread<br>Quantity – 10<br>Category – Food<br>*Boundary*<br>Item name – Kings meal White Bread<br>Quantity – 1<br>Category – Food<br>*Erroneous*<br>Item name – Kings meal White Bread<br>Quantity – A<br>Category – wef23 | On the inventory page I will click the add button. On the add window I will input the test data into the relevant fields. I will then click add. | The new item, with its details should be present on the inventory table on the UI and as an actual record in the actual database. The same result should be given for the boundary data set.<br>An error message should be displayed when the erroneous data set is entered into the fields as it is an invalid input. |
| 2 | Demonstrate the requirement : The solution must be able to take out stock items | *Normal*<br>Item name – Kings meal White Bread<br>Quantity – 10<br>Expiry date – 20/11/16<br>*Boundary*<br>Item name – Kings meal White Bread<br>Quantity – 1<br>Expiry date – 1/1/0<br>*Erroneous*<br>Item name – Kings meal White Bread<br>Quantity – (-10)<br>Expiry date – FF/11/16 | I will firstly click on the transfer item button. I will search for the item, by typing in the item name in the item name field. I will then select the item and select a quantity from the search results and click transfer items. When prompted to, I will add the expiry dates to the item. | The transferred item should now show on the for sale list on the UI and on the for sale table in the database. The item should no longer show up in the inventory list on the UI and on the database, since all the items have been removed. This should be the case for the normal and boundary data sets. However for the erroneous data set, an error message should be shown. The quantity cannot be negative and the expiry date should not contain letters in. |
| 3 | Demonstrate the requirement: The solution must be accessible from different computers | *Normal*<br>Computer in location A – internet connection<br>Computer in location B – internet connection<br>*Boundary* | In location A I will add an item on the inventory page and close the solution. In location B I will open the solution to check if the new item is present. | The new item added should be present in location B, with exactly the same details as the input in location A. This should be the outcome for both the normal and boundary scenarios. The boundary scenario should result in the computer at location B taking longer to load the items. In the erroneous |

| | | | | |
|---|---|---|---|---|
| | | Computer in location A – internet connection<br>Computer in location B – very slow internet connection<br>*Erroneous*<br>Computer in location A – internet connection<br>Computer in location B – no internet | | scenario, the computer at location B shouldn't be able to present any data due to lack of internet connection. If any data is displayed it should outdated data and shouldn't include anything to do with the input at location A. |
| 4 | Demonstrate the requirement: The solution should be useable by inexperienced users | Input new item into inventory. Move the item to for sale. Update its expiry date. Lookup performance information about that particular items history. | I will invite staff members at the shop who feel they are inexperienced with computers to use the solution. If they are able to complete the given task, then the solution can be labelled as useable for inexperienced users. | The staff member(s) should be able to perform the given task. |
| 5 | Demonstrate the requirement : The solution must be able to predict when products will sell well | *Normal*<br>The transfer's data for each month will equal apart from January, whose will be greater.<br>*Boundary*<br>The transfer's data for each month will 0 apart from January, whose will be greater than 0.<br>*Erroneous*<br>The transfer's data for each month will be 0. | After going to the performance page, I will search for the product the test data set is held on and press the performance analysis button. | For the normal data, the analysis should show me the best time of the year to sell the product. This should be January for the normal data set.<br>For the Boundary data set, the analysis should return January again.<br>The Erroneous data should return an error as no transfer data is being held by the system. |
| 6 | Demonstrate the requirement : The solution must be able to | *This tests should take place with an empty database*<br>*Normal* | I will setup the data for each item using database directly. After I will open the solution and go to the performance page. | The top 5 performing items table should now contain the 5 items in descending order for transfers for this month for the normal data.<br>For the boundary data, the item with transfers of greater than 1 should be at the top of the |

| | | | | |
|---|---|---|---|---|
| | find popular products | 5 items with random performance transfers for the current month between 1 and 100.<br>*Boundary*<br>4 items with performance transfers of 1 and 1 item with performance transfers of greater than 1.<br>*Erroneous*<br>The transfer's data for each month will be 0. | | table, whilst the other items should be underneath the table in any order.<br>For the Erroneous data, the table should be in an empty state. |
| 7 | Demonstrate the requirement :<br>It must be able to find unpopular products | *This tests should take place with an empty database*<br>*Normal*<br>5 items with random performance transfers for the current month between 1 and 100.<br>*Boundary*<br>4 items with performance transfers of 10. 1 item with performance transfers of less than 10.<br>*Erroneous*<br>The transfer's data for each month will be 0. | I will setup the data for each item using database directly. After I will open the solution and go to the performance page. | The top 5 worst performing table should show the 5 items in ascending order for transfers for this month for the normal data. For the boundary data, the item with transfers of less than 10 should be at the top of the table, whilst the other items should be underneath the table in any order.<br>For the Erroneous data, the table should be in an empty state. |
| 8 | Demonstrate the requirement:<br>The solution must keep a log of all actions that have taken place by users | *This tests should take place with an empty database*<br><br>Item name – Kings meal White Bread<br>Quantity – 10<br>Category – Food | I will navigate to the inventory page and click on the add item button. I will then enter the test data into entry box. After it is added I will navigate back to the home page and click on the log button. | The log dialog should show a table showing the action I have just made. It should include the date of the action, a description of the action (for example, [username] has added [quantity] [item name] to inventory), the First Name of the user who made the action (this should be myself) and the role of that person (in this case it will be Admin). |

| | | | | |
|---|---|---|---|---|
| 9 | Test the validity of data entered into inventory | *Normal*<br>Item name – Kings meal White Bread<br>Quantity – 10<br>Category – Food<br>*Boundary*<br>Item name – Kings meal White Bread<br>Quantity – 1<br>Category – Food<br>*Erroneous*<br>Item name – Kings meal White Bread<br>Quantity – A<br>Category – wef23 | On the inventory page I will click the add button. On the add window I will input the test data into the relevant fields. I will then click add. | As with test number 1 , the system should be able to input the Normal and Boundary data into the database, however when the erroneous data is entered , the system should give an error message, telling the user that their input is invalid , or not allow the input to be processed. |
| 10 | Test the solutions ability to create new user accounts | *Username* – Arun.Patel<br>*First Name*- Arun<br>*Last Name*- Patel<br>*Password*- passwordTest<br>*Role* – Senior Staff | On the create an account page, I will enter the test data into the relevant fields. Once I have clicked on the submit button I will look at the database and check to see if the new account is shown in the database. | When I click on the submit button, a success message should be shown. When I check the database, a new record should have been created showing that the new account has been created with the exact details as the test data. |
| 11 | Test the solutions ability to authenticate account logins | *Normal*<br>*Username* – Arun.Patel<br>*Password*- passwordTest<br>*Erroneous*<br>*Username* – Arun.Patel<br>*Password*- test | This test will be conducted after completing test number 10.<br>On the login page main page, I will enter the test data into the relevant field. | Entering the normal data should cause a success message to be shown upon pressing the login button.<br>When entering the erroneous data, a failure message should be shown and the system should not continue to show the main page. |
| 12 | Test the solutions ability to implement permissions. | *Normal*<br>*User group role – Manager*<br>*Erroneous*<br>*User group role- Staff* | Whilst logged in with a user account of the test data, I will try to open the log window. | Whilst logged into an account with a user group corresponding to the normal test data (manager group), when I try to click on the log window, I should be shown the log window.<br>Whilst logged in with an account with the staff user group (erroneous test data) I should be |

| | | | | |
|---|---|---|---|---|
| | | | | presented with an error when I click on the log window button |

# 8 Signatures

Confirmation of design stage: _____ Date of signature-

# Development

At this point, I will now begin developing the solution. In this section of my report I will document my progress in creating the solution. I will be developing in a modular approach. These modules include, an interface module, a database module, database connectivity module (including Create, Read, Update and Delete) and an AI and machine learning module. To start with I will begin with the interface module.

# 1 Interface

I will be replicating my interface designs using the C# programming language and the Windows Presentation Foundation, which makes use of XAML (Extensible Application Markup Language) to design interfaces.

### *1.01 XAML*

XAML is a declarative markup language used to describe the Graphical User Interface (GUI) of an application. Although most of the XAML will be created by the GUI designer in Visual Studio, I will still be altering and creating a lot of XAML myself.

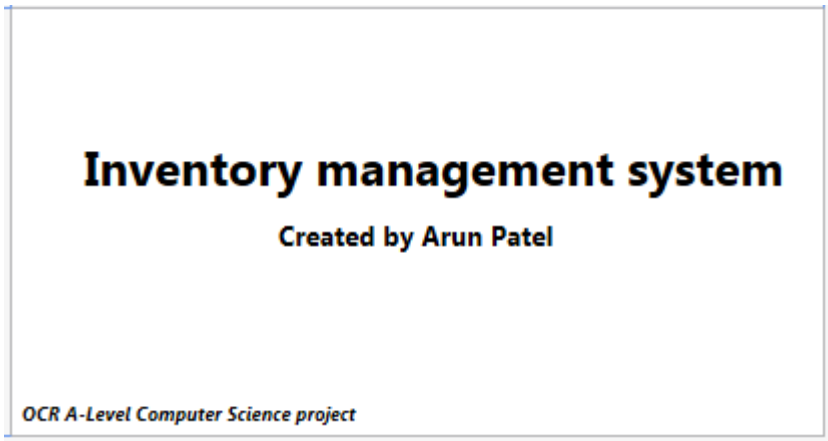
As described in the design section, the interface will feature three main windows, the splash window, login window and the main window. To start with I began designing the splash screen.

The XAML for the initial splash screen design was the following:

## 1.1 Splash Screen

```
        Title="SplashScreen" Height="212" Width="405" BorderBrush="Transparent"
        WindowStyle="None" AllowsTransparency="True" WindowStartupLocation="CenterScreen"
        MouseLeftButtonDown="WindowMouseLeftButtonDown" ResizeMode="NoResize" >

    <Grid>
        <Label x:Name="label" Content="Inventory management system" HorizontalAlignment="Left"
Margin="31,57,0,0" VerticalAlignment="Top" RenderTransformOrigin="-2.5,-0.462" FontSize="24" FontWeight="Bold"
               Foreground="Black"/>
        <Label x:Name="label_Copy" Content="Created by Arun Patel" HorizontalAlignment="Left" Margin="128,99,0,0"
               VerticalAlignment="Top" RenderTransformOrigin="-2.5,-0.462" FontSize="13.333" FontWeight="Bold"
               Foreground="Black"/>
        <Label x:Name="label_Copy1" Content="OCR A-Level Computer Science project " HorizontalAlignment="Left" Margin="0,190,0,0"
               VerticalAlignment="Top" RenderTransformOrigin="-2.5,-0.462" FontSize="9.333" FontStyle="Italic" FontWeight="Bold"
               Foreground="Black" />
    </Grid>

</Window>
```

This produced the following,



I then added a background to the splash screen and by using the background property on the grid.

```
<Grid>
    <Grid.Background>
        <ImageBrush ImageSource="Resources/inventoryBLUR.jpg"/>
    </Grid.Background>
```

The image, inventoryBLUR, is an image I editing in Photoshop of a warehouse with a Gaussian blur added. I also changed the colour of the text on the splash screen to white by making use of this property on each label. I could have made use of styling but chose not to since there are only 3 labels.

```
Foreground="White"/>
```

These changes produced the following result.

The following XAML made the splash screen fade into the screen rather than just appear.

```
<Window.Triggers>
    <EventTrigger RoutedEvent="Window.Loaded">
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Duration="00:00:01" Storyboard.TargetProperty="Opacity" From="0" To="1"  />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger>
</Window.Triggers>
```

The following XAML is the finalised splash screen window.

```
        Title="SplashScreen" Height="212" Width="405" BorderBrush="Transparent" WindowStyle="None"
        AllowsTransparency="True" WindowStartupLocation="CenterScreen" MouseLeftButtonDown="WindowMouseLeftButtonDown" ResizeMode="NoResize" >
    <Window.Triggers>
        <EventTrigger RoutedEvent="Window.Loaded">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation Duration="00:00:01" Storyboard.TargetProperty="Opacity" From="0" To="1"  />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Window.Triggers>
    <Grid>
        <Grid.Background>
            <ImageBrush ImageSource="Resources/inventoryBLUR.jpg"/>
        </Grid.Background>
        <Label x:Name="label" Content="Inventory management system" HorizontalAlignment="Left" Margin="31,57,0,0" VerticalAlignment="Top"
               RenderTransformOrigin="-2.5,-0.462"
               FontSize="24" FontWeight="Bold" Foreground="White"/>
        <Label x:Name="label_Copy" Content="Created by Arun Patel" HorizontalAlignment="Left" Margin="128,99,0,0" VerticalAlignment="Top"
               RenderTransformOrigin="-2.5,-0.462"
               FontSize="13.333" FontWeight="Bold" Foreground="White"/>
        <Label x:Name="label_Copy1" Content="OCR A-Level Computer Science project " HorizontalAlignment="Left" Margin="0,190,0,0" VerticalAlignment="Top"
               RenderTransformOrigin="-2.5,-0.462"
               FontSize="9.333" FontStyle="Italic" FontWeight="Bold" Foreground="White" />
    </Grid>
</Window>
```

## 1.2 Login window

Now that I have designed my splash screen I will now design my login screen. The following is the XAML code used to create the window.

```
        Title="Login" Height="212" Width="405" BorderBrush="Transparent" WindowStyle="None"
        AllowsTransparency="True" WindowStartupLocation="CenterScreen" MouseLeftButtonDown="WindowMouseLeftButtonDown" ResizeMode="NoResize" >
    <Window.Resources>
        <Style x:Key="MouseOverLabelStyle" TargetType="Button" >
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border Background="Transparent">
                            <ContentPresenter/>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="Foreground" Value="#FF3176D8"></Setter>
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Foreground" Value="Green"/>
                </Trigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

    <Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <StackPanel HorizontalAlignment="Left" Height="212" VerticalAlignment="Top" Width="124" Background="#FFA0A0A0">
                <Label x:Name="title_label" Content="Inventory management" HorizontalAlignment="Left" VerticalAlignment="Top" Grid.Column="0" Grid.Row="0"
                       FontSize="10.667" Foreground="Black" />
                <Button x:Name="forgotten_password_left_button" Content="forgotten password?" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top"
                        FontSize="10.667" FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="forgotten_password_button_Click"/>
                <Button x:Name="help_button" Content="Help" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" FontSize="10.667"
                        FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="help_button_Click" />
                <Button x:Name="about_button" Content="About" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top"
                        FontSize="10.667" FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="about_button_Click" />
                <Button x:Name="create_button" Content="Create Account" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top"
                        FontSize="10.667" FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="create_button_Click" />
                <Button x:Name="home_button" Content="Home" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top"
                        FontSize="10.667" FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="home_button_Click" />
            </StackPanel>
            <ContentControl x:Name="mainContentControl" Margin="124,0,0,0" />
```

This section of the XAML code

```
<Window.Resources>
    <Style x:Key="MouseOverLabelStyle" TargetType="Button" >
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Border Background="Transparent">
                        <ContentPresenter/>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
        <Setter Property="Foreground" Value="#FF3176D8"></Setter>
        <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Foreground" Value="Green"/>
            </Trigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
```

, defines a style and template for buttons on the login screen. It tells the program to completely change the look of a button to nothing but text. When the users mouse is over the button then change the colour of the text (of the button) to green from its set colour.

This produces the following result on a button in the login screen (note – the grey background here is independent of the forgotten password button):

When the user's mouse is not over the button

When the users mouse if over the button.

The following XAML sets up the forgotten password button and assigns the created style to it.

```
<Button x:Name="forgotten_password_left_button" Content="forgotten password?" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top"
        FontSize="10.667" FontStyle="Italic" Style="{StaticResource MouseOverLabelStyle}" Height="20" Click="forgotten_password_button_Click"/>
```

```
<Border BorderBrush="Black" BorderThickness="0.5">
    <Grid>
        <StackPanel HorizontalAlignment="Left" Height="212" VerticalAlignment="Top" Width="124" Background="#FFA0A0A0">
```

The XAML above creates a black border around the edges of the login window (since I defined the window to have no Window Style). The buttons on the login window are placed in a stack panel, so they are appear sequentially from top to bottom.

The finalised logins screen is the following:

In order to make the login screen appear once the splash screen has closed and logic for moving the splash screen, I used the following code in the c# (.cs) file associated with the splash screen XAML file.

```
DispatcherTimer timer;
public SplashScreen()                          /*Initialise a timer once the window loads*/
{
    InitializeComponent();
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(5);
    timer.Tick += timer_Tick;
    timer.Start();
}

private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    DragMove();                                /*Allows the user to move the window around screen with mouse*/
}

void timer_Tick(object sender, EventArgs e)  /*Every 5 seconds this method is called*/
{
    Login lg = new Login();                    /*Instantiate a login window*/
    lg.Show();                                 /*Show the login window to the user*/
    timer.Stop();
    this.Close();                              /*Close the current (splash screen) window -- free memory*/
}
```

## 1.3 Main window

The main window has the following XAML.

```
Title="MainWindow" Height="683.039" Width="730.814" BorderBrush="Transparent"
WindowStyle="None" AllowsTransparency="True" WindowStartupLocation="CenterScreen"
MouseLeftButtonDown="WindowMouseLeftButtonDown" ResizeMode="NoResize">
```

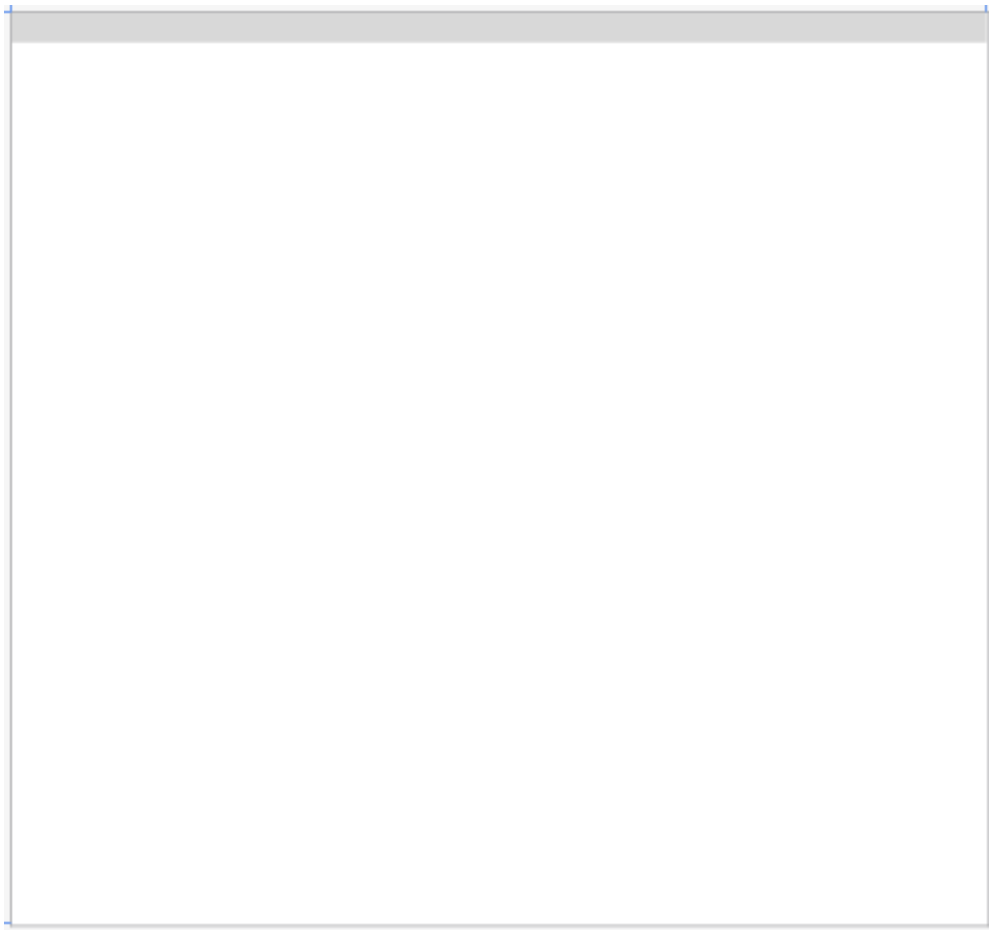
This line of XAML, creates a borderless plain window that starts in the centre of the screen. It has a mouse left button down event, which provides functionality for moving the window around the screen. The C# code for handling this event is shown below.

```
private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    DragMove();    /*Move the window when the left mouse button is down*/
}
```

My main windows grid contains the following. It creates a border around the window and grey toolbar at the top of the screen.

```
<Border BorderBrush="Black" BorderThickness="0.5">
    <Grid>
        <Rectangle Fill="#FFD8D8D8" HorizontalAlignment="Left" Height="22"  VerticalAlignment="Top" Width="731"/>
    </Grid>
</Border>
```

Here is the result that this produces:



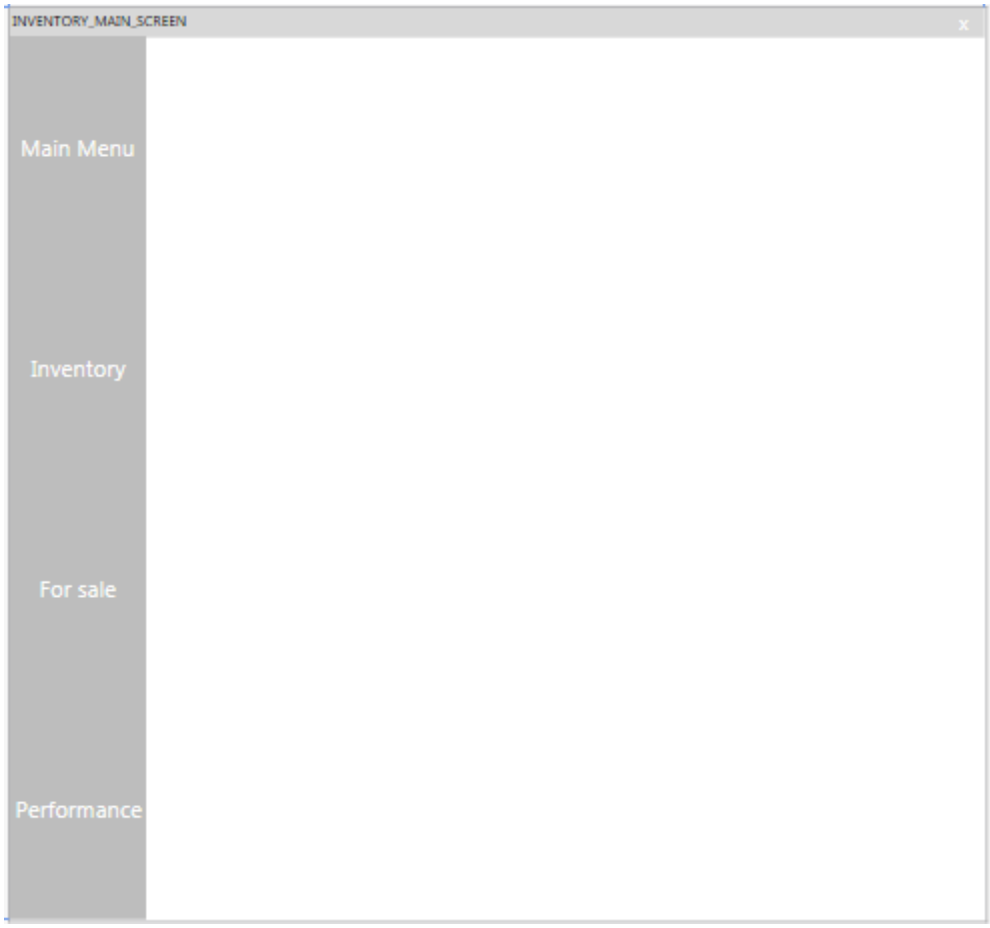
I have also created a template and style for two types of buttons the main window.

```
<Window.Resources>
    <Style x:Key="SideBarButtons" TargetType="Button" >
        <Setter Property="Background" Value="Transparent" />
        <Setter Property="Foreground" Value="White" />
        <Setter Property="FontSize" Value="15" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Button}">
                    <Border Background="{TemplateBinding Background}">
                        <Grid>
                            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Grid>
                    </Border>

                    <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter Property="Background" Value="#FF61ABE8" />
                            <Setter Property="Foreground" Value="White" />
                        </Trigger>
                        <Trigger Property="IsPressed" Value="True">
                            <Setter Property="Background" Value="#FF1C8DEA" />
                            <Setter Property="Foreground" Value="White" />
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
    <Style x:Key="CloseButton" TargetType="Button" >
        <Setter Property="Background" Value="Transparent" />
        <Setter Property="Foreground" Value="White" />
        <Setter Property="FontSize" Value="15" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Button}">
                    <Border Background="{TemplateBinding Background}">
                        <Grid>
                            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Grid>
                    </Border>

                    <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter Property="Background" Value="#FF61ABE8" />
                            <Setter Property="Foreground" Value="White" />
                        </Trigger>
                        <Trigger Property="IsPressed" Value="True">
                            <Setter Property="Background" Value="#FF1C8DEA" />
                            <Setter Property="Foreground" Value="White" />
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</Window.Resources>
```

This creates buttons that are illuminated blue when hovered over and made even darker when clicked.

```
    <Grid>
            <Rectangle Fill="#FFD8D8D8" HorizontalAlignment="Left" Height="22"
VerticalAlignment="Top" Width="731"/>



            <Button x:Name="close_button" Content="x" Margin="699,0,-1,0"
VerticalAlignment="Top" BorderBrush="Transparent"   Style="{StaticResource
CloseButton}" BorderThickness="0" Click="close_button_Click"  />
            <TextBlock x:Name="windowtitle" HorizontalAlignment="Left"
Margin="2,2,0,0" TextWrapping="Wrap" Text="INVENTORY_MAIN_SCREEN"
VerticalAlignment="Top" RenderTransformOrigin="0.336,0.322" Width="539"
FontSize="10.667" TextDecorations="{x:Null}"/>
            <StackPanel Margin="0.5,21,627,0" Background="#FFBDBDBD">
                <Button x:Name="Main_Menu_button" Content="Main Menu"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"  BorderThickness="0"
FontSize="17"  Style="{StaticResource SideBarButtons}" Height="165"
Click="Main_Menu_button_Click"/>
                <Button x:Name="Inventory_button" Content="Inventory"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"  BorderThickness="0"
FontSize="17"  Style="{StaticResource SideBarButtons}" Height="165"
Click="Inventory_button_Click"/>
                <Button x:Name="For_Sale_button" Content="For sale"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"  BorderThickness="0"
FontSize="17"  Style="{StaticResource SideBarButtons}" Height="165"
Click="For_Sale_button_Click"/>
                <Button x:Name="Performance_button" Content="Performance"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" BorderThickness="0"
FontSize="17"  Style="{StaticResource SideBarButtons}" Height="165"
Click="Performance_button_Click"/>
            </StackPanel>
        </Grid>
```

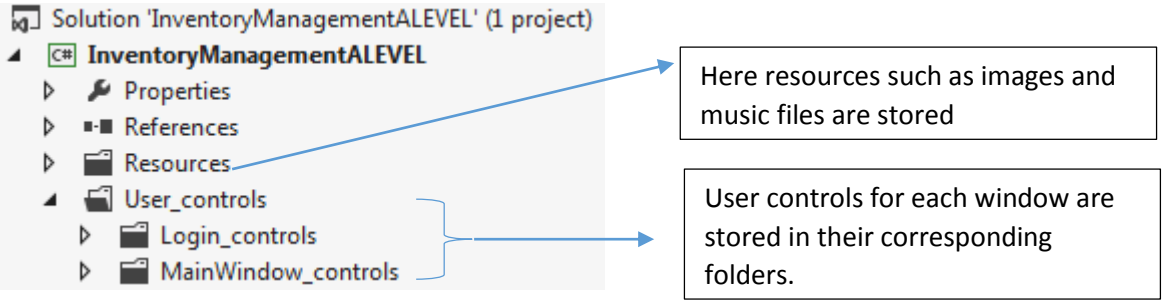
This creates the following window.



## 1.4 Login window user controls

In order to dynamically change the content of an interface such as the login window, I will be hiding and showing user controls onto the window itself. The benefit of this is that user will not be exposed to numerous amounts of popup windows but will easily navigate through a single window with information being rendered onto it dynamically.

In order to keep my work organised, I have made use of folders within my visual studio solution.




### *1.4.1 Main login user control*

In order for the user control to fit into the required space on the login screen, I have used the following dimensions, shown below in XAML code.

```
d:DesignHeight="212" d:DesignWidth="281">
```

This means that all the content in the user control I created will only be shown in the white space of the login screen (the right hand side of the image shown below).

The content of the user control can be easily be created with the following XAML.

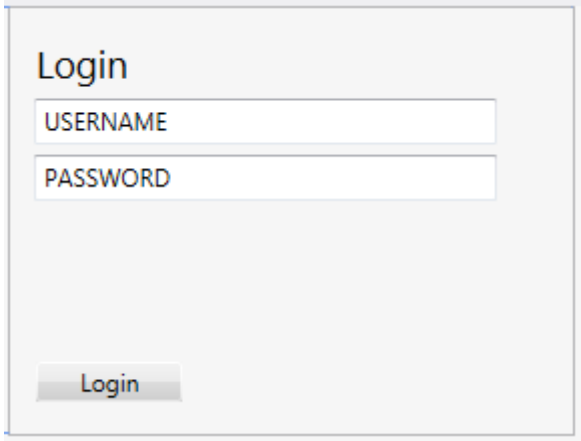
```
<Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <Label x:Name="Login_label" Content="Login" HorizontalAlignment="Left"
Margin="8,10,0,0" VerticalAlignment="Top" FontSize="18.667"/>
            <TextBox x:Name="username_textBox" HorizontalAlignment="Left" Height="23"
Margin="12,45,0,0" TextWrapping="Wrap" Text="USERNAME" VerticalAlignment="Top"
Width="231" GotFocus="username_textBox_GotFocus"/>
            <PasswordBox x:Name="password_passwordBox" HorizontalAlignment="Left"
Height="23" Margin="12,73,0,0" VerticalAlignment="Top" Width="231" />
            <TextBox x:Name="password_textBox" HorizontalAlignment="Left" Height="23"
Margin="12,73,0,0" TextWrapping="Wrap" Text="PASSWORD" VerticalAlignment="Top"
Width="231" GotFocus="password_textBox_GotFocus" />
            <Button x:Name="login_button" Content="Login" HorizontalAlignment="Left"
Margin="12,175,0,0" VerticalAlignment="Top" Width="75" Click="login_button_Click"
BorderBrush="Transparent" BorderThickness="0" />
            <Label x: Name="error label" Content="" HorizontalAlignment="Left"
Margin="10,101,0,0" VerticalAlignment="Top" Width="250" FontSize="9" Height="39"
FontStyle="Italic" Foreground="Red"/>
        </Grid>
    </Border>
```

The user control contains a textbox control, called “password_textbox” on top of a password box control called “password_passwordbox”. This is due to the fact that the password box control, by default, doesn’t allow for the password box to display a default value. To get around this I used a text box control to display a default message, “password”. Once the user has clicked on the password text box I have set it clear its value, disable itself and set the users focus on to the password box control. The error label is used to display an error message when the user enters an incorrect login and password combination.

The c# code for these events are shown below.

```
private void username_textBox_GotFocus(object sender, RoutedEventArgs e)
{
    if (username_textBox.Text == "USERNAME")           /*Clear the default text if its still there*/
    {
        username_textBox.Text = "";
    }
}

private void password_textBox_GotFocus(object sender, RoutedEventArgs e)
                                                      /*this is called if the user clicked on the password textbox*/
{
    password_textBox.Visibility = Visibility.Hidden;  /*Hide the default textbox form the user*/
    Keyboard.Focus(password_passwordBox);             /*Set the focus to the password box control*/
}
```

The following image is of the finalised user control.




In order to draw this user control onto the main login window I have included the following line of XAML inside the main login window grid.

```
<ContentControl x:Name="mainContentControl" Margin="124,0,0,0" />
```

This allows me to dynamically draw user controls onto login windows main screen by changing the value of the content control.

In the login screens c# code, I created a new global instance of the user control I have just created.

```
public partial class Login : Window
{
    LoginMainCtrl loginMainIns = new LoginMainCtrl();
```

The benefit of making this a global variable, instead of instantiating the user control each time its needed , is that it is far more efficient , less resources are needed to create the instance as it is only instantiated once at the runtime.

Furthermore, the data entered in the user control will remain, for example if the user enters half of their username and then clicks on the about button and then returns to the main page again, their username will remain. However, if I chose to instantiate the user control each time it's needed, the user's half entered username would not remain, due the user viewing a different instance of the user control.

I have chosen for the user control I have just created to be shown when the login screen first loads and when the user clicks on the home button on the login main window. Below is the c# code to allow this to happen.

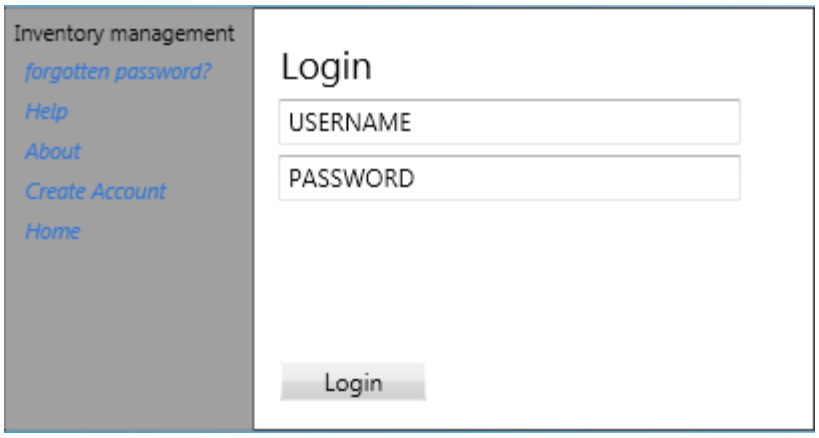
```
public Login()
{
    InitializeComponent();
    this.mainContentControl.Content = loginMainIns; //set the content control to the main login control instance
}
```

This is placed in the constructor of the login windows class.

```
private void home_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = loginMainIns;
}
```

This section of code does the same as the above but is called when the user clicks on the home button.

When running, this is the result produced when the user control is rendered onto the login screen.




For now , when the login button is clicked , it causes the program to show the main window screen.

*1.4.2 Create account user control*

Following on from the previous user control I created, this user control will have the same dimensions as shown below.

```
d:DesignHeight="212" d:DesignWidth="281">
```

The content of the user control is shown below in XAML,

```
<Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <Label x:Name="CreateAccount_label" Content="Create an account"
HorizontalAlignment="Left" Margin="6,10,0,0" VerticalAlignment="Top"
FontSize="18.667"/>

            <StackPanel HorizontalAlignment="Left" Height="137" Margin="10,45,0,0"
VerticalAlignment="Top" Width="231" >
                <TextBox x:Name="username_textBox" HorizontalAlignment="Left"
Height="23" Margin="0,0,0,0" TextWrapping="Wrap" Text="USERNAME"
VerticalAlignment="Top" Width="231" GotFocus="username_textBox_GotFocus"/>
                <TextBox x:Name="firstname_textBox" HorizontalAlignment="Left"
Height="23"  Margin="0,5,0,0"  TextWrapping="Wrap" Text="FIRST NAME"
VerticalAlignment="Top" Width="231" GotFocus="firstname_textBox_GotFocus"/>
                <TextBox x:Name="lastname_textBox" HorizontalAlignment="Left"
Height="23" Margin="0,5,0,0" TextWrapping="Wrap" Text="LAST NAME"
VerticalAlignment="Top" Width="231" GotFocus="lastname_textBox_GotFocus"/>
                <TextBox x:Name="password_textBox" HorizontalAlignment="Left"
Height="23" Margin="0,5,0,0" TextWrapping="Wrap" Text="PASSWORD"
VerticalAlignment="Top" Width="231" GotFocus="password_textBox_GotFocus"/>
                <TextBox x:Name="role_textBox" HorizontalAlignment="Left" Height="23"
Margin="0,5,0,0" TextWrapping="Wrap" Text="ROLE" VerticalAlignment="Top" Width="231"
GotFocus="role_textBox_GotFocus"/>
            </StackPanel>
            <Button x:Name="submit_button" Content="Submit" HorizontalAlignment="Left"
Margin="10,185,0,0" VerticalAlignment="Top" Width="75" BorderBrush="Transparent"
BorderThickness="0" Click="submit_button_Click" />
            <Button x:Name="clear_button" Content="Clear" HorizontalAlignment="Left"
Margin="90,185,0,0" VerticalAlignment="Top" Width="75" BorderBrush="Transparent"
BorderThickness="0" Click="clear_button_Click" />
        </Grid>

    </Border>
```

Each textbox has its own got focus event, used to detect when the user has clicked on the textbox, so that I can clear its contents if the textbox contains default text (as previously done with the previous user control). The clear button resets each field to their default text.

The c# code associated with this XAML is shown below:

```
private void ClearOnFirst(ref TextBox tb , string TextToCheck)
/*Pass by reference the textbox being cleared and its default text*/
{
    if (tb.Text == TextToCheck) /*If the default text exists , then clear it*/
    {
        tb.Text = "";
    }

}
private void username_textBox_GotFocus(object sender, RoutedEventArgs e)
    /*When the user clicks on a textbox call the ClearOnFirst method */
{
    ClearOnFirst(ref username_textBox , "USERNAME");
}

private void firstname_textBox_GotFocus(object sender, RoutedEventArgs e)
{
    ClearOnFirst(ref firstname_textBox, "FIRST NAME");
}

private void lastname_textBox_GotFocus(object sender, RoutedEventArgs e)
{
    ClearOnFirst(ref lastname_textBox, "LAST NAME");
}

private void password_textBox_GotFocus(object sender, RoutedEventArgs e)
{
    ClearOnFirst(ref password_textBox, "PASSWORD");
}

private void role_textBox_GotFocus(object sender, RoutedEventArgs e)
{
    ClearOnFirst(ref role_textBox, "ROLE");
}

private void clear_button_Click(object sender, RoutedEventArgs e)
{
    /*Sets all of the textboxes text to its default values*/
    username_textBox.Text = "USERNAME";
    firstname_textBox.Text = "FIRST NAME";
    lastname_textBox.Text = "LAST NAME";
    password_textBox.Text = "PASSWORD";
    role_textBox.Text = "ROLE";
}
```

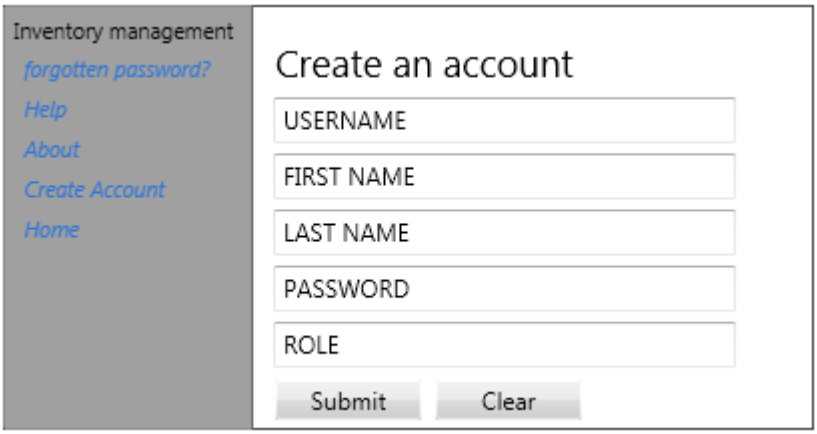
In order to have this user control render itself onto the login window, I created a new global variable instance of the user control in the main login windows C# file, as shown below:

```
LoginCreateAccount loginCreateAccountIns = new LoginCreateAccount();
```

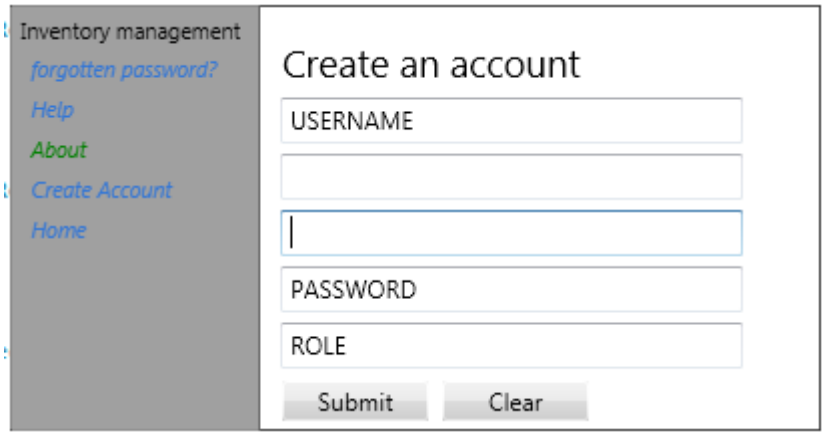
When the user clicks on the create account button, the program executes the following code, causing the user control to show:

```
private void create_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = loginCreateAccountIns;
}
```

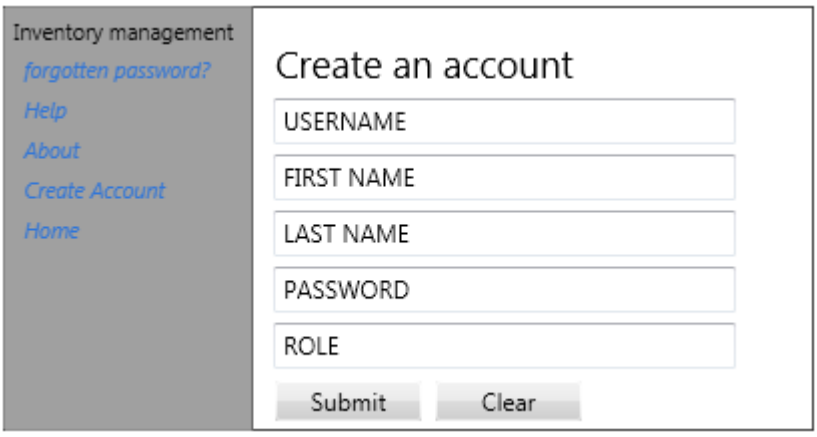
The image below shows the login screen displaying the user control I have just created.




When the user clicks on the textbox, the text is cleared, as shown below.




Clicking the clear button, causes the textboxes text to reset, as shown below.




*1.4.3 Forgotten password user control*

As with the other user controls, this one will have the following dimensions:

```
d:DesignHeight="212" d:DesignWidth="281">
```

The XAML for this user control is the following:

```
<Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <Label Content="FORGOT_PASSWORD" HorizontalAlignment="Left"
Margin="79,63,0,0" VerticalAlignment="Top" Height="65" Width="125"/>
        </Grid>

    </Border>
```

A global variable instance was created in the login windows main C# file:

```
LoginForgotPassword loginForgotPasswordIns = new LoginForgotPassword();
```

The following code, shows the user control when the user clicks on the forgotten password button:

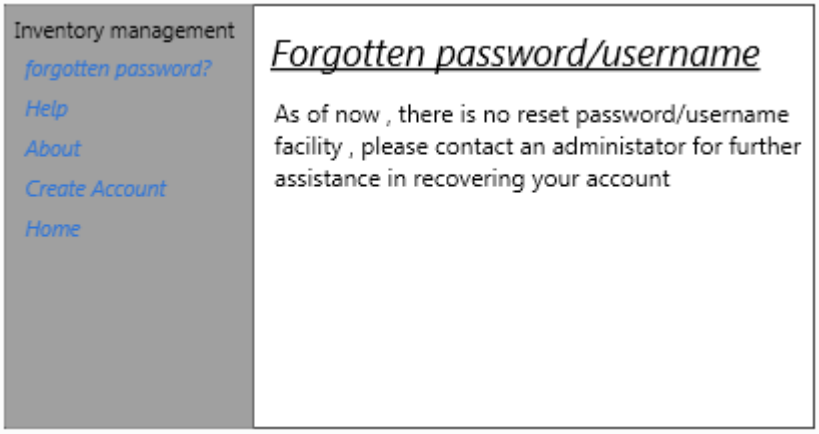
```
private void forgotten_password_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = loginForgotPasswordIns;
}
```

The image below is of the program running and displaying the forgotten password user control.

I then updated the content of grid with the following:

```
<Grid>
            <TextBlock x:Name="maintitle_TextBlock" Text="Forgotten password/username"
HorizontalAlignment="Left" VerticalAlignment="Top" FontStyle="Italic"
FontSize="18.667" TextDecorations="Underline" Margin="8,10,0,0"/>
            <TextBlock x:Name="maininfo_textBlock" HorizontalAlignment="Left"
Margin="10,45,0,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="166"
Width="270">
                As of now , there is no reset password/username facility , please
contact an administator for further assistance in
                recovering your account
            </TextBlock>
        </Grid>
```

This produces the following:




*1.4.4 Login help user control*

As with the forgotten password user control, I will be creating a blank user control to start with

Its dimensions are declared as the following:

```
d:DesignHeight="212" d:DesignWidth="281">
```

Its contents are as follows:

```
<Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <Label Content="HELP" HorizontalAlignment="Left" Margin="88,56,0,0"
VerticalAlignment="Top" Height="65" Width="125"/>
        </Grid>
    </Border>
```

The global variable instance of this user control in the login screens main C# file is shown below

```
LoginHelp loginHelpIns = new LoginHelp();
```

This user control is shown when the user clicks on the help button

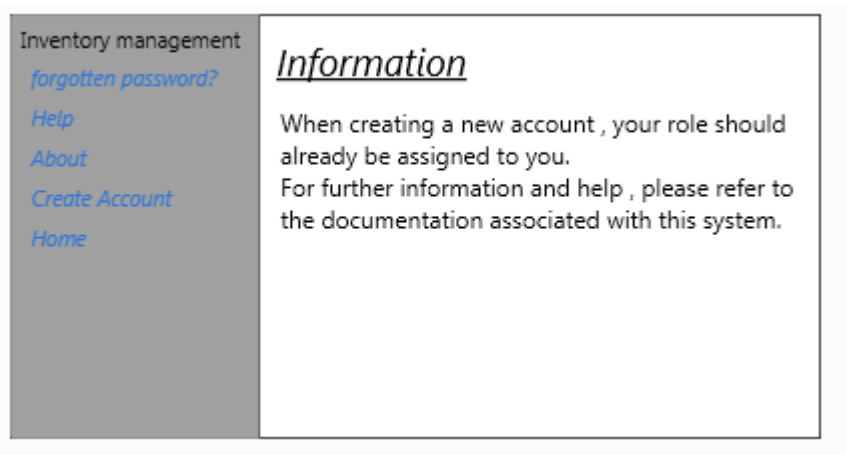
```
private void help_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = loginHelpIns;
}
```

The image below shows the user control being displayed when the program is running:

I then updated the content of grid with the following:

```
   <Grid>
            <TextBlock x:Name="maintitle_TextBlock" Text="Information"
HorizontalAlignment="Left" VerticalAlignment="Top" FontStyle="Italic"
FontSize="18.667" TextDecorations="Underline" Margin="8,10,0,0"/>
            <TextBlock x:Name="maininfo_textBlock" HorizontalAlignment="Left"
Margin="10,45,0,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="166"
Width="270">
                When creating a new account , your role should already be assigned to
you.
                <LineBreak/> For further information and help , please refer to the
documentation associated
                with this system.
            </TextBlock>
        </Grid>
```

This produced the following result:




*1.4.5 Login about user control*

The dimensions of this user control are declared as the following:

```
d:DesignHeight="212" d:DesignWidth="281">
```

The contents are as follows:

```
<Border BorderBrush="Black" BorderThickness="0.5">
        <Grid>
            <Label Content="ABOUT" HorizontalAlignment="Left" Margin="57,56,0,0"
VerticalAlignment="Top" Height="65" Width="125"/>
        </Grid>
    </Border>
```

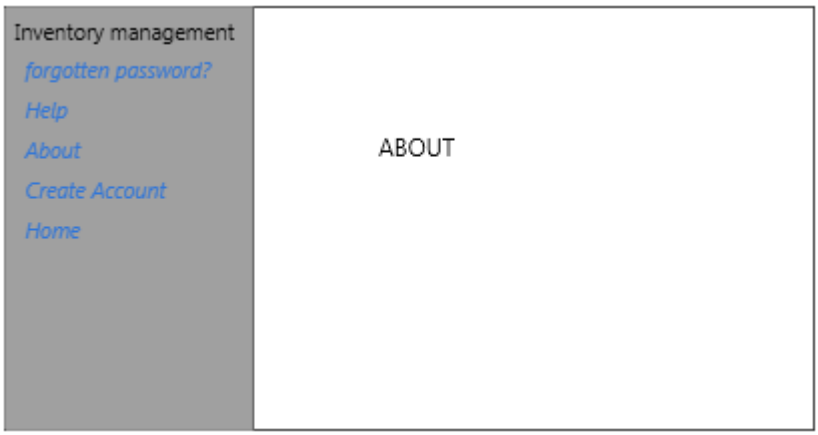
The global variable instance of this user control in the login screens main C# file is shown below

```
LoginAbout loginAboutIns = new LoginAbout();
```

This user control is shown when the user clicks on the about button

```
private void about_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = loginAboutIns;
}
```

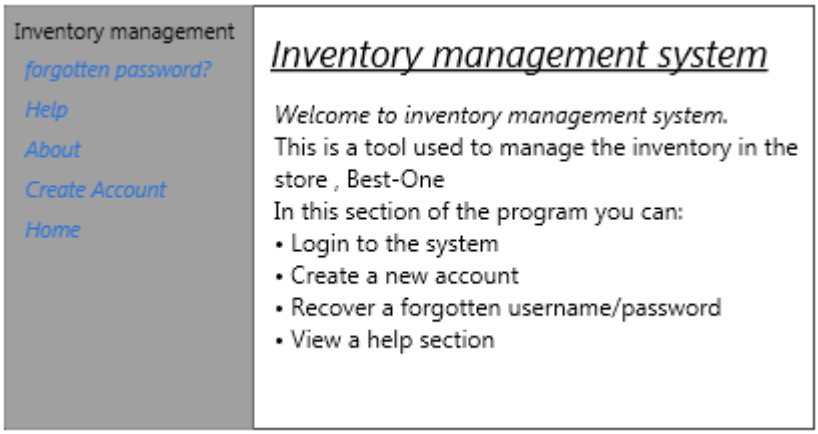
The image below shows the user control being displayed when the program is running:



I then updated the content of grid with the following:

```
<TextBlock x:Name="maintitle_TextBlock" Text="Inventory management system"
HorizontalAlignment="Left" VerticalAlignment="Top" FontStyle="Italic"
FontSize="18.667" TextDecorations="Underline" Margin="8,10,0,0"/>
            <TextBlock x:Name="maininfo_textBlock" HorizontalAlignment="Left"
Margin="10,45,0,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="166"
Width="270">
                <Italic>Welcome to inventory management system.</Italic>
                <LineBreak/>This is a tool used to manage the inventory in the store ,
Best-One
                <LineBreak/>In this section of the program you can:
                <LineBreak/>• Login to the system
                <LineBreak/>• Create a new account
                <LineBreak/>• Recover a forgotten username/password
                <LineBreak/>• View a help section
</TextBlock>
```

This produced the following result:



The main login screen C# file now looks like the following:

```
public partial class Login : Window
{
        /*Global var instances for each usercontrol*/
    LoginMainCtrl loginMainIns = new LoginMainCtrl();
    LoginCreateAccount loginCreateAccountIns = new LoginCreateAccount();
    LoginForgotPassword loginForgotPasswordIns = new LoginForgotPassword();
    LoginAbout loginAboutIns = new LoginAbout();
    LoginHelp loginHelpIns = new LoginHelp();

    public Login()
    {
        InitializeComponent();
        /*Set the contents of the page to the login main page user control*/
        this.mainContentControl.Content = loginMainIns;
    }
    private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
    {
        /*Allows the user to drag the screen around the window*/
        DragMove();
    }

    private void forgotten_password_button_Click(object sender, RoutedEventArgs e)
    {
        /*Set the contents of the page to the forgotten password user control
         When the user clicks on the forgotten password button*/
        this.mainContentControl.Content = loginForgotPasswordIns;
    }

    private void home_button_Click(object sender, RoutedEventArgs e)
    {
        this.mainContentControl.Content = loginMainIns;
    }

    private void create_button_Click(object sender, RoutedEventArgs e)
    {
        this.mainContentControl.Content = loginCreateAccountIns;
    }

    private void about_button_Click(object sender, RoutedEventArgs e)
    {
        this.mainContentControl.Content = loginAboutIns;
    }

    private void help_button_Click(object sender, RoutedEventArgs e)
    {
        this.mainContentControl.Content = loginHelpIns;
    }
}
```

In order to access the user controls folder within code, I used the following import:

```
using InventoryManagementALEVEL.User_controls.Login_controls;
```

## 1.5 Main window user controls

Similarly to the login windows user controls, I will store the main windows user controls in the main window controls, which is stored in the user controls folder.

### *1.5.1 Main page user control*

This user control is displayed when the main window firsts loads.

Its dimensions are as follows:

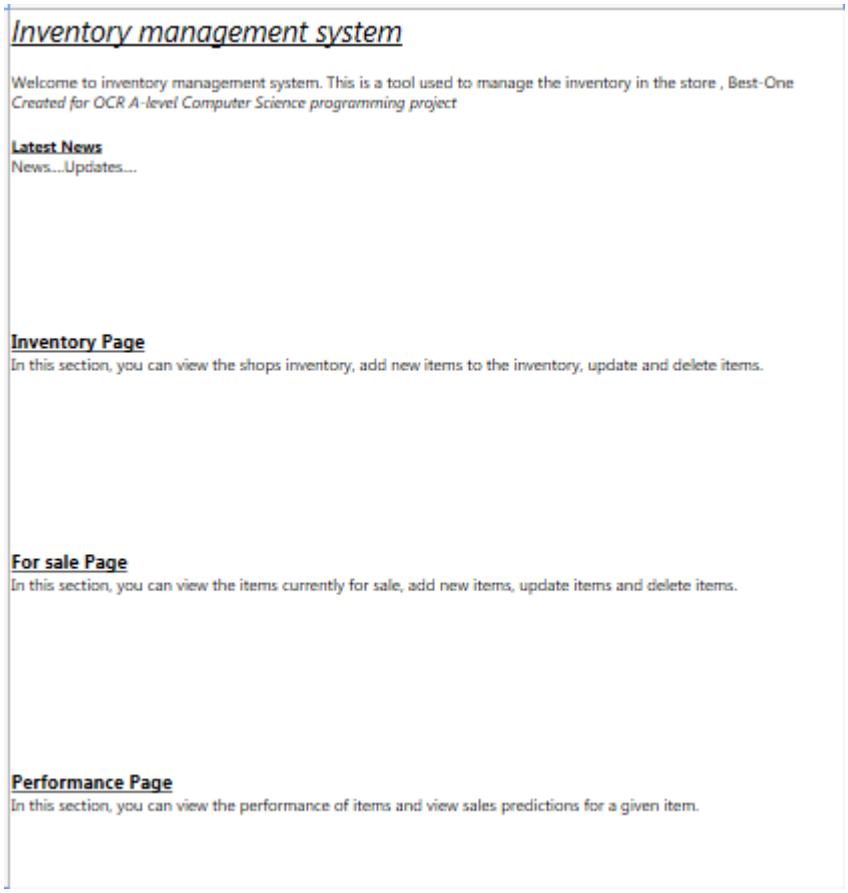
```
d:DesignHeight="658" d:DesignWidth="625">
```

Its contents are as follows:

```
<TextBlock x:Name="maintitle_TextBlock" Text="Inventory management system"
HorizontalAlignment="Left" VerticalAlignment="Top" FontStyle="Italic" FontSize="22"
TextDecorations="Underline" Margin="1,0,0,0"/>
        <TextBlock x:Name="maininfo_textBlock" HorizontalAlignment="Left"
Margin="1,45,-1,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="188"
Width="625">
             <Span FontSize="12">Welcome to inventory management system. This is a
tool used to manage the inventory in the store , Best-One </Span>
            <LineBreak/><Italic>Created for OCR A-level Computer Science programming
project  </Italic>
            <LineBreak/><LineBreak/><Bold><Underline>Latest News</Underline></Bold>
            <LineBreak/>News....Updates....
        </TextBlock>
        <TextBlock x:Name="Inventoryinfo_textBlock" HorizontalAlignment="Left"
Margin="1,238,-1,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="160"
Width="625">
             <Span FontSize="14"><Bold><Underline>Inventory
Page</Underline></Bold></Span>
             <LineBreak/>In this section, you can view the shops inventory, add new
items to the inventory, update and delete items.
        </TextBlock>
        <TextBlock x:Name="ForSaleinfo_textBlock" HorizontalAlignment="Left"
Margin="1,403,-1,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="160"
Width="625">
             <Span FontSize="14"><Bold><Underline>For sale
Page</Underline></Bold></Span>
             <LineBreak/>In this section, you can view the items currently for sale,
add new items, update items and delete items.
        </TextBlock>
        <TextBlock x:Name="Performance_textBlock" HorizontalAlignment="Left"
Margin="1,568,-1,0" TextWrapping="Wrap"  VerticalAlignment="Top" Height="90"
Width="625">
             <Span FontSize="14"><Bold><Underline>Performance
Page</Underline></Bold></Span>
             <LineBreak/>In this section, you can view the performance of items and
view sales predictions for a given item.
        </TextBlock>
```

As shown above, the main page user control consists of a series of text blocks used to display information. As of now, the main page will not contain anything else.

The following is shown as a result of the XAML just created.




In order to have this user control be displayed on the main window, in the main window C# file I created a global variable of an instance of this user control:

```
public partial class MainWindow : Window
{
    MainPage mainPageIns = new MainPage();
```

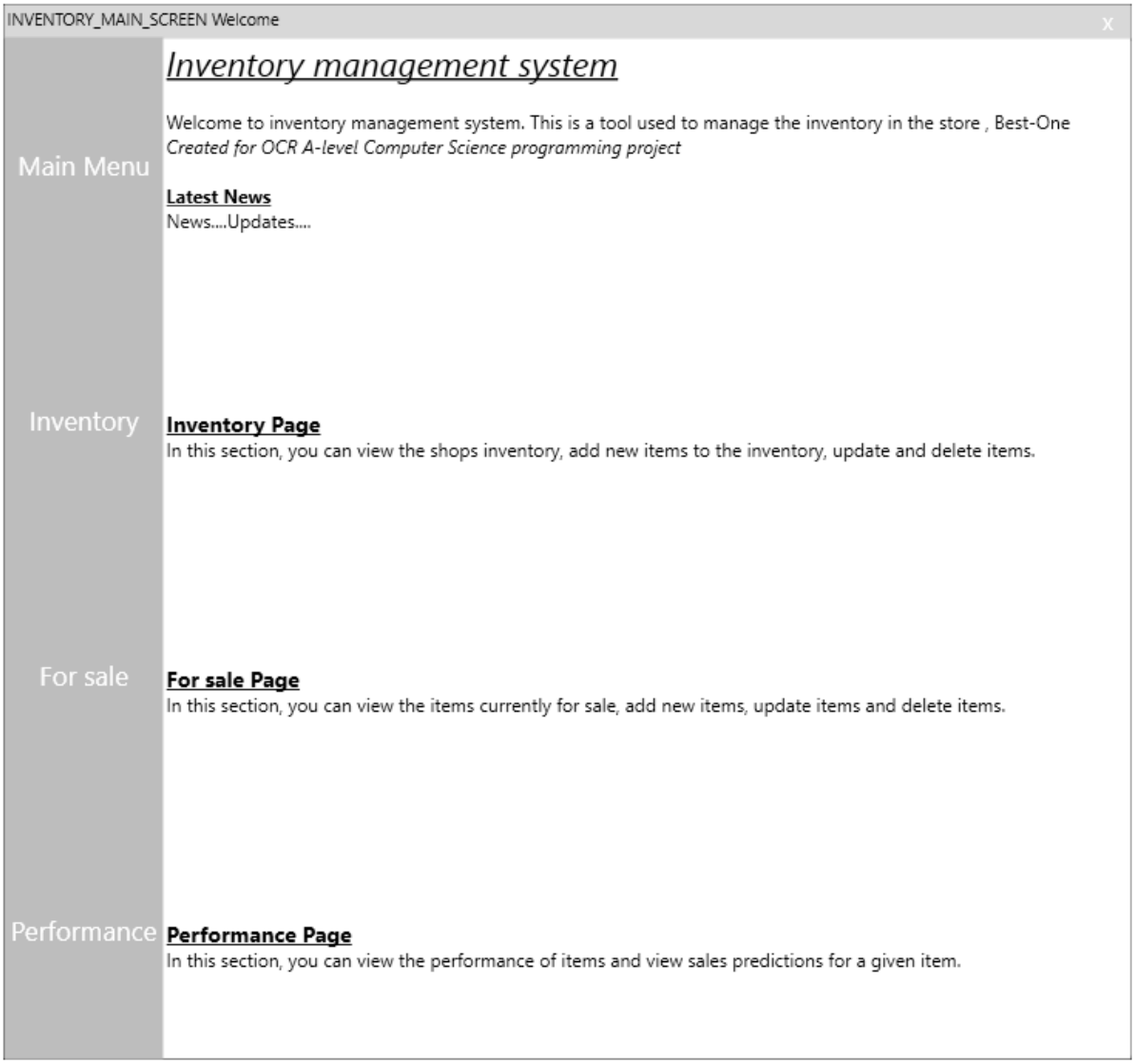
To have this user control show when the window loads I have added the following code to the main windows constructor:

```
public MainWindow(string _username)
{
    InitializeComponent();
    //Set the content control to the main page instance when the page loads
    this.mainContentControl.Content = mainPageIns;
```

When the user clicks on the main menu button on the main window, the user control should be shown, therefor I have added the following code to the button clicked event method for the main menu button:

```
private void Main_Menu_button_Click(object sender, RoutedEventArgs e)
{
    //set the content control the main page instance
    this.mainContentControl.Content = mainPageIns;
}
```

The following is an image of the program running showing the main page:




*1.5.2 Main Inventory user control*

This user control is the page which will display the main inventory table to the screen and allow the user to manipulate this table. For now I will be creating the UI for this page. It will have no functionality.

This page will have the same dimensions as the main page user control. It will have two buttons at the top of page, an item button and delete selected item(s) button at the top of the page underneath a title and subtitle. Underneath this will be a table displaying the items name, category and quantity.

The XAML to achieve this is the following

```
<TextBlock x:Name="maintitle_TextBlock" Text="Inventory management system"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" FontStyle="Italic"
FontSize="22" TextDecorations="Underline"  />
                    <TextBlock x:Name="subtitle_TextBlock" Text="Inventory table"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" FontStyle="Italic"
FontSize="19" TextDecorations="Underline"  />
                    <Button x:Name="add_button" Content="Add new item"
Margin="2,3,0,0" VerticalAlignment="Top" HorizontalAlignment="Left"
BorderBrush="Transparent" BorderThickness="0" Click="add_button_Click"  Height="18"
Width="76"/>
                    <Button x:Name="delete_button" Content="Delete selected item(s)"
Margin="2,3,0,0" VerticalAlignment="Top" HorizontalAlignment="Left"
BorderBrush="Transparent" BorderThickness="0" Click="delete_button_Click" Height="18"
Width="122"/>
```

To setup the table on the user control I used a data grid control:

```
<DataGrid x:Name="inventory_dataGrid"   HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
          AutoGenerateColumns="False" CanUserAddRows="False" HeadersVisibility="Column" CanUserSortColumns="False"
          ScrollViewer.CanContentScroll="True" ScrollViewer.VerticalScrollBarVisibility="Auto" ScrollViewer.HorizontalScrollBarVisibility="Auto"
```

This sets up an empty read only data grid that is equipped with a scrollbar, if the number of items displayed in the table exceeds the height of the page.

To define the columns of the table, I setup data grid columns:

```
<DataGrid.Columns>
    <DataGridTextColumn
        IsReadOnly="False"
        Header="Item Name"
        Binding="{Binding Path=ItemName}"
        Width="*"
        />
    <DataGridTextColumn
        IsReadOnly="True"
        Header="Category"
        Binding="{Binding Path=CategoryName}"
        Width="*"/>
    <DataGridTextColumn
        IsReadOnly="False"
        Header="Quantity"
        Binding="{Binding Path=Quantity}"
        Width="*"
        />
```

The Binding Path refers to the data to bind to the data grid column. This will be setup to link to the data in a database later.

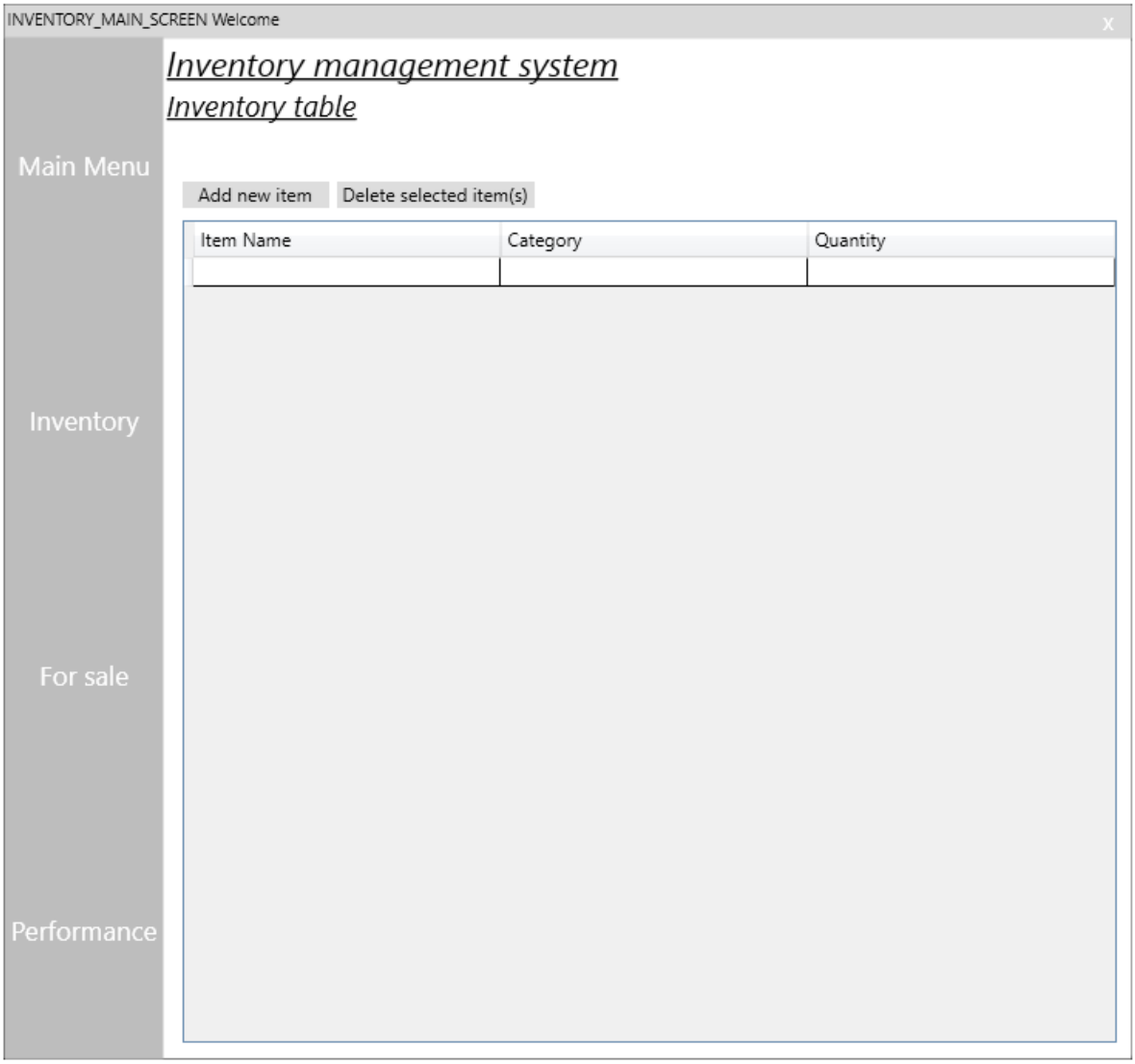
As with the previous user control, I have created a global instance of the user control in the main window C# file.

```
MainPage mainPageIns = new MainPage();
MainInventory mainInventoryIns = new MainInventory();
```

To have it be displayed when the user clicks on the inventory button I have setup the inventory button to set the Content control to the inventory instance.

```
private void Inventory_button_Click(object sender, RoutedEventArgs e)
{
    this.mainContentControl.Content = mainInventoryIns;
}
```

The following is an image of the program running showing the main page:




I have decided to have every button have no border. The reason for this is because I prefer the appearance of a borderless button and its look is more consistent across different versions of Windows containing different .NET versions.

```
BorderBrush="Transparent" BorderThickness="0"
```

The XAML above is required to achieve this effect. I could create a XAML style at a later date, for all buttons to save having to type this for every button.

*1.5.3 Main For sale page user control*

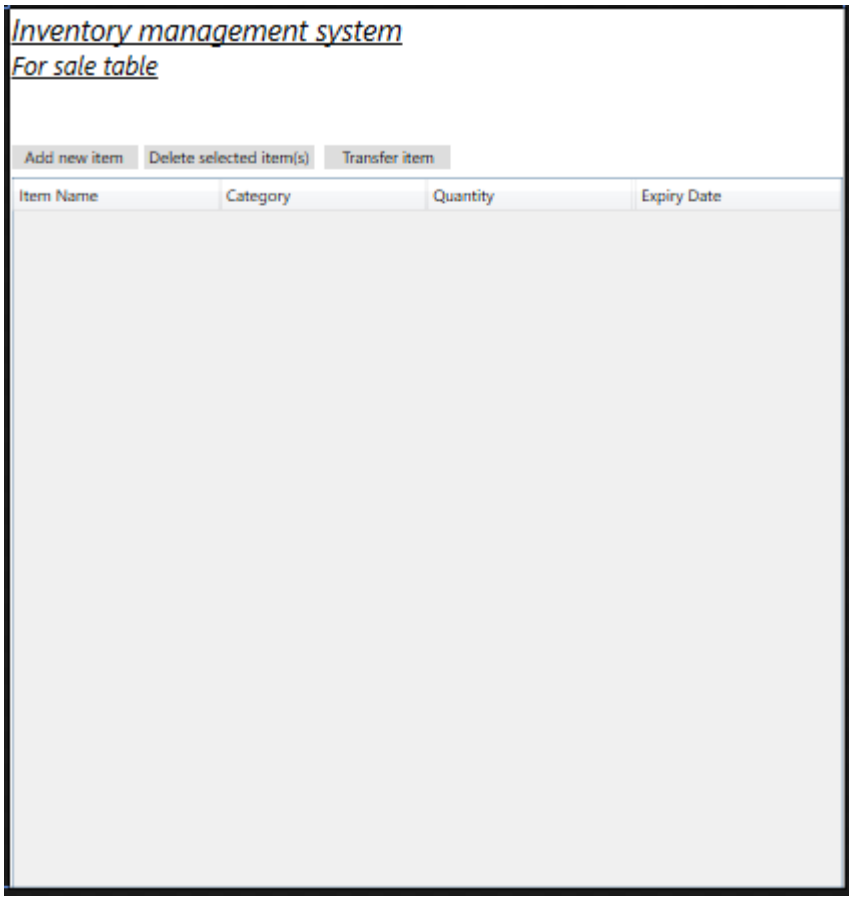
This user control will feature a similar look to the inventory user control.

In addition to the 'add and delete' buttons it will also have a transfer button. The data grid will also have a sell by date column.

The data grid columns section of the XAML is shown below

```
<DataGrid.Columns>
    <DataGridTextColumn
        IsReadOnly="False"
        Header="Item Name"
        Binding="{Binding Path=ItemName}"
        Width="*"
        />
    <DataGridTextColumn
        IsReadOnly="True"
        Header="Category"
        Binding="{Binding Path=CategoryName}"
        Width="*"/>
    <DataGridTextColumn
        IsReadOnly="False"
        Header="Quantity"
        Binding="{Binding Path=Quantity}"
        Width="*"
        />
    <DataGridTextColumn
        IsReadOnly="False"
        Header="Expiry Date"
        Binding="{Binding Path=ExpiryDate}"
        Width="*"

        />
```

The image bellow shows the for sale user control in the designer view:




*1.5.4 Performance user control*

This user control will be created at a later date.

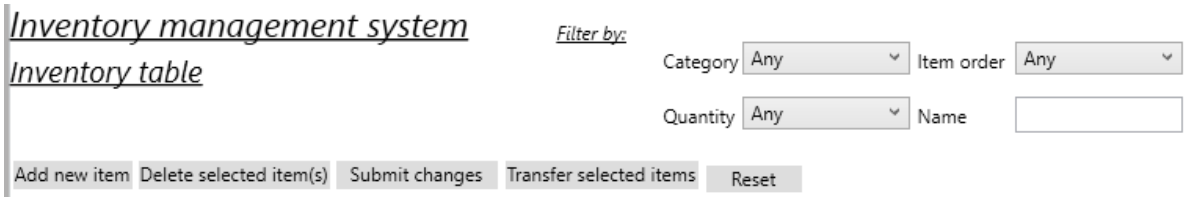
# 2 Interface changes

After completing the initial interface. I have decided to make changes to the interface. Firstly, I will add a filter section to the inventory and for sale user controls as well as more buttons. In addition, I have noticed that the interface is too small, therefor I will make the interface resizable, with facilities to minimise, maximise the interface.  I will also be making changes to the appearance of the user controls and windows I have created as I feel the current appearance is not engaging or professional.

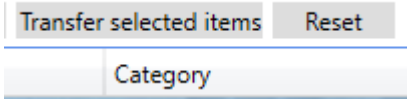
## 2.1 Adding the filter section to the inventory user control

This task will only involve adding more XAML to the user control.

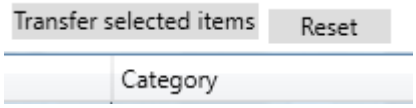
The following image shows the top of the inventory user control:




There is an issue here, the reset button is not In line with the other buttons. In the designer view, the buttons are aligned but when the program is running the buttons do not align.



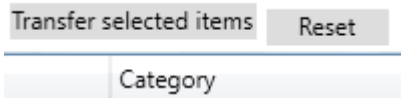

- The buttons in design view



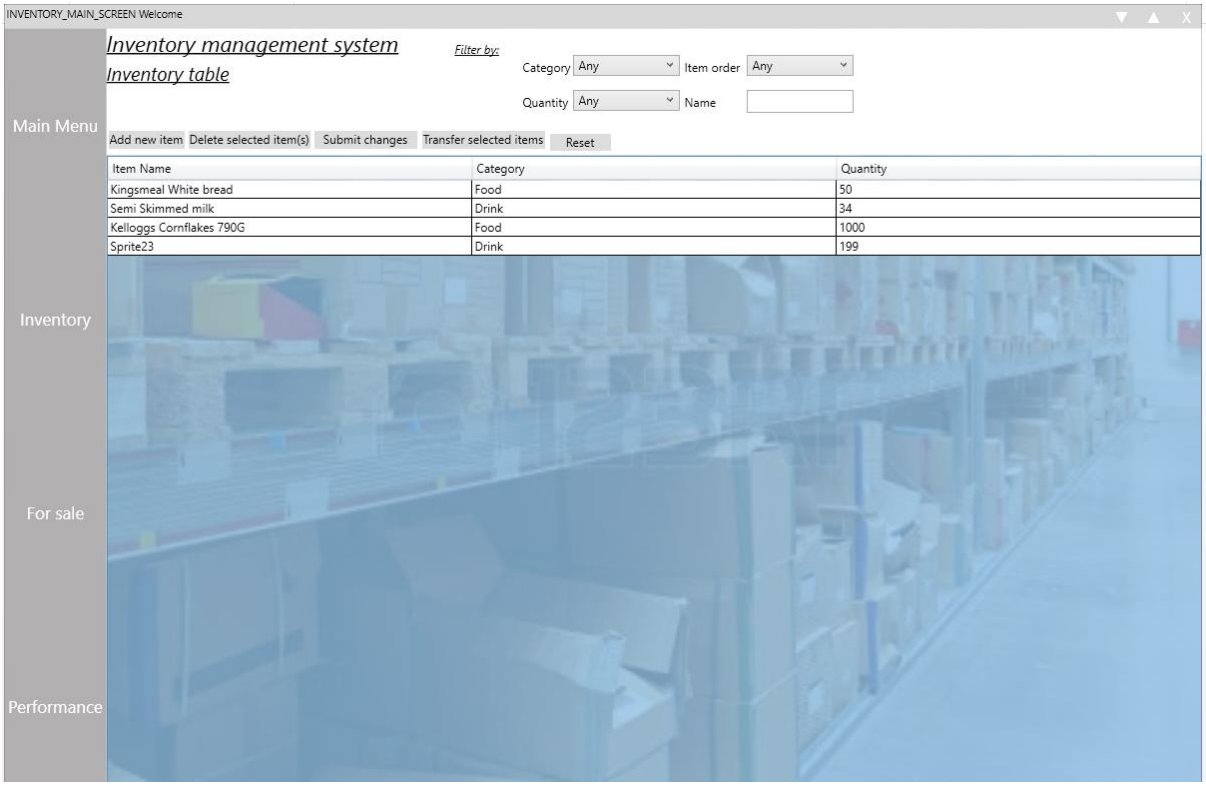

- The buttons when the program is running

To resolve this I have set the height property of the reset button to 18, which is identical to the height property of the Transfer selected items button.

```
Width="61" Height="18"
```



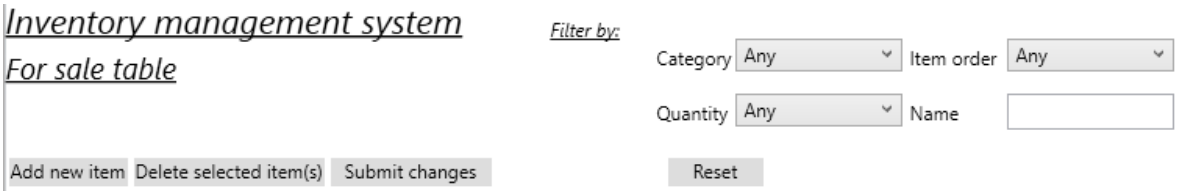

Testing this fix has shown to make the problem worse so I have reverted the button back to its original setting.




## 2.2 Adding the filter section to the for sale user control

This task involved copying the XAML created in the previous task into the for sale control.

The following image shows the top of the for sale user control:




## 2.3 Adding a maximise and minimise button

To achieve this, I will standard buttons with the style used for the close button. The contents of these buttons will be: ▲for maximise and ▼for minimise.

```
<Button x:Name="maximize_button" Content="▲"    BorderBrush="Transparent"   Style="{StaticResource CloseButton}" BorderThickness="0"
<Button x:Name="minimize_button" Content="▼"    BorderBrush="Transparent"   Style="{StaticResource CloseButton}" BorderThickness="0"
```

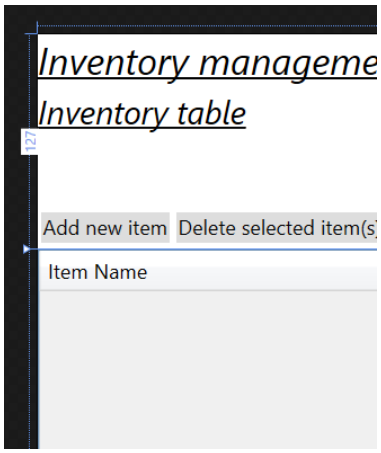
The XAML above creates the buttons, with the close button style. I have moved the buttons around to make them appear in the top right hand corner of the window next to the close button.

Below is an image showing how they look when the program is running:

The logic for these buttons will created after the window is able to resize itself properly (this is done in 2.5)

## 2.4 Adding a background to Inventory and for sale table

Firstly I have added a grid row definition to separate the data grid and the top half of the control. I did this by clicking on the edge of the grid on the side of the place where I wanted to place the row in the designer view, as shown below:




This generates the following XAML in the grid:

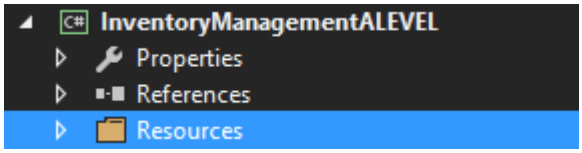
```
<Grid.RowDefinitions>
    <RowDefinition Height="127"/>
    <RowDefinition/>
</Grid.RowDefinitions>
```

Visual studio also generates assigns a 'Grid.Row' property for each control in the interface. For example, the XAML below is used with the datagrid to have it placed in the bottom half of the grid, or row 1 (counting from row 0).

```
Grid.Row="1">
```

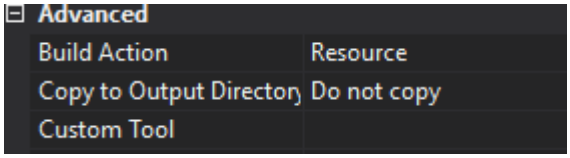
I will now set the following image to the background of the table in the user control:

To do this, I created a resources folder in my project in visual studio:




I then right clicked on the folder and clicked add existing item and navigated to the image. The image is now added to the project and ready to use.

In the properties box for this image, visual studio has set the following settings by default:



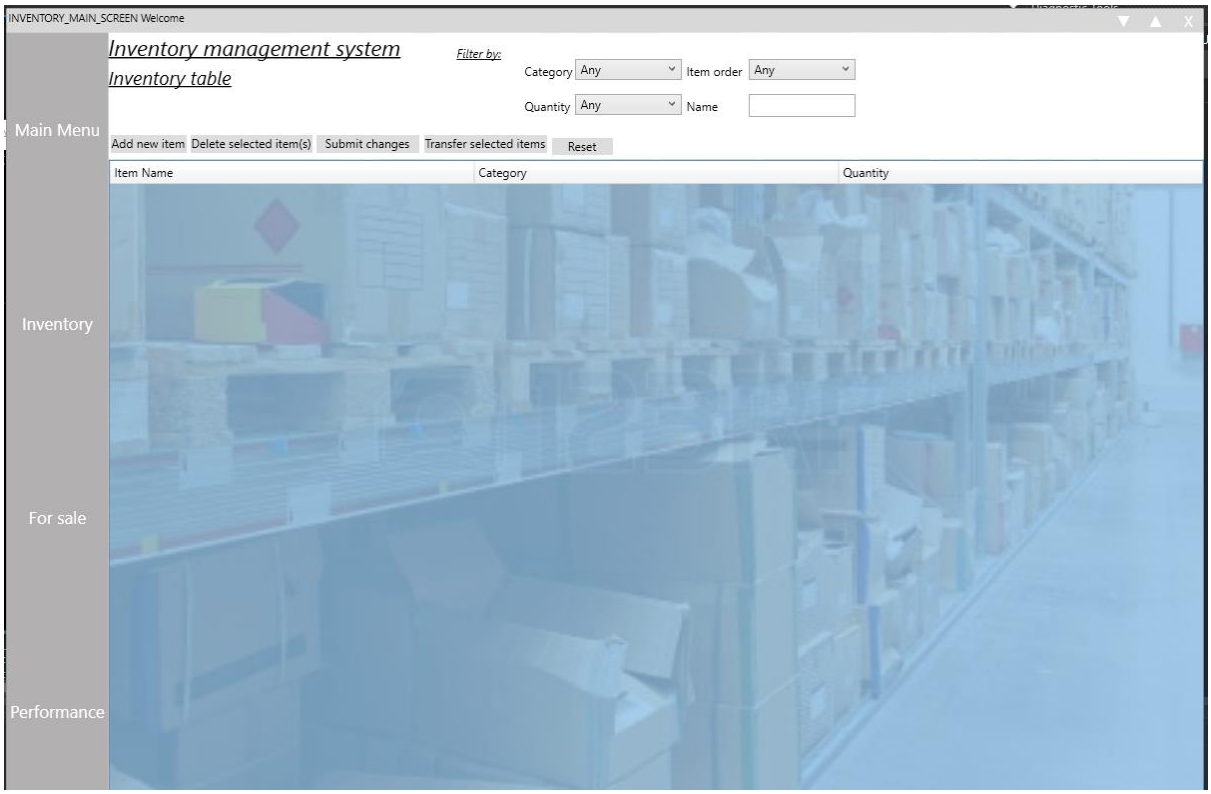

By default the build action is set to Resource, so the image is packaged into the executable file and does have to be included in the release folder.

To set the background of the datagrid to the image, I navigated the properties settings of the datagrid, selected brush, image brush and selected the image from the Image source drop down box.

This generated the following XAML:

```
<DataGrid.Background>
    <ImageBrush ImageSource="/InventoryManagementALEVEL;component/Resources/MainPageBackImageBlueOver.jpg" Stretch="UniformToFill"/>
</DataGrid.Background>
```

The image below shows the program running on the inventory page:




To achieve the same effect with the for sale table, the exact same steps were taken.

## 2.5 Making the interface resizable

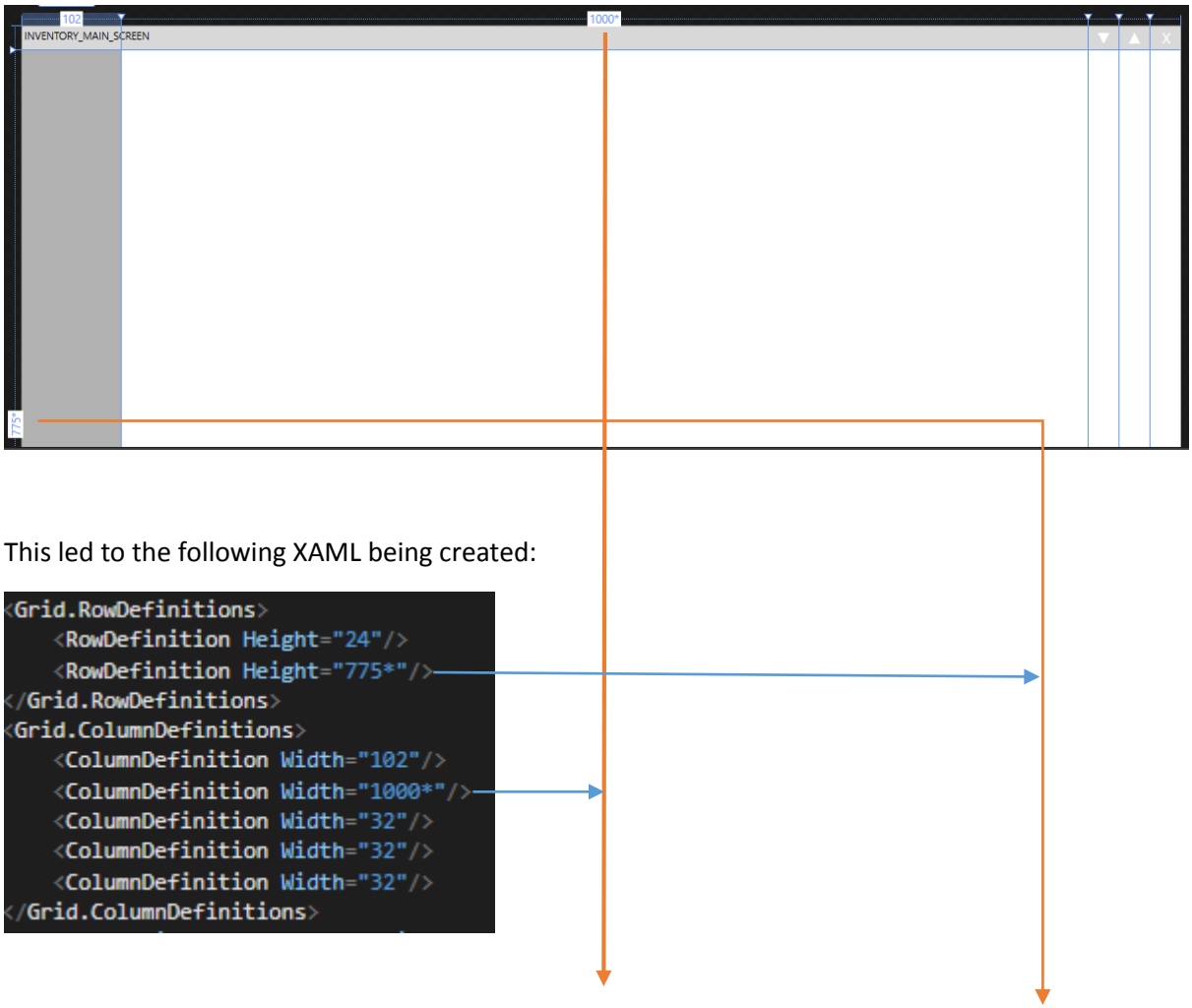
In order to have the contents of each user control resize proportionally with the main window, I need to place each control in the interface into a grid (which it currently is by default) and setup grid columns and rows. I can then place the controls into these rows and columns and set their height and width to the size of the space of the cell in the grid they are in. This way, when the window resizes, so does the grids columns and rows, meaning the controls inside these grids will also resize with the window.

To start with I have changed the default size of the main window to 800x1200. To allow the page to be resized I have added a resize grip to the window and specified a minimum size the page can shrunk to, the following XAML achieved this:

```
MinHeight="683" MinWidth="860" ResizeMode="CanResizeWithGrip">
```

The sidebar buttons I have created previously were contained in a stack panel. The stack panel in WPF does resize when its container resizes, so I will have to place the side bar buttons in a grid panel, which is able to resize.

Using the designer, I have created rows and columns on the main grid:




This led to the following XAML being created:

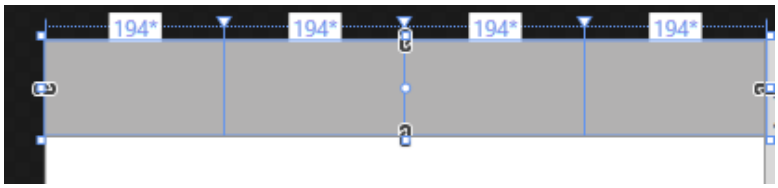
```
<Grid.RowDefinitions>
    <RowDefinition Height="24"/>
    <RowDefinition Height="775*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="102"/>
    <ColumnDefinition Width="1000*"/>
    <ColumnDefinition Width="32"/>
    <ColumnDefinition Width="32"/>
    <ColumnDefinition Width="32"/>
</Grid.ColumnDefinitions>
```

Row and column definitions with a * in its assignment are variable size. Those without are fixed size. Those column and row definitions are shown above by the orange arrows. When the window resizes

these rows and columns will expand and shrink whilst the others won't, meaning their child controls should resize proportionally.

As with before, visual studio has replaced the controls on the window into their correct row and column.

The grey sidebar in the image above is a nested grid I have created




The nested grids XAML is shown below. It is placed in the first row its parent grid and set to the same background colour as the previous stack panel.

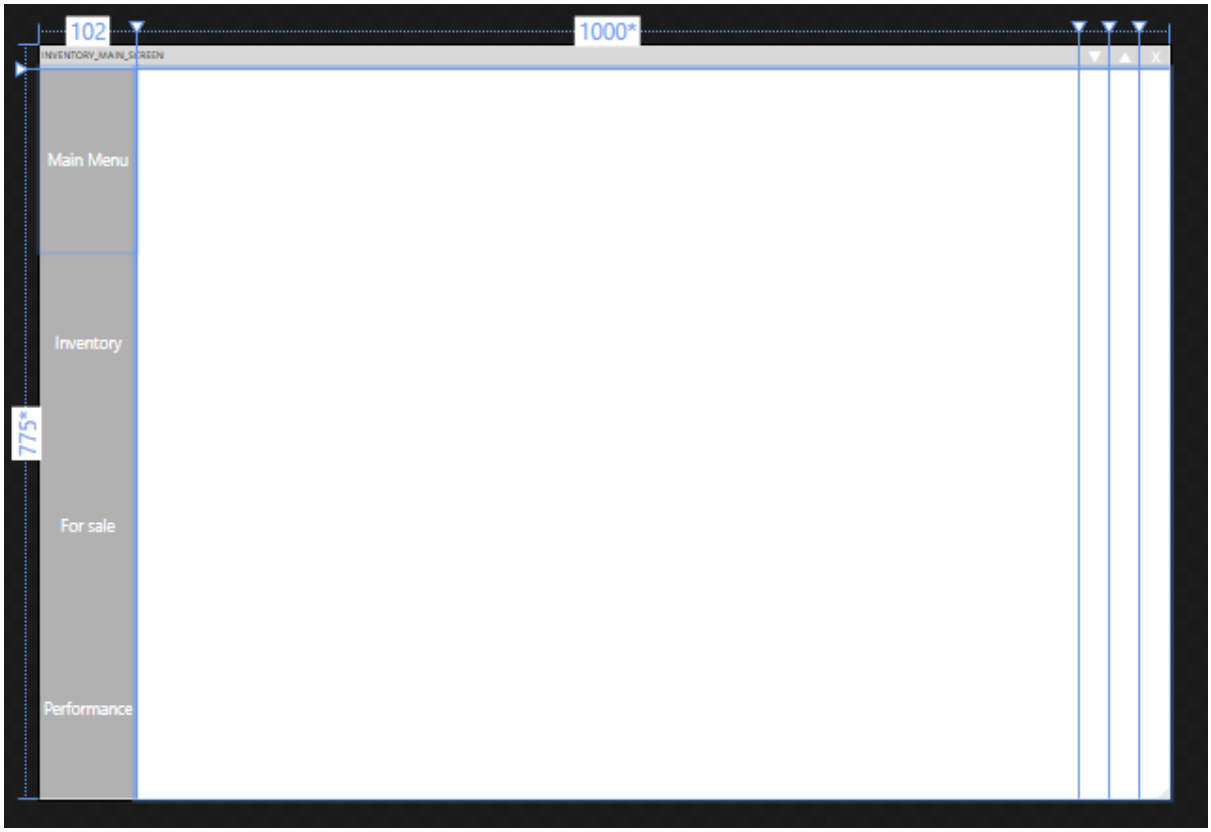
```
    <Grid HorizontalAlignment="Stretch" Grid.Row="1" Background="#FFB2B1B1" >
        <Grid.RowDefinitions>
            <RowDefinition Height="194*"/>
            <RowDefinition Height="194*"/>
            <RowDefinition Height="194*"/>
            <RowDefinition Height="194*"/>
        </Grid.RowDefinitions>
    </Grid>
</Grid>
```

Each row definition of this grid is of the same size and is set to stretch and shrink as needed.

```
    </Grid.RowDefinitions>
    <Button x:Name="Main_Menu_button"  Content="Main Menu"  VerticalAlignment="Stretch"  BorderThickness="0" FontSize="17"  Style="{StaticResource SideBarButtons}" Click="Main_Menu_button_Click" Grid.Row="0" />
    <Button x:Name="Inventory_button" Content="Inventory"  VerticalAlignment="Stretch"  BorderThickness="0" FontSize="17"  Style="{StaticResource SideBarButtons}" Click="Inventory_button_Click" Grid.Row="1"/>
    <Button x:Name="For_Sale_button" Content="For sale"   VerticalAlignment="Stretch"  BorderThickness="0" FontSize="17"  Style="{StaticResource SideBarButtons}" Click="For_Sale_button_Click" Grid.Row="2" />
    <Button x:Name="Performance_button" Content="Performance"  VerticalAlignment="Stretch" BorderThickness="0" FontSize="17"  Style="{StaticResource SideBarButtons}" Click="Performance_button_Click"  Grid.Row="3" />
</Grid>
```

The XAML above adds the buttons from the stack panel into this new grid. Each button has a grid row property in ascending order from 0 to 3.

The result of this is shown below: in the designer view.




The content control fits into the white space in the image above. It spans across the 3 columns (which are holding the 'minimise, maximise and close' buttons) which in row 1. Now any user control should expand and shrink inside this space as the window gets larger and smaller. However in order for this to happen, I have to sort the contents of each user control into columns and rows.

In the main page user control I have changed the background to the image used in the inventory page datagrid. This was achieved by adding the following XAML to the main grid

```
<Grid.Background>
    <ImageBrush ImageSource="/InventoryManagementALEVEL;component/Resources/MainPageBackImageBlueOver.jpg"/>
</Grid.Background>
```

Since the default size of the main window has changed, I have updated the dimensions of the user control:

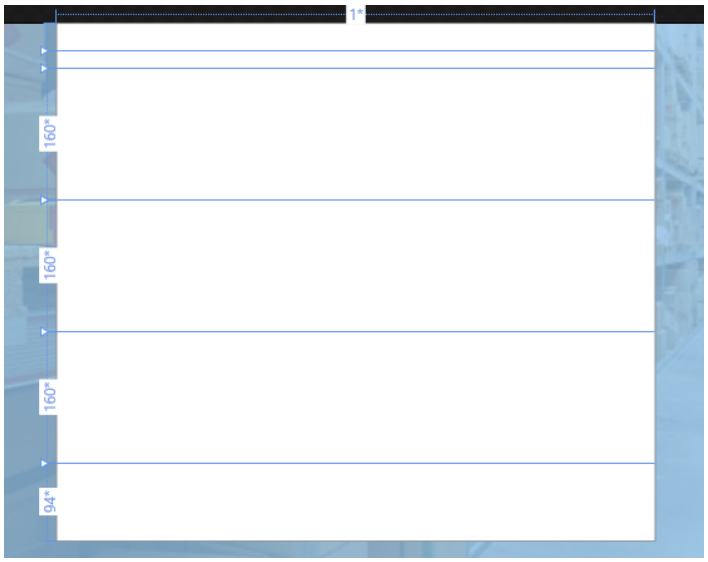
```
d:DesignHeight="658" d:DesignWidth="1080">
```

I have used a nested grid to create a white space in the centre of the page.

```
<Grid Margin="182,0,182,40" Background="White" >
```

The image below is of the user control in the designer view:

Similarly to the sidebar on the main window, the contents of the main page was previously in a stack panel. I will need to change this to a grid, in order for the user control to expand and shrink with the window.




```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="33*"/>
    <RowDefinition Height="21*"/>
    <RowDefinition Height="160*"/>
    <RowDefinition Height="160*"/>
    <RowDefinition Height="160*"/>
    <RowDefinition Height="94*"/>
</Grid.RowDefinitions>
```

The images above show the rows and columns I have created, that have been made to resemble the layout of the stack panel I previously created.

In order to have the old text be displayed in the grid, I will have to assign their row and column manually.

I have also decided to have all the text in the centre of the grid.

To add the title of the main page to the grid I used the following XAML:

```
<TextBlock x:Name="maintitle_TextBlock" TextAlignment="Center"  HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
           Text="Inventory management system" FontStyle="Italic" FontSize="22" TextDecorations="Underline" Grid.Row="0"/>
```

`Grid.Row="0"`, Places the title in the first row of the grid, at the top. `TextAlignment="Center"` causes the text to be placed in the centre of the row.

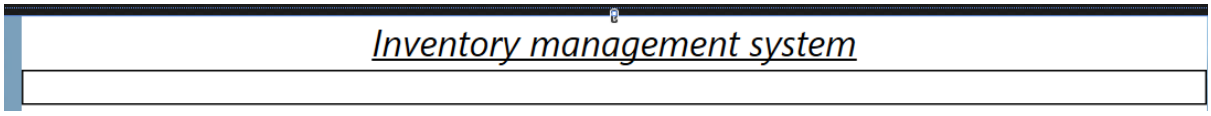
To add the rest of the text I added the text blocks in the same way but changed the row for it was placed in.

The gap in-between the first and third rows are to be used as a tool bar. For now, this tool bar will contain a button will launch the log window when it's developed. To create the toolbar I will create a border around the row and add buttons to the row.

The following XAML creates a border around the specific row:

```
<Border BorderBrush="Black" BorderThickness="1" Grid.Row="1"/>
```

This created the following effect, as shown in the designer view:

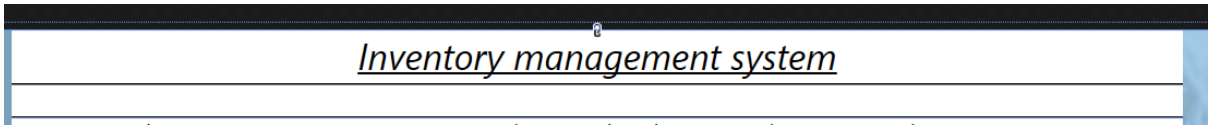



This is not the effect I wanted as the edges of the border are showing, which doesn't fit with the borderless white grid.

To fix this I have changed the border thickness value. It now looks like this:

```
BorderThickness="0,1"
```

This creates a border with thickness 1 on the horizontal borders and a border with thickness for the vertical borders, creating the effect I wanted:

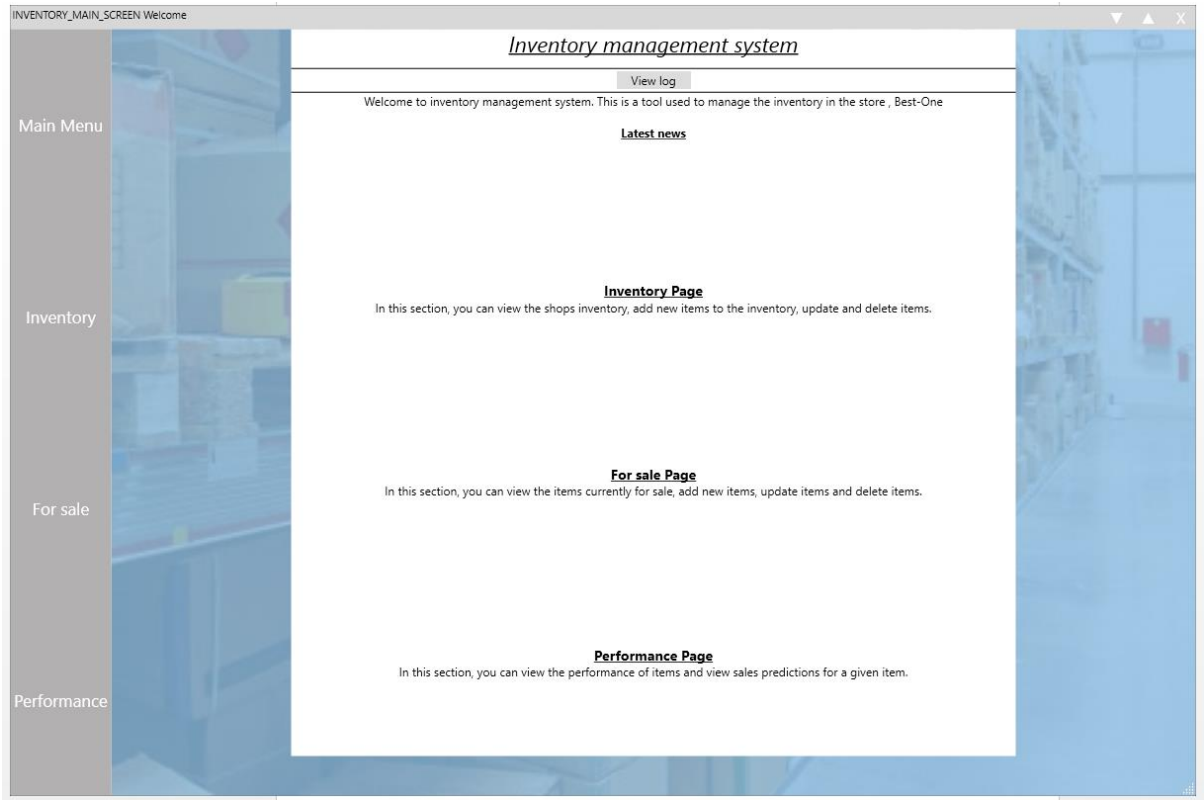



To create a button to go inside this section I used the following XAML:

```
<Button x:Name="log_button" Content="View log" HorizontalAlignment="Center" Grid.Row="1" BorderBrush="Transparent"
        BorderThickness="0" VerticalAlignment="Center" Width="75"/>
```

This creates a button inside the middle of the border.

The main window and main screen now looks the following when the programming is running:

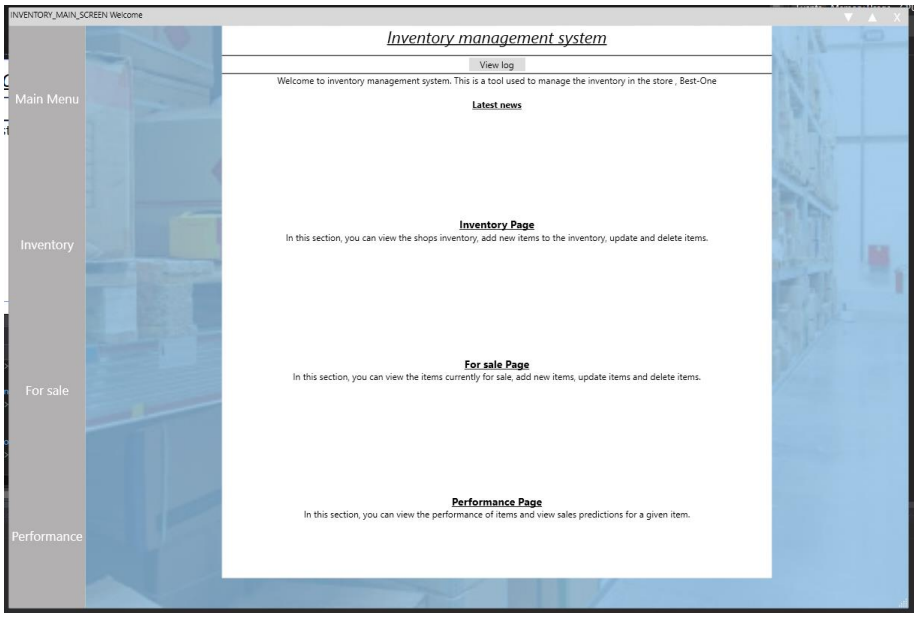



Adding the following XAML to each control in the grid makes the control use all the available space in the grid cell it has been placed in:

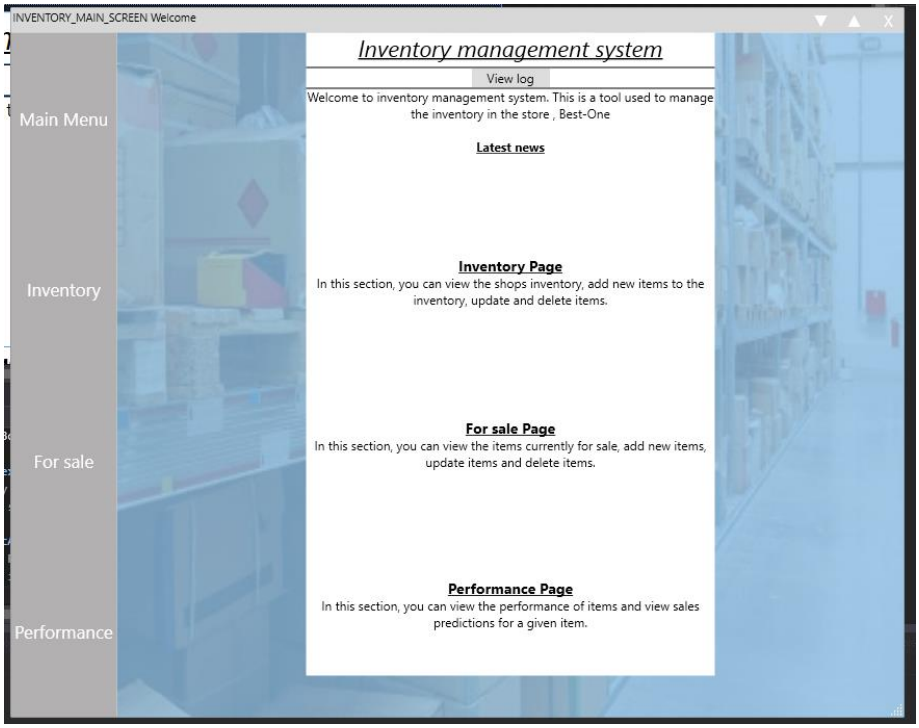
```
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
```

This means that the control will resize dynamically as the window changes size.

The following image shows the program running with the main page showing:




When the window is resized to its minimal size, it looks like the following:




Now, the program resizes, the contents of the main page also resize with the window. I will now adjust the inventory and for sale pages to achieve the same effect.

In its current state, the inventory page and the for sale page contain a single grid divided into two. The top half contains the buttons and text, whilst the bottom half is just the data grid.

Since the default size of the main window has changed, I have updated the dimensions of the user control:

Making the data grid expand and shrink with the window will be easy, I just need to set the horizontal and vertical alignment property to stretch.

```
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
```

In the designer view, this causes the data grid to fill up the entire grid cell:

In the top half of the grid I have setup a nested grid, with columns and rows defined as follows:

Unlike the data grid, the buttons in this grid shouldn't change its dimensions as its cell gets larger or smaller. To setup this behaviour I added the following XAML to each button, as well as the Title and Sub title text block controls:

```
VerticalAlignment="Top" HorizontalAlignment="Left"
```

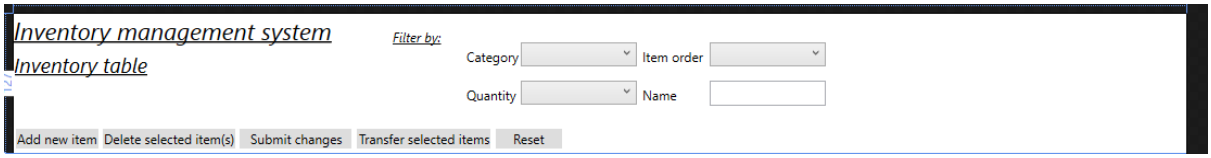
I also set the background of this grid to White.

To add the rest of the filter section back I have setup another grid inside this nested grid and moved it to right until it was in the centre of the white section.

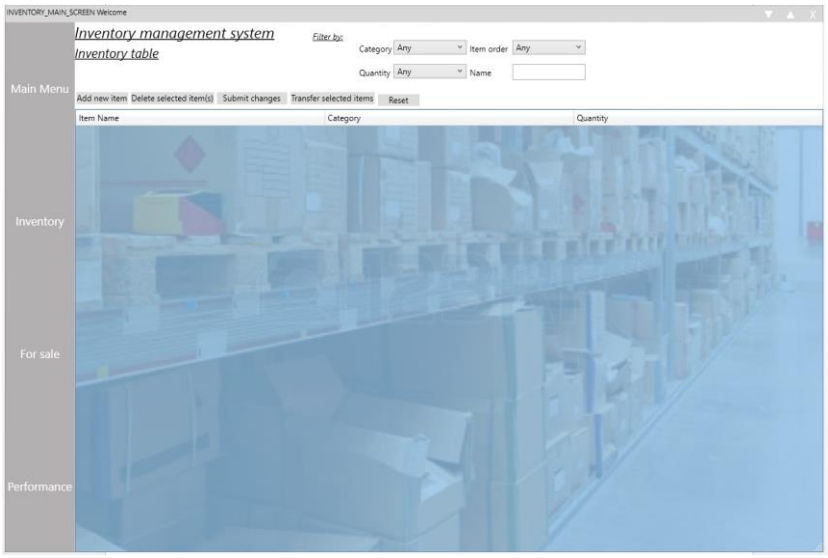
```
<Grid Margin="336,0,0,0" VerticalAlignment="Stretch" HorizontalAlignment="Stretch">
```

The Margin definition allows the grid to move to the right by 336. I have added the XAML for the filter section back into this new grid without setting up column or row definitions as I do not want this section to move when resizing the window.

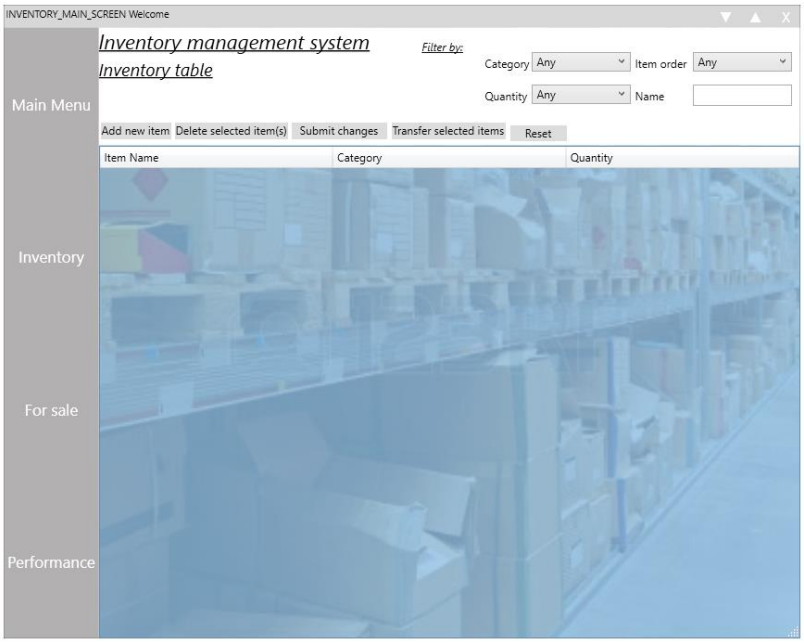
Due to the increased amount of space on the interface, I have decided to change the appearance of the filter section to make use of this free space. This was done by moving the controls around the grid using the designer. The new upper section of this page now looks as follows in the designer view:




When the program is running, showing the inventory page, in its default size, it looks as follows:



When the window is resized to its smallest state, it looks as follows:




As shown, the inventory page resizes as needed.

The same process was carried out on the' for sale page' and the same effect was achieved.

No work was done on the performance page as it currently doesn't exist.

To make the window minimise and maximise when clicking the buttons created in 2.3, in the mouse click event handlers for these buttons I added the following code:

```
1 reference
private void maximize_button_Click(object sender, RoutedEventArgs e)
{
    this.WindowState = WindowState.Maximized;

}

1 reference
private void minimize_button_Click(object sender, RoutedEventArgs e)
{
    this.WindowState = WindowState.Minimized;
}
```

This code worked but created two issues; when maximised there was no way to minimise the window and once maximised the window would cover up the windows task bar completely.

The issues are shown below in the image:

The windows task bar is not visible in this full screen screenshot. Also, once in this state there is no way to restore the window back to its original state.

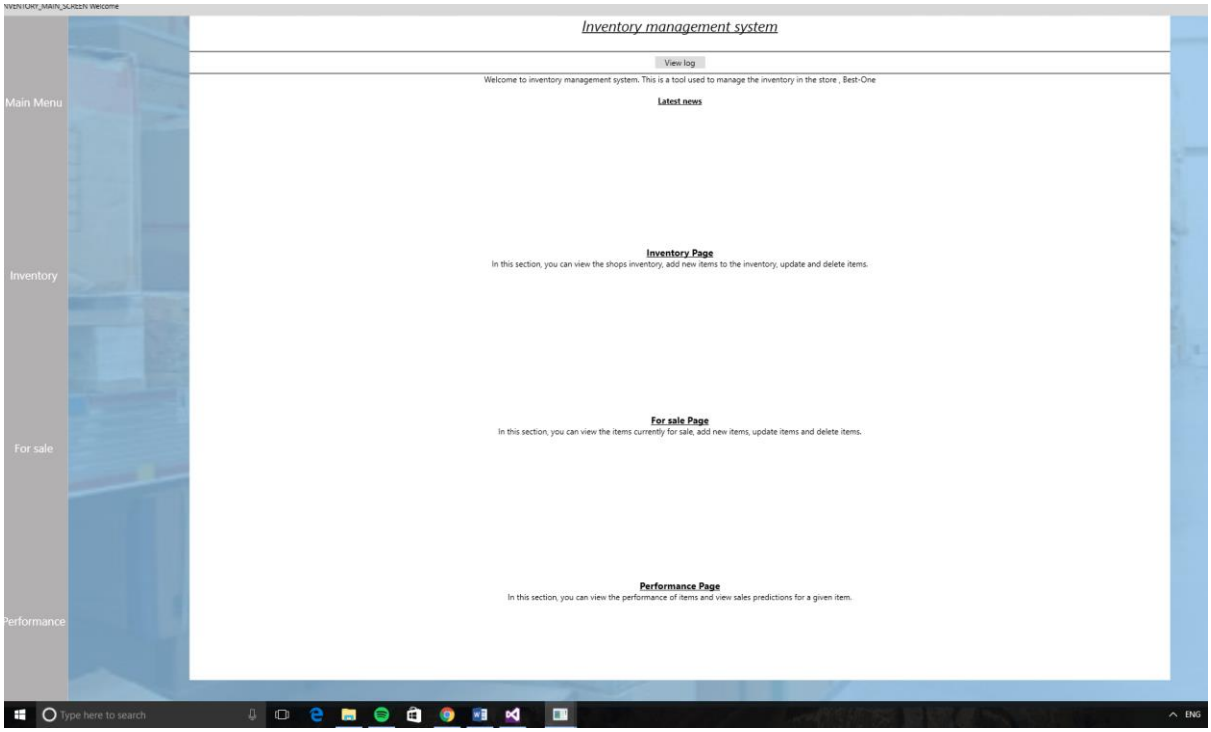
To create a way to minimise the window, I have decided to allow this action to happen when the user clicks on the top bar. To do this I created a left mouse click event on the top bar.

```
MouseLeftButtonDown="topbar_Rectangle_MouseLeftButtonDown"/>
```

Fixing the window task bar disappearing bug just involved adding the following lines of code to the windows constructor:

```
public MainWindow()
{
    InitializeComponent();
    //Set the content control to the main page instance when the page loads
    this.mainContentControl.Content = mainPageIns;
    //sets the maximum width and heigh to the size of the users screen when maximising.
    this.MaxHeight = SystemParameters.MaximizedPrimaryScreenHeight;
    this.MaxWidth = SystemParameters.MaximizedPrimaryScreenWidth;
```

This full screen screenshot shows how the program looks when maximised:



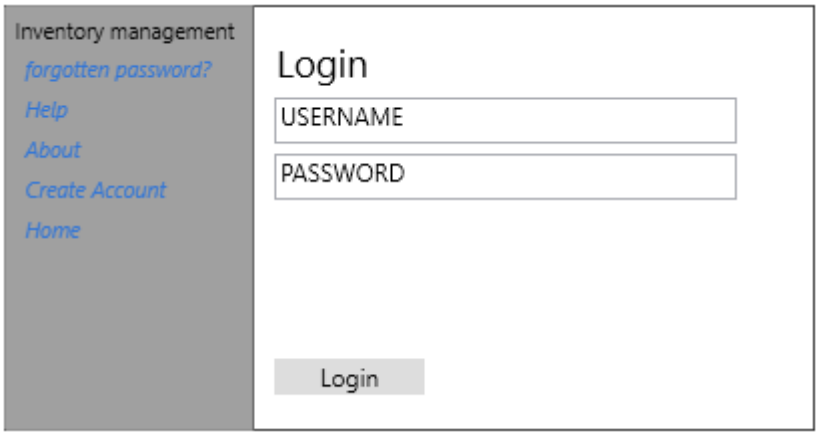

The task bar is now visible clicking on the top bar, the window is resized back to its previous dimensions.

## 3 Interface prototype

Now that the main interface is complete, a prototype version of the solution can be showed.

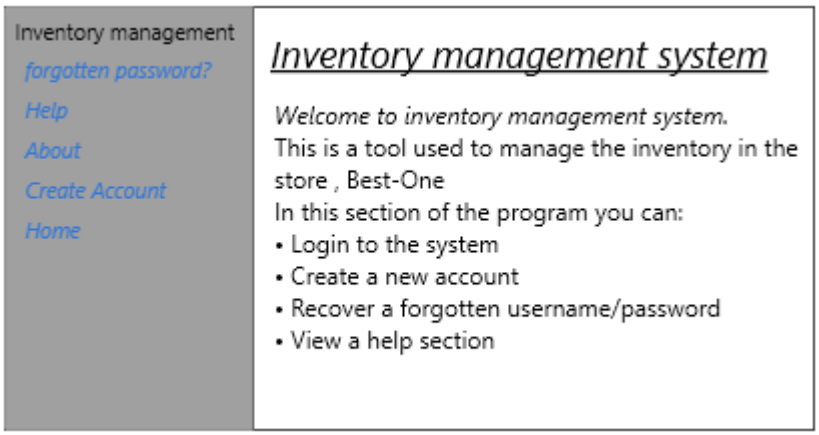
The splash screen is showed when the program is launched:

After a short amount of time the login screen is shown:



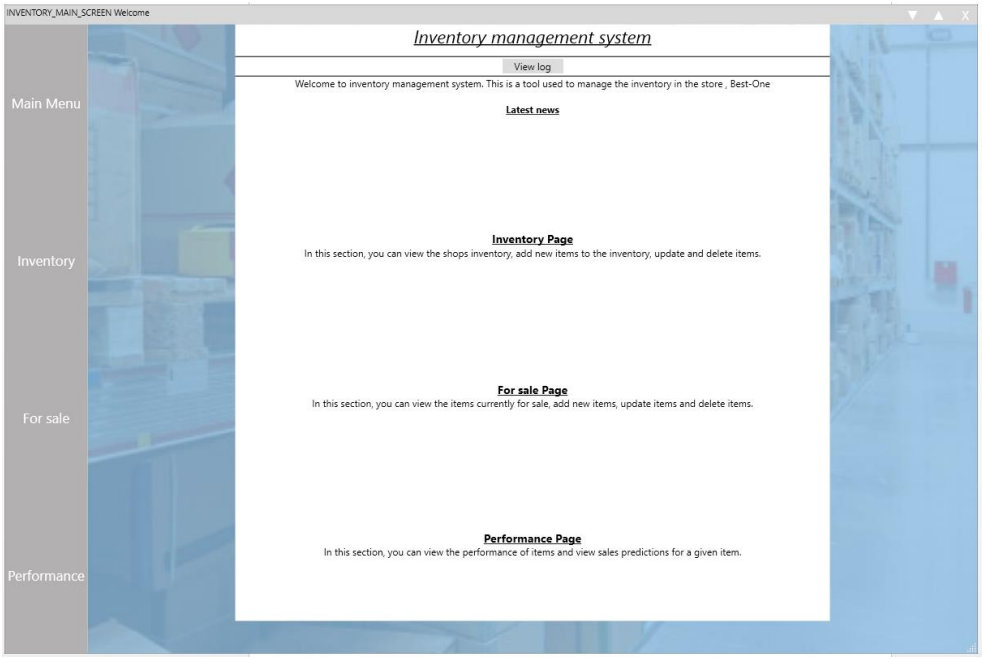

Cycling through the options on the left work correctly, each showing their correct information.

The login screen showing the about screen:

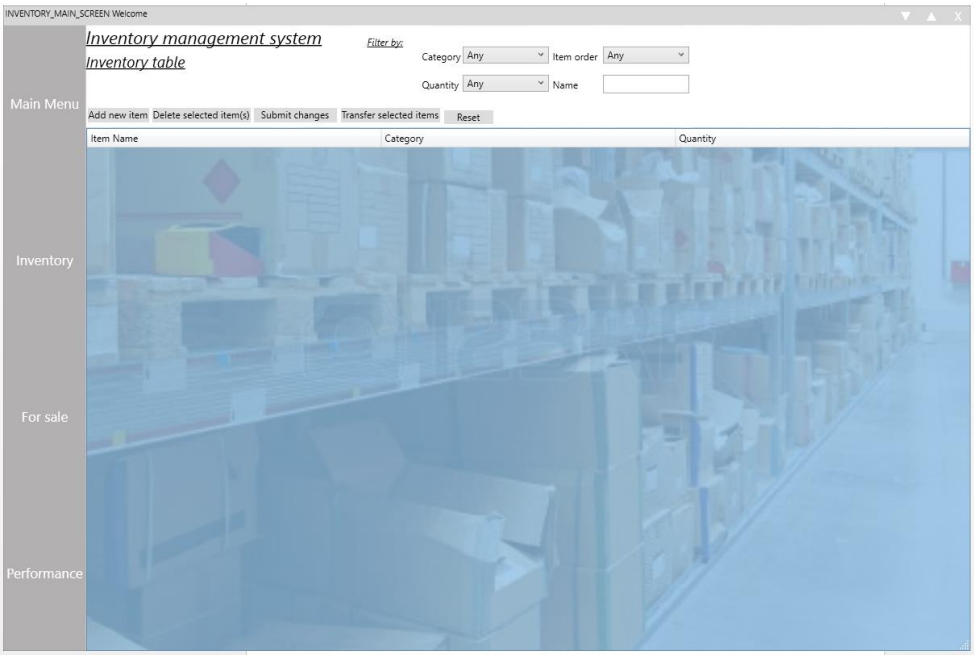



Clicking on the login button causes the main window to show and the login screen is removed.

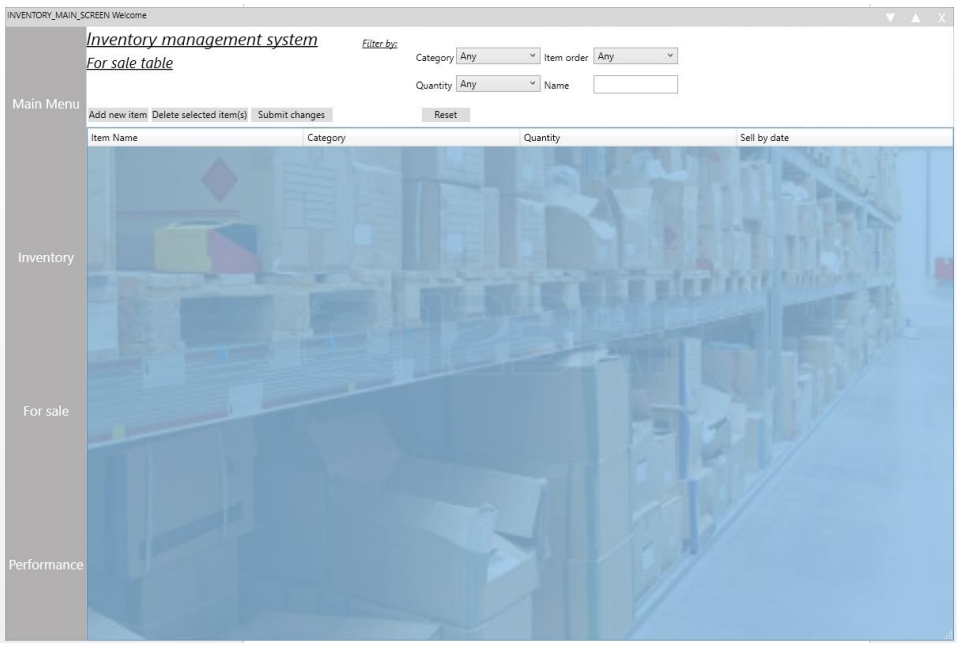
The main screen, main page is then shown to me:




Clicking the inventory page causes the inventory page to show:



Clicking on the for sale page, causes the for sale page to be shown:



# 4 Database configuration

## 4.1 SQL Server

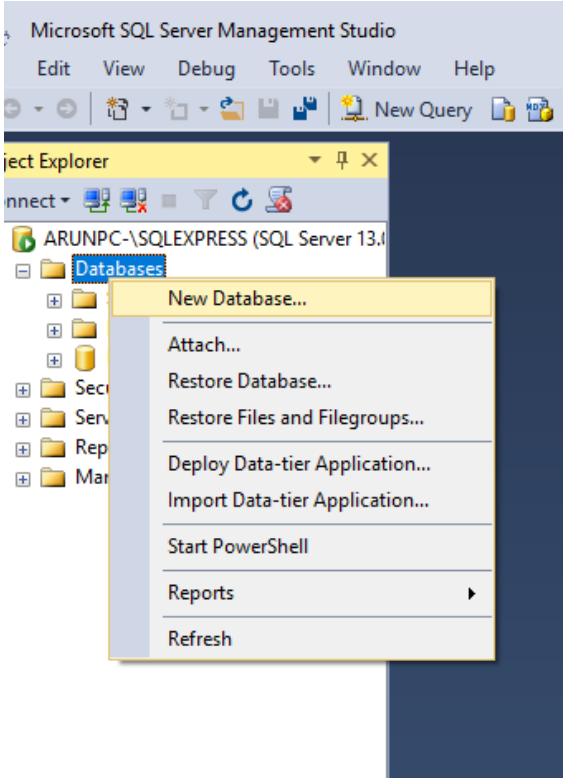
Microsoft Azure database makes use of Microsoft SQL Server Database. At time of designing my solution I was unaware that Microsoft Azure database does not offer a free option (as it previously did). **As a result, I will now be using Microsoft SQL Server Express, to create an SQL database. Once the solution is being used by my client I will then be able to upload the detached database file to a SQL server database host such as Microsoft Azure.**

SQL Server is a Microsoft product used to manage and store information. Initially, the database is created and stored in a server ran locally on the machine used to create it. This creates a database that can be queried using SQL (Structured Query Language) in the form of an '.mdf' file. This file can be 'dethatched' from the server so that the database can be accessed without having to run a server. This means, as long as the 'mdf file' is present and my solution can read it, the data can be read from any computer whilst the database is not hosted.

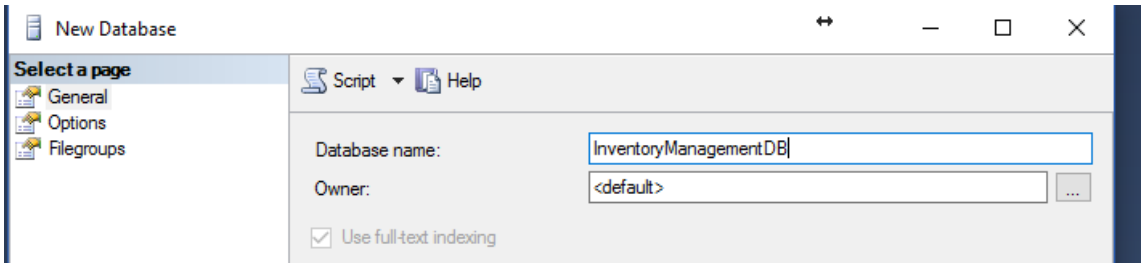
To setup my database I will need to download and install SQL server express.

After installing this software I opened the Microsoft SQL Server Management Studio. This tool was installed with the SQL server express package and allows me to create and manage my database (whilst it is attached to the server). The SQL server instance was created and started during the installation, so I just needed to click on the connect button to connect to the server.

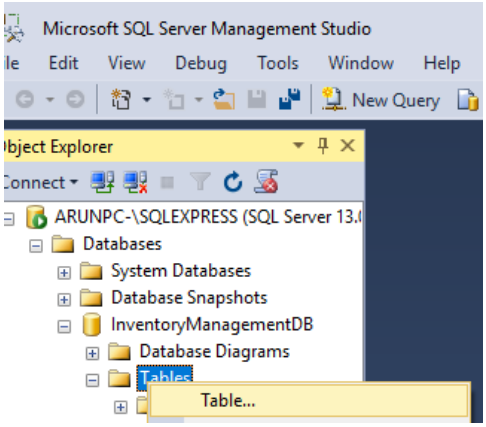
To create the database I clicked on the 'add new database' button:



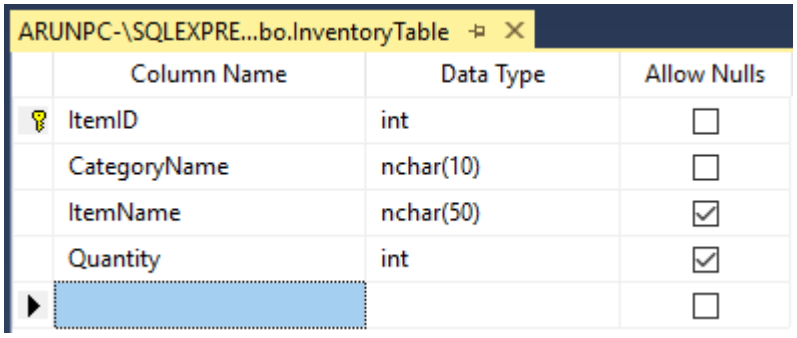
I was then presented with the following dialog, where I entered the name of the database and clicked on the 'ok' button to create the database:




The new database is then shown in the object explorer.



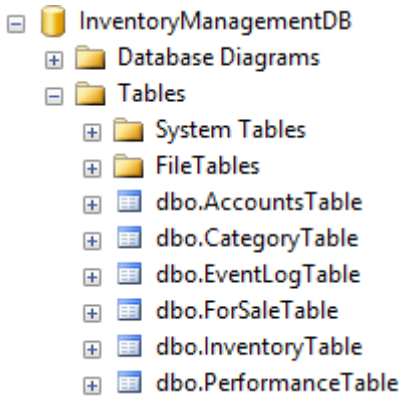

To create a new table in the database I clicked on the table button in the Tables folder. I was then presented with the following design view for the table.




The setup above, creates the inventory table that I designed in the design section of this report (design section 3.3). Shown below

When closing the table, I was prompted to name the table. I chose to name the table 'Inventory Table', as planned in the design section. I did the same for every table in the Database ERD diagram.

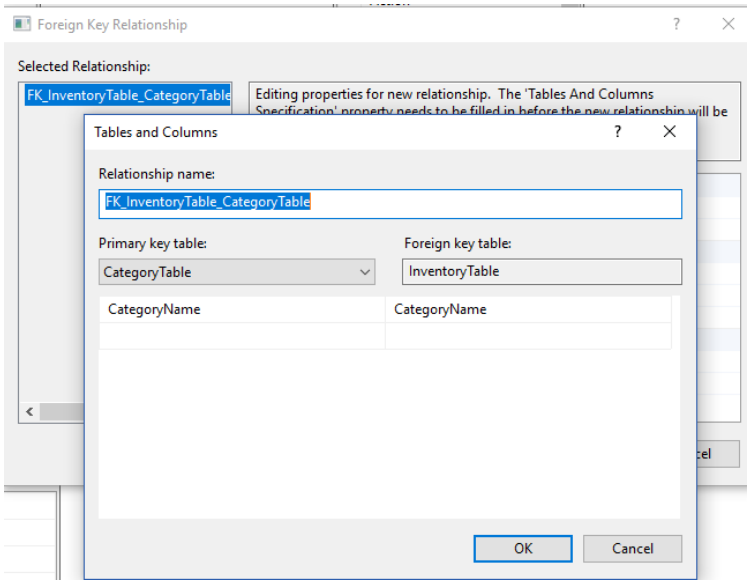
After refreshing the tables list, I was then shown a list of tables in the database.




## 4.2 Table relationships

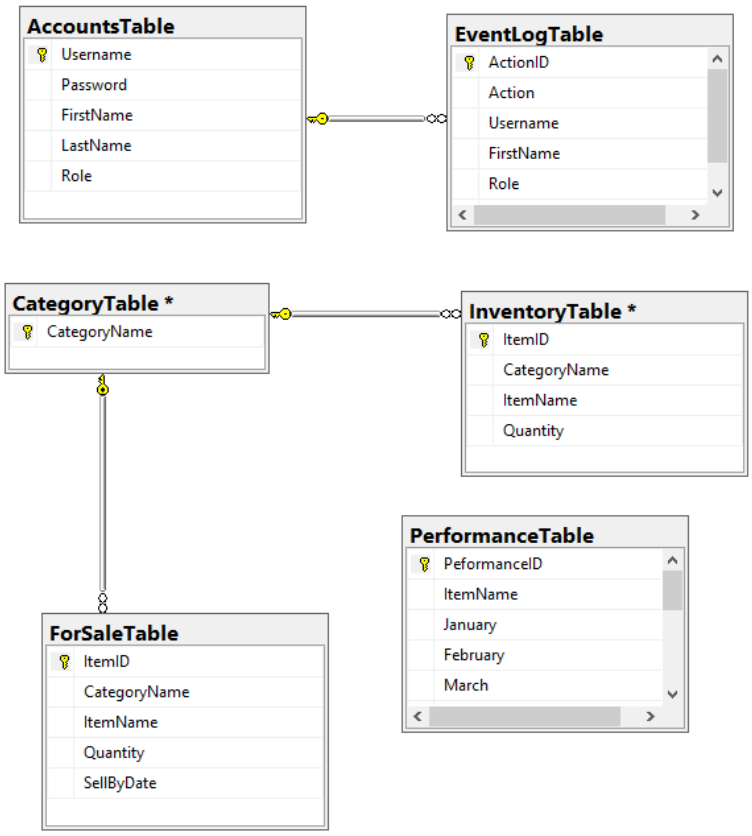
To create relationships between the existing tables in the SQL server database, I created a database diagram, which contained all the tables in the database. This provided an easy to view visual representation of the tables I had just created, which made it easier to check if they were setup correctly.

To create a relationship between two tables in the diagram, I left clicked on the foreign key of one table to the primary key of another. I was then shown this window:




Here, I can choose the exact primary and foreign key from each table and specify the name for the relationship. In every case, the default values were correct.

After setting up relationships as described in my database ERD, the database diagram looked as follows:

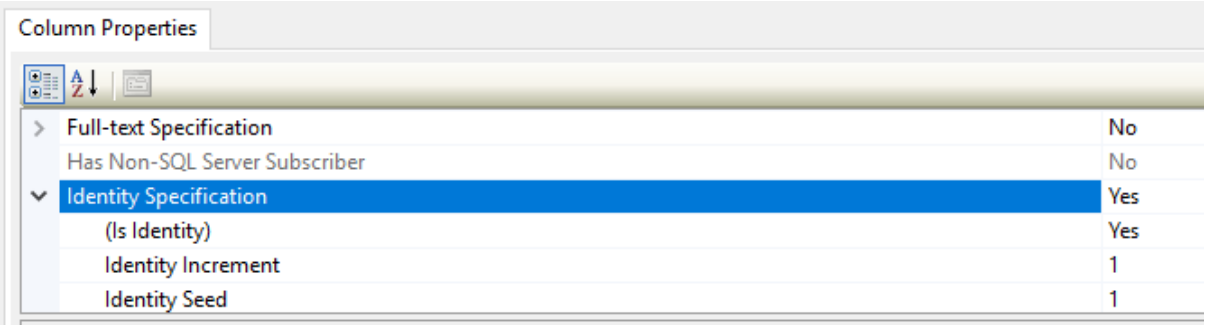



## 4.3 Auto-numbering fields and testing relationships

Some primary keys, such as ItemID should be auto generated. When a new record is created this field should be an integer value of the last ItemID incremented by 1.

To create this behaviour, I right clicked the inventory table and clicked on design view. This view was first shown to me when creating the table.

In the column properties tab, I enabled the identity specification property. By default the primary key is set to increment by 1, which is the behaviour I wanted.




I then closed the design view and saved the new changes.

To test this worked, I created test data in the inventory table. I did this by right clicking the inventory and clicking on the edit top 200 rows button. I was shown this image.

ARUNPC-\SQLEXPRE...bo.InventoryTable

| | ItemID | CategoryName | ItemName | Quantity |
|---|---|---|---|---|
| ▶* | NULL | NULL | NULL | NULL |

I can then filled out the category name, item name and quantity.

The test data I used is shown below:

ARUNPC-\SQLEXPRE...bo.InventoryTable

| | ItemID | CategoryName | ItemName | Quantity |
|---|---|---|---|---|
| | NULL | Bread | Kings meal Bre | 100 |

If the relationship works, I should be presented with an error, when I press enter (which causes the data to be uploaded to the database).

As expected I was shown the following error:

ARUNPC-\SQLEXPRE...bo.InventoryTable

| | ItemID | CategoryName | ItemName | Quantity |
|---|---|---|---|---|
| | NULL | Bread | Kings meal Bre | 100 |
| * | NULL | NULL | NULL | NULL |

Microsoft SQL Server Management Studio

No row was updated.

The data in row 1 was not committed.
Error Source: .Net SqlClient Data Provider.
Error Message: The INSERT statement conflicted with the FOREIGN KEY constraint "FK_InventoryTable_CategoryTable1". The conflict occurred in database "InventoryManagementDB", table "dbo.CategoryTable", column 'CategoryName'.
The statement has been terminated.

Correct the errors and retry or press ESC to cancel the change(s).

OK Help

Which shows that the relationship does work. To be able to add an item to the inventory table I need to setup a category. I will create the categories “Any”, “Drink”, “Food” and “Household”. To this, I opened the edit view for the category table and created records for each category as shown below:

| | CategoryName |
|---|---|
| | Any |
| | Drink |
| | Food |
| | Household |
| ▶* | NULL |

Now when I re-enter the data into the inventory table as before but using the category Food instead, I shouldn't be presented with any error and the itemID field should be auto generated.

ARUNPC-\SQLEXPRE...bo.InventoryTable

| | ItemID | CategoryName | ItemName | Quantity |
|---|---|---|---|---|
| | 60 | Food | Kingsmeal Brea... | 100 |

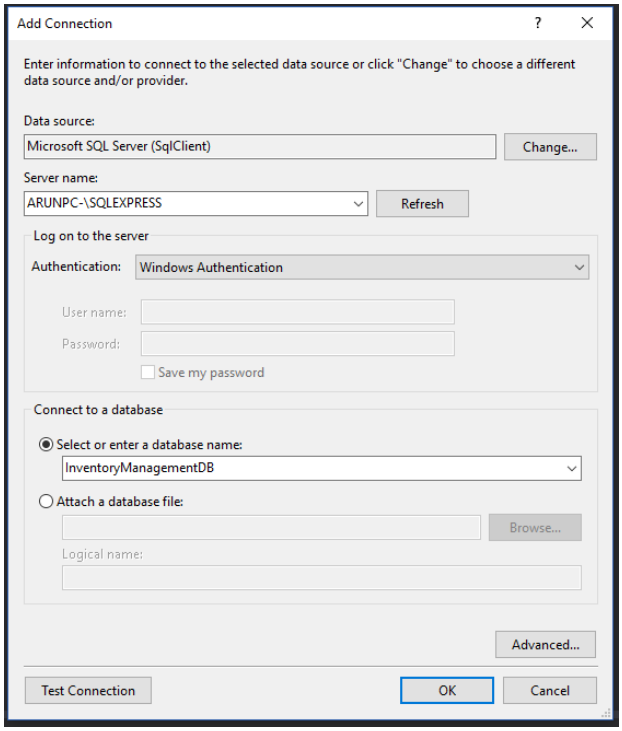
The image above shows that the item was successfully added. This shows that the relationship works and that the itemID field is auto generated as required. Its value is 60 as the previous item, which was deleted, had item ID 60.

## 4.4 Connecting the database

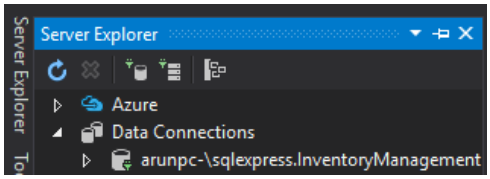
### *4.4.1 LINQ to SQL*

In order to allow the program to communicate with the database, I will use LINQ to SQL. It allows relational data to be managed as objects in C#. This simplifies managing the data in the database from the program as each record in a table can be treated as a class, making it seem more familiar for myself and future programmers working on the project.

To connect Visual studio to the SQL server instance, in the server explorer tab, I right clicked the data connections heading and clicked add connection. I then added the SQL server from the Server name drop down list, which was automatically populated. I then selected the database from the database name drop down box and clicked the ok button.

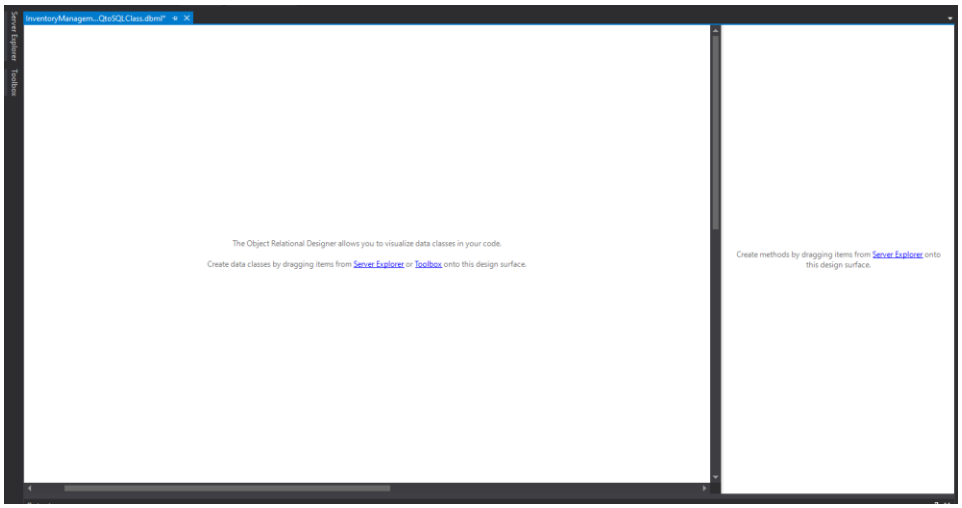



The database then showed up in the server explorer window:

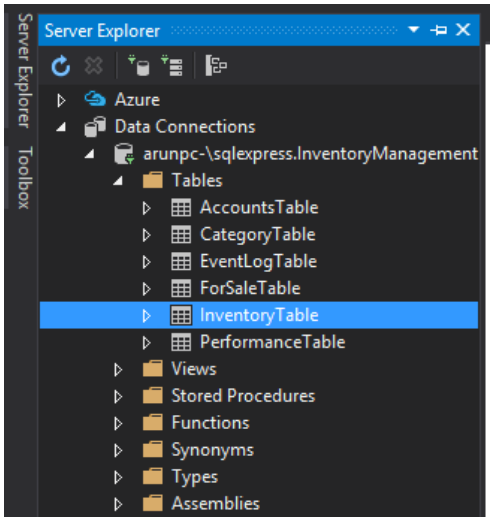


To setup LINQ to SQL I added a LINQ to SQL designer to the project in Visual studio.

I was then shown this LINQ to SQL design view:

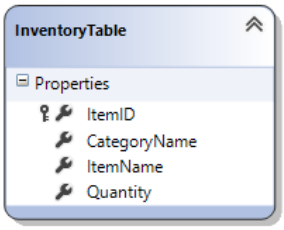



In the server explorer, I opened the table folder, showing the tables I setup in the SQL Server Management Studio shown below:

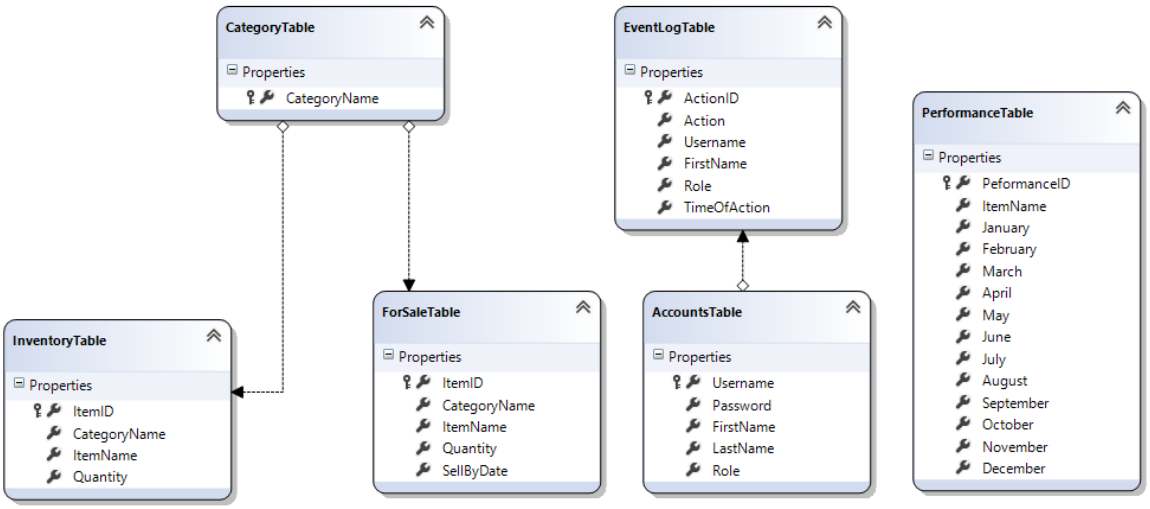



To create a LINQ to SQL object for each table, I dragged the table into the designer. As a result, an object for the table was created and LINQ to SQL was linked to the SQL server.

The image below shows the contents of the designer after doing this:



After doing this for every table, the designer then looked like this.




It automatically detected relationships and created a relationship between each object, shown by the connecting arrows.

Now that LINQ to SQL is setup, I will setup a class called database link (as shown in my class diagram in the design section 6.2), to handle all database operations.

I decided to name this class SQLServerAccessor.

Inside this class I created a global data context variable, which is set to the connection string used by the LINQ to SQL designer:

```
public InventoryManagementLINKQtoSQLClassDataContext DataContextInv
    = new InventoryManagementLINKQtoSQLClassDataContext(Properties.Settings.Default.InventoryManagementDBConnectionString);
```

The data context allows me to query the database using LINQ syntax or by using lambda expressions.

## 4.5 Setting up CRUD operations

### *4.5.1 Reading from the database*

To start with I will attempt to implement a SQL “SELECT” statement.

A LINQ query can be made to return a list of the items returned in the query. This list can then be set to the item source of a data grid. I will use this to populate the inventory data grid with a known an item in the inventory table. This will be the kings meal bread item I added earlier using the SQL management studio.

In the SQLServerAccessor class I created the following function.

```
public List<InventoryTable> SearchForItemInventory(string name)
{
    //returns a list of all items with the item name the same as the parameter
    return (from x in DataContextInv.InventoryTables
            where x.ItemName.Equals(name)
            select x).ToList();
}
```

The function returns a list (of inventory table objects) of all the items in the Inventory table but only those items who have an Item Name that is the same as the passed string.

In the main window inventory user control class, I created a global variable instance of the SQLServerAccessor class. Using inheritance was not an option here since the user control inherits from another class already.

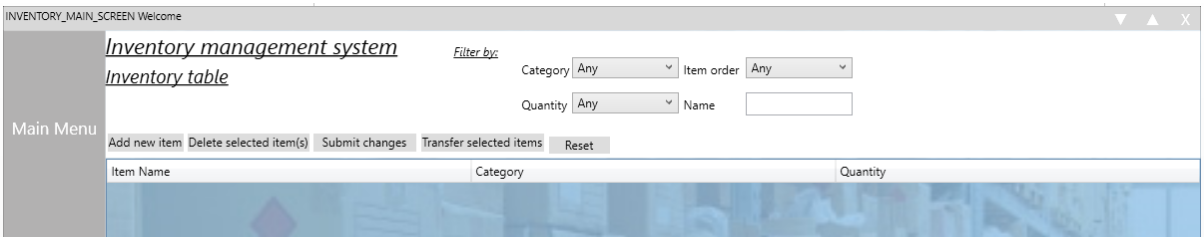
```
SQLServerAccessor SqlLink = new SQLServerAccessor();
```

In the constructor of the user control I added the following line of code:

```
inventory_dataGrid.ItemsSource = SqlLink.SearchForItemInventory("Kingsmeal Bread");
```

This should cause the data grid to show the item when the user control loads.

This did not work and the data grid remained empty, as shown below:

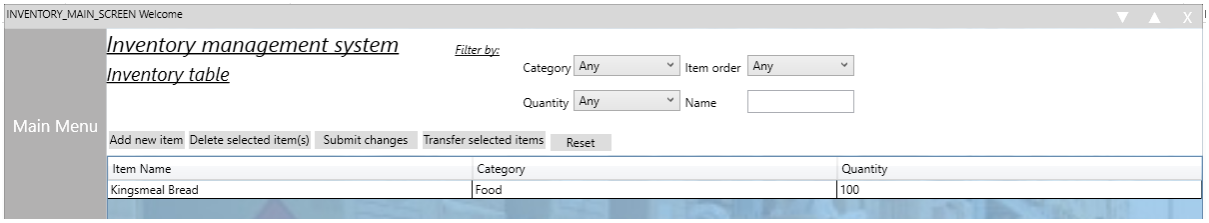



This is because the data bind used for the data grid did not fully match the object it was being set to. The inventory table contained a property, ItemID, which the datagrid was not set to bind to. To fix this I added the following to the data grids XAML

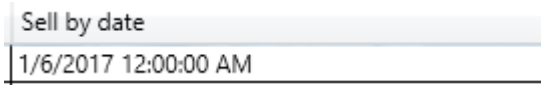
```
<DataGridTextColumn
    IsReadOnly="False"
    Header="ItemID"
    Binding="{Binding Path=ItemID}"
    Width="*"
    Visibility="Hidden"
    />
```

The visibility of this column is set to column as it not useful for the user to see.

This time, when I loaded the inventory page, the item was shown:



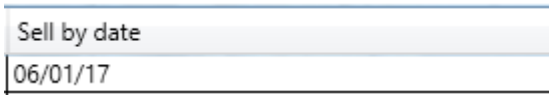

When reading from the for sale table, an unwanted extra piece of information was added to the sell by dates. The Sell by dates a time next to the date as shown below:




To remove this, I added a string format in XAML for the column:

```
Binding="{Binding Path=SellByDate , StringFormat=\{0:dd/MM/yy\} }"
```

Now, the sell by date looks as follows:



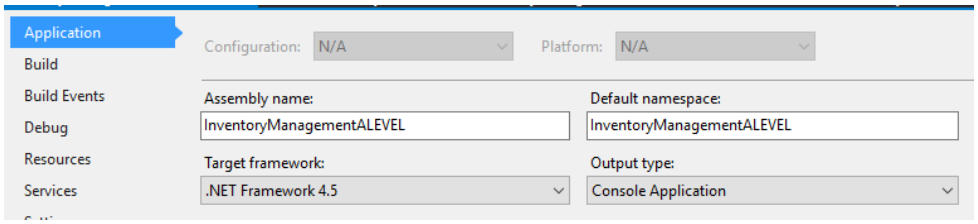

*4.5.2 Creating new items*

To create new items and have them be saved to the database, I will need to implement an SQL "INSERT" statement.

To do this I will need to create a new instance of the Inventory table object. I can then set its properties to items details.

To make debugging easier, I have changed the program type of the project to a Console application. This will allow me to see the console output so that I can perform actions such as Console.Writeline.

This was done in the properties section of the program, I set the output type to Console Application:




I created the function below to insert an item into the database given the items details as parameters.

```
public bool addNewItemInventory(string _itemName, int _quantity, string Category, string username)
{
    //the function returns true if the item was successfully inserted and false if not
    try
    {
        //instantiate a new inventory table object
        InventoryTable newItem = new InventoryTable();
        //set its properties to the parameters
        newItem.ItemName = _itemName;
        newItem.Quantity = _quantity;
        newItem.CategoryName = Category;
        DataContextInv.InventoryTables.InsertOnSubmit(newItem);
        DataContextInv.SubmitChanges();
        return true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }


}
```

In the inventory page user controls constructor, I added the following line of code, which should cause the item Coke, with quantity 100 and category Drink to be added to the database when the user control loads:

```
SqlLink.addNewItemInventory("Coke",100,"Drink");
```

After running the program, I closed it. This should be enough for the line of code to be triggered. I then checked the SQL Server Management Studio and looked at the top 200 rows for the Inventory table. The result is shown below:

ARUNPC-\SQLEXPRE...bo.InventoryTable

| ItemID | CategoryName | ItemName | Quantity |
|---|---|---|---|
| 60 | Food | Kingsmeal Brea... | 100 |
| 61 | Drink | Coke ... | 100 |

As shown, the item has been inserted successfully, with all details correctly added.

These functions can be adapted to work with the For Sale table.

*4.5.3 Creating new items from the User Interface to Inventory and the for sale table*

Now that the program is able to insert new items, I will setup an interface to allow users to do this.

To do this I will create a new window. This window will be used add new items to the inventory table.

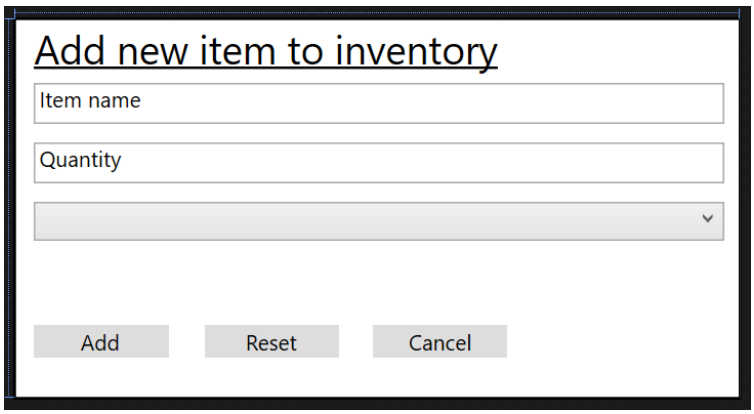
It will be a custom window (where window style is set to none like all the other interfaces I have created so far) and will be fixed size.

```
<TextBlock x:Name="maintitle_TextBlock" Text="Add new item to inventory" HorizontalAlignment="Left" VerticalAlignment="Top"  FontSize="22" TextDecorations="Underline" Margin="10,1,0,0"/>
<StackPanel HorizontalAlignment="Left" Height="93" Margin="10,35,0,0" VerticalAlignment="Top" Width="385">
    <TextBox x:Name="ItemName_textBox" Height="23" Margin="0,0,0,0" TextWrapping="Wrap" Text="Item name" GotFocus="ItemName_textBox_GotFocus"/>
    <TextBox x:Name="Quantity_textBox" Height="23"  Margin="0,10,0,0" TextWrapping="Wrap" Text="Quantity" GotFocus="Quantity_textBox_GotFocus"/>
    <ComboBox x:Name="Category_comboBox" Margin="0,10,0,0" SelectedIndex="0"/>
</StackPanel>
<Button x:Name="add_button" Content="Add" HorizontalAlignment="Left" Margin="10,170,0,0" VerticalAlignment="Top" Width="75" Click="add_button_Click" BorderBrush="Transparent" BorderThickness="0"/>
<Button x:Name="reset_button" Content="Reset" HorizontalAlignment="Left" Margin="105,170,0,0" VerticalAlignment="Top" Width="75" Click="reset_button_Click" BorderBrush="Transparent" BorderThickness="0"/>
<Button x:Name="cancel_button" Content="Cancel" HorizontalAlignment="Left" Margin="199,170,0,0" VerticalAlignment="Top" Width="75" Click="cancel_button_Click" BorderBrush="Transparent" BorderThickness="0"/>
```

I created a stack panel to hold the data entry fields, two textboxes for the Item name and quantity and a combo box for selecting the category of the item.

As with my other interfaces, it has code to allow it be moved around the screen and to clear the contents of any textboxes with default text when the user clicks on it. The reset button just restores the textboxes to their default text value.

In the designer view, this is what the interface looked like:

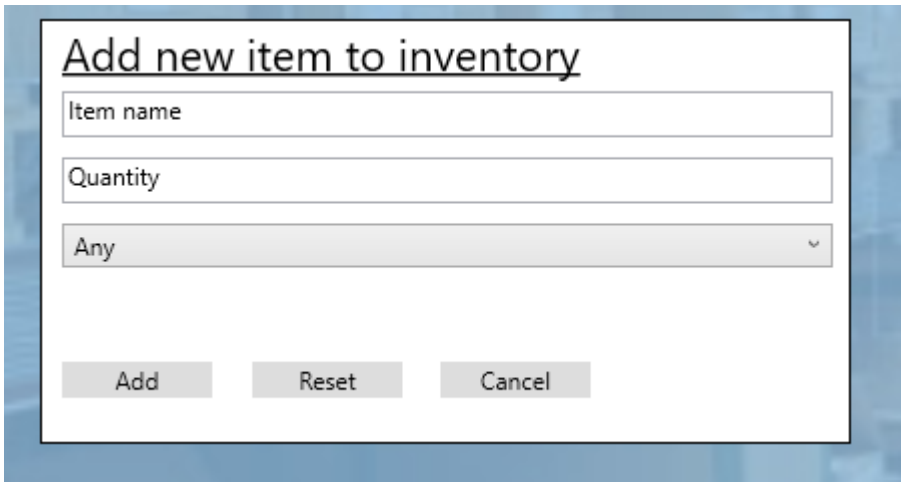



This window should be shown when the user clicks on 'the add new item' button in the inventory page. To do this, in' the add new item' button, mouse click, event handler I added the following code.

```
private void add_button_Click(object sender, RoutedEventArgs e)
{
    AddNewItemWindow Addwin = new AddNewItemWindow();
    Addwin.Show();
}
```

The code above creates an instance of the add item window and then shows it.

The image below is of the add new item window being shown after the add new item button was clicked.




By default, the category drop down list is set to any. This drop down box should display a list of all the categories in the category table. This will involve a simple "SELECT" query.

In the SQLServerAccessor class, I created the following function:

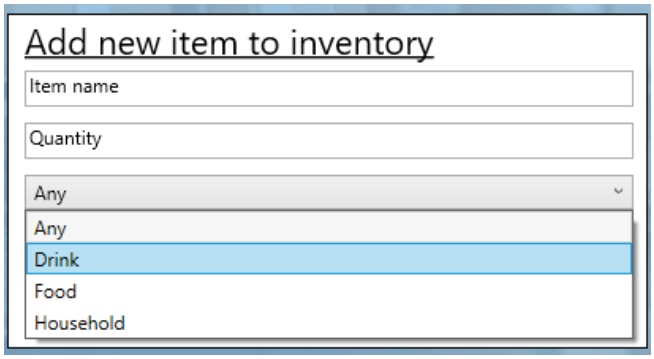
```
public IEnumerable<CategoryTable> getCategories()
{
    return from s in DataContextInv.CategoryTables
           select s;

}
```

This will return all the categories in the category table.

In the constructor of this window I added the following code:

```
public AddNewItemWindow()
{
    InitializeComponent();
    //populate the category combobox with all the categories in the category table
    Category_comboBox.ItemsSource = SQLlink.getCategories().ToList();
    Category_comboBox.DisplayMemberPath = "CategoryName";
    Category_comboBox.SelectedValuePath = "CategoryName";
    Category_comboBox.SelectedIndex = 0;

}
```

This code just populates the drop down list with the categories in the category table as shown below:

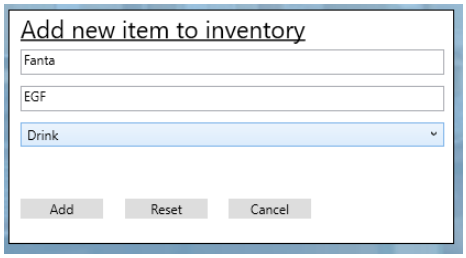



Provided the inputs in the Item Name and Quantity are valid, when clicking the Add button, the add new item to inventory function in the SQLServerAccessor class can be called, passing the values from the input fields in this window.

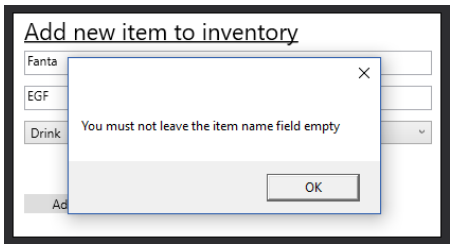
```
private void add_button_Click(object sender, RoutedEventArgs e)
{
    Console.WriteLine(Category_comboBox.Text);
    int passedval;
    //validate the input fields, is the name field empty? is the quantity an integer?
    if ((ItemName_textBox.Text != null) && (int.TryParse(Quantity_textBox.Text, out passedval)))
    {
        try
        {
            //insert the new item
            if (SQLlink.addNewItemInventory(ItemName_textBox.Text, Convert.ToInt32(Quantity_textBox.Text), Category_comboBox.Text.Replace(" ", string.Empty), mainUsername))
            {
                MessageBox.Show("Added new item");
                this.Close();
            }
        }
        catch (Exception en) { Console.WriteLine(en.ToString()); }

    }
    else if(ItemName_textBox.Text != null)
    {
        MessageBox.Show("You must not leave the item name field empty");
    }
    else if (int.TryParse(Quantity_textBox.Text, out passedval))
    {
        MessageBox.Show("The quantity you have entered is not an integer");
    }

}
```

The code above inserts the inputted item into the database. It also validates the input to make sure the item name is not empty and the quantity is an integer.  If the input is not valid a suitable message is shown to the user.

Trying to enter the following input should result in an error being generated:




When clicking on the add button the following message was shown:




This is not the correct error that should have been shown.

This was a logic error in the else if statement used to check if the item name field was empty. The amended else if statement is shown below:

```
else if(ItemName_textBox.Text == null)
```

Testing the same input caused no warning message to be showed. To debug this issue, I have setup a break point.

```
39          private void add_button_Click(object sender, RoutedEventArgs e)
40          {
41              Console.WriteLine(Category_comboBox.Text);
42              int passedval;
43              //validate the input fields, is the name field empty? is the quantity an integer?
44              if ((ItemName_textBox.Text != null) && (int.TryParse(Quantity_textBox.Text, out passedval)))
45              {
46                  try
47                  {
48                      //insert the new item
49                      if (SQLlink.addNewItemInventory(ItemName_textBox.Text, Convert.ToInt32(Quantity_textBo
50                      {
51                          MessageBox.Show("Added new item");
52                          this.Close();
53                      }
54                  }
55                  catch (Exception en) { Console.WriteLine(en.ToString()); }
56
57              }
58              else if(ItemName_textBox.Text == null)
59              {
60                  MessageBox.Show("You must not leave the item name field empty");
61              }
62              else if (int.TryParse(Quantity_textBox.Text, out passedval))
63              {
64                  MessageBox.Show("The quantity you have entered is not an integer");
65              }
```

When the highlighted line of code is executed, the program should stop at this line and allow me to step through the code. The inspector output will show me the value of variables, allowing me to identify the error in the code.

When the program is ran, and the line of code is ran, the program stops.

Using the buttons above, I can step through the code.

When I do this I can view the value of local variables being used:

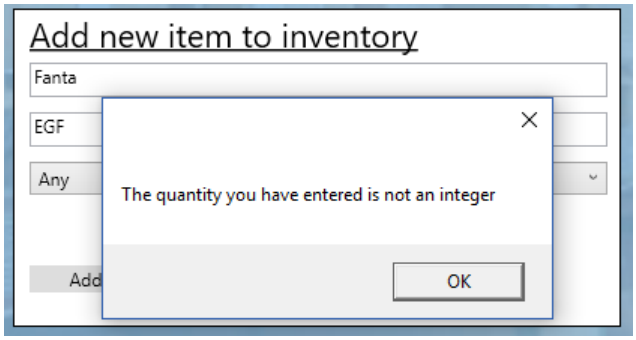
Locals

| Name | Value |
|---|---|
| System.Windows.Controls.TextBox.Text.get returned | "Fanta" |
| System.Windows.Controls.TextBox.Text.get returned | "EGF" |
| int.TryParse returned | false |
| this | {InventoryManagementALEVEL.AddNewItemWindow} |
| sender | {System.Windows.Controls.Button: Add} |
| e | {System.Windows.RoutedEventArgs} |
| passedval | 0 |

The issue was that the final else if statement should have been a 'NOT' statement. If the value int parse method returns false then the value is not an integer, instead I was testing for it was true. The corrected else if statement is shown again.

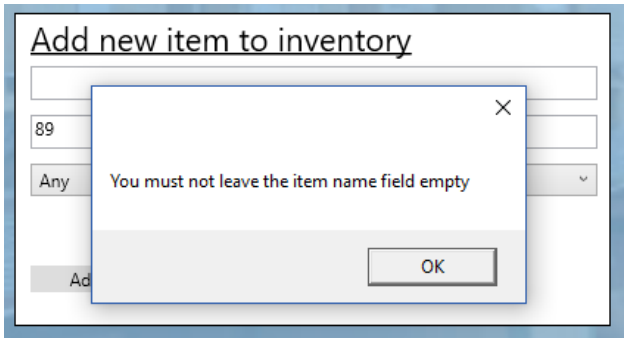
```
else if (!int.TryParse(Quantity_textBox.Text, out passedval))
```

When inputting the previous inputs as before I was shown the following error message:



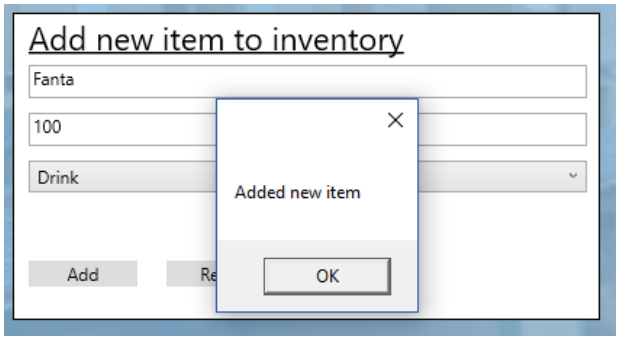

This is the correct error message that should be shown.

Leaving the item name empty produces the following error:




Using the inputs, Fanta, 100 and Drink, I will test to see if the item is added to the database.

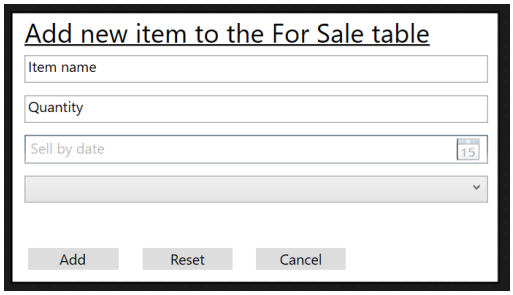
The image below shows the success message shown when I clicked the add button:




In the SQL Server Management Studio, the new record was generated and shown below:

| | | | | |
|---|---|---|---|---|
| | 65 | Drink | Fanta ... | 100 |

The same process was done to produce a window for adding items directly into the for sale table. This table requires items to have a specified sell by date. I used a date picker control to allow the user to input a date:




To set the date picker default text, I used a XAML style.

```
<Style TargetType="{x:Type DatePickerTextBox}">
    <Setter Property="Text" Value="Sell by date" />
</Style>
```

The date picker should only be enabled if the selected category is either Food or Drink.

To do this, I setup an event to be triggered when the combo box is closed, so when the user has selected a category, the event is fired. In the event handler for this, I added the following code

```
private void Category_comboBox_ContextMenuClosing(object sender, ContextMenuEventArgs e)
{
    //enable the date picker if they have chosen food or drink
    if (Category_comboBox.Text == "Food" || Category_comboBox.Text == "Drink")
    {
        SellByDate_datePicker.IsEnabled = true;
    }
}
```

I then setup the LINQ query to insert the new item into the for sale table. The function I have created below is stored in the SQLServerAccessor class:

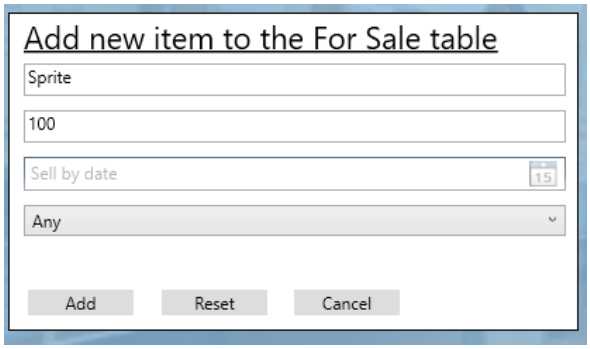
```
public bool addNewItemForSale(string _itemName, int _quantity, string Category, DateTime SellByDate)
{
    try
    {
        ForSaleTable newItem = new ForSaleTable();
        newItem.ItemName = _itemName;
        newItem.Quantity = _quantity;
        newItem.CategoryName = Category;
        newItem.SellByDate = SellByDate;
        DataContextInv.ForSaleTables.InsertOnSubmit(newItem);
        DataContextInv.SubmitChanges();

        return true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }

}
```

In the Add button click event handler function, I validated the inputs and called the function above to insert the item.

```
private void add_button_Click(object sender, RoutedEventArgs e)
{
    int passedval;
    Console.WriteLine(Category_comboBox.Text);
    //validate the input fields, is the name field empty? is the quantity an integer?
    if ((!string.IsNullOrWhiteSpace(ItemName_textBox.Text)) && (int.TryParse(Quantity_textBox.Text, out passedval)))
    {
        try
        {
            //insert the new item
            if (SQLlink.addNewItemForSale(ItemName_textBox.Text, Convert.ToInt32(Quantity_textBox.Text), Category_comboBox.Text.Replace(" ", string.Empty), SellByDate_datePicker.SelectedDa
            {
                MessageBox.Show("Added new item");
                this.Close();
            }
        }
        catch (Exception en) { Console.WriteLine(en.ToString()); }

    }
    else if (string.IsNullOrWhiteSpace(ItemName_textBox.Text))
    {
        MessageBox.Show("You must not leave the item name field empty");
    }
    else if (!int.TryParse(Quantity_textBox.Text, out passedval))
    {
        MessageBox.Show("The quantity you have entered is not an integer");
    }
}
```

To test this works I will use following test data: Item name: Sprite, Quantity: 100, sell by date 01/01/2017 and category Drink.

Before choosing the drink category, the date picker is not enabled:

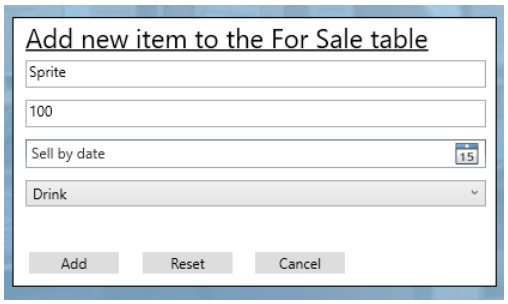



However, when I chose the drink category the date picker remained disabled.

To fix this I have changed the event being triggered by the combo box and removed any white space from combo boxes text.

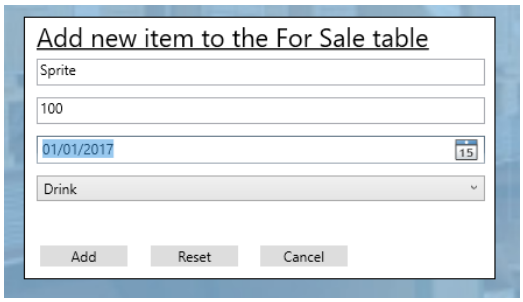
```
private void Category_comboBox_DropDownClosed(object sender, EventArgs e)
{
    Console.WriteLine(Category_comboBox.Text.Replace(" ", ""));
    //enable the date picker if they have chosen food or drink
    if (Category_comboBox.Text.Replace(" ", "") == "Food" || Category_comboBox.Text.Replace(" ", "") == "Drink")
    {
        SellByDate_datePicker.IsEnabled = true;
    }
}
```

I entered the same data into the fields of the window and chose the drink category:

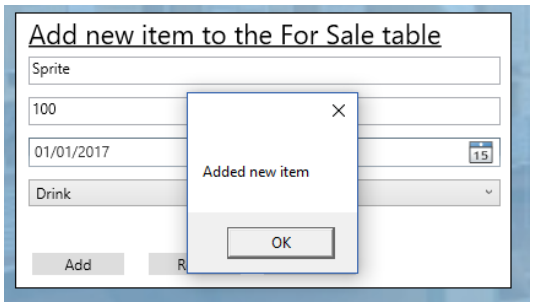



The date picker enabled this time, as intended.

After selecting the sell by date:




Clicking add generated a success message:




Checking the records in the SQL Server Management Studio showed the new item added to the For Sale table:

| | | | | | |
|---|---|---|---|---|---|
| 22 | Drink | Sprite | ... | 100 | 2017-01-01 |

The image above shows the record in the database, exactly as entered in the program.

*4.5.4 Auto refreshing the database data into the data grids*

In order for the data in the data grids to stay up to date, it must be fetched from the database again. For example, if the program loaded and the program fetched the data whilst another second

instance of the program then adds another item to the database after, the data grids in the first instance won't show the new item, it must be fetched again to show this new item.

I will set the data grids to fetch new data every second to give an almost real time experience. In the inventory table I created a new instance of the dispatcher timer class:

```
DispatcherTimer dispatcherTimer = new DispatcherTimer();
```

I created the function to be called every time the timer ticks:

```
private void dispatcherTimer_Tick(object sender, EventArgs e)
```

I then setup the timer in the user controls constructor:

```
dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
dispatcherTimer.Start();
```

This setups the timer and causes it to tick every second.

In the SQLServerAccessor, I created a function to return every item in the inventory table:

```
public List<InventoryTable> getNewInventoryTable()
{
    return (from x in DataContextInv.InventoryTables
            select x).ToList();
}
```

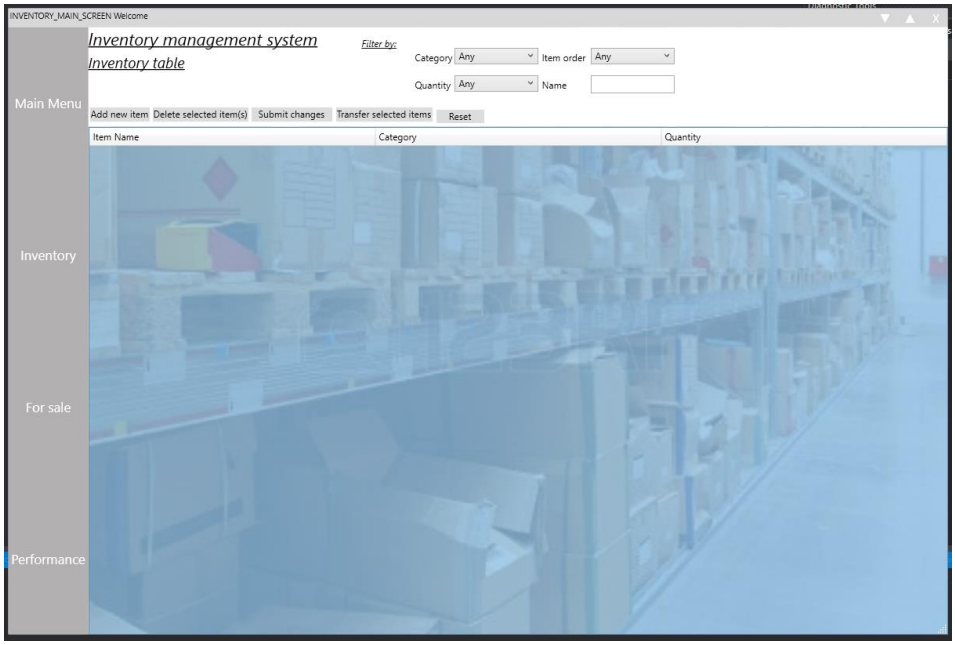
In the timer tick method I called the function I just created and set the data grids item source to the value it returned:

```
private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    inventory_dataGrid.ItemsSource = SqlLink.getNewInventoryTable();
```

This should cause the data grid to update every second.

To test this works I will load the program and leave it on the inventory page. Then I will add a new item to the inventory table via the SQL Server Management Studio. If the new item shows up in the table then the auto update feature works.

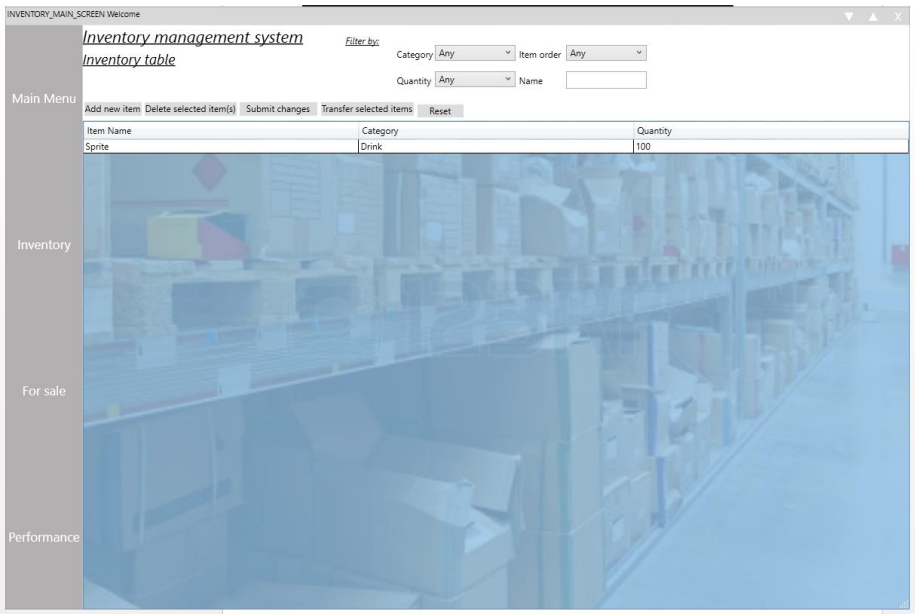
Before any items were added to the table, the inventory page showed a blank table, as shown below:




In the SQL Server Management studio I added the new item, shown below:

| | | | | | |
|---|---|---|---|---|---|
| | 66 | Drink | Sprite | ... | 100 |

In the inventory page, the item was shown almost instantly:




For the for sale page, I took the same approach, using timers. However the query function in the SQLServerAccessor differed, to select the items in the for sale page instead of the. The LINQ query used for the for sale table is shown below:

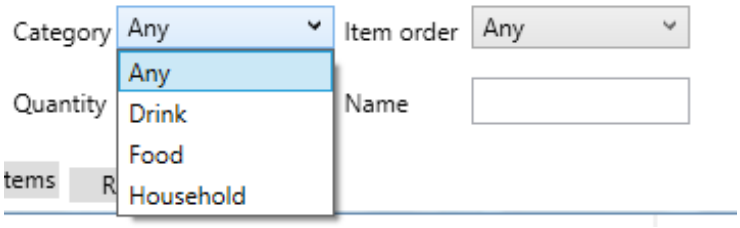
```
return (from x in DataContextInv.ForSaleTables
        select x).ToList();
```

### *4.5.5 Setting up filters*

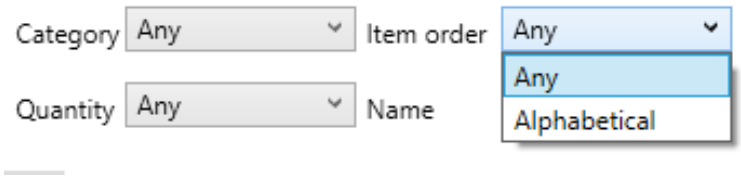
In order to setup filters I need to setup the filter controls. I will do this programmatically inside the user controls constructor:

```
public MainInventory()
{
    InitializeComponent();

    //populate the data grid with the inventory table initially
    inventory_dataGrid.ItemsSource = SqlLink.DataContextInv.InventoryTables.ToList();
    //populat the category combo box with all the categories in the category table.
    category_comboBox.ItemsSource = SqlLink.getCategories().ToList();

    category_comboBox.DisplayMemberPath = "CategoryName";
    category_comboBox.SelectedValuePath = "CategoryName";
    category_comboBox.SelectedIndex = 0;

    item_order_comboBox.Items.Add("Any");
    item_order_comboBox.Items.Add("Alphabetical");
    item_order_comboBox.SelectedIndex = 0;

    quantity_comboBox.Items.Add("Any");
    quantity_comboBox.Items.Add("Ascending");
    quantity_comboBox.Items.Add("Descending");
    quantity_comboBox.SelectedIndex = 0;


    dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
    dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
    dispatcherTimer.Start();
}
```

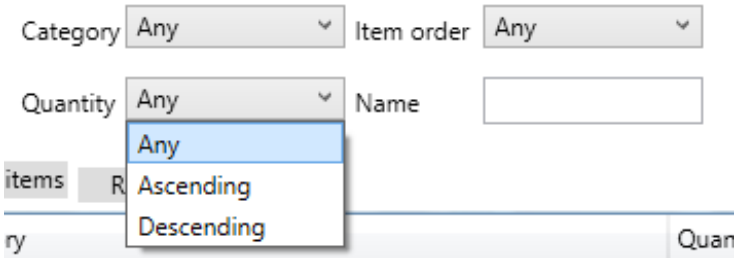
Shown below is the category combo working as intended:




The item order combo box showing the data entered programmatically:




And the quantity combo box:



Now that I can get an input from the user based on how to filter the data in the tables, I can create queries to reflect on these inputs.

I have altered the get inventory table function in the SQLServerAccessor class to take two parameter, the name of the item and the category. These parameters are to be taken from the filter controls.

If the passed name parameter is the name of the item and the category parameter is “Any” then the function should return items that have a similar name to the item name parameter but from any category. This is achieved by using the following LINQ query:

```
if (name != "" && category == "Any")
{
    //returns the items which are similar or exactly the same as the input name but can come from any category
    return (from x in DataContextInv.InventoryTables
            where x.ItemName.Contains(name)
            select x).ToList();
}
```

Similarly if the item name is given and the category is not “Any”, then the query should return items that have a name similar to the item name parameter and that are in the given category:

```
else if (name != "" && category != "Any")
{
    //returns the items which are similar or exactly the same as the input name and are in the input category
    return (from x in DataContextInv.InventoryTables
            where x.ItemName.Contains(name) &&
            x.CategoryName.Equals(category)
            select x).ToList();
}
```

If the name is not given but a category is given then it should return all the items in the given category:

```
else if (name == "" && category != "Any")
{
    //returns items in the input category
    return (from x in DataContextInv.InventoryTables
            where x.CategoryName.Equals(category)
            select x).ToList();
}
```

If none of these conditions are met then no filter has been selected and so the function should return all the items:

```
else
{
    //no filter , return everything
    return (from x in DataContextInv.InventoryTables
            select x).ToList();
}
```

The same set of tests are completed on inputs from the filter section of the for sale table. The query function for that table instead returns items from the for sale table.

Since these query functions are called every second, when the user selects a filter, the tables should filter almost instantly.

In the user controls timer tick method, I have moved its code into another method called update. I will then call this method from the timer tick method. I have done this to keep the timer tick method tidy. If the timer tick is used for another purpose at a later date, it will make it easier to view the new code:

```
updateAll();
```

In this method I will call the 'get inventory table' function in the SQLServerAccessor class, using it to set the data grids item source. I will pass the values of the filter section controls to this function.

```
inventory_dataGrid.ItemsSource = SqlLink.getNewInventoryTable(category_comboBox.Text.Replace(" ", string.Empty), name_textBox.Text);
```

Finally, I need to setup filtering for the order of items, in ascending or descending order based on quantity or alphabetical order.

To do this, I need to use sort descriptions on the data grid. Firstly, I will clear the current sort description being used on the data grid and then set every column in the data grid to have no sort direction:

```
inventory_dataGrid.Items.SortDescriptions.Clear();
foreach (var cols in inventory_dataGrid.Columns)
{
    cols.SortDirection = null;
}
```

The foreach loop is a short hand way of looping around the grid instead of using a standard for loop.

If the item order combo box is set to alphabetical, then first column (item name) of the data grid needs to be ordered in alphabetical order:

```
if (item_order_comboBox.Text == "Alphabetical")
{
    //arrange the first column (item name) in alphabetical order
    var col = inventory_dataGrid.Columns[0];
    inventory_dataGrid.Items.SortDescriptions.Add(new SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

    col.SortDirection = ListSortDirection.Ascending;
}
```

If the quantity combo box is set to ascending or descending, then the third column (quantity) needs to be ordered in an ascending or descending way.

```
else if (quantity_comboBox.Text == "Ascending")
{
    //arrange the third column (quantity) in ascending order
    var col = inventory_dataGrid.Columns[2];
    inventory_dataGrid.Items.SortDescriptions.Add(new SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

    col.SortDirection = ListSortDirection.Ascending;
}
else if (quantity_comboBox.Text == "Descending")
{
    //arrange the third column (quantity) in descending order
    var col = inventory_dataGrid.Columns[2];
    inventory_dataGrid.Items.SortDescriptions.Add(new SortDescription(col.SortMemberPath, ListSortDirection.Descending));

    col.SortDirection = ListSortDirection.Descending;
}
inventory_dataGrid.Items.Refresh();
```

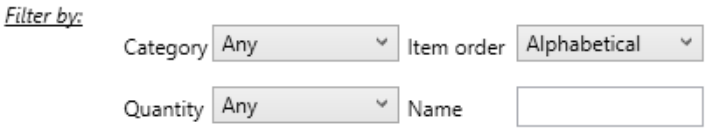
The last line of code in the image above refreshes the data grid.

To test the filters, I will add several items to the inventory table using the program itself.

The new items are shown below:

| Item Name | Category | Quantity |
|---|---|---|
| Sprite | Drink | 100 |
| Coke | Drink | 50 |
| Kingsmeal White Bread | Food | 34 |
| Phone charger USB | Any | 10 |
| Semi Skimmed milk | Drink | 56 |

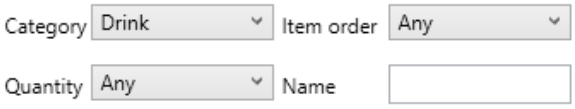
Firstly, I will try to order the items in alphabetical order:




This caused the items to be sorted into alphabetical order:

| | | |
|---|---|---|
| Coke | Drink | 50 |
| Kingsmeal White Bread | Food | 34 |
| Phone charger USB | Any | 10 |
| Semi Skimmed milk | Drink | 56 |
| Sprite | Drink | 100 |

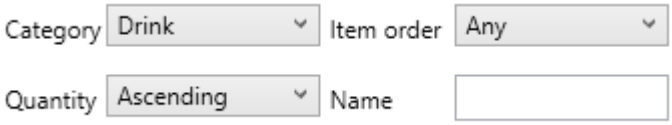
Now I will only show items that are in the drink category:




As a result, only items in the drink category were shown:

| Item Name | Category | Quantity |
|---|---|---|
| Sprite | Drink | 100 |
| Coke | Drink | 50 |
| Semi Skimmed milk | Drink | 56 |

Now, I will keep the category filter enabled and try to order the items that have made it through the filter by quantity in ascending order:




The result is shown below:

| | | |
|---|---|---|
| Coke | Drink | 50 |
| Semi Skimmed milk | Drink | 56 |
| Sprite | Drink | 100 |

To test the item name filter, I have added two new items with item names: “Coke Vanilla” and “Coke Cherry”.

| | | |
|---|---|---|
| Coke Cherry | Drink | 12 |
| Coke Vanilla | Drink | 14 |

When using item name: “Coke”:

*Filter by:*

Category Any Item order Any

Quantity Any Name coke

The items left in the in the data grid are:

| Item Name | Category | Quantity |
|---|---|---|
| Coke | Drink | 50 |
| Coke Cherry | Drink | 12 |
| Coke Vanilla | Drink | 14 |

To make the filter even more precise I have changed the item Name filter to “Coke Cherry”.

This causes only the coke cherry item to be shown:

| Item Name | Category | Quantity |
|---|---|---|
| Coke Cherry | Drink | 12 |

*4.5.6 Updating database data based on the data grids data*

There is an issue with the way the data grids work in its current state. If the user tries to select a record in the data grid or try to change the values of a field in the record, the data grid automatically deselects it. The reason for this, is because, the data grids are changing their item source every second, in order to get up to date data. To fix this, I need to disable automatic updating on the data grid whilst the user has selected a record in the data grid.

This can be achieved by determining whether the user has selected a row or not. If the user has selected a row then don’t call the update all function (called in the timer tick method):

```
private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    //if the user has selected an item , then disable automatic updating
    if (inventory_dataGrid.SelectedItem == null)
    {
        updateAll();
    }
```

To re-enable automatic updating on the data grid, I need to detect that the user has either pressed enter on the record (causing it to de select and change the records objects properties), or clicked elsewhere. The only other place the user can click on in the user control is the main grid itself. To detect that the user has clicked on the main grid, I setup a mouse left click event on the main grid. When this event is fired, it should deselect the data grid. To do this, in the event handler for this event I added the following code:

```
private void GridMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    //deselect the datagrid, if the user clicks on the main grid
    inventory_dataGrid.UnselectAll();
}
```

I also want the user to know that the data grid is not updating anymore and that it is in an edit mode.

I have created a label to go above the data grid. By default, I have set its content to nothing, so that it is not visible to the user. It also has the font colour red.

```
<Label x:Name="editmode_label" Content=""
```

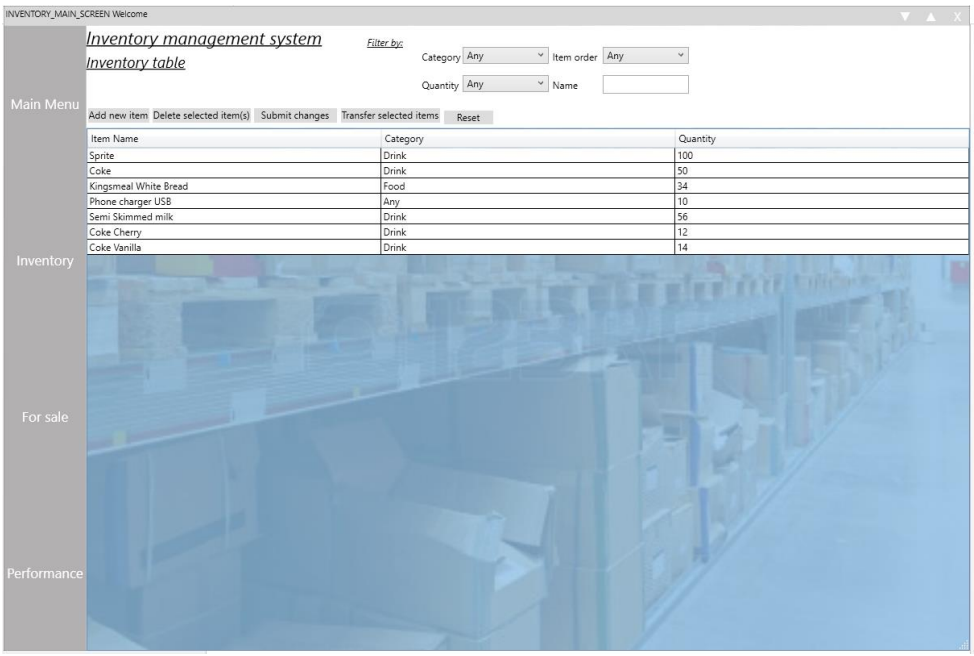
When the user is editing the data grid, I will set the text of the label to “edit mode enabled”:

```
private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    //if the user has selected an item , then disable automatic updating
    if (inventory_dataGrid.SelectedItem == null)
    {
        updateAll();
    }
    else
    {
        editmode_label.Content = "edit mode enabled";
        Console.WriteLine("focused");
    }
}
```

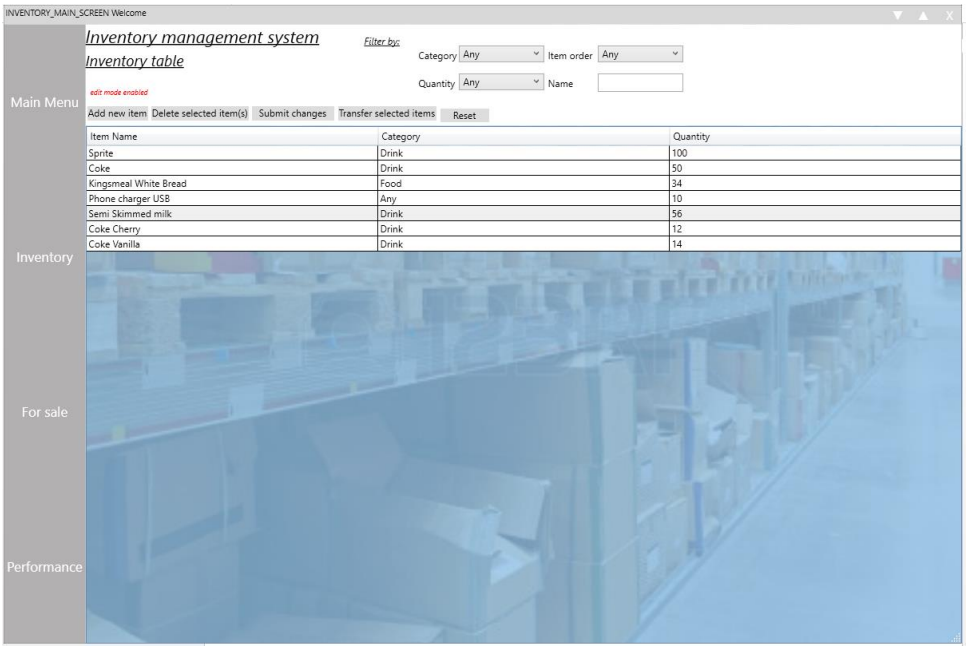
If the grid is auto updating then edit mode is not enabled anymore and so the label should be set to nothing again to hide it from the user:

```
private void updateAll()
{
    editmode_label.Content = "";
```

When no item is selected in the data grid the user control looks as follows:




When an item is selected, the red edit mode label is shown and the row is not de selected by itself since the grid is not auto updating:




The edit mode label is shown as intended and the item is not deselected.

To actually submit the changes the user has made to items, I will call an update method when the user clicks on the submit changes button.

In the submit changes button event handler method I added the following line of code:

```
SqlLink.DataContextInv.SubmitChanges();
```

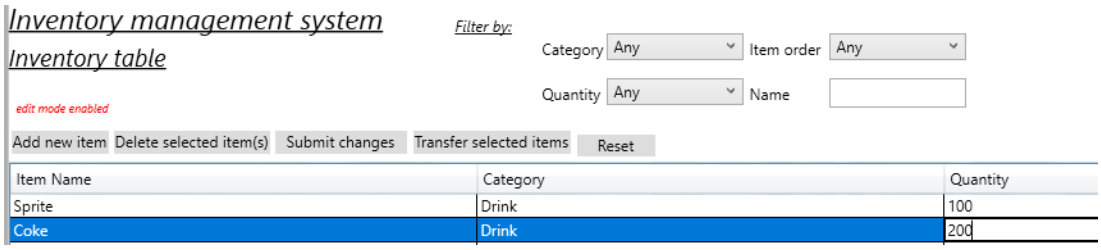
This causes LINQ to SQL to update the database based on the table objects in the program.

To test this works, I will change the quantity of an item and click on the submit changes button. After clicking the button, the grid should resume auto updating (since it would deselect any items) and fetch data from the data base. If the changes I made persist then the program updated the record successfully.

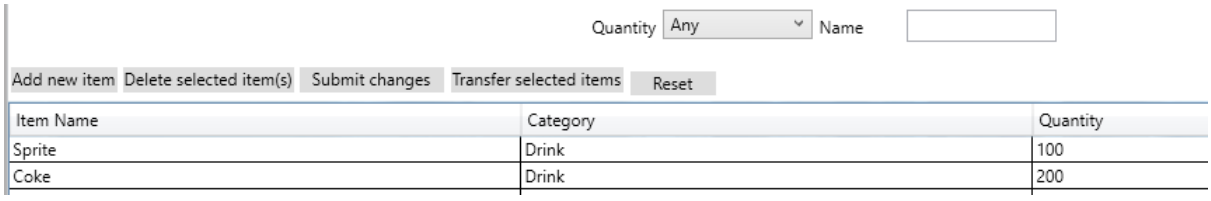
I will change the quantity of coke to 200.The coke items record in the data grid before is shown below:

| | | |
|---|---|---|
| Coke | Drink | 50 |

Before clicking the submit changes button:




I then pressed the enter button and clicked the submit changes button:




Edit mode disabled itself and the data grid resumed auto updating. The new quantity remained at 200 meaning the program successfully updated the data grid.

I did the same for the for sale table. The same method was called in the update all method of the for sale table:

```
SqlLink.DataContextInv.SubmitChanges();
```

*4.5.7 Deleting Data*

Users are able to delete more than one item at a time. As with updating items, when the user selected a single item or multiple items, the data grid should enter edit mode. This behaviour will be present as the user will still be selecting items, which is the trigger for entering edit mode.

In every case, the program assume that the user has selected more than one item. If this isn't the case then it will deal with the one item only.

Firstly, I will setup a method in the SQLServerAccessor class. This method will take a list or array of item IDs corresponding the items selected by the user. It will then delete the items that correspond to these items:

```
public bool deleteItemsInventory(int[] itemIDs)
{
    //deletes a list of items from inventory
    try
    {
        //loop through the list of itemIDs
        for (int x = 0; x < itemIDs.Length; x++)
        {
            //This uses a lambda expression query to create a inventory table object that corresponds to the current item id
            InventoryTable deleteitem = DataContextInv.InventoryTables.FirstOrDefault(e => e.ItemID.Equals(itemIDs[x]));
            //delete the item
            DataContextInv.InventoryTables.DeleteOnSubmit(deleteitem);
            DataContextInv.SubmitChanges();
        }
        return true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
}
```

I created another method here that does the same thing but for the for sale table.

In order to call this method, I will need to convert the selected items in the data grid into an array of item IDs corresponding to the items selected. Once the delete is finished, it should update the table causing the deleted record to be removed from the data grid and cause auto updating to resume:

```
private void delete_button_Click(object sender, RoutedEventArgs e)
{
    //Check that an item has actually been selected to be deleted
    if (inventory_dataGrid.SelectedItem != null)
    {
        //setup a list of item IDs , which have a datatype of integer
        List<int> itemIDs = new List<int>();
        //loop through the selected items
        foreach (InventoryTable item in inventory_dataGrid.SelectedItems)
        {
            //add the item ID of each item to the itemID list
            itemIDs.Add(item.ItemID);
            Console.WriteLine(item.ItemID.ToString());
        }

        if (SqlLink.deleteItemsInventory(itemIDs.ToArray()))
        {
            MessageBox.Show("deleted");
            Console.WriteLine("Updating table on delete");
            //update the tables data to reflect the delete, also resumes auto updating
            inventory_dataGrid.ItemsSource = SqlLink.getNewInventoryTable(category_comboBox.Text.Replace(" ", string.Empty), name_textBox.Text);
        }
    }
}
```

The same method was used in the for sale table delete button event handler. It also took the selection from the for sale table data grid.

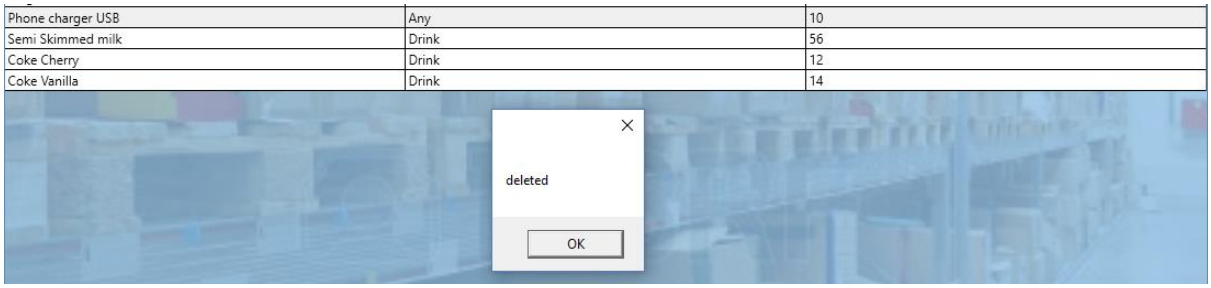
To test deleting items, I will delete a selected item from the inventory data grid. After selecting the item, I will click delete selected items. After a success message should be shown and the data grid no longer have the item shown.

The data grid before any items are deleted:

| Item Name | Category | Quantity |
|---|---|---|
| Sprite | Drink | 100 |
| Coke | Drink | 200 |
| Kingsmeal White Bread | Food | 34 |
| Phone charger USB | Any | 10 |
| Semi Skimmed milk | Drink | 56 |
| Coke Cherry | Drink | 12 |
| Coke Vanilla | Drink | 14 |

I will now delete the phone charger USB item.

Shown below is the success message shown when I selected the item clicked the delete selected items button:

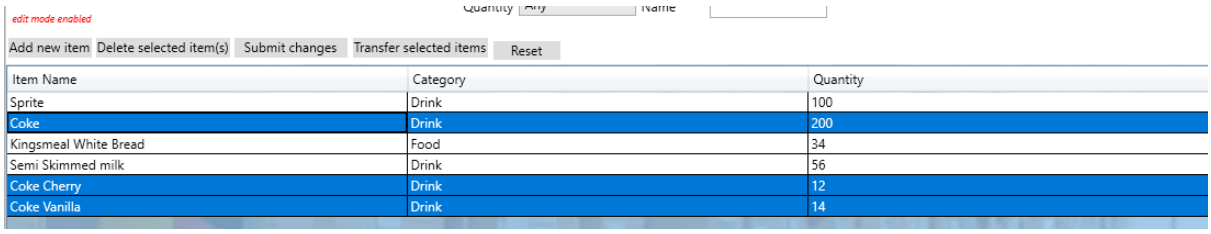



The item is no longer shown in the data grid:

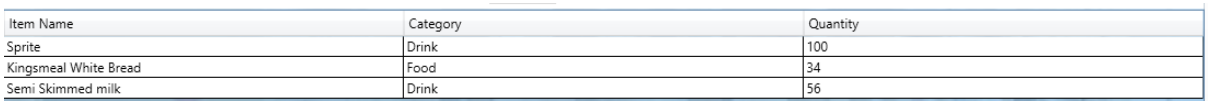
| | | |
|---|---|---|
| Sprite | Drink | 100 |
| Coke | Drink | 200 |
| Kingsmeal White Bread | Food | 34 |
| Semi Skimmed milk | Drink | 56 |
| Coke Cherry | Drink | 12 |
| Coke Vanilla | Drink | 14 |

Now I will delete all three coke items at once:

I have selected all three items by holding down the control button on my keyboard and clicking the three coke items:




I then clicked the delete selected items button and was shown the success message. The three items are now removed from the data grid:



## 5 Creating user accounts

Each of user of the system should have a unique account. When the program is first started, they will be required to login into system, using their username and chosen password. This will allow the user to be identified by the system, so that actions they make whilst using the system can logged. In addition they will be given a role. Permissions are to be setup which determine what the user can do with the system.

I have already created a create account user control for the login window. I only need to take the inputs from this user control and insert a new record into the databases accounts table with the details from the input.

Usernames must be unique so I need to check if the username entered by the username is taken or not. I can do this by querying for the specific username, if the query returns anything then the username is taken.

In the SQLServerAccessor, I created the following method:

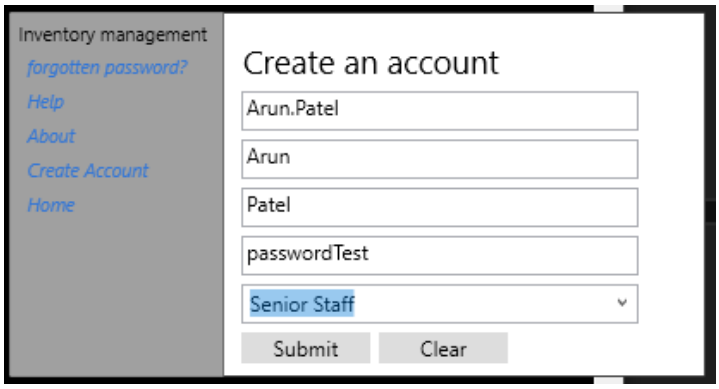
```
public bool CreateNewAccount(string username, string firstname, string lastname, string password, string role)
{
    //search for an existing user with the same username
    var accQuery = from u in DataContextInv.AccountsTables
                   where u.Username.Equals(username)
                   select u;
    if (accQuery.Count() == 0)  //if the query returns more than 0 , the username is taken
    {
        //create a new Accounts table instance and fill it with the passed values
        AccountsTable newuser = new AccountsTable();
        newuser.Username = username;
        newuser.FirstName = firstname;
        newuser.LastName = lastname;
        newuser.Password = password;
        newuser.Role = role;
        DataContextInv.AccountsTables.InsertOnSubmit(newuser);
        DataContextInv.SubmitChanges();
        return true;
    }
    else //the username was taken so return false
    {
        return false;
    }

}
```

The parameters of this method will need to originate from the input fields on the login create an account user control.

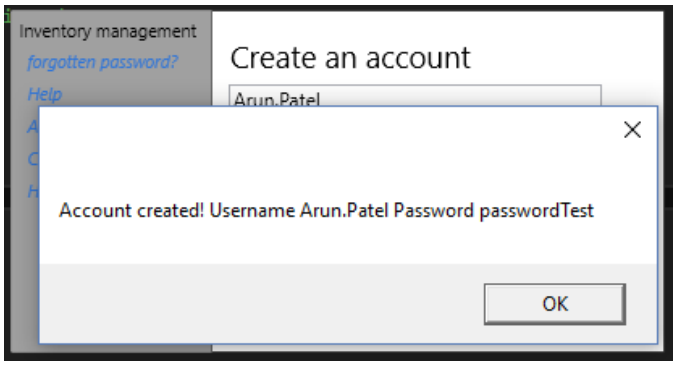
In the event handler for submit button in the login create an account user control, I will call the CreateNewAccount method, passing the values of each input control:

```
private void submit_button_Click(object sender, RoutedEventArgs e)
{
    //check each field is not its default value
    if ((username_textBox.Text != "USERNAME") && (firstname_textBox.Text != "FIRST NAME")
        && (lastname_textBox.Text != "LAST NAME") && (password_textBox.Text != "PASSWORD")
        && (role_ComboBox.Text != "ROLE")
        )
    {
        if (SqlLink.CreateNewAccount(username_textBox.Text, firstname_textBox.Text, lastname_textBox.Text, password_textBox.Text, role_ComboBox.Text))
        {
            MessageBox.Show("Account created! Username " + username_textBox.Text + " Password " + password_textBox.Text);
        }
        else
        {
            MessageBox.Show("Username is already taken , please choose another");
        }
    }
    else
    {
        MessageBox.Show("You must fill out all fields");
    }

}
```

To test this, I will create a new account for myself:




After clicking on the submit button, I was shown the following message:




In the SQL Server Management Studio, the new account is shown as a record in the Accounts table:

| | | | | | |
|---|---|---|---|---|---|
| Arun.Patel ... | passwordTest ... | Arun | Patel | Senior Staff ... | |

When a user attempts to login to the system, the program needs to authenticate their login by checking the password they have entered is correct for the associated username.

**This test demonstrates that test number 10 of my test plan was successful.**

To check if the password was correct or the username was valid, I can query the database for the exact username and password entered by the user. If the query returns nothing then either password was incorrect or the username doesn't exist, so the login was not authenticated. This behaviour has been implemented by the following method in the SQLServerAccessor class:

```
public bool AuthenticateLogin(string Username, string Password)
{
    //the method returns true if the login was authenticated
    Console.WriteLine("authenticating login");
    //Query the accounts table for a username with the entered username and the entered password
    var accQuery = from u in DataContextInv.AccountsTables
                   where u.Username.Equals(Username) &&
                   u.Password.Equals(Password)
                   select u;
    if (accQuery.Count() != 0)
        //if the query returns nothing , the username and password does not exist together in a record
        //so the username or password was incorrect
    {
        Console.WriteLine("Authenticated");
        return true;
    }
    else
    {
        Console.WriteLine("Authentication failed");
        return false;
    }
}
```

Now, when the user clicks on the login button, I can call this function, passing the username and password entered. If the function returns false I can set the error label to a suitable message, if it returns true I can instantiate the main window and show it the user whilst closing the login window.

In order for the main window to be aware of who the user is, I will require the username to be passed to the window when instantiating it. I can do this by setting the parameter of its constructor to the username:

```
public MainWindow(string _username)
```

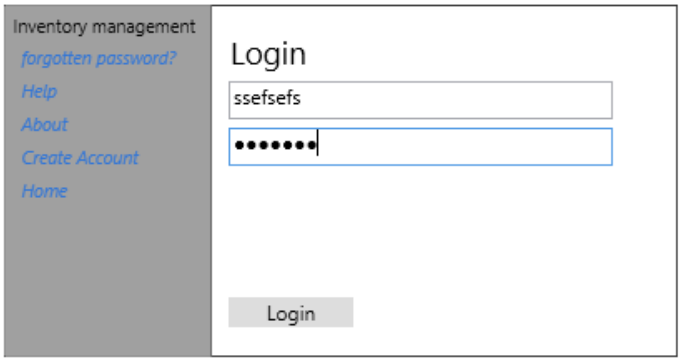
When the window is loaded I will show a welcome message to the user:

```
MessageBox.Show("Welcome " + _username);
windowtitle.Text += " Welcome " + _username;
```

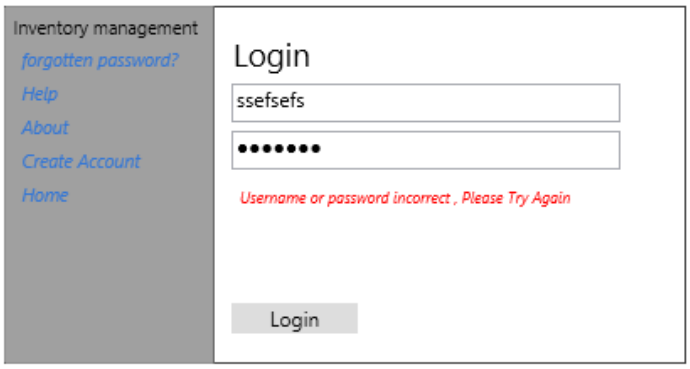
The new login button event handler is shown below:

```
private void login_button_Click(object sender, RoutedEventArgs e)
{
    if (SqlLink.AuthenticateLogin(username_textBox.Text, password_passwordBox.Password))
    {
        //create an instance of the main window and show it if the login was authenticated
        MainWindow mw = new MainWindow(username_textBox.Text);
        mw.Show();
        var myWindow = Window.GetWindow(this);
        myWindow.Close();
    }
    else
    {
        error_label.Content = "Username or password incorrect , Please Try Again";
    }
}
```

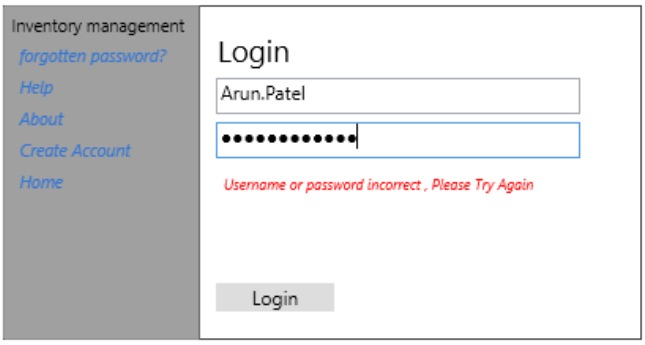
To test this, I will firstly enter a random username and password. This shouldn't authenticate and should cause the error label to show an error message:



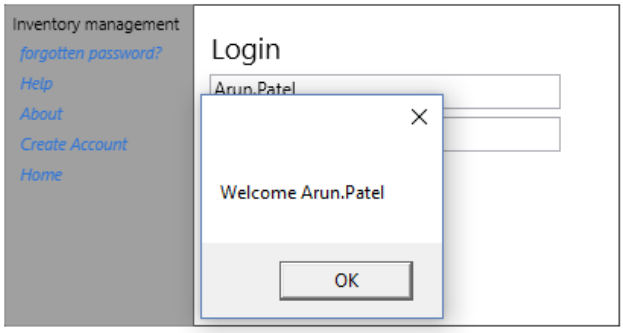

After clicking the login button:



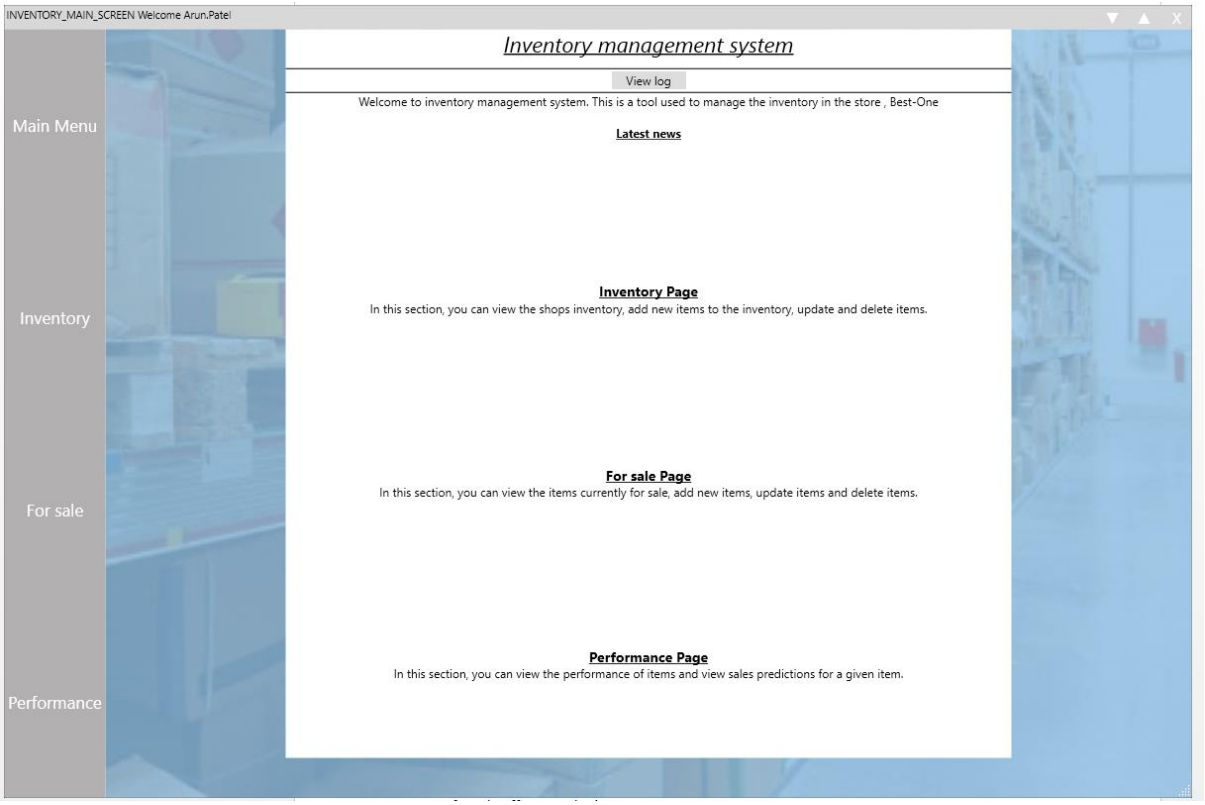

This behaviour is correct. I will now enter the username and password I created previously:




After clicking the login button, the main screen loaded and I was shown a welcome message:




**This test demonstrates that test number 11 of my test plan was successful.**

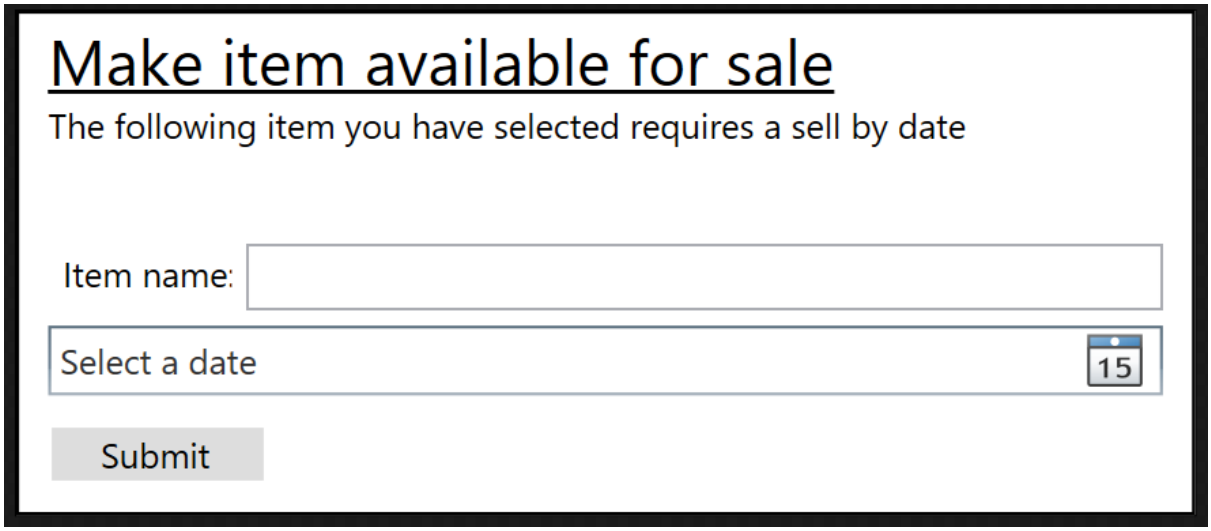



## 6 Transferring items from inventory to the for sale table

Items in the inventory table will need to be transferred to the for sale table when they have been made available for sale. For some items, a sell by date is required. In addition, the user should be transfer a specific quantity of an item at time instead of all of the item if needed.

To start with I will create a new window. This window will be used to enter the sell by date of those items being transferred.

The created window is shown below in the designer view:




This window will be given the item to add the sell by date to, therefor in the constructor of this window, I have added the parameter of an Inventory table object:

```
public transferItemWindow(InventoryTable item)
{
    InitializeComponent();
    itemname_textBox.Text = item.ItemName;
```

The item name textbox is then set the name of the item as shown above.

Back in the inventory user control, in the event handler for the transfer item button, I have created to lists. One list for items that don't require a sell by date and another for those that do:

```
List<InventoryTable> ItemsToTrans = new List<InventoryTable>();
List<InventoryTable> ItemsToTransSellByDate = new List<InventoryTable>();
```

To determine which items need to go in which list, I will iterate through the users selection (on the data grid) and check to see if the category of the item is either food or drink. If it in these categories then I will add the item to the list of items that require a sell by date (ItemsToTransSellByDate), if it isn't then I will add it to the other list:

```
List<InventoryTable> ItemsToTrans = new List<InventoryTable>();//no sell by date list
List<InventoryTable> ItemsToTransSellByDate = new List<InventoryTable>(); //sell by date list
if (inventory_dataGrid.SelectedItem != null)
{
    if (inventory_dataGrid.SelectedItems != null)
    {
        //iterate through the users selection
        foreach (InventoryTable item in inventory_dataGrid.SelectedItems)
        {
            //if the item is a food or drink item , add it to the sell by date list
            if ((item.CategoryName.Replace(" ", "") == "Food") || (item.CategoryName.Replace(" ", "") == "Drink"))
            {
                ItemsToTransSellByDate.Add(item);
            }
            else
            {
                ItemsToTrans.Add(item);
            }
        }
    }
}
```

I can then unselect the data grid, to allow for automatic updating to resume:

```
inventory_dataGrid.UnselectAll();
updateAll();
```

If the sell by date list has items in, then I need to instantiate the transfer window, sending each item to it. To do this I will iterate through the list, on each iteration I will create a new instance of the transfer window, sending the item to it as a parameter.

```
if (ItemsToTransSellByDate.Count != 0)
    //if there are any items to transfer with a sell by date
{
    foreach (InventoryTable itemToPushWithSellByDate in ItemsToTransSellByDate)
    {
       //iterate through the list and instantiate the new window , sending in each item as a parameter
        transferItemWindow transitemsellby = new transferItemWindow(itemToPushWithSellByDate);
        transitemsellby.MainUserName = _MainUsername;
        transitemsellby.Show();
    }
```

In the transfer window submit button event handler, I need to check that the date picker has not been left empty before transferring the item.

```
if (sellbydate_datepicker.SelectedDate == null)
{
    MessageBox.Show("You must enter a sell by date in order to submit");
}
```

To actually transfer the item, I will create a new method in the SQLServerAccessor class.

In some cases, the quantity of the item being transferred may be less than the entire quantity of the item held in inventory. I will need to calculate the difference between the quantity of the item in the inventory table and the quantity of the item being transferred. The items record in inventory should then be updated, so that its quantity is reduced and the transferred quantity of that item should be transferred to the for sale table.

To do this I will create an inventory table object set to the value of the item to be transferred, that is currently in inventory.

```
//get the transferred item from inventory first
var oldItem = DataContextInv.InventoryTables.Single(x => x.ItemID == itemToTransfer.ItemID);
```

If the quantity of the item being transferred is less than the quantity of the item in inventory then I need to update the item in inventory first and then create a for sale table object with the relevant details from the parameters of this method:

```
public void TransferItemsToForWITHsellByDate(InventoryTable itemToTransfer, DateTime sellbydate)
```

```
if (itemToTransfer.Quantity < oldItem.Quantity)
{
    //update the quantity of the item in inventory
    oldItem.Quantity = oldItem.Quantity - itemToTransfer.Quantity;
    ForSaleTable transitem = new ForSaleTable();
    transitem.ItemName = itemToTransfer.ItemName;
    transitem.Quantity = itemToTransfer.Quantity;
    transitem.SellByDate = sellbydate;
    transitem.CategoryName = itemToTransfer.CategoryName;
    DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
    DataContextInv.SubmitChanges();

}
```

If the user is transferring all of the item then the item just needs transferring:

```
else
{
    ForSaleTable transitem = new ForSaleTable();
    transitem.ItemName = itemToTransfer.ItemName;
    transitem.Quantity = itemToTransfer.Quantity;
    transitem.SellByDate = sellbydate;
    transitem.CategoryName = itemToTransfer.CategoryName;
    DataContextInv.InventoryTables.DeleteOnSubmit(oldItem);
    DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
    DataContextInv.SubmitChanges();

}
```

I have also setup another method for use when transferring an item that doesn't require a sell by date:

```
public void TransferItemsToForSaleNoSellBy(List<InventoryTable> ListOfItemsToTransfer)
{
    foreach (InventoryTable nextItem in ListOfItemsToTransfer)
    {
        var oldItem = DataContextInv.InventoryTables.Single(x => x.ItemID == nextItem.ItemID);
        Console.WriteLine("is " + nextItem.Quantity + " < " + oldItem.Quantity);
        if (nextItem.Quantity < oldItem.Quantity)
        {
            oldItem.Quantity = oldItem.Quantity - nextItem.Quantity;
            ForSaleTable transitem = new ForSaleTable();
            transitem.ItemName = nextItem.ItemName;
            transitem.Quantity = nextItem.Quantity;
            DateTime? newdate = null;
            transitem.SellByDate = newdate;
            transitem.CategoryName = nextItem.CategoryName;
            DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
            DataContextInv.SubmitChanges();
        }
        else
        {
            ForSaleTable transitem = new ForSaleTable();
            transitem.ItemName = nextItem.ItemName;
            transitem.Quantity = nextItem.Quantity;
            DateTime? newdate = null;
            transitem.SellByDate = newdate;
            transitem.CategoryName = nextItem.CategoryName;
            DataContextInv.InventoryTables.DeleteOnSubmit(oldItem);
            DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
            DataContextInv.SubmitChanges();
        }
    }

}
```
#

Back in the transfer window, once it has been confirmed that the date picker is not empty, I can then call the transfer items with sell by date method in the SQLServerAccessor:

```
private void submit_button_Click(object sender, RoutedEventArgs e)
{
    if (sellbydate_datepicker.SelectedDate == null)
    {
        MessageBox.Show("You must enter a sell by date in order to submit");
    }
    else
    {
        SqlLink.TransferItemsToForWITHsellByDate(globItem, sellbydate_datepicker.SelectedDate.Value
        this.Close();
    }
}
```

The window is closed once the item has been transferred.

Once the for loop in the inventory user control has finished, the items that didn't require a sell by date can then be transferred to the for sale page by calling the transfer items button without sell by date method in the SQLServerAccessor class:

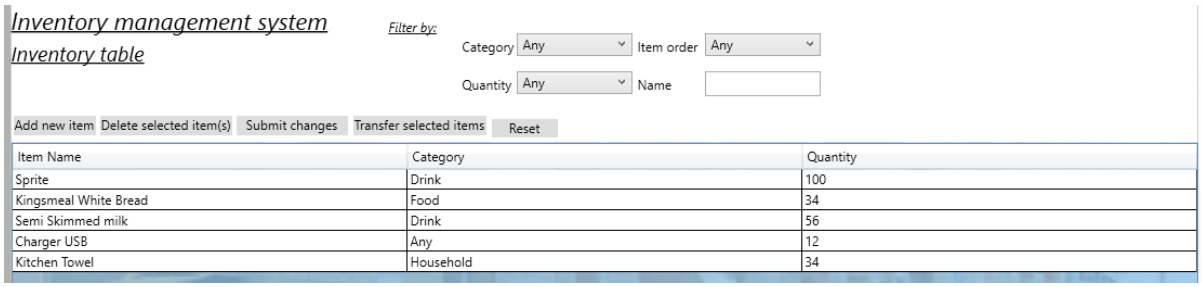
```
//transfer the items that dont have a sell by date
SqlLink.TransferItemsToForSaleNoSellBy(ItemsToTrans
```

If no items were selected that required a sell by date, then the same method can be called directly:

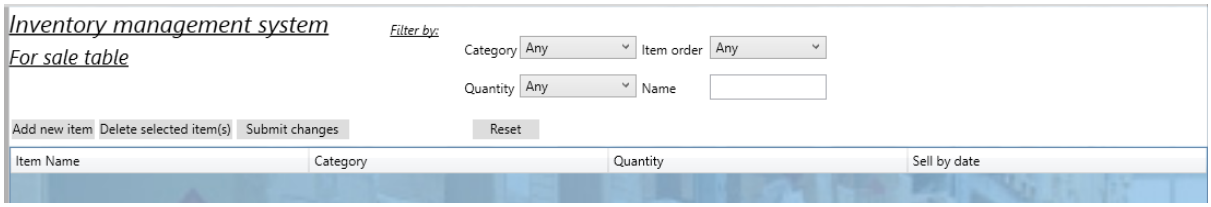
```
else
{
    //only transfer items that have no sell by date
    SqlLink.TransferItemsToForSaleNoSellBy(ItemsToTrans,
```

To test this feature, I will firstly transfer a specified quantity of an item .

The images below shows the inventory data grid and for sale table before any transfers have been made:

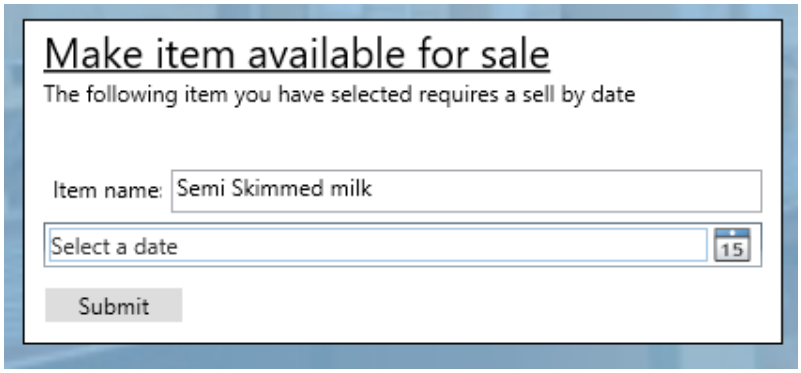



,

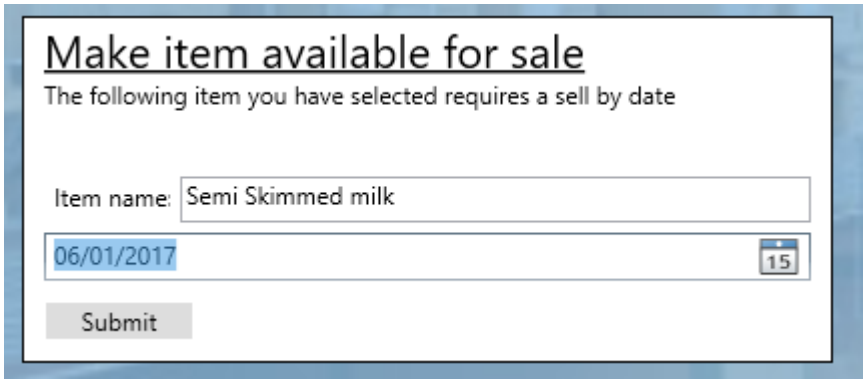



I have selected 10 semi skimmed milk:

When clicking the transfer button the transfer window was shown:



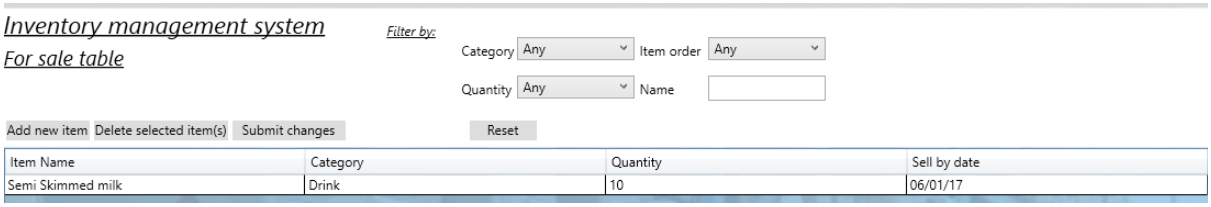

I added a sell by date to the item:



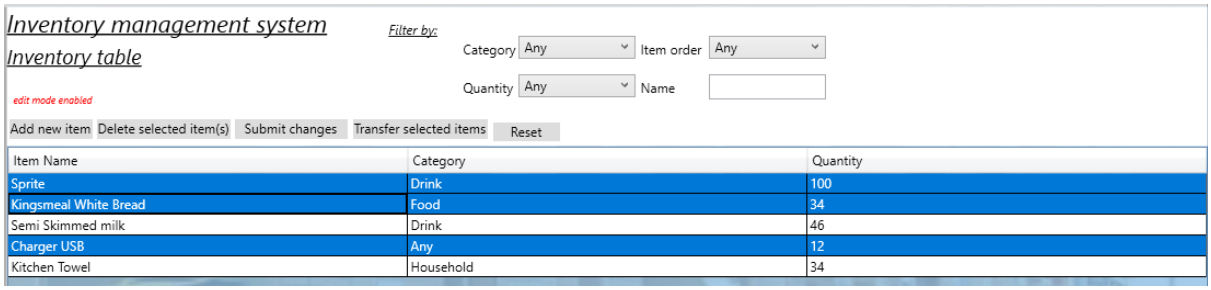

After clicking submit button the semi skimmed milk item now had a quantity 10 less than it did before (46):

| | | |
|---|---|---|
| Semi Skimmed milk | Drink | 46 |

The 10 items transferred were now shown in the for sale data grid with the inputted sell by date:




I will now transfer multiple items at the same time. I have selected 3 items to transfer:

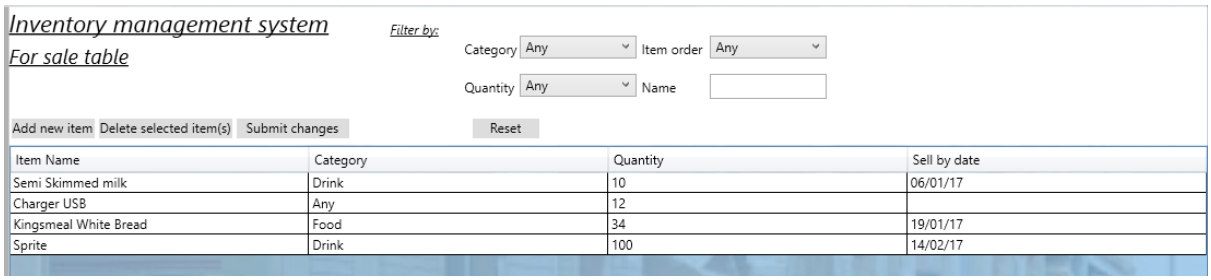



When clicking submit I should be asked to enter the sell by date for the Sprite and Kings Meal white bread. The result of clicking the transfer item button is shown below:

Entering the sell by date for Kings Meal white bread:

I then added the sell by date for Sprite

The for sale table data grid now looks as follows:



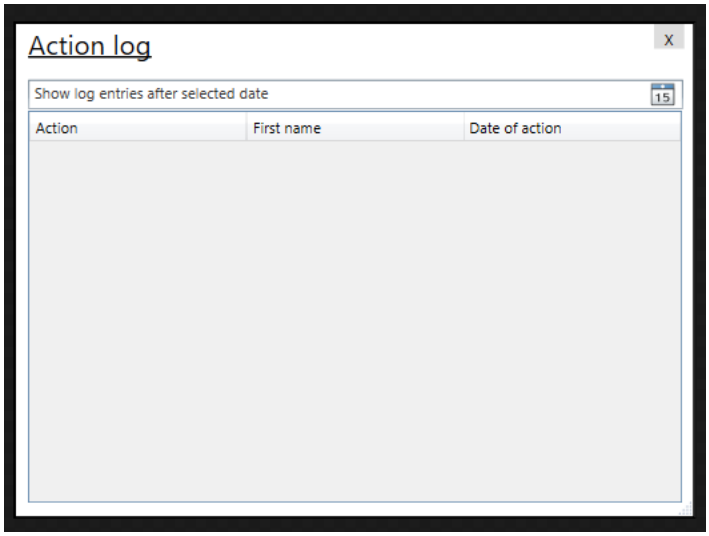

One use of only transferring a set amount of an item, is when some of the same items have different sell by dates. For example, 10 Kings Meal white bread might have a sell by date different to the remaining 30. The user can add the 10 items first, enter the sell by date then add the remaining 30 with different sell by dates.

## 7 Action logging

One of the requirements for this solution was that it should be able to log the actions of users. The log feature accessed by pressing the log button on the main page.

Firstly I need to create a window for showing the log.

The window I have created is shown below in the designer view:

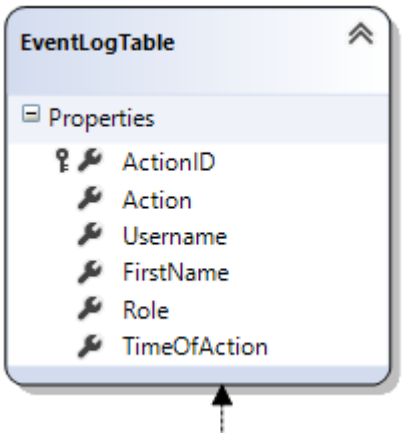



As with before, I have used a style to set the default text of the date picker:

```
<Style TargetType="{x:Type DatePickerTextBox}">
    <Setter Property="Text" Value="Show log entries after selected date" />
    <Setter Property="IsReadOnly" Value="True"/>
</Style>
```

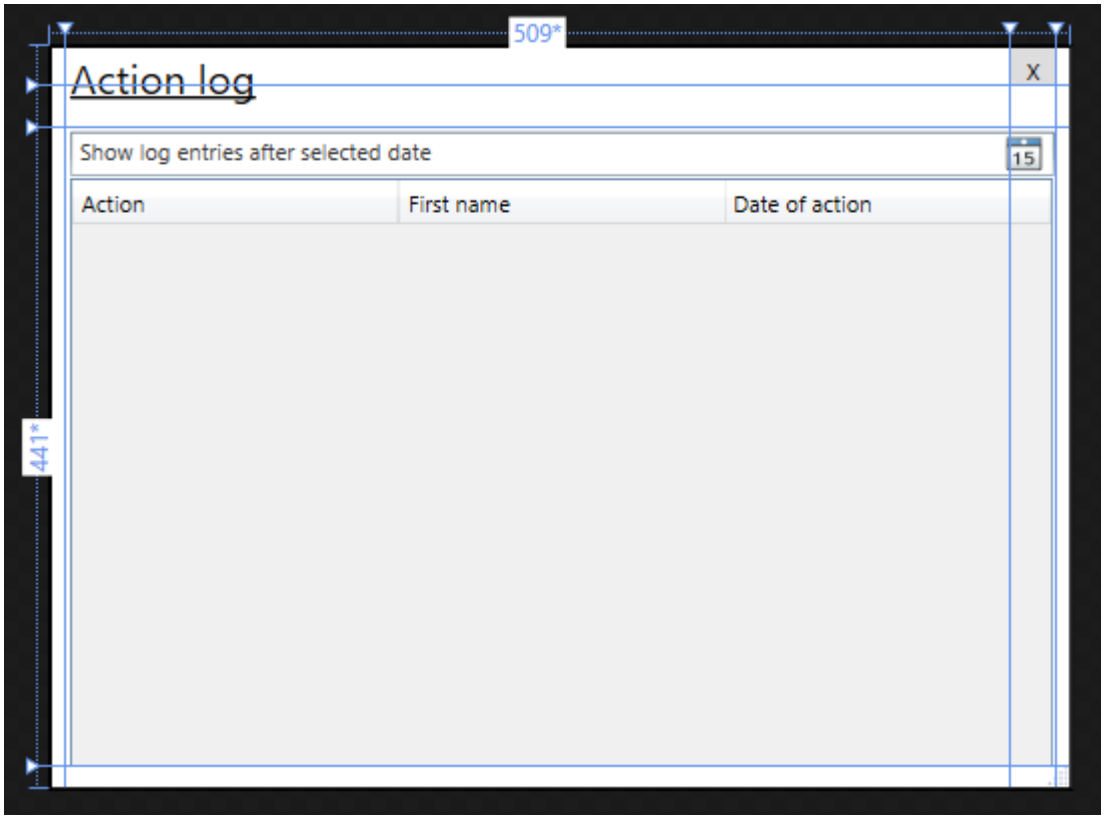
The data grids columns were created to bind to the Event log table object, so I set the binding path equal to the properties of this object:

```
<DataGrid.Columns>
    <DataGridTextColumn
    IsReadOnly="False"
    Header="Action"
    Binding="{Binding Path=Action}"
    Width="*"
    />
    <DataGridTextColumn
    IsReadOnly="False"
    Header="First name"
    Binding="{Binding Path=FirstName}"
    Width="*"
    />
    <DataGridTextColumn
    IsReadOnly="False"
    Header="Date of action"
    Binding="{Binding Path=TimeOfAction, StringFormat=\{0:dd/MM/yy\}}"
    Width="*"
    />

</DataGrid.Columns>
```




I used the string format again to prevent the time of the date from showing in the column.

This window is also resizable so I have setup grid columns and rows as shown below:




To populate the event log table with event logs, I need to generate a log entry when an action takes place, such as adding an item into inventory.

I will setup a log entry for adding an item into inventory. Since the event requires the username of the user, I will add this a parameter to inventory adding method:

```
public bool addNewItemInventory(string _itemName, int _quantity, string Category, string username)
```

In the add new item window, which is the window which called the add new item to inventory method, I have created a global variable called username:

```
public partial class AddNewItemWindow : Window
{
    SQLServerAccessor SQLlink = new SQLServerAccessor();
    public string mainUsername;
```

This global variable is then passed to the add new item method when it's called:

```
(SQLlink.addNewItemInventory(ItemName_textBox.Text, Convert.ToInt32(Quantity_textBox.Text), Category_comboBox.Text.Replace(" ", string.Empty), mainUsername))
```

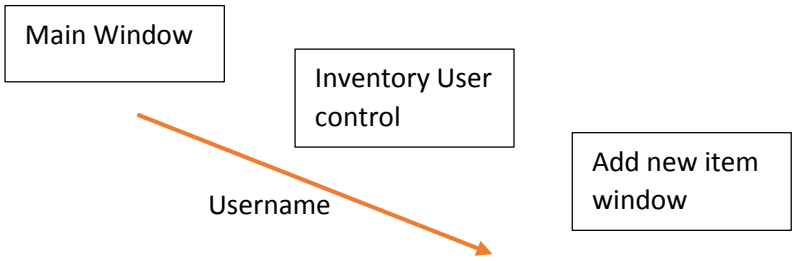
The add new item window is instantiated by the inventory user control.

In the inventory user control I added a global variable called username to it:

```
SQLServerAccessor SqlLink = new SQLServerAccessor();

PeformanceAnalysis perf = new PeformanceAnalysis();
DispatcherTimer dispatcherTimer = new DispatcherTimer();

public string _MainUsername;
```

This user control is instantiated by the main window, which already is given the username of the user. It also contains a global variable for the user name which is set the actual username during instantiation.

To summarise, the following diagram shows how the username is given to the add new item window




I have created a new method in the SQLServerAccessor class for adding new log entry. The parameters passed to it include the username of the user and message for the log entry. The message will depend on the action being made.

To call this function for adding new items to inventory I used the following function call:

```
string message = username + " Has added " + _itemName + " to inventory," + " quantity " + _quantity + " category " + Category;
AddLogEntry(username, message);
```

This function call was made for every method that performed an action in the SQLServerAccessor class, however the message for each one differed depending on what the action did.

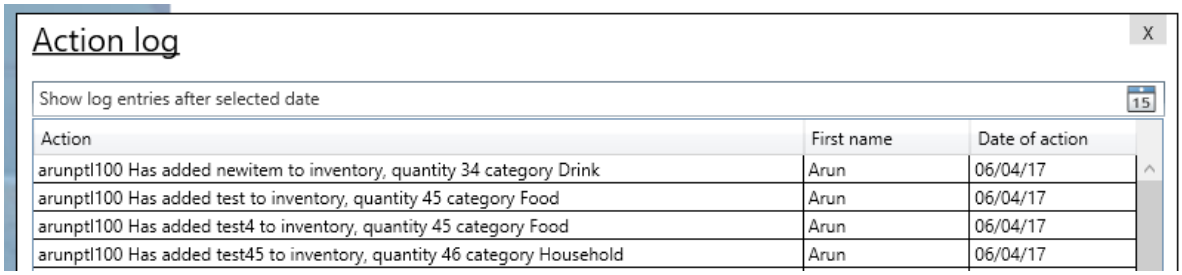
The user will have the option to only show actions that took place after a certain date. In the SQLServerAccessor I added a method that returns the log entries that took place after a date that is passed to the method:

```
public List<EventLogTable> GetLogEntriesAfterDate(DateTime ShowAferDate)
{
    //A list of the event logs
    List<EventLogTable> eventLog = DataContextInv.EventLogTables.ToList();
    //A list of all the event logs that are after the given date , initially it is empty
    List<EventLogTable> sortedLog = new List<EventLogTable>();
    //iterate through eventlog
    foreach (EventLogTable Log in eventLog)
    {
        Console.WriteLine(Log.TimeOfAction.Date);
        //if the date of the log is after the given date
        if (Log.TimeOfAction.Date > ShowAferDate.Date)
        {
            //add it to the sorted log
            sortedLog.Add(Log);
        }
    }
    return sortedLog;
}
```

In the log window, I added a calendar closing event to the date picker. This event will be fired when the user closes the date picker. In the event handler for this event I set the item source of log data grid to the result of calling the get log entries after date method in the SQLServerAccessor class:

```
private void log_DatePicker_CalendarClosed(object sender, RoutedEventArgs e)
{
    if (log_DatePicker.SelectedDate != null)
    {
        log_dataGrid.ItemsSource = SqlLink.GetLogEntriesAfterDate(log_DatePicker.SelectedDate.Value);
        log_dataGrid.Items.Refresh();
    }
}
```

After adding an item to inventory, the log window looks as follows:

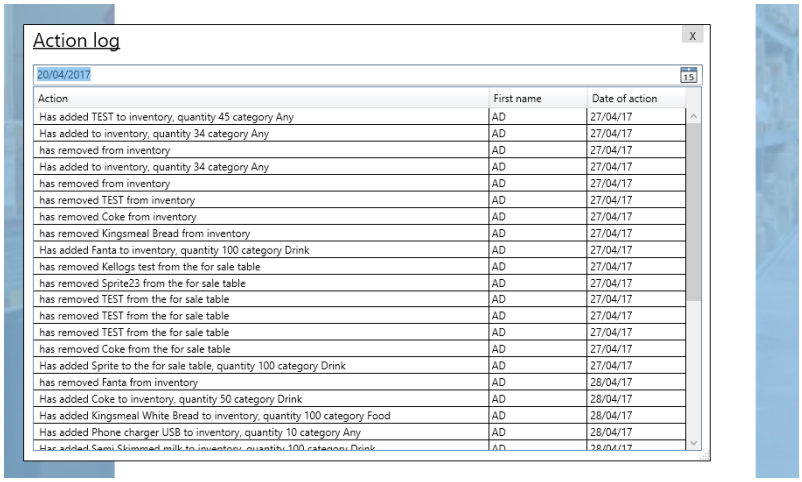


Action log

Show log entries after selected date

| Action | First name | Date of action |
|---|---|---|
| arunptl100 Has added newitem to inventory, quantity 34 category Drink | Arun | 06/04/17 |
| arunptl100 Has added test to inventory, quantity 45 category Food | Arun | 06/04/17 |
| arunptl100 Has added test4 to inventory, quantity 45 category Food | Arun | 06/04/17 |
| arunptl100 Has added test45 to inventory, quantity 46 category Household | Arun | 06/04/17 |

To only show log entries after the 20/04/17 I set this date in the date picker, which resulted in only these action logs showing:



Action log

20/04/2017

| Action | First name | Date of action |
|---|---|---|
| Has added TEST to inventory, quantity 45 category Any | AD | 27/04/17 |
| Has added to inventory, quantity 34 category Any | AD | 27/04/17 |
| has removed from inventory | AD | 27/04/17 |
| Has added to inventory, quantity 34 category Any | AD | 27/04/17 |
| has removed from inventory | AD | 27/04/17 |
| has removed TEST from inventory | AD | 27/04/17 |
| has removed Coke from inventory | AD | 27/04/17 |
| has removed Kingsmeal Bread from inventory | AD | 27/04/17 |
| Has added Fanta to inventory, quantity 100 category Drink | AD | 27/04/17 |
| has removed Kellogs test from the for sale table | AD | 27/04/17 |
| has removed Sprite23 from the for sale table | AD | 27/04/17 |
| has removed TEST from the for sale table | AD | 27/04/17 |
| has removed TEST from the for sale table | AD | 27/04/17 |
| has removed TEST from the for sale table | AD | 27/04/17 |
| has removed Coke from the for sale table | AD | 27/04/17 |
| Has added Sprite to the for sale table, quantity 100 category Drink | AD | 27/04/17 |
| has removed Fanta from inventory | AD | 28/04/17 |
| Has added Coke to inventory, quantity 50 category Drink | AD | 28/04/17 |
| Has added Kingsmeal White Bread to inventory, quantity 100 category Food | AD | 28/04/17 |
| Has added Phone charger USB to inventory, quantity 10 category Any | AD | 28/04/17 |
| Has added Semi Skimmed milk to inventory, quantity 100 category Drink | AD | 28/04/17 |

## 8 Implementing CRUD permissions

In the design section (5.2) I outlined the permissions each user group will have. The CRUD permission has been copied below:

| Permission group | Permissions |
|---|---|
| Admin | Full access to any features of the solution. It is only to be used for debugging purposes. It shouldn't be used by the end users, they shouldn't have access to this user group either. End users can use this permission group at their own risk |
| Manager group | This user group will have the most functionality. They will have access to all features of the solution intended for the end user. Whilst they won't have direct access to the actual database, storing the data, they will have access to the database through the solution.<br>As well as this, they will have access to administrative tools such as tracking who has made changes to the inventory list or for sale list. The actual tracking log will be available to the user in this permissions group.<br>They will also have access to the item analysis section of the solution. Here they can view the performance of products.<br>They will not have access to the developer console. |
| Senior Staff group | This permission group offers most functionality of the solution but misses out some features reserved only for the manager group.<br>Users in this permission group will not be able to update items in the inventory table and will not be able to track who has made changes to any of the tables.<br>In regards to the for sale table, users in this group will be able to perform CRUD operations. In addition, they will be able to move items from inventory to the for sale table. |
| Staff group | This permission group is the most basic group the solution has to offer and offers limited functionality. It Is aimed towards staff members who are new to the shop or staff members who have little experience with computers. They will be able to add or remove items from inventory table and for sale table. |

### 8.1 Manager Permissions

This user group is the only user group available to users that is able to access the log table.

Since each user control only stores the username of each user, I will need to create a method that returns the user group corresponding to a username.

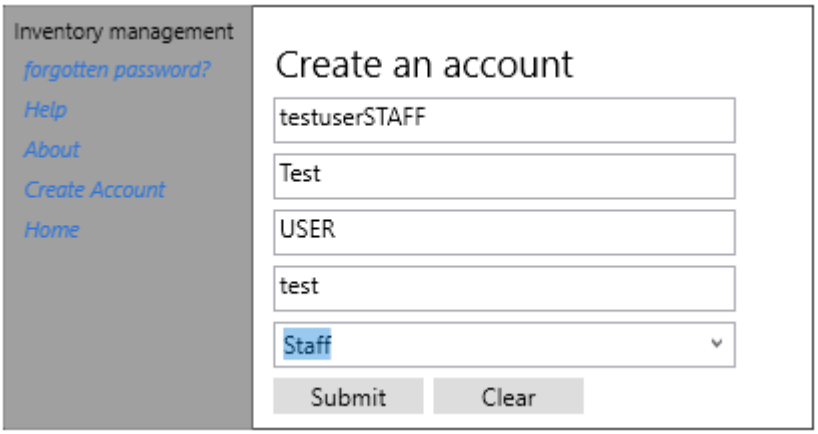
I have created a method in the SQLServerAccessor class that returns the user group of a passed username:

```
public string GetAccountType(string username)
{
    //returns the user group of the passed username
    var UserData = DataContextInv.AccountsTables.Single(x => x.Username == username);
    //remove any whitespace in the returned query
    return UserData.Role.Replace(" ","");
}
```

To implement this permission, when the user clicks on the log button, I will check the user group of the user, if they aren't in the manager or admin group then the program shouldn't show the log window.

```
private void log_button_Click(object sender, RoutedEventArgs e)
{
    //only show the log window if the user is an admin or manager
    if (SqlLink.GetAccountType(MainUsername) == "Manager" || SqlLink.GetAccountType(MainUsername) == "Admin")
    {
        logWindow logwin = new logWindow();
        logwin.Show();
    }
}
```

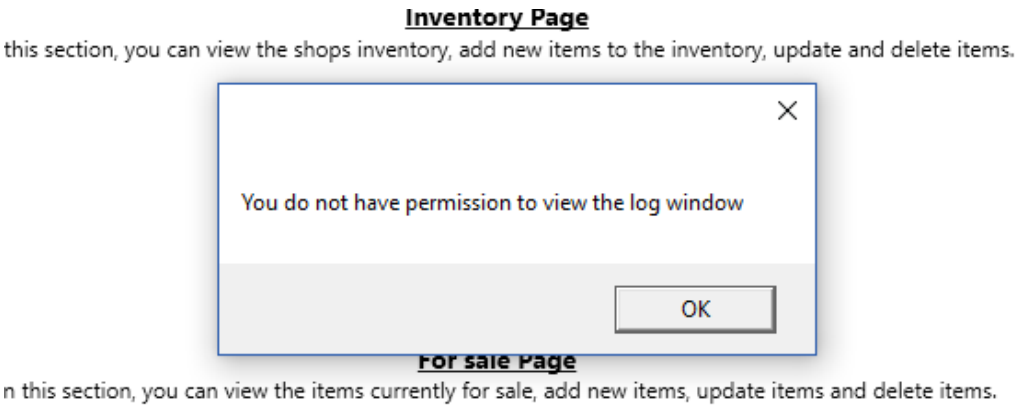
To test this works I will create an account with the user staff user group:




When trying to open the log window whilst logged into this account, nothing happened. This is the intended behaviour. However, I have decided display a message to the user saying that they do not have permission to view the log window. This should make it clear to use why the log window isn't opening.

```
else
{
    MessageBox.Show("You do not have permission to view the log window");
}
```

When logged into the test user account and I try to access the log window, the following message is shown:




When I login using the manager user I created previously, the log window is shown.

This user group is also the only user group available to users that is able to access the performance page. To setup this permission, in the constructor of the main window, I can check if the user is a staff member or senior staff member (the two groups that don't have permission to this feature). If they are, then I can disable the performance button, preventing the user from accessing this feature:

```
//disable the performance button if the user doesnt have permission to access it
if (AccountType == "Staff" || AccountType == "Senior Staff")
{
    Performance_button.IsEnabled = false;
}
```

When logged in as a staff member, when clicking on the performance page button, nothing happens. This is because the user group does not have permission to use this feature.

**This test demonstrates that test number 12 of my test plan was successful.**

## 8.2 Senior Staff Permissions

Users in this group are allowed to update items in the inventory table and for sale table as well as transfer items from inventory to the for sale table.

When the methods for doing these actions are called, the program should check that the user group is not the staff member group (since every other user group is able to use these features).

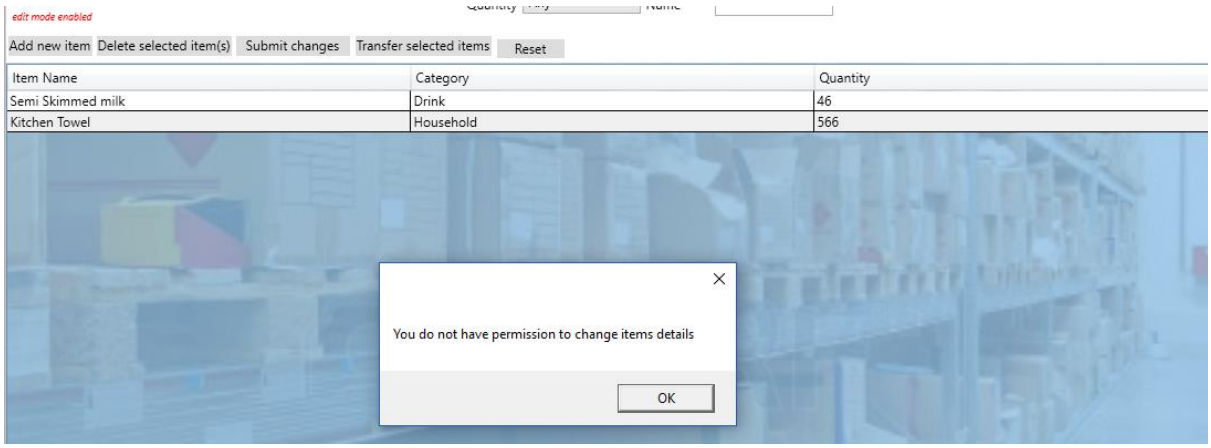
I added this check to the event handlers of the submit buttons in the inventory and for sale table:

```
if (SqlLink.GetAccountType(_MainUsername) != "Staff")
{
```

The same check was added to the transfer items button event handler.

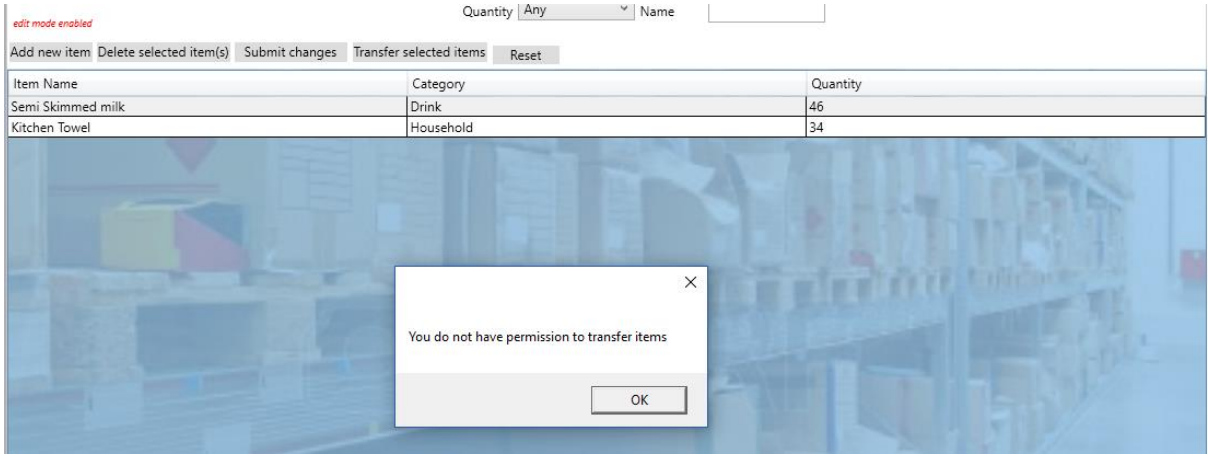
Whilst logged in as the staff member user account, I will try to update an items details.

When clicking on the submit changes button, the following message was shown:




The change was not updated in the database:

When clicking the transfer items button, the following message was shown:




## 8.3 Staff Permissions

No work needs to be done to implement this permissions group. The features a staff user can now access as a result of the permissions already implemented now reflect the permissions they have.

# 9 Performance analysis

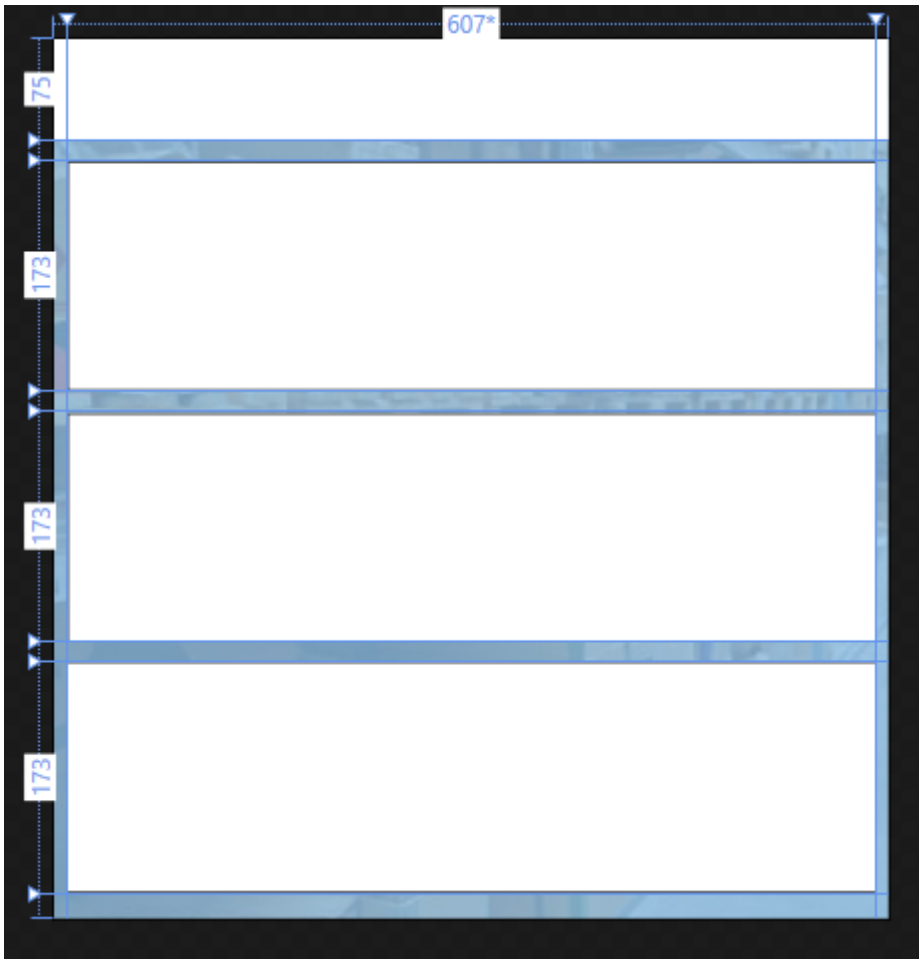
## 9.1 Performance page user control

To begin with I will need to design the performance page user control. It will consist of the three sections. The first section , showing the top 5 best performing items, the middle section showing the top 5 worst performing items and the bottom half dedicated to the performance analysis of an individual item.

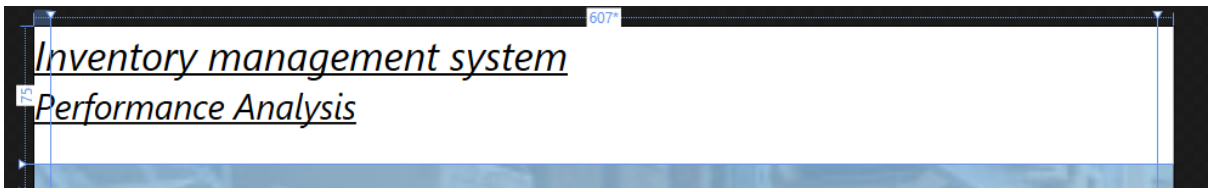
To start with, I have created rows and columns on the user control, creating a top title section and the top middle and bottom sections. Each section will be filled with its own nested grid.

I then filled each space with a white grid. Setting their horizontal and vertical alignments to stretch, causing it to use all of the available space in the grid cell.

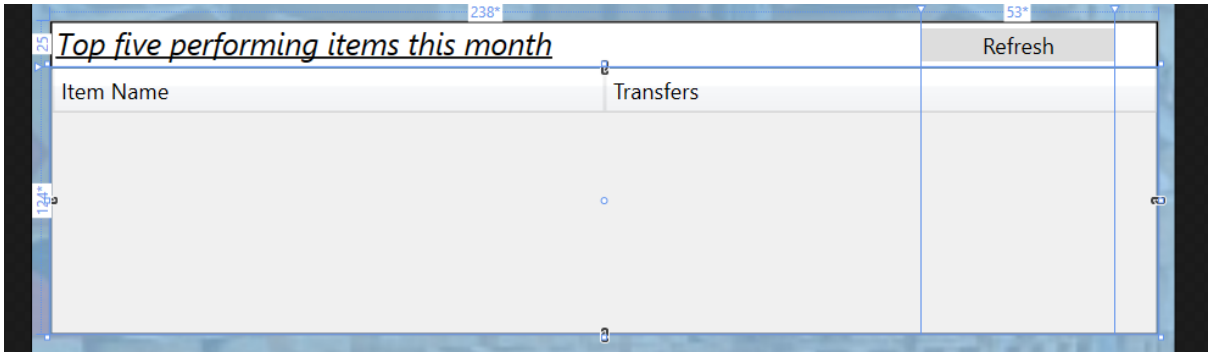
The user control now looks as follows in the designer view:



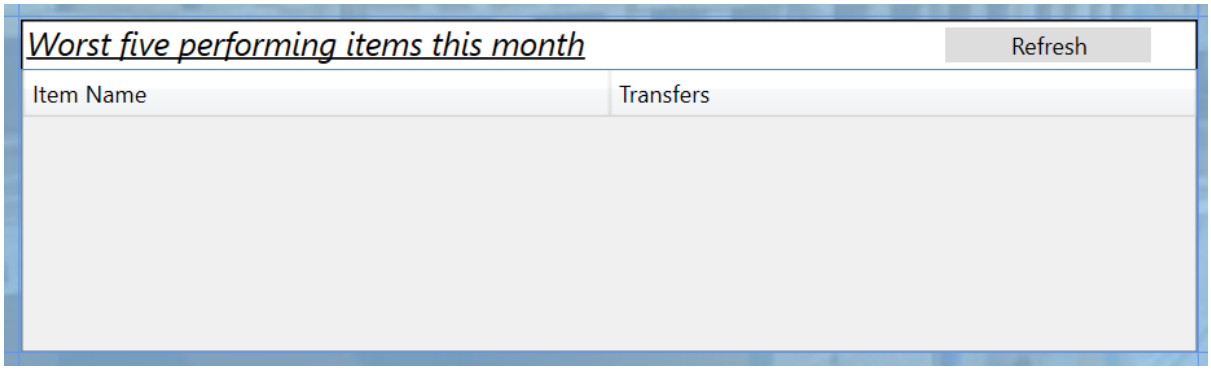

In the top section I added a title and sub title:




In the first section I added a data grid, which will actually display the top five items. After the controls were placed correctly I added row and column definitions to the grid to allow for resizing:




I did the same for the middle section, which displays the top five worst performing items:

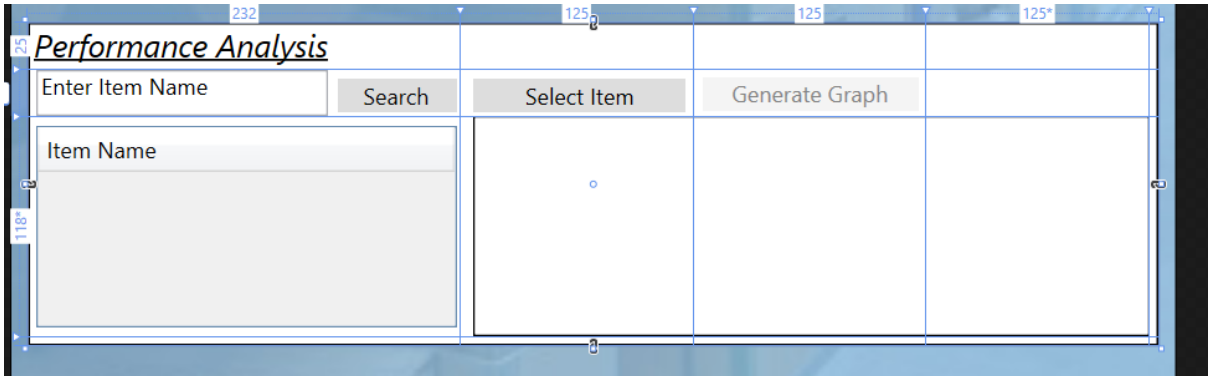



The binding paths for these data grids are as follows:

```
<DataGridTextColumn
IsReadOnly="True"
Header="Item Name"
Binding="{Binding Path=ItemName}"
Width="*"
/>
<DataGridTextColumn
IsReadOnly="True"
Header="Transfers"
Binding="{Binding Path=Transfers}"
Width="*"
```

I will create a new object to bind to the data grid at a later time.

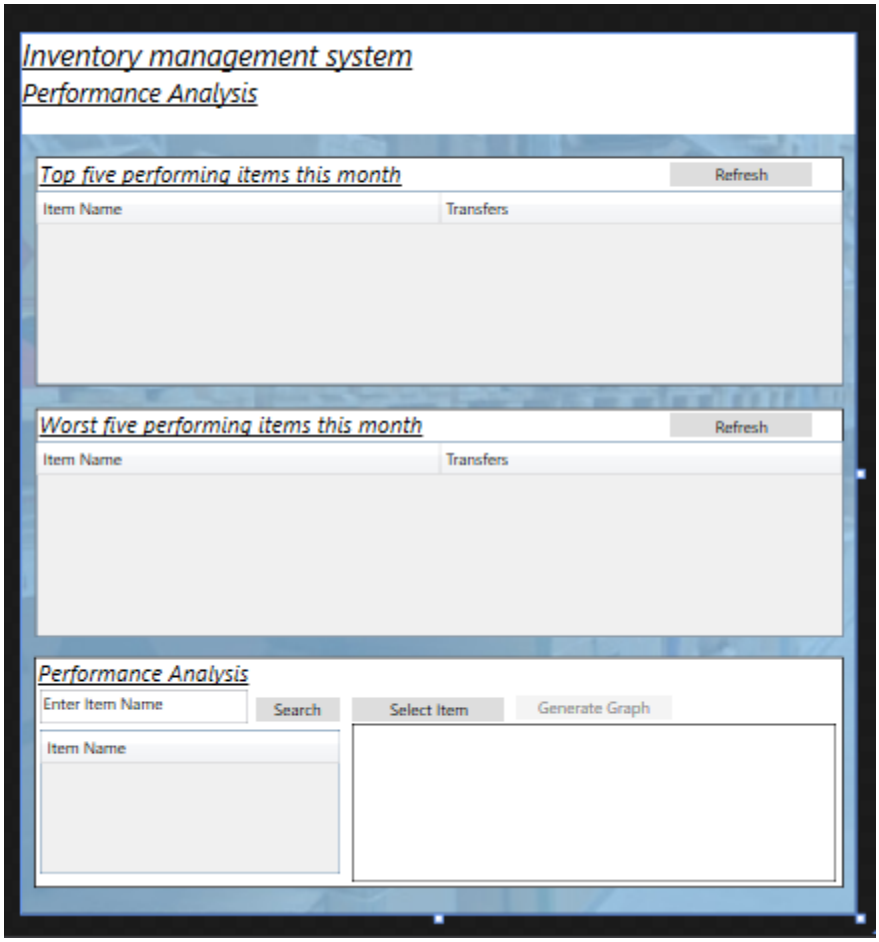
Finally the last section contains a search function, to bring up an analysis of a particular item:



The section allows the user to generate a performance graph. However this button should only be enabled once an item has been selected. Therefore, initially, the button should be disabled:

```
<Button x:Name="generate_graph_button"  IsEnabled="False"
```

The image below shows the performance page user control in the design view:




## 9.2 Creating performance entries

To do all performance operations I have created a new class called PerformanceAnalysis:

```
class PeformanceAnalysis
```

With an instance of the SQLServerAccessor class:

```
SQLServerAccessor SqlLink = new SQLServerAccessor();
```

**The success of an item is measured by how many times it is transferred from inventory to the for sale table (the item is in high demand if it constantly needs to be replenished on the shops shelves).**

Therefore, when an item is being transferred I need to update the performance table's record for that item, incrementing the transfers made for the current month by the amount being transferred

In the SQLServerAccessor class I have created a new method called generate performance log, taking the for sale table object for the item that has been transferred. The method will create a new performance record if none exists for the item, adding the quantity of the item being transferred to

the transfers for the current month column ( for example , if the current month was January then it would add it to the January field).

The method name and parameters are shown below:

```
public void GeneratePerformanceLog(ForSaleTable ActItem)
```

Firstly I will query the performance table to check if a record for the item being transferred already exists:

```
var PerfQuery = from u in DataContextInv.PerformanceTables
                where u.ItemName == ActItem.ItemName
               select u;
if (PerfQuery.Count() == 0) // then a perf record for the item doesnt exist
{
```

If there isn't a record for the particular item, then I need to instantiate a new performance table object, setting it up for the item and adding the number of items being transferred to the current month.

After creating a new instance of the performance table object, I set its item name property to the name of the item being transferred shown below:

```
PerformanceTable PerformanceLog = new PerformanceTable();
PerformanceLog.ItemName = ActItem.ItemName;
```

To add the number of transfers to the current month's column, I need to get the current month. The following line of code sets the string variable to the name of the current month:

```
//returns the current month , e.g. January
string currentMonth = DateTime.Now.ToString("MMMM");
```

For example m if the current month variables value is January, then I need to add the quantity of the item to the January field:

```
if (currentMonth == "January")
{
    PerformanceLog.January += ActItem.Quantity.Value;
}
```

If it was February then the quantity needs to be added to February column:

```
if (currentMonth == "February")
{
    PerformanceLog.February = ActItem.Quantity.Value;
}
```

However, if the perf query showed that there was an existing record for the item, then the records current month value needs to be incremented by the quantity of the item:

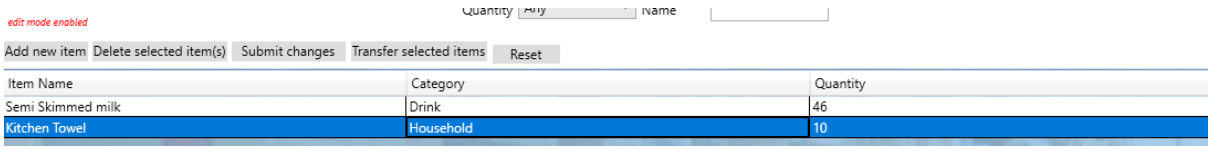
```
else
{
    //get the actual record in the performance table
    var PerfData = DataContextInv.PerformanceTables.Single(x => x.ItemName == ActItem.ItemName);
    string currentMonth = DateTime.Now.ToString("MMMM");
    if (currentMonth == "January")
    {
        //increment the value by the quantity of item
        PerfData.January += ActItem.Quantity.Value;
    }
```

This function will need to be called every time an item is transferred to the for sale table:

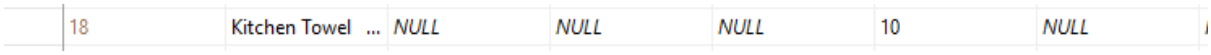
```
GeneratePerformanceLog(transitem);
```

To test this method, I will transfer a small amount of a new item. I will then check in the SQL Server Management Studio to check if the record was created. I will then transfer the rest of the item to the for sale table. The record should now be updated, showing the incremented value for the current month.

Transferring 10 Kitchen towels to the for sale table:

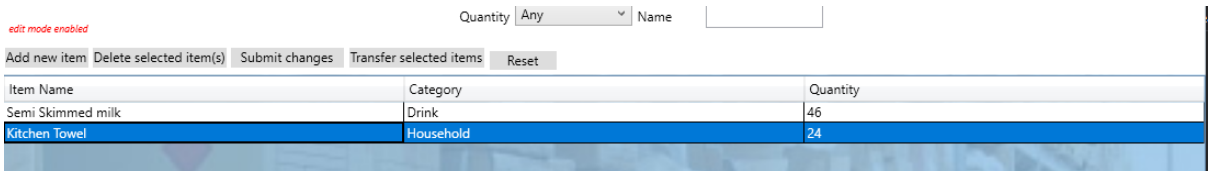



The record was created and shown in the performance table:

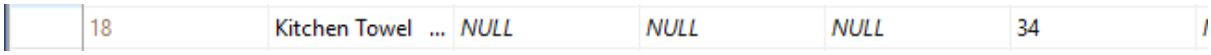



The field with 10 in corresponds to the April column and the number of items that were transferred.

Transferring the rest of the Kitchen Towels:




The record was successfully updated in the performance table:




The April column now shows the correct incremented value.

## 9.3 Top five best performing items

Firstly, I have created a class inside the performance analysis class for the performance object that the data grid will bind to:

```
public class PerformanceObj
{
    2 references
    public string ItemName { get; set; }
    24 references
    public int Transfers { get; set; }
}
```

I also created a new method inside the Performance analysis class which returns a list of these objects:

```
public List<PerformanceObj> GetTopFiveThisMonth()
{
```

Inside this method I have created three lists. One list holds all of the records in the performance table.  One list will hold the top five items for the current month as performance table objects. The last list will hold the top five items for the current month as performance objects, this list is returned at the end of the method.

```
//get every performance record
List<PerformanceTable> AllPerformanceItems = SqlLink.GetPerformanceItems();
//the top five items
List<PerformanceTable> TopFivePerformance = new List<PerformanceTable>();
//top five items as performance objects
List<PerformanceObj> TopFivePerformanceObjects = new List<PerformanceObj>();
```

Now, I can order the all performance items list in descending order of transfers for the current month:

```
//get the current month
string currentMonth = DateTime.Now.ToString("MMMM");

if (currentMonth == "January")
{
    //orders the list in descending order for the January column
    AllPerformanceItems = AllPerformanceItems.OrderByDescending(o => o.January).ToList();
}
```

If the current month was February:

```
if (currentMonth == "February")
{
    AllPerformanceItems = AllPerformanceItems.OrderByDescending(o => o.February).ToList();
}
```

Now that the list has been ordered in descending order, add the first five items (top five) to the top five performance list:

```
int x = 1;
//add the first five items to the top five performance list
foreach (PerformanceTable perfElement in AllPerformanceItems)
{
    if (x <= 5)
    {
        TopFivePerformance.Add(perfElement);
        x++;
    }
}
```

Now, I need to iterate through the top five items list and convert it into a performance object:

```
foreach (PerformanceTable conv in TopFivePerformance)
{
    PerformanceObj nextobj = new PerformanceObj();
    //setup the performance object
    nextobj.ItemName = conv.ItemName;
    if (currentMonth == "January")
    {
        nextobj.Transfers = conv.January.Value;
    }
```

The code above is an example of when the current month was January, it contains other conditions to for other months, such as:

```
if (currentMonth == "February")
{
    nextobj.Transfers = conv.February.Value;
}
```

Once the performance object is setup, it is added to the final list:

```
TopFivePerformanceObjects.Add(nextobj);
```

Once all five items have been converted, the list is returned:

```
return TopFivePerformanceObjects;
```

In the performance page user control, I created an instance of the performance class:

```
PeformanceAnalysis perf = new PeformanceAnalysis();
```

In the constructor on the user control I have the set the item source of the top five items data grid to the result of calling the get top five items method in the Performance analysis class:

```
public MainPerformance()
{
    InitializeComponent();
    TopFive_dataGrid.ItemsSource = perf.GetTopFiveThisMonth();
```

To test this works I have manually added five items to the performance table with random values for each month:

| PeformanceID | ItemName | January | February | March | April | May | June | July | August | September | October | November | Decemb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | Coke ... | 23 | 324 | 34 | 35 | 56 | 456 | 56 | 456 | 56 | 456 | 56 | 456 |
| 20 | Fanta ... | 34 | 456 | 56 | 56 | 56 | 34 | 435 | 456 | 34 | 345 | 456 | 56 |
| 21 | White Bread ... | 456 | 435 | 34 | 12 | 123 | 12 | 5 | 1 | 23 | 34 | 23 | 34 |
| 22 | Sprite ... | 234 | 34 | 91 | 92 | 23 | 1 | 2 | 4 | 54 | 3 | 343 | 3 |
| 23 | Coke Cherry ... | 345 | 34 | 3 | 3 | 2 | 43 | 32 | 3 | 4 | 21 | 12 | 4 |

When I run the program, the performance page should show the items in order for April.

The image below shows the performance page showing the correct top five items:

*Top five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| Sprite | 92 |
| Fanta | 56 |
| Coke | 35 |
| White Bread | 12 |
| Coke Cherry | 3 |

## 9.4 Top five worst performing items

To get the top five worst performing items I just need to alter the method used to get the top five best performing items.

Instead of ordering all the items for the current month in descending order, it should be done in ascending order:

```
AllPerformanceItems = AllPerformanceItems.OrderBy(o => o.January).ToList();
```

The code above, orders all the performance items in ascending order for the January column.

The rest of the method is the same as method for getting the top five best performing items.

Whilst still using the items setup in the performance table as before, the performance page top five worst items data grid looks as follows:

*Worst five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| Coke Cherry | 3 |
| White Bread | 12 |
| Coke | 35 |
| Fanta | 56 |
| Sprite | 92 |

To refresh the top five best or top five worst items data grids, in the refresh button click event handler for each grid, I called the method in the Performance Analysis again, setting the value it returned to the data grids item source:

```
private void topfiveworst_refresh_button_Click(object sender, RoutedEventArgs e)
{
    TopFiveWorst_dataGrid.ItemsSource = perf.GetTopFiveWorstThisMonth();
    TopFiveWorst_dataGrid.Items.Refresh();
}
```

## 9.5 Determining if an item will sell well in the current month

To determine whether an item will be successful in the month that it is being transferred, I will implement the algorithm I designed in section 6.3.

The method will return a Boolean value, false if the item will sell well during the current month and true if the item wont. It will also take the name of the item as a parameter:

```
public bool GetPerformanceThisMonthAsBool(string ItemName)
{
```

In SQLServerAccessor class I have created a method to search for a performance record for the item name passed to the method:

```
public List<PerformanceTable> GetPerformanceItemsByLIKEName(string itemName)
{
    return (from x in DataContextInv.PerformanceTables
            where x.ItemName.Contains(itemName)
            select x).ToList();
}
```

Firstly I will use the method above to check that a performance record actually exists for the item first. If it doesn't exist then the method should just return true.

```
//if a performance record for the
if (SqlLink.GetPerformanceItemsByName(ItemName).Count != 0)
{
```

```
else { return false; }
```

Next I will get the actual performance record for the item:

```
var PerfData = SqlLink.DataContextInv.PerformanceTables.Single(x => x.ItemName == ItemName);
```

And get the current month:

```
//gets the current month
string currentMonth = DateTime.Now.ToString("MMMM");
```

I then setup an integer array, where index 0 represents the transfers made in January and index 1 the transfers made in February and so on:

```
int[] TransfersMonths = new int[12];
TransfersMonths[0] = PerfData.January.Value;
TransfersMonths[1] = PerfData.February.Value;
TransfersMonths[2] = PerfData.March.Value;
TransfersMonths[3] = PerfData.April.Value;
TransfersMonths[4] = PerfData.May.Value;
TransfersMonths[5] = PerfData.June.Value;
TransfersMonths[6] = PerfData.July.Value;
TransfersMonths[7] = PerfData.August.Value;
TransfersMonths[8] = PerfData.September.Value;
TransfersMonths[9] = PerfData.October.Value;
TransfersMonths[10] = PerfData.November.Value;
TransfersMonths[11] = PerfData.December.Value;
```

If the current month is January, then the method will iterate through the array, checking if any other index is greater than index 0. If any are, then the method should return true, since it's not likely that the item will sell the best during this month:

```
if (currentMonth == "January")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        //if any other month had more sales than the current month , then its not likely to sell best in this month
        if (TransfersMonths[0] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

The same logic was used for other months, for example, if the current month is February then the algorithm uses the following:

```
if (currentMonth == "February")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[1] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

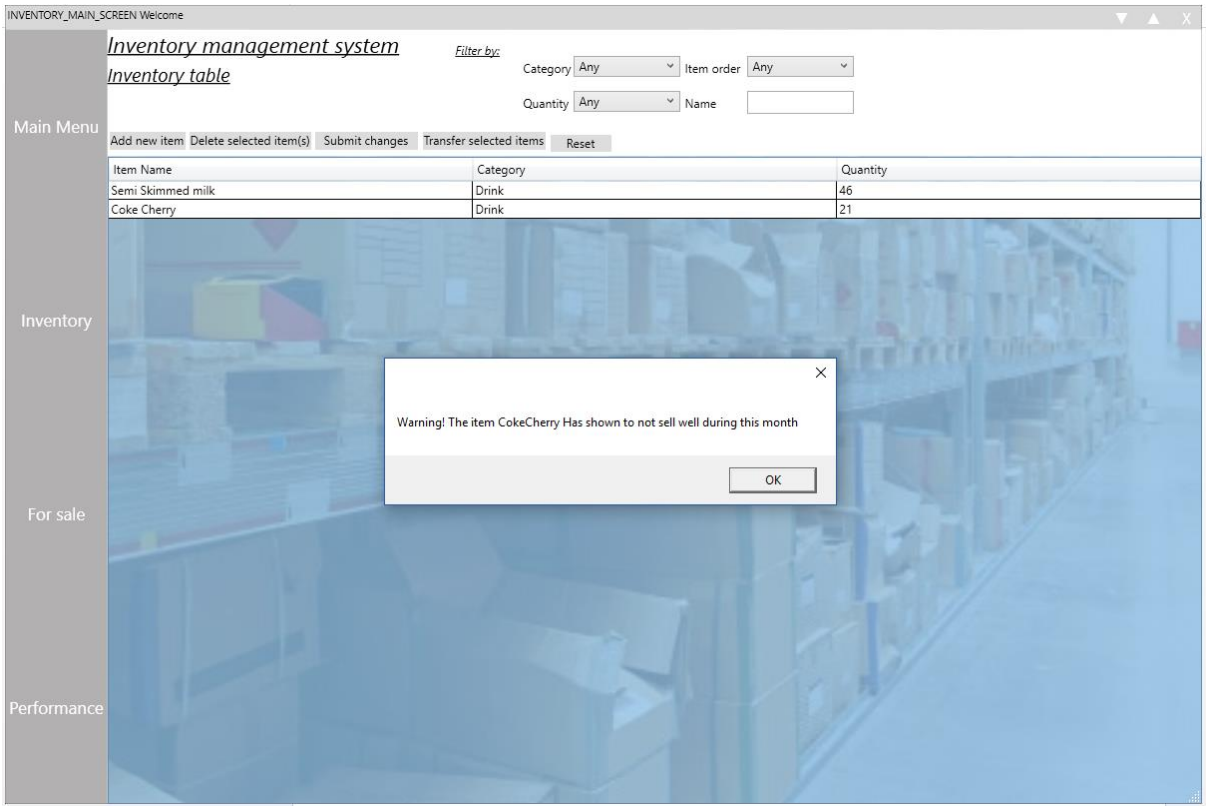
Whenever an item is being transferred to the for sale table this method should be called, so that user is aware that the item is not likely to sell well if the data hints so. Therefore, I will call this method when a transfer is taking place:

```
if (perf.GetPerformanceThisMonthAsBool(itemToPushWithSellByDate.ItemName))
{
    MessageBox.Show("Warning! The item " + itemToPushWithSellByDate.ItemName.Replace(" ", "") + " Has shown to not sell well during this month");
}
```

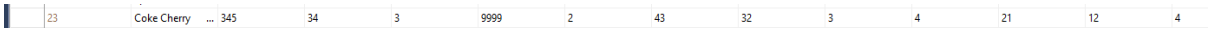
To test this works, I will setup an item in inventory with performance data that shows it won't sell this month.

I will use the Coke Cherry item. The performance data is setup to have it with the least number of transfers in the current month, April. This means, when trying to transfer the item, the program should tell the user that this month is not the best month for this item.

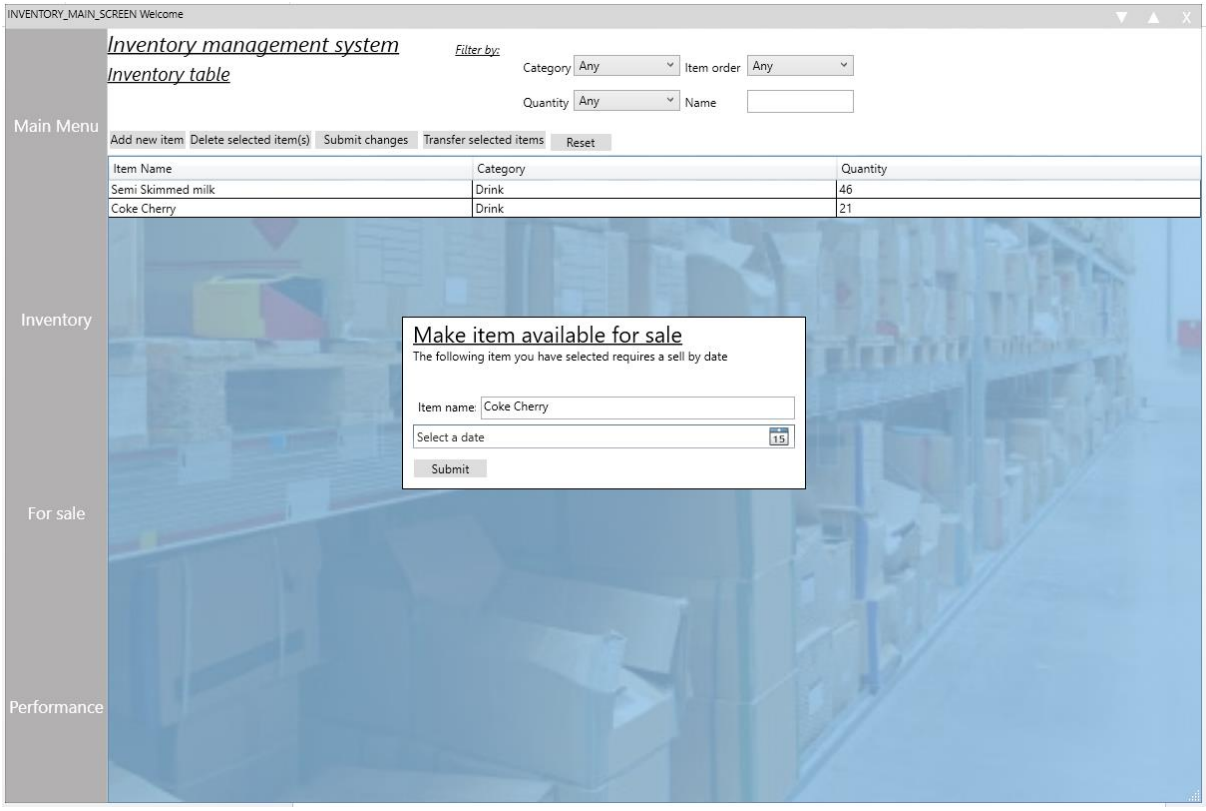
After clicking the transfer button whilst Coke cherry was selected:



Now, if I set the performance data of Coke Cherry to have maximum transfers in April the message should not be displayed:



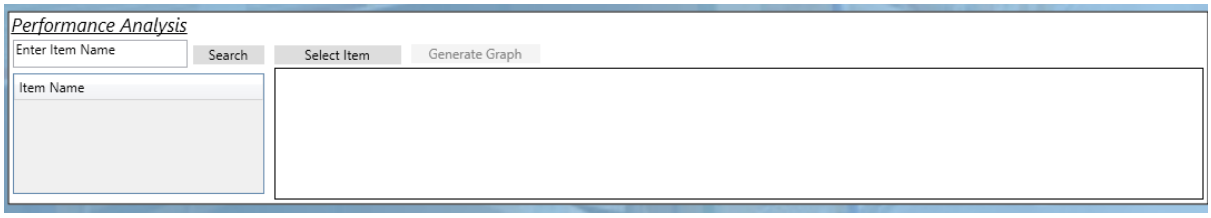

After clicking on the transfer button, no message was shown:




This shows that method works as intended.

## 9.6 Individual item performance analysis

This section relates to the bottom section of the performance page user control:




To start with, I will build the search for an item section. When the user has entered an item name and clicked the search button the data grid underneath the item name text should be populated with the search results. After that, the user can select an item from the data grid to perform an analysis on.

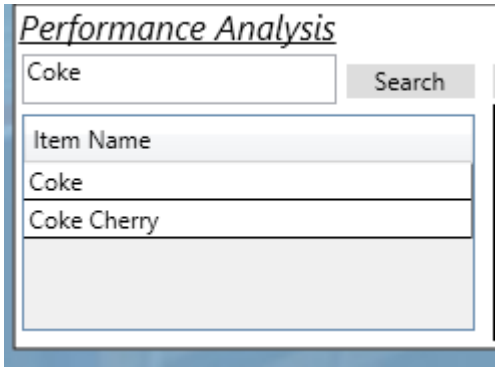
In the SQLServerAccessor I have created a method which calls an existing method. It returns all the performance records which have a similar or exact name to the passed item name:

```
public List<PerformanceTable> GetSearchResults(string name)
{
    return SqlLink.GetPerformanceItemsByLIKEName(name);
}
```

To populate the data grid, with item names similar to the input, in the search buttons mouse click event handler I set the item source of the data grid to the result of calling the get search results method in the SQLServerAccessor:

```
private void submit_search_button_Click(object sender, RoutedEventArgs e)
{
    if (search_textBox.Text != "Enter Item Name")
    {
        //populate the data grid with the performance records similar to the input item name
        SearchResults_dataGrid.ItemsSource = perf.GetSearchResults(search_textBox.Text);
        SearchResults_dataGrid.Items.Refresh();
    }
}
```

If I enter Coke in the item name, the data grid looks as follows:




When the user clicks on the select item button, the program should generate an analysis message for that item. It should say, which month the item has shown to sell best in and which month the item has shown to sell worst in.

In the Performance Analysis class, I have created a new method, called generate performance text. It returns a string value, corresponding the message it just generated. The item to be analysed will also be passed to it:

```
public string GeneratePerformanceAnalysisText(PerformanceTable PerfData)
```

I then setup the following variables. As with the previous method (in 9.5) an array has been setup which corresponds to the current month:

```
int[] TransfersMonths = new int[12];
TransfersMonths[0] = PerfData.January.Value;
TransfersMonths[1] = PerfData.February.Value;
TransfersMonths[2] = PerfData.March.Value;
TransfersMonths[3] = PerfData.April.Value;
TransfersMonths[4] = PerfData.May.Value;
TransfersMonths[5] = PerfData.June.Value;
TransfersMonths[6] = PerfData.July.Value;
TransfersMonths[7] = PerfData.August.Value;
TransfersMonths[8] = PerfData.September.Value;
TransfersMonths[9] = PerfData.October.Value;
TransfersMonths[10] = PerfData.November.Value;
TransfersMonths[11] = PerfData.December.Value;
int lastHighestVal = 0;
int lastHighestIndex = 0;
int lastLowestVal = 0;
int lastLowestIndex = 0;
string worstMonth = "";
string bestMonth = "";
```

Next, the method will iterate through transfer month array

```
for (int pos = 0; pos < TransfersMonths.Length; pos++)
{
```

On the first iteration, the last highest and lowest value and index should be set to current value and index being iterated through:

```
//if its the first iteration then setup the variables
if (pos == 0)
{
    lastHighestIndex = pos;
    lastHighestVal = TransfersMonths[pos];
    lastLowestIndex = pos;
    lastLowestVal = TransfersMonths[pos];
}
```

On later iterations, if the value being iterated through has a greater value than the value stored in the last highest value variable, then the last highest value should be updated to the current value being iterated through and the last highest index to the current index:

```
//if the month being iterated through has had more transfers than last highest month
else if(TransfersMonths[pos] > lastHighestVal)
{
    //update the last highest values and index
    lastHighestVal = TransfersMonths[pos];
    lastHighestIndex = pos;
}
```

Similarly if the current value being iterated through is less than the last lowest value, then the last lowest values and index should be updated:

```
//if the month being iterated through has had less transfers than last highest month
else if (TransfersMonths[pos] < lastLowestVal)
{
    //update the last lowest values and index
    lastLowestVal = TransfersMonths[pos];
    lastLowestIndex = pos;
}
```

Now that index of the best and worst performing months for the item have been found , the index needs to be linked to the actual month the index corresponds to.

For example, if the last highest index was 0, the month was January:

```
if (lastHighestIndex == 0 )
{
    bestMonth = "January";
}
```

Or if the last lowest index was 5, the month was June:

```
if (lastLowestIndex == 5)
{
    worstMonth = "June";
}
```

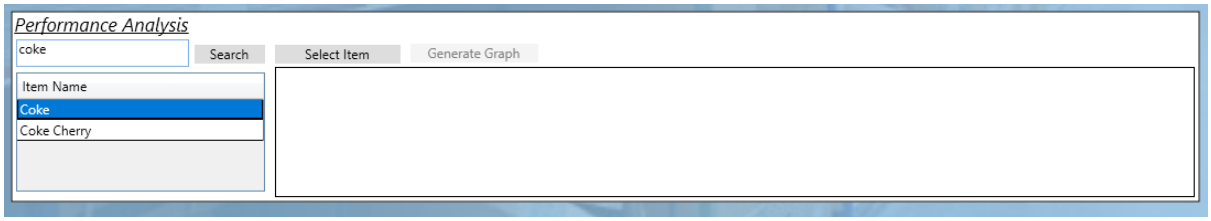
The worst and best months have now been identified, all that is left to do if formulate the analysis message. This has been done by concatenating together the best and worst month strings together with other text:

```
return (PerfData.ItemName.Replace(" ","") + "  has shown to sell best in the month " + bestMonth + " and sells worst in the month " + worstMonth );
```

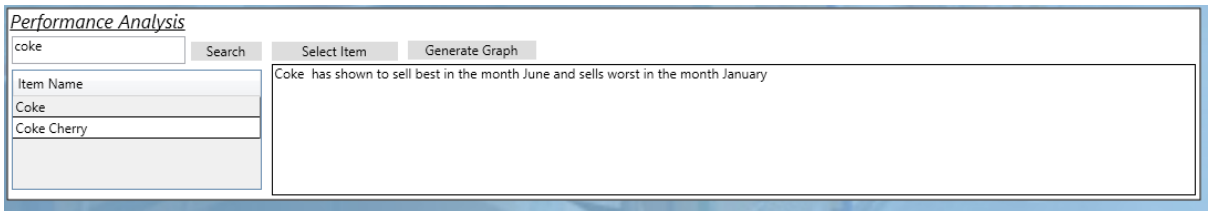
When the user clicks on the select item button in the performance page user control, the method I have just created will be called in the mouse click event handler. Once the message has been generated and set the user should be allowed to generate a performance graph:

```
if (SearchResults_dataGrid.SelectedItem != null)
{
    //just take the first item
    foreach (PerformanceTable item in SearchResults_dataGrid.SelectedItems)
    {
        perfItem = item;
        break;
    }
    report_textBox.Text = perf.GeneratePerformanceAnalysisText(perfItem);
    generate_graph_button.IsEnabled = true;
}
```

To test this I will select the coke item:




After clicking on the select item button the message was generated:



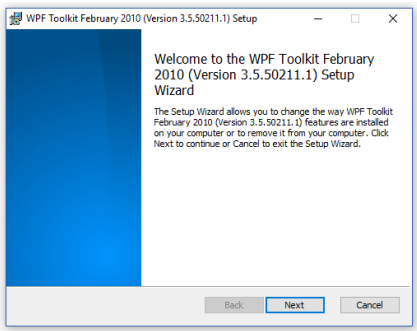

Based on the performance record for coke, the message is accurate:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | Coke | ... | 23 | 324 | 34 | 35 | 56 | 456 | 56 | 456 | 56 | 456 | 56 | 456 |

## 9.7 Performance graphs

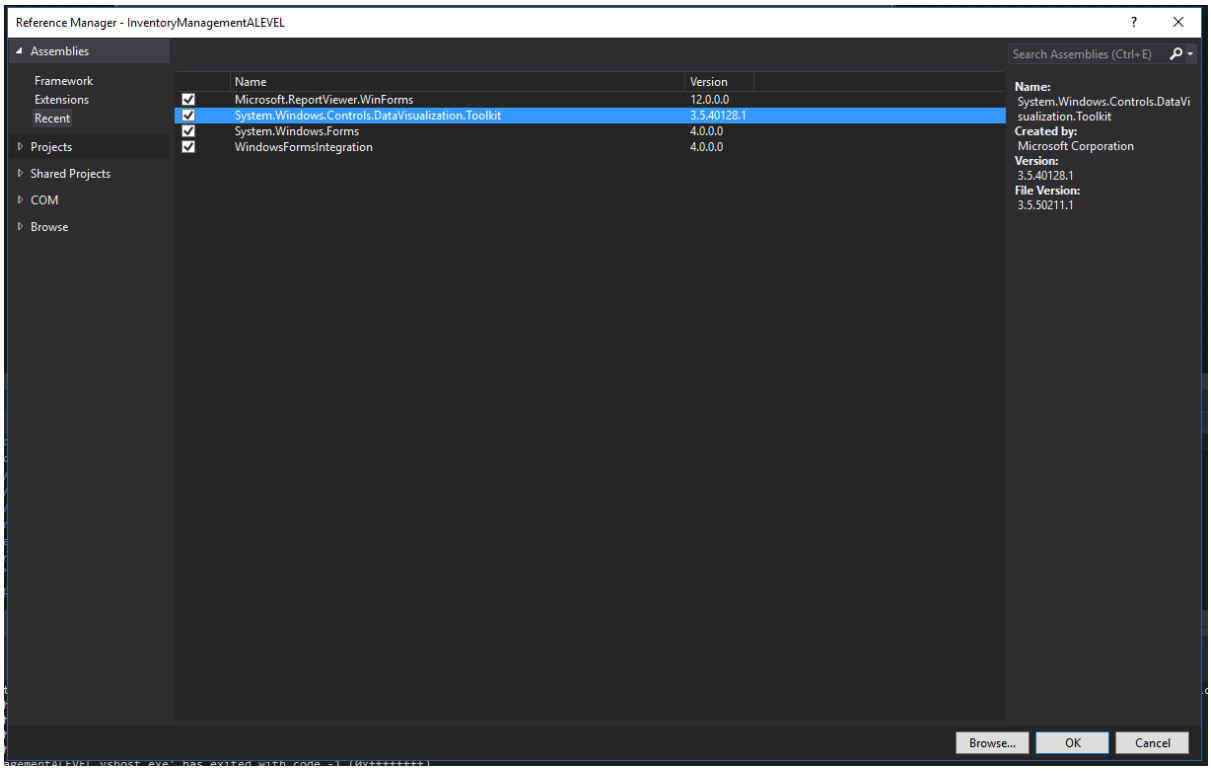
In order to draw graphs in WPF, I will use the WPFtoolkit library. “The WPF Toolkit is a collection of WPF features and components that are being made available outside of the normal .NET Framework ship cycle”.

To install the WPFtoolkit I just had to run an installer I had downloaded:



In visual studio, I created a new window, called graph display window. Unlike my previous windows, this window will retain the default windows style.

To add the WPF toolkit to my project, I added a reference to it:



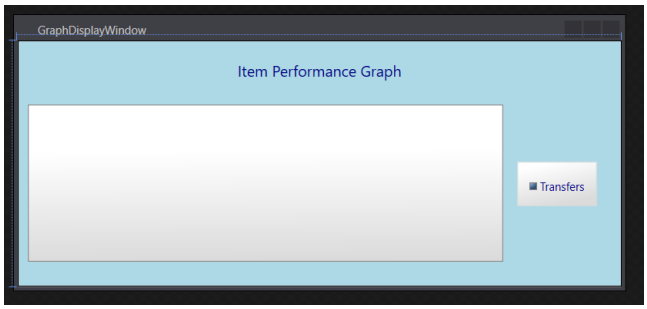

In the graph display windows XAML I added the following lines of XAML to allow me to access the tool kit through XAML:

```
xmlns:DV="clr-namespace:System.Windows.Controls.DataVisualization;assembly=System.Windows.Controls.DataVisualization.Toolkit"
xmlns:DVC="clr-namespace:System.Windows.Controls.DataVisualization.Charting;assembly=System.Windows.Controls.DataVisualization.Toolkit"
```

Inside the main, grid the following XAML caused a line graph to appear:

```
<DVC:Chart Name="Perf_Chart"
   HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
   Background="LightBlue"
   Foreground="DarkBlue"
   Title="Item Performance Graph"
           >
    <DVC:LineSeries Title="Transfers"
    IndependentValueBinding="{Binding Path=Key}"
    DependentValueBinding="{Binding Path=Value}">
    </DVC:LineSeries>
</DVC:Chart>
```

In the design view it looked as follows:



The constructor of this window requires that the item being graphed be passed to it:

```
public GraphDisplayWindow(PerformanceTable perfData)
```

To setup a link between the x and y axis I added the following code in the constructor. On the X axis, the months of the year should be displayed. On the Y axis, the total number of transfers for each month should be displayed:

```
//links the x axis (month) to the transfers for that month ( y axis)
((LineSeries)Perf_Chart.Series[0]).ItemsSource =
new KeyValuePair<string, int>[]{
new KeyValuePair<string, int>("January", perfData.January.Value),
new KeyValuePair<string, int>("February", perfData.February.Value),
new KeyValuePair<string, int>("March", perfData.March.Value),
new KeyValuePair<string, int>("April", perfData.April.Value),
new KeyValuePair<string, int>("May", perfData.May.Value),
new KeyValuePair<string, int>("June", perfData.June.Value),
new KeyValuePair<string, int>("July", perfData.July.Value),
new KeyValuePair<string, int>("August", perfData.August.Value),
new KeyValuePair<string, int>("September", perfData.September.Value),
new KeyValuePair<string, int>("October", perfData.October.Value),
new KeyValuePair<string, int>("November", perfData.November.Value),
new KeyValuePair<string, int>("December", perfData.December.Value),};
```

To generate graphs, I just need to instantiate the window passing the item being analysed. I will do this is in the mouse click event handler for the generate graph button.

Firstly I have created a global variable instance of the performance table object containing the item being analysed:

```
PerformanceTable perfItem = new PerformanceTable();
```

In the constructor of the user control, in mouse click event for the select item button I set the value of the instance:

```
perfItem = item;
```

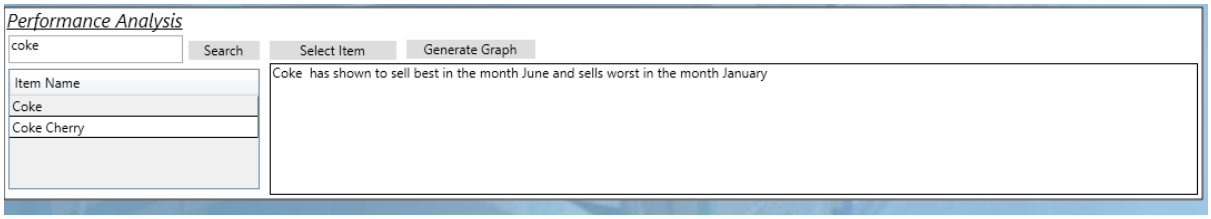
As shown previously.

I then instantiated the graph window in the mouse click event handler for the generate graph button:

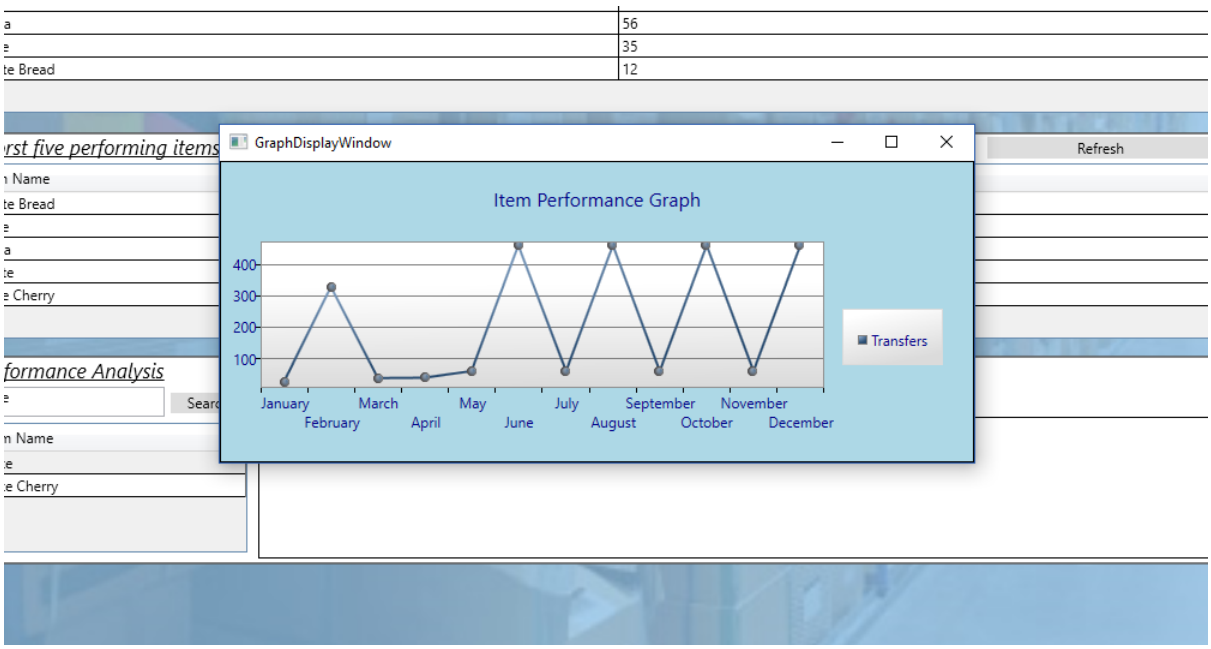
```
private void generate_graph_button_Click(object sender, RoutedEventArgs e)
{
    GraphDisplayWindow gdw = new GraphDisplayWindow(perfItem);
    gdw.Show();
}
```

To test the graph, I will load the performance page, select the coke item again and click on the generate graph button:

Selecting coke:




After clicking the generate graph button:



# Evaluation

## 1 Testing the robustness of the solution

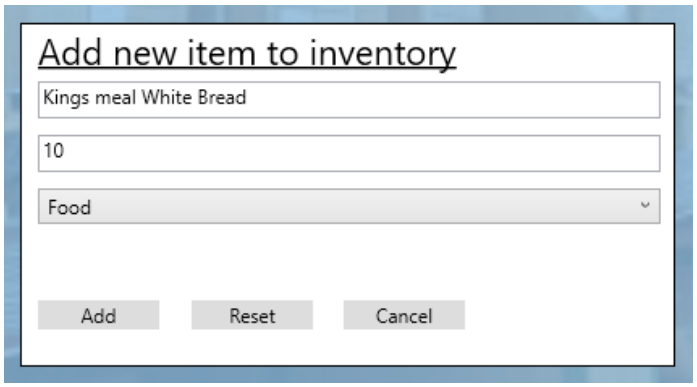
### 1.1 testing against test plan

To begin with I will test the solution against the test plan I created in the design section of this document (Design section 7.1). This will demonstrate that the solution has successfully met requirements.

*Test 1*

*Demonstrate the requirement: The solution must be able to input new stock items.*

Using the normal data set:




The new item was shown in the inventory table after clicking the add button:

| | | |
|---|---|---|
| Kings meal White Bread | Food | 10 |

In the actual database, the record was created and is shown below:

| | | | | |
|---|---|---|---|---|
| | 76 | Food | Kings meal White Bread | ... | 10 |

Using the boundary data set:

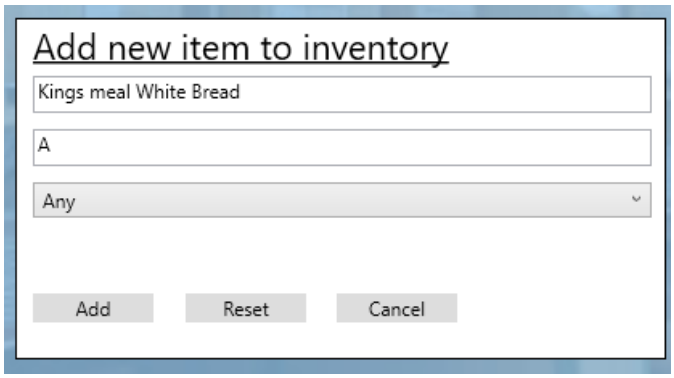
The new item was shown in the inventory table after clicking the add button:

| | | |
|---|---|---|
| Kings meal White Bread | Food | 1 |

In the actual database, the record was created and is shown below:

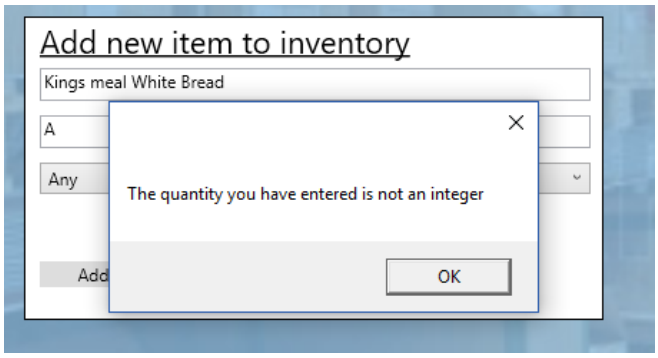
| | | | |
|---|---|---|---|
| 78 | Food | Kings meal White Bread... | 1 |

Using the Erroneous data set:




It was not possible to enter the category specified in the erroneous data set.

Pressing the add button caused the following error message to show:




The expected results were met and it can be concluded that test 1 was successful.

*Test 2*

*Demonstrate the requirement: The solution must be able to take out stock items*

The action I will take is slightly different from the action described in the test plan, this because the solution has been developed to transfer items in a different way.

Using the normal data set:

I will first add the item to inventory:

| | | |
|---|---|---|
| Kings meal White Bread | Food | 10 |

I then added the sell by date:

After clicking submit the item was no longer in the inventory table:

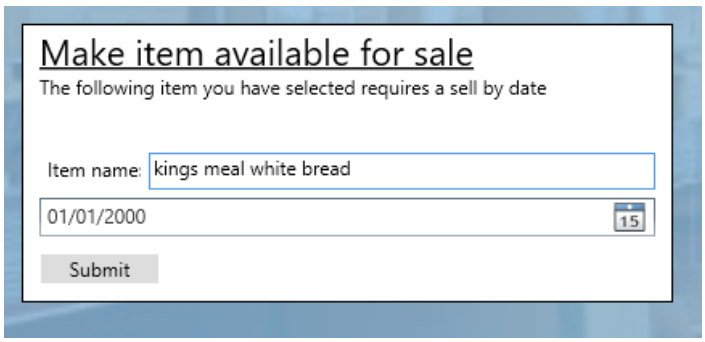
| Item Name | Category | Quantity |
|---|---|---|
| Coke Cherry | Drink | 21 |

And is now showing in the for sale table:

| | | | |
|---|---|---|---|
| Kings meal White Bread | Food | 10 | 20/11/16 |

Using the boundary data set:

It was not possible to enter a year of 0, it reset to 2000 when entering 0:




This is not the expected result.

Using the erroneous data set:

I was able to add an item to the inventory table with a negative quantity, this is not the expected behaviour.

Add new item | Delete selected item(s) | Submit changes | Transfer selected items | Reset

| Item Name | Category | Quantity |
|---|---|---|
| Coke Cherry | Drink | 21 |
| kings meal white bread | Food | -10 |

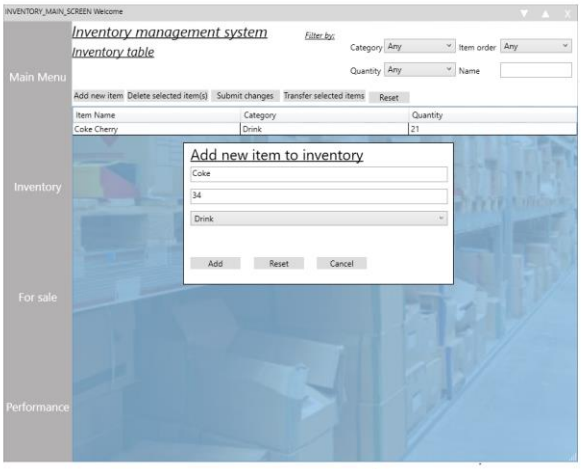
Overall, test number 2 was unsuccessful. The expected result was only achieved when using the normal data set.

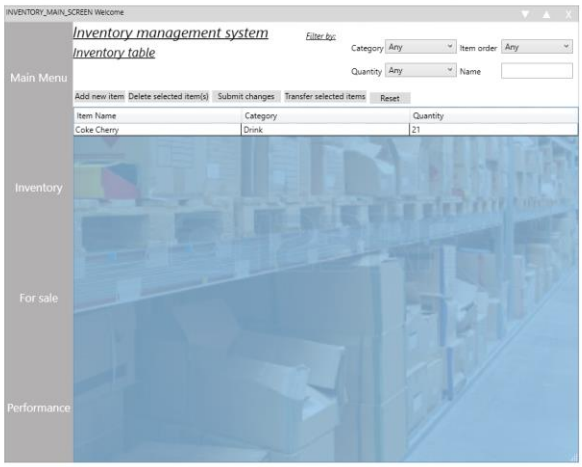
*Test 3*

*Demonstrate the requirement: The solution must be accessible from different computers*

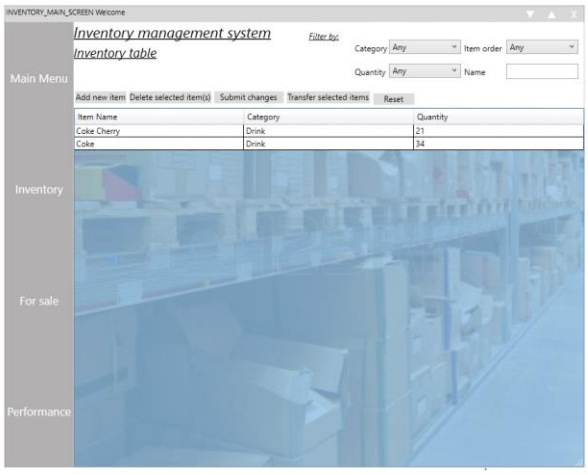
Since the solution uses a SQL server database installed on a single machine, instead of using different computers I will have to use different instances of the solution running on the same machine. However, this should be suitable enough to emulate the solution running on different machines. In addition, I will only be able to carry out this test using normal data set due to this limitation.

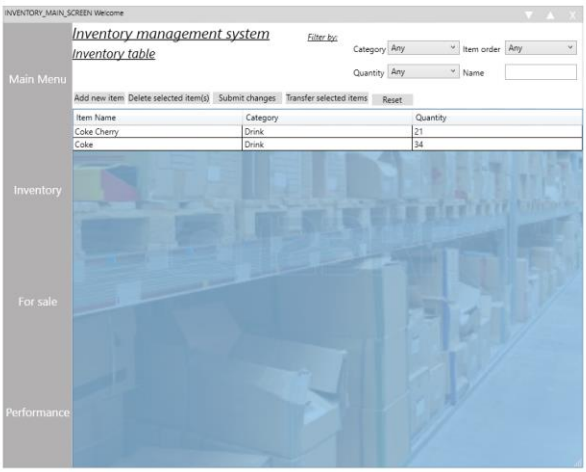
The image below shows the two instances open side by side:




When I click the add button on the left instance, the new item should appear in the inventory table of both instances as shown below:



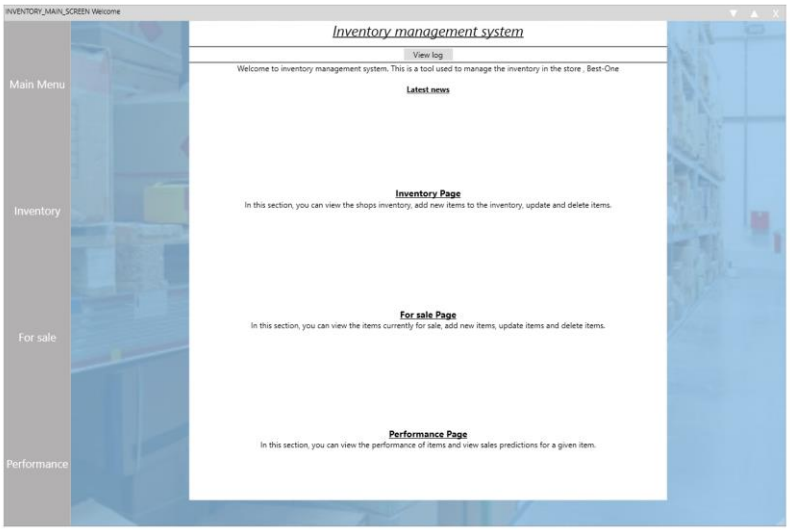

Despite the limitations of the solution during this test, the test was successful and showed that the solution can be used from different computers

*Test 4*

Demonstrate the requirement: The solution should be useable by inexperienced users

**This test will be very similar to usability testing that I will be conducting shortly.**

To begin with, the user was given the solution in a logged in state, open as shown below:

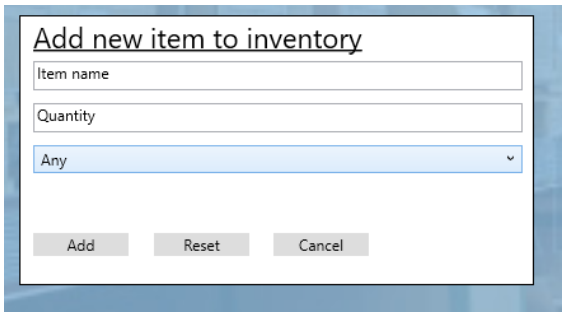



The first task the user was required to do was to input a new item into the inventory page.

Immediately the user wasn't sure were to start and required assistance from me to be shown to go to the inventory page.

Once there the user was able to click on the add item button

From there the user was able to enter the item data and click add, causing it to be added to inventory. However, the user was initially confused as to what the drop down box was asking for. After a short time the user figured out that it was asking for the category of the item, they did not require any intervention from me.




For now I ended the test there. I would be performing more usability tests at a later date in much more detail.

However, the test was successful despite the user being stuck initially, the user completed the task eventually.

*Test 5*

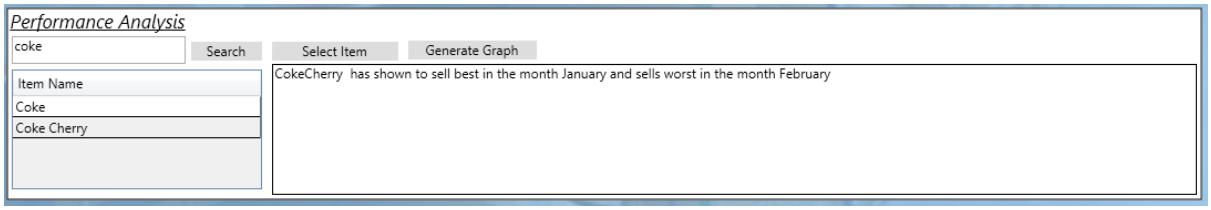
*Demonstrate the requirement: The solution must be able to predict when products will sell well*

Using the normal data set:

I setup normal data set using the SQL Server Management Studio.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Coke Cherry | ... | 45 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

After searching and selecting this item in the performance page, the program told me that the best month was January:

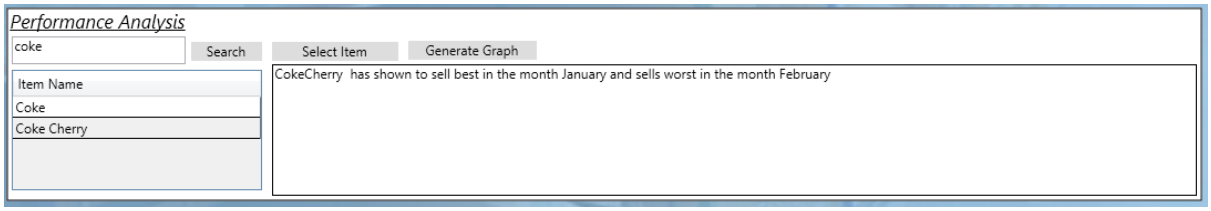



Using the boundary data set:

The performance record has been changed to following boundary data set:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 23 | Coke Cherry | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

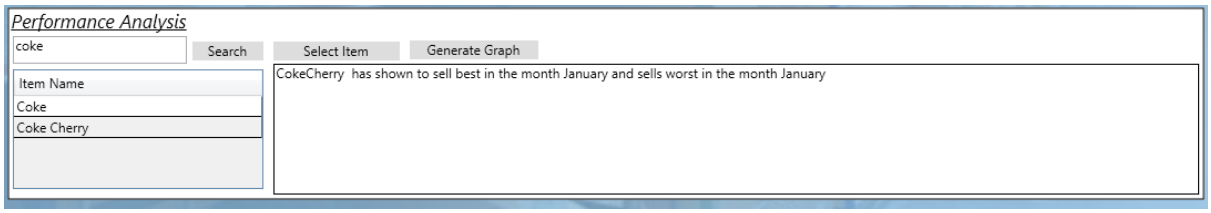
When I select the item in the performance page, the program told me that the best month was January:




Using the erroneous data set:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 23 | Coke Cherry | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

When I clicked the select item button, the output was the same as the normal and boundary data set:




This is not the intended behaviour of the solution. It should have produced an error message.

Overall test number 5 has been partially successful, when using the normal and erroneous data sets, the expected result was achieved however when using the erroneous data set, the expected result was not met.

*Test 6*

Demonstrate the requirement: The solution must be able to find popular products

As with the previous test I will enter the data sets using the SQL Server Management Studio:

Using the normal data set:

| PeformanceID | ItemName | January | February | March | April | May | June | July | August | September | October | November | Decemb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | Coke ... | 23 | 324 | 34 | 35 | 56 | 456 | 56 | 456 | 56 | 456 | 56 | 456 |
| 20 | Fanta ... | 34 | 456 | 56 | 56 | 56 | 34 | 435 | 456 | 34 | 345 | 456 | 56 |
| 21 | White Bread ... | 456 | 435 | 34 | 12 | 123 | 12 | 5 | 1 | 23 | 34 | 23 | 34 |
| 22 | Sprite ... | 234 | 34 | 91 | 92 | 23 | 1 | 2 | 4 | 54 | 3 | 343 | 3 |
| 23 | Coke Cherry ... | 67 | 45 | 7 | 41 | 24 | 2 | 4 | 55 | 21 | 22 | 45 | 5 |

The top five performing items table is shown below:

*Top five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| Sprite | 92 |
| Fanta | 56 |
| Semi Skimmed milk | 46 |
| Coke Cherry | 41 |
| Coke | 35 |

This is the expected result.

Using the boundary data set:

| April |
|---|
| 35 |
| 1 |
| 1 |
| 1 |
| 1 |

The top five performing items table is shown below:

*Top five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| Coke | 35 |
| Fanta | 1 |
| White Bread | 1 |
| Sprite | 1 |
| Coke Cherry | 1 |

This is the expected result.

Using the erroneous data set:

| April |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

The top five performing items table is shown below:

*Top five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| Coke | 0 |
| Fanta | 0 |
| White Bread | 0 |
| Sprite | 0 |
| Coke Cherry | 0 |

Although this wasn't the expected result, the result is still acceptable.

Overall test number 6 was successful.

*Test 7*

Demonstrate the requirement: It must be able to find unpopular products

Using the normal data set:

| | PeformanceID | ItemName | January | February | March | April | May | June | July | August | September | October | November | Decemb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 19 | Coke ... | 23 | 324 | 34 | 35 | 56 | 456 | 56 | 456 | 56 | 456 | 56 | 456 |
| | 20 | Fanta ... | 34 | 456 | 56 | 56 | 56 | 34 | 435 | 456 | 34 | 345 | 456 | 56 |
| | 21 | White Bread ... | 456 | 435 | 34 | 12 | 123 | 12 | 5 | 1 | 23 | 34 | 23 | 34 |
| | 22 | Sprite ... | 234 | 34 | 91 | 92 | 23 | 1 | 2 | 4 | 54 | 3 | 343 | 3 |
| | 23 | Coke Cherry ... | 67 | 45 | 7 | 41 | 24 | 2 | 4 | 55 | 21 | 22 | 45 | 5 |

The top five worst performing items table is shown below:

*Worst five performing items this month* Refresh

| Item Name | Transfers |
|---|---|
| kings meal white bread | 1 |
| White Bread | 12 |
| Coke | 35 |
| Coke Cherry | 41 |
| Semi Skimmed milk | 46 |

This is the expected result.

Using the boundary data set:

| April |
|---|
| 35 |
| 1 |
| 1 |
| 1 |
| 1 |

The top worst performing items table is shown below:

*Worst five performing items this month* Refresh

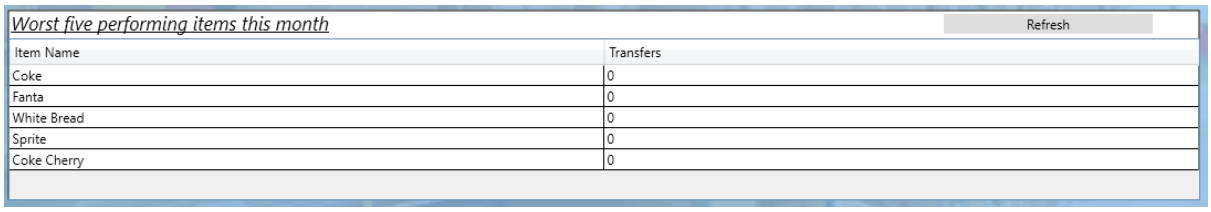
| Item Name | Transfers |
|---|---|
| Fanta | 1 |
| White Bread | 1 |
| Sprite | 1 |
| Coke Cherry | 1 |
| Coke | 35 |

This is the expected result.

Using the erroneous data set:

| April |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

The top worst performing items table is shown below:




| Item Name | Transfers |
|---|---|
| Coke | 0 |
| Fanta | 0 |
| White Bread | 0 |
| Sprite | 0 |
| Coke Cherry | 0 |

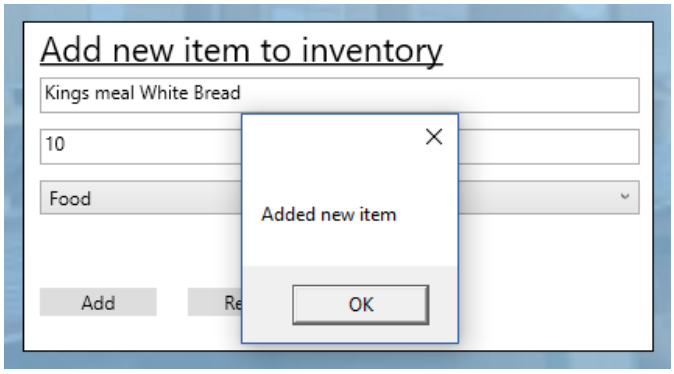
Similarly to test number 6, this wasn't the expected result but the result is still acceptable.

Overall test number 7 was successful.

*Test 8*

Demonstrate the requirement: The solution must keep a log of all actions that have taken place by users

Adding the item to the inventory page:




Viewing the log entry:

| | | |
|---|---|---|
| Arun.Patel Has added Kings meal White Bread to inventory, quantity 10 category Food | Arun | 30/04/17 |

The action was successfully added therefore the test was successful.

*Test plan summary*

To summarise the testing I have just done I have created the following table, which summarises the outcome of each test.

| **Test No** | **Outcome** |
|---|---|
| 1 | Successful |
| 2 | Unsuccessful |
| 3 | Successful |
| 4 | Successful |
| 5 | Partially successful |
| 6 | Successful |
| 7 | Successful |
| 8 | Successful |
| 9 | Successful |
| 10 | Successful |
| 11 | Successful |
| 12 | Successful |

*Note – Although tests 9-12 are not demonstrated here, they have been demonstrated during the development stage of this document*

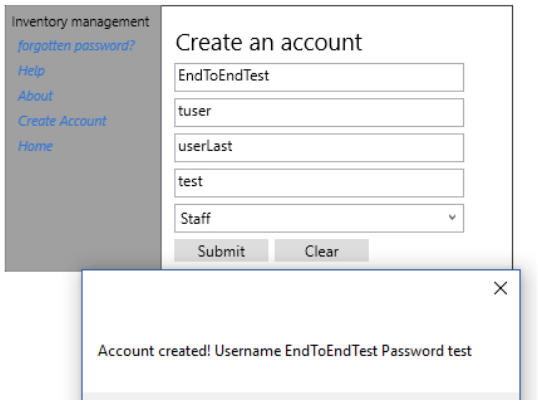
## 1.2 End to End test

This test involves a complete run through of the program. I have created the following plan to outline the tasks I will need to complete during this test.

1) Create an account with the role staff member
2) Login using this account
3) Attempt to access the performance page
4) Attempt to access the log window
5) Create a new account with the role Manager
6) Add 3 items to the inventory
    - Two items with the category food or drink
7) Transfer 10 of the one of the items to the for sale table
    - This must be the one of the food or drink items
    - A sell by date must be added to this item.
8) View the items in the for sale table
9) Transfer the rest of the item
10) View the log to see the action you have made
11) Filter the log to only show log entries after April this year
12) Filter the inventory table to only show items that are in the category drink and order them in ascending quantity order
13) View the performance information for the item just transferred
14) Generate a performance graph for this item

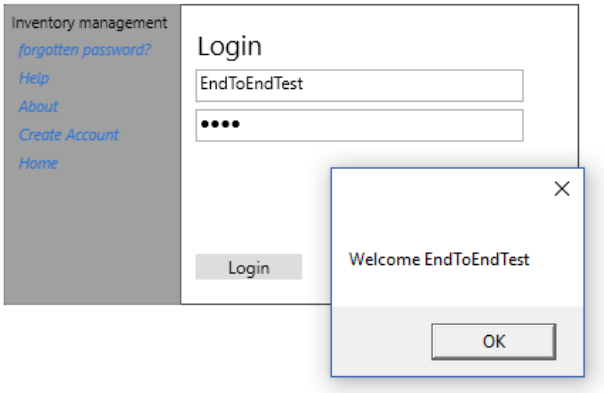
*Task 1*

Creating an account with the staff member role:




*Task 2*

Logging in using the account I just created:



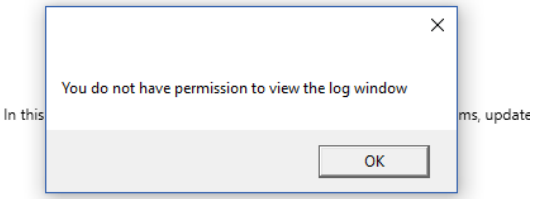

*Task 3*

Clicking on the performance button resulted in nothing happening, this is the intended behaviour since the user group of the account I am using does not have permission to access.

This could be made clearer to the user that this is case by showing the user a message box.

*Task 4*

When clicking on the log window, an error message was generated:

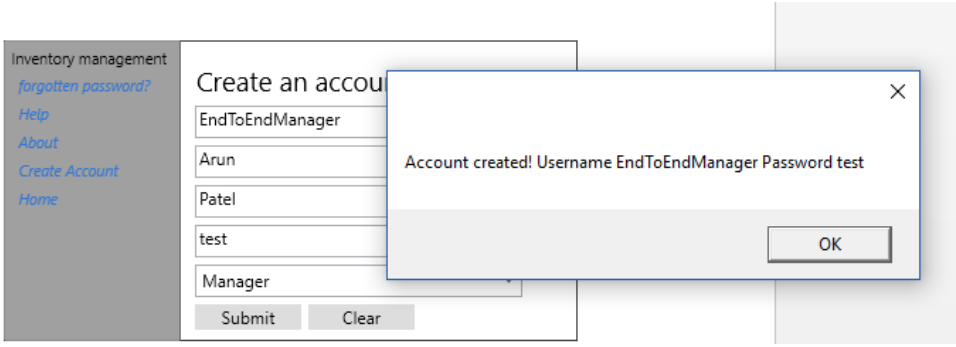



This is the correct behaviour for the user group of the account I am using.

*Task 5*

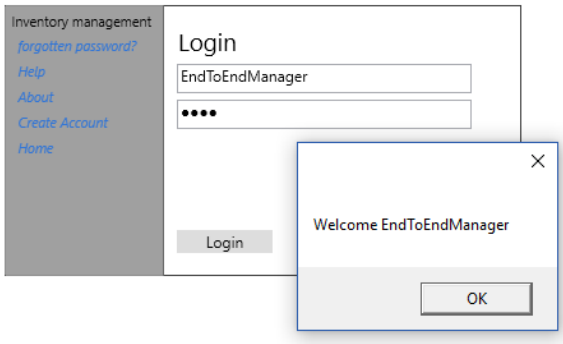
There is no way to log out of the program, returning to the login window. Instead I had to close the program and re open it to the login window.

Creating an account with the manager user group:




*Task 6*

Firstly, I logged into the account I created:




I then added 3 items the inventory table:

| | | |
|---|---|---|
| Kings meal White Bread | Food | 10 |
| Tango | Drink | 23 |
| Lined Paper | Household | 100 |

*Task 7*

Selecting 10 of the Tango Drink:

| | | |
|---|---|---|
| Tango | Drink | 10 |

Selecting 10 of a particular item the tables could be confusing to users as it is not straightforward. However once the user knows how to do this, It is an easy to choose a set number of items.

After clicking on the transfer button, I was prompted to add the sell by date, I added a random sell by date to it.

After the items quantity in the inventory page had been reduced to 13:

| | | |
|---|---|---|
| Tango | Drink | 13 |

This is correct since 23-10 = 13

*Task 8*

In the for sale page, the item was now there with quantity 10 and the date I added:

| | | | |
|---|---|---|---|
| Tango | Drink | 10 | 28/10/17 |

*Task 9*

To transfer the rest of the items, I selected the items record in the inventory table, without changing its quantity, and clicked on the transfer button. I was prompted to enter the sell by date again, I chose the same date as last time.

In the for sale table a new record was made, It did not merge with the previous record:

| | | | |
|---|---|---|---|
| Tango | Drink | 10 | 28/10/17 |
| Tango | Drink | 13 | 28/10/17 |

This is the intended behaviour. If the sell by dates of the two records were different it would make more sense. Maybe in the future, the solution could merge the two records if the sell by dates are the same.

*Task 10*

To view the log, I went back to the main page and clicked on the log button. The new log entries were at the bottom of the log list.

| | | |
|---|---|---|
| Created a new account for the user EndToEndTest | tuser | 30/04/17 |
| Created a new account for the user EndToEndManager | Arun | 30/04/17 |
| EndToEndManager Has added Tango to inventory, quantity 23 category Drink | Arun | 30/04/17 |
| EndToEndManager Has added Lined Paper to inventory, quantity 100 category Household | Arun | 30/04/17 |
| EndToEndManager has made Tango available for sale and added the sell by date of 28/10/2017 | Arun | 30/04/17 |

*Task 11*

To filter the log entries I entered the date into the date picker at the top of the top of the page:

Now the first log entry in the list was after this date:

Action log

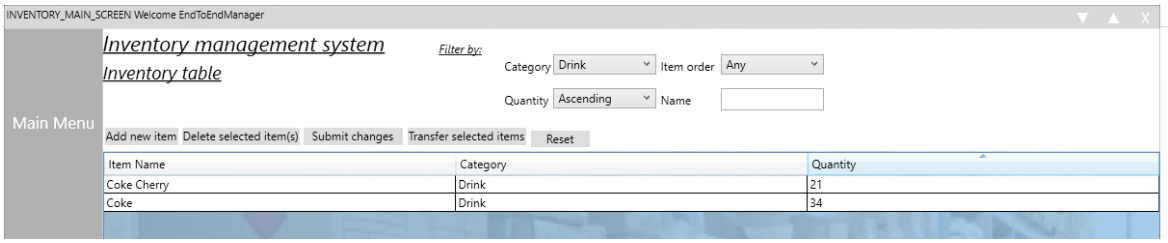
X

01/04/2017

| Action | First name | Date of action |
|---|---|---|
| arunptl100 Has added newitem to inventory, quantity 34 category Drink | Arun | 06/04/17 |
| arunptl100 Has added test to inventory, quantity 45 category Food | Arun | 06/04/17 |
| arunptl100 Has added test4 to inventory, quantity 45 category Food | Arun | 06/04/17 |
| arunptl100 Has added test45 to inventory, quantity 46 category Household | Arun | 06/04/17 |
| arunptl100 has removed test45 from inventory | Arun | 06/04/17 |
| arunptl100 has removed test4 from inventory | Arun | 06/04/17 |
| arunptl100 has removed test from inventory | Arun | 06/04/17 |
| arunptl100 Has added test to inventory, quantity 45 category Food | Arun | 06/04/17 |
| arunptl100 has made test available for sale and added the sell by date of 21/04/2017 00:00:00 | Arun | 06/04/17 |
| arunptl100 has removed test from the for sale table | Arun | 06/04/17 |
| has removed rt from inventory | AD | 07/04/17 |
| Has added testing to inventory, quantity 45 category Drink | AD | 10/04/17 |
| has removed testing from inventory | AD | 10/04/17 |
| Has added retre to inventory, quantity 45 category Drink | AD | 10/04/17 |
| Has added 345 to inventory, quantity 435 category Drink | AD | 10/04/17 |
| has made 345 available for sale and added the sell by date of 12/04/2017 00:00:00 | AD | 10/04/17 |
| has made retre available for sale and added the sell by date of 13/04/2017 00:00:00 | AD | 10/04/17 |
| has removed retre from the for sale table | AD | 10/04/17 |

*Task 12*

To setup a filter on the inventory table, I went back to the inventory table. In the filter section I selected the category Drink and selected the quantity ordering section to be in ascending order:

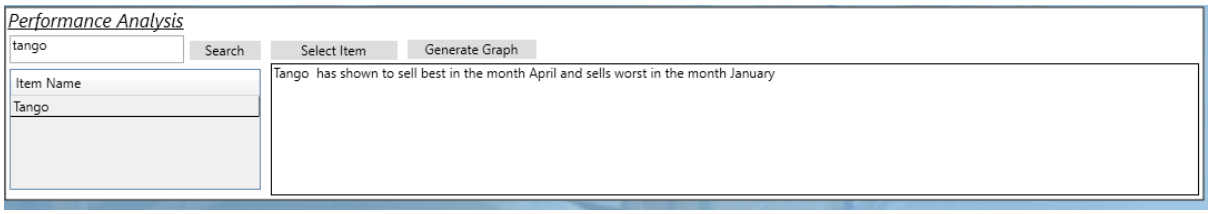



As shown in the image above, the items left in the table are only in the category drink and are in an ascending order of quantity.

*Task 13*

To view the performance information, I went to the performance page. In the item name field at the bottom of the page,

When I clicked on the select item button, a performance analysis message was generated:

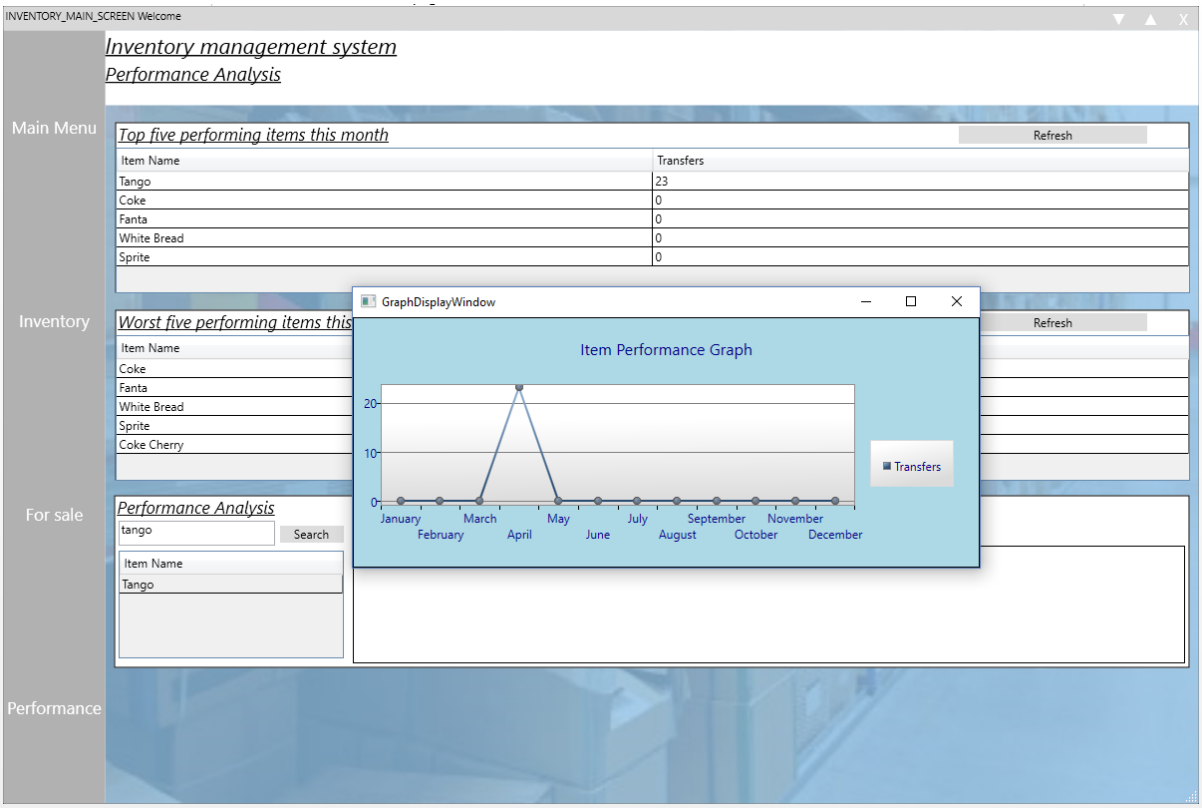



The message is correct, since the item has only been transferred in the month April. However the worst month information is incorrect since every other month also has 0 transfers.

*Task 14*

To generate a graph I just clicked on the generate graph button, with the tango item selected.

This resulted in the following graph being displayed:




This concludes the end to end test of the solution.

## 1.3 Usability testing

The purpose of usability testing is to test how easy it to use the solution. This will be done by having real users test the solution by completing a series of tasks that I will create before the test. This will allow me to see where the user encounters problems and experiences confusion. As well as observing the user complete the task, I will also ask the users to complete a short survey after they have completed each task.

### *1.3.1 Usability test*

The tasks that each user will need to complete is shown below:

1) Create an account
    - Choose the manager user group
2) Add 3 new item to inventory
    - One item must be a food item
    - One item must be a drink item
    - The last item can be in any category
3) Filter the inventory table to only show items that are food items
4) Select the food items. Transfer 1 of these items to the for sale table.
    - Add a random sell by date to the item
5) Check that the item has been added to the for sale table
6) Transfer the rest of the items in inventory to the for sale page
    - Adding random sell by dates when needed
7) Lookup the performance analysis for one of the items transferred.
8) Generate a performance report.

*1.3.2 Usability survey*

Inventory Management Usability survey

The purpose of this survey is to gain an understanding of how easy it is to use solution I have created.

1) What would you rate the solution out of 10?

2) Are there any improvements that could be made to the account creation process?

3) Out of ten, how easy was it to add an item to inventory?

4) Please describe any issues you had when transferring items from inventory to the for sale table

5) Are there any improvements that could be made to the transferring items process?

6) Please describe any other improvements that could be made to the solution

Thank you for completing this survey

*User 1 observation*

User 1 is my client, Purnesh Patel.

Initial reactions from my client as the program loaded was that it was very professional looking and appealing. This was after seeing the splash and login screen.

My client was quick to navigate to the account creation page. He then filled out the required fields, choosing the manager user group. After creating the account the success message box confirmed the username and password created. My client was not happy about this, he thought this could have been a security issue, if other staff members were looking at the screen whilst the account was being created, then they would be able to see the password too.

He then went back to the main login page and logged into the system using the account he just created. He then maximised the page using the upward arrow button. It was here I had to intervene to tell my client how to restore the window back to its previous state (by clicking on the rectangle at the top of the page).

After, he went to the inventory page and added 3 items with no issues. He was disappointed to see that there was no option to add more categories.

Choosing filtering options seemed straightforward as my client seemed to have no issues with doing this.

This biggest problem came with trying to select 1 item. I had to intervene and show my client how to select only 1 item. However once I did this, he was able to transfer the item and add a sell by date. He verified that it actually transferred by navigating to the for sale page.

Going back to the inventory page and transferring the rest of the item was also straightforward. He chose to add a different sell by date to these items. He said this was useful as it is often the case that he will have multiple items of different sell by dates.

Generating a performance report was straightforward for the client. He also thought that the performance graph was a very nice visual representation of an items performance. However he thought the performance analysis lacked enough detail.

*User 1 survey results*

After completing the tasks I had given to my client, I gave my client the usability survey complete. His results are shown below:

1) *What would you rate the solution out of 10?*
    - *6/10*
2) *Are there any improvements that could be made to the account creation process?*
    - Yes, when I created an account has been created, it was worrying to see the password I entered show up on the screen. Other staff members around at the time may be able to see this password when it's shown. They could access my account with permission. Please remove this feature.
3) *Out of ten, how easy was it to add an item to inventory?*
    - 9/10
4) *Please describe any issues you had when transferring items from inventory to the for sale table*
    - The main issue I had was that it not obvious at all for how to select a set quantity of an item. Selecting all of an item was much easier.

5) *Are there any improvements that could be made to the transferring items process?*
    - Yes, the transferring process has been made very well, but there is not enough help tools or instructions for transferring a set amount of an item instead of all of it. The program is going to be used by people who do not have much computer experience at all and I think they would struggle to use this feature.
6) *Please describe any other improvements that could be made to the solution*
    - The performance analysis of an item is not in enough depth. Trends can be shown over the year but it would also be useful if the analysis gave an items trends over weeks or days.
    - It is not practical to go around the shop every night updating the for sale table with the actual quantity of items out for sale. Maybe the mobile app could built at a later date that we spoke about a while ago.
    - There is not enough help facilities. If someone doesn't know how to use the program, there is no way of learning how to use it, unless they are being taught by someone who know how to use the program.

*User 2 observation*

User 2, Sameer Patel, is a senior staff member at the shop. He has a lot of experience with computers and shouldn't run into any issues with using the solution.

He created an account with no issues. He didn't raise the issue about showing the username and password once the account was created.

There was also no issues with adding items to the inventory table. He also asked if there was a way to add more categories or even sub categories.

The only issue this user had was selecting one of the item to transfer instead of all that item. This is an issue that every user has experienced so far. He mentioned that there is no way of knowing that there is a way to actually select a single item or a set quantity.

The user was also disappointed with the performance analysis of items. He said whilst it gave useful information for a particular item, the information was not in enough depth, he said that as well as being able to view performance information for a particular month , it should also give performance information across a week or even days. He also added that the performance graph was very informative but was basic.

*User 2 survey results*

As with user 1, I asked user 2 to complete the same survey after they competed the tasks.

1) *What would you rate the solution out of 10?*
    - *8/10*
2) *Are there any improvements that could be made to the account creation process?*
    - This section of the program seems to work well the way it is. I had no issues with this process.
3) *Out of ten, how easy was it to add an item to inventory?*
    - 10/10
4) *Please describe any issues you had when transferring items from inventory to the for sale table*

- Selecting a single item was easy to do once I knew how to do it. The program did not make it clear how to do this.

5) *Are there any improvements that could be made to the transferring items process?*
    - More help is needed for new users. Apart from that, the process was easy and straightforward.
6) *Please describe any other improvements that could be made to the solution*
    - The performance analysis section needs to be in more depth.
    - The performance graph should allow you to view the exact transfers made when the mouse hovers over the line.

## 2 Success criteria evaluation

In this section I will evaluate whether the success criteria/ requirements of the solution have been met or not.

### *The solution must be able to input new stock items.*

This requirement has been met. During User testing, both users were able to do this task without any issues. During the end to end test of the system, I was able to complete this task. I also tested this task whilst testing the system against the test plan (Test 1). This requirement has been fully met.

This also links back to the success criteria I set out in the analysis stage of this document. It stated that, "If the item is shown to any user after the item has been added then it would have been successfully added and retained by the solution". This has been demonstrated in all testing conducted, including testing conducted during the development of the solution. Items added into inventory page persisted once the solution was closed, re opened and used by different accounts.

### *The solution must be able to take out stock items.*

This requirement was demonstrated during all types of testing I conducted. During usability testing, it was made clear that this feature of the solution was not implemented well and that it was difficult to take out a specific amount of an item instead of all of the item. For that reason, this requirement has been partially met.

In the solutions success criteria, it was stated that "Items should be able to move from inventory to the shops shelves. In the solution, this involves moving it from one table to another". Testing I have conducted has proven that this happens. Items move from the inventory table to the for table, allowing for the solution to differentiate properly between items that are currently in inventory and items that are for sale.

### *The solution must be accessible from different computers*

As shown during testing against the test plan, it was difficult to test this feature of the solution. I did however, test using two instances of the solution at the same time to emulate using it across different computers. This was successful but didn't fully show that the solution can be used across different computers. Therefore, this requirement has been partially met.

Going back to the solutions success criteria, it stated that, "Data held about items should be accessible on one machine and another, in different locations and on another network". This hasn't been demonstrated in all testing done. This is due to the fact that I did not have access to machines in different locations or on different networks.

*The solution should be useable by inexperienced users*

Usability testing and testing against the test plan showed that this requirement has not been met. Every user who tried the solution has said that the solution was not suitable for new users due to it lacking help information. This requirement has not been met. This was shown to be mainly because of the fact that the solution lacked any help tools such as tool tips or a user guide. These are things that could be implemented at a later data however.

*It must be able to find popular products*

During an end to end test of the solution and testing against the test plan. I tested fully the solutions ability to find popular items, with success every time. For that reason, this requirement has been fully met.

As stated in the solutions success criteria, "When an empty database used by the solution is filled with the random items with random sales data, the solution should be able to find the five most popular items in the data being held". This was exactly proved when I conducted testing of the solution against the test plan. The items were in the correct order.

*It must be able to find unpopular products*

As with the previous requirement, this requirement was demonstrated during an end to end test of the solution and testing against the test plan. It too was also very successful, meaning this requirement has been fully met.

As stated in the solutions success criteria, "When an empty database used by the solution is filled with the random items with random sales data, the solution should be able to find the five most unpopular items in the data being held." This was exactly proved when I conducted testing of the solution against the test plan. The items were in the correct order.

*It must be able to predict when products will sell well*

This requirement was partially met. Usability testing showed that predictions or item analysis was not in enough depth. However end to end and test plan testing showed that predictions worked despite not being in enough depth.

The solutions success criteria stated that, "When a database used by the solution is holding the sales data about of a single item, the solution should be able to predict when the item will sell best, giving a particular date range". This has been demonstrated in the end to end test, when an item was selected, the solution was able to tell the user when the particular item was likely to sell best, given in the particular month.

*Data stored in the solution must be valid*

This requirement was mostly demonstrated during development, when I would test validation rules I setup on data entry fields such as entering account details or adding items to inventory. I also tested validation whilst testing the solution against the test plan. This requirement has been met.

*The solution must keep a log of all actions that have taken place by users*

This requirement was demonstrated during testing against the test plan and during an end to end test of the solution. No issues have been found using this feature except that log entries show oldest log entries first instead of the latest log entries. This requirement was met.

In the success criteria of the solution, it stated that, “When the user completes an action such as creating a new account, an action event should be added to a log stored by the solution. It would include the time of the action and the username of user completing the action”. In addition to demonstrating this, the solution also is able to filter the log entries based on a particular date to log entries after, it also is able store first name of the user who made the action, making it easier for the username to quickly identify the person who made the action.

## 3 Unmet success criteria

In this section I will go through every requirement that was either not met or partially met and address how they could fixed in further development.

### *The solution must be able to take out stock items.*

In terms of functionality this requirement was met, however in terms of usability it was not met.

### *The solution must be accessible from different computers*

This requirement was partially met due to limitations of hardware available to test it. Once the SQL Server Database is being hosted by a provider such as Azure, I would be able to test the solution being ran on different computers.

### *The solution should be useable by inexperienced users*

This requirement could be resolved by adding more help facilities such as tool tips, help dialogs and documentation. This would allow users to learn how to use the program independently rather than having someone else be there to teach them.

### *It must be able to predict when products will sell well*

This requirement was partially met. In the further development, I would need to record the transfers taking place daily or for a given time per item. This way, I could go in enough detail to cover selling trends of an item over a couple of hires to multiple years. This would give a full range of detail to this feature.

## 4 Resolving usability issues

### 4.1 Account creation message

One issue that was found during usability testing was there was a potential security issue when the user creates an account. When the user submits their account details causing a new account to be created, a message box is shown, telling the user the username they have chosen and the password they chose. The issue with this is that other people watching the user will also see this message, also knowing their password.

To resolve this, I will remove this message from being shown. In the future, an emailing system could be developed to send the user an email confirming their account details.

## 4.2 Transferring some but not all of an item

This seemed to be the main issue with the usability of the system. When users were asked to transfer a single item (instead of all of an item) from inventory to the for sale table, I was forced to intervene and explain how to do this.

Looking back now, I can see that it not clear how to do this. If not told how to do it, it is not obvious that the user is required to select the items record and alter its quantity on the data grid directly then click on the transfer button.

Although, initially it wasn't clear how to use this feature, once users were told how to use this feature, users found it easy to use and didn't have issues after.

To resolve this issue, I would outline the exact the steps required in order to use this feature in documentation created in the future. I would also add more helpful information inside the program, such as tool tips. For example, when the user's mouse is hovering over the transfer button, a tool tip should appear saying that a specific quantity of the item can be transferred by changing the quantity field of the items record before clicking the transfer button.

# 5 Maintenance issues and limitations of the solution

## 5.1 Maintenance issues

### *5.1.1 Database expansion*

As the number of items being stored in the system increases, the amount of database storage being used will increase. Once database storage has ran out, new items will no longer be able to be saved to the database. This is true if the database is being hosted by a SQL Server provider such as Microsoft Azure.

In order to re gain the ability to save new items to the database again the user will need to upgrade their database package to the next available tier. This will allow for the capacity of the database to increase, allowing for more items to be stored in the database. Other perks may be introduced when upgrading the database package such as increased performance. The time taken to read or write data to the database may also decrease, resulting in a better user experience as users are not left waiting for data to fetch or wrote.

On the other hand, upgrading the database package involves a cost to the user. To upgrade to the next available package, an increase in the subscription cost may be required in order to access better features they have to offer.

### *5.1.2 Updates*

As of now, once the solution is updated and new features are implemented, there is no way to send these updates the user.

When an update is released, I would have to physically send the update to the user, either in person or by sending them a download link. This is both time consuming for me and the user. In addition to this, I would have no control over the installation of this update, the user may not properly install the update or new version, resulting in the program breaking.

## 5.2 Solution limitations

### *5.2.1 Internet connection*

The solution requires an internet connection to function. Without an internet connection, the program has no way of getting data from the database. If the user loses their internet connection, they will lose access to their data, meaning they will have no access to their inventory data, for sale data or performance data. They would have to resort to using their old system whilst an internet connection is recovered.

Furthermore, a slow internet connection would result in the solution being much slower as the time taken to fetch data or write date to the database would increase. A faster internet connection would be optimal, so that waiting times are kept to a minimal, making the business more efficient.

### *5.2.2 Link to actual sales data*

Currently, the solution has no link to actual sales data. The reason for this, is that I had no way of interfacing with sales hardware such as a bar code scanner. I have instead used another form of measuring the success of a product, the number of times an item is transferred from inventory to the for sale table. However, knowing the exact number of times an item had been sold would be far more beneficial.

### *5.2.3 Manual updating of data at the end of every day*

Another issue that arose from not having actual sales data, is that the solution has no way of knowing the actual quantity of items out for sale at the end of day. Since items are being sold, the quantity of an item out for sale at the end of the day may be less than it was at the start of the day. For that reason, the user has to manually check the quantity of each item at the end of each day updating their quantity values.

## 5.3 Resolving solution limitations

### *5.3.1 Resolving Internet connection limitations*

The solution will always be dependent on an internet connection, however, when an internet connection is not available, a local version of the database could be used instead. This local version of the database could be a snapshot taken of the main database at the start of the week. This way, the changes the user will still have access to their data (although it may not be completely up to date). Changes made to the local database could then be updated/merged to the actual database when an internet connection is restored.

### *5.3.2 Resolving the sales data limitation*

The only way around this issue would be to have the ability to interface with the current sales system. If I can take data from that, I could automatically update the sales data for each item and perform much deeper performance analysis's on items.

### *5.3.3 Resolving the issue of having to manually update tables*

Similarly to the previous issue, this issue could be resolved by directly interfacing with the sales system. This way, once a sale is made I could link the sale to items for sale record and alter its quantity. This way, data would stay up to date throughout the day.

Another way of resolving this issue would be to develop a smart phone application. This wouldn't be as beneficial as interfacing with the sales system as it would still require the manual entry of data from the shop at the end of every day. However, this would be made easier for the user to do , since they could update data as they walk around the shop instead of having to check the quantity of an item and go back to their computer to update the data.

# Code listing

## LoginAbout.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.Login_controls
{
    /// <summary>
    /// Interaction logic for LoginAbout.xaml
    /// </summary>
    public partial class LoginAbout : UserControl
    {
        public LoginAbout()
        {
            InitializeComponent();
        }
    }
}
```

## LoginCreateAccount.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.Login_controls
{
    /// <summary>
    /// Interaction logic for LoginCreateAccount.xaml
    /// </summary>
    public partial class LoginCreateAccount : UserControl
    {
        SQLServerAccessor SqlLink = new SQLServerAccessor();

        public LoginCreateAccount()
        {
            InitializeComponent();
        }
        private void ClearOnFirst(ref TextBox tb , string TextToCheck)
        /*Pass by reference the textbox being cleared and its default text*/
        {
            if (tb.Text == TextToCheck) /*If the default text exists , then clear it*/
            {
                tb.Text = "";
            }

        }
        private void username_textBox_GotFocus(object sender, RoutedEventArgs e)
            /*When the user clicks on a textbox call the ClearOnFirst method */
        {
            ClearOnFirst(ref username_textBox , "USERNAME");
        }

        private void firstname_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            ClearOnFirst(ref firstname_textBox, "FIRST NAME");
        }

        private void lastname_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            ClearOnFirst(ref lastname_textBox, "LAST NAME");
        }

        private void password_textBox_GotFocus(object sender, RoutedEventArgs e)
```

```
        {
            ClearOnFirst(ref password_textBox, "PASSWORD");
        }



        private void clear_button_Click(object sender, RoutedEventArgs e)
        {
            /*Sets all of the textboxes text to its default values*/
            username_textBox.Text = "USERNAME";
            firstname_textBox.Text = "FIRST NAME";
            lastname_textBox.Text = "LAST NAME";
            password_textBox.Text = "PASSWORD";
            role_ComboBox.Text = "ROLE";
        }

        private void submit_button_Click(object sender, RoutedEventArgs e)
        {
            //check each field is not its default value
            if ((username_textBox.Text != "USERNAME") && (firstname_textBox.Text !=
"FIRST NAME")
                && (lastname_textBox.Text != "LAST NAME") && (password_textBox.Text !=
"PASSWORD")
                && (role_ComboBox.Text != "ROLE")
                )
            {
                if (SqlLink.CreateNewAccount(username_textBox.Text,
firstname_textBox.Text, lastname_textBox.Text, password_textBox.Text,
role_ComboBox.Text))
                {
                    MessageBox.Show("Account created! Username " +
username_textBox.Text + " Password " + password_textBox.Text);
                }
                else
                {
                    MessageBox.Show("Username is already taken , please choose
another");
                }
            }
            else
            {
                MessageBox.Show("You must fill out all fields");
            }

        }
    }
}
```

## LoginForgotPassword.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.Login_controls
{
    /// <summary>
    /// Interaction logic for LoginForgotPassword.xaml
    /// </summary>
    public partial class LoginForgotPassword : UserControl
    {
        public LoginForgotPassword()
        {
            InitializeComponent();
        }
    }
}
```

## LoginHelp.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.Login_controls
{
    /// <summary>
    /// Interaction logic for LoginHelp.xaml
    /// </summary>
    public partial class LoginHelp : UserControl
    {
        public LoginHelp()
        {
            InitializeComponent();
        }
    }
}
```

## LoginMainCtrl.xaml

```
namespace InventoryManagementALEVEL.User_controls.Login_controls
{
    /// <summary>
    /// Interaction logic for LoginMainCtrl.xaml
    /// </summary>
    public partial class LoginMainCtrl : UserControl
    {

        public LoginMainCtrl()
        {
            InitializeComponent();
        }
        SQLServerAccessor SqlLink = new SQLServerAccessor();
        private void login_button_Click(object sender, RoutedEventArgs e)
        {
            if (SqlLink.AuthenticateLogin(username_textBox.Text,
password_passwordBox.Password))
            {
                //create an instance of the main window and show it if the login was
authenticated
                MainWindow mw = new MainWindow(username_textBox.Text);
                mw.Show();
                var myWindow = Window.GetWindow(this);
```

```
                myWindow.Close();
            }
            else
            {
                error_label.Content = "Username or password incorrect , Please Try
Again";
            }
        }


        private void username_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            //if the default text is shown clear it on mouse click
            if (username_textBox.Text == "USERNAME")
            {
                username_textBox.Text = "";
            }
        }

        private void password_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            //causes the password field to be shown and set to the focus
            password_textBox.Visibility = Visibility.Hidden;
            Keyboard.Focus(password_passwordBox);
        }


    }
}
```

## MainForSale.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.MainWindow_controls
{
    /// <summary>
    /// Interaction logic for MainForSale.xaml
    /// </summary>
    public partial class MainForSale : UserControl
    {

        SQLServerAccessor SqlLink = new SQLServerAccessor();
        DispatcherTimer dispatcherTimer = new DispatcherTimer();
        public string _MainUsername;




        public MainForSale()
        {
            InitializeComponent();


            //populate the data grid with the for sale table initially
            for_sale_dataGrid.ItemsSource =
SqlLink.DataContextInv.ForSaleTables.ToList();
            //populate the category combo box with all the categories in the category
table.
            category_comboBox.ItemsSource = SqlLink.getCategories().ToList();
            category_comboBox.DisplayMemberPath = "CategoryName";
            category_comboBox.SelectedValuePath = "CategoryName";
            category_comboBox.SelectedIndex = 0;
```

```
            item_order_comboBox.Items.Add("Any");
            item_order_comboBox.Items.Add("Alphabetical");
            item_order_comboBox.SelectedIndex = 0;

            quantity_comboBox.Items.Add("Any");
            quantity_comboBox.Items.Add("Ascending");
            quantity_comboBox.Items.Add("Descending");
            quantity_comboBox.SelectedIndex = 0;



            dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
            dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
            dispatcherTimer.Start();
        }
        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            if (for_sale_dataGrid.SelectedItem == null)
            {
                editmode_label.Content = "";
                for_sale_dataGrid.ItemsSource =
SqlLink.getNewForSaleTable(category_comboBox.Text.Replace(" ", string.Empty),
name_textBox.Text);
                for_sale_dataGrid.Items.SortDescriptions.Clear();
                foreach (var cols in for_sale_dataGrid.Columns)
                {
                    cols.SortDirection = null;
                }
                if (item_order_comboBox.Text == "Alphabetical")
                {
                    //arrange the first column (item name) in alphabetical order
                    var col = for_sale_dataGrid.Columns[0];
                    for_sale_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

                    col.SortDirection = ListSortDirection.Ascending;
                }
                else if (quantity_comboBox.Text == "Ascending")
                {
                    //arrange the third column (quantity) in ascending order
                    var col = for_sale_dataGrid.Columns[2];
                    for_sale_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

                    col.SortDirection = ListSortDirection.Ascending;
                }
                else if (quantity_comboBox.Text == "Descending")
                {
                    //arrange the third column (quantity) in descending order
                    var col = for_sale_dataGrid.Columns[2];
                    for_sale_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Descending));

                    col.SortDirection = ListSortDirection.Descending;
                }

                for_sale_dataGrid.Items.Refresh();
            }
            else
            {
                editmode_label.Content = "edit mode enabled";
                Console.WriteLine("focused");
```

```
            }

        }


        private void add_button_Click(object sender, RoutedEventArgs e)
        {
            AddNewItemWindowForSale additemforsale = new AddNewItemWindowForSale();
            additemforsale.mainUsername = _MainUsername;
            additemforsale.Show();
        }
        private void delete_button_Click(object sender, RoutedEventArgs e)
        {

            //Check that an item has actually been selected to be deleted
            if (for_sale_dataGrid.SelectedItem != null)
            {
                //setup a list of item IDs , which have a datatype of integer
                List<int> itemIDs = new List<int>();
                //loop through the selected items
                if (for_sale_dataGrid.SelectedItems != null)
                {
                    foreach (ForSaleTable item in for_sale_dataGrid.SelectedItems)
                    {
                        //add the item ID of each item to the itemID list
                        itemIDs.Add(item.ItemID);
                        Console.WriteLine(item.ItemID.ToString());
                    }

                    if (SqlLink.deleteItemsForSale(itemIDs.ToArray() , _MainUsername))
                    {
                        MessageBox.Show("deleted");
                        Console.WriteLine("Updating table on delete");
                        //update the tables data to reflect the delete, also resumes
auto updating
                        for_sale_dataGrid.ItemsSource =
SqlLink.getNewForSaleTable(category_comboBox.Text.Replace(" ", string.Empty),
name_textBox.Text);
                    }
                }
            }
        }
        private void submit_changes_button_Click(object sender, RoutedEventArgs e)
        {
            if (SqlLink.GetAccountType(_MainUsername) != "Staff")
            {
                SqlLink.DataContextInv.SubmitChanges();
            }
        }
        private void reset_button_Click(object sender, RoutedEventArgs e)
        {
            category_comboBox.SelectedIndex = 0;
            item_order_comboBox.SelectedIndex = 0;
            quantity_comboBox.SelectedIndex = 0;
            name_textBox.Text = "";
        }

        private void Grid_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            for_sale_dataGrid.UnselectAll();
        }
    }
```

```
}
```

## MainInventory.xaml

```
namespace InventoryManagementALEVEL.User_controls.MainWindow_controls
{
    /// <summary>
    /// Interaction logic for MainInventory.xaml
    /// </summary>
    public partial class MainInventory : UserControl
    {
        SQLServerAccessor SqlLink = new SQLServerAccessor();

        PeformanceAnalysis perf = new PeformanceAnalysis();
        DispatcherTimer dispatcherTimer = new DispatcherTimer();

        public string _MainUsername;

        public MainInventory()
        {
            InitializeComponent();

            //populate the data grid with the inventory table initially
            inventory_dataGrid.ItemsSource =
SqlLink.DataContextInv.InventoryTables.ToList();
            //populate the category combo box with all the categories in the category
table.
            category_comboBox.ItemsSource = SqlLink.getCategories().ToList();

            category_comboBox.DisplayMemberPath = "CategoryName";
            category_comboBox.SelectedValuePath = "CategoryName";
            category_comboBox.SelectedIndex = 0;

            item_order_comboBox.Items.Add("Any");
            item_order_comboBox.Items.Add("Alphabetical");
            item_order_comboBox.SelectedIndex = 0;

            quantity_comboBox.Items.Add("Any");
            quantity_comboBox.Items.Add("Ascending");
            quantity_comboBox.Items.Add("Descending");
            quantity_comboBox.SelectedIndex = 0;


            dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
            dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
            dispatcherTimer.Start();
        }


        private void updateAll()
        {
            editmode_label.Content = "";

            inventory_dataGrid.ItemsSource =
SqlLink.getNewInventoryTable(category_comboBox.Text.Replace(" ", string.Empty),
name_textBox.Text);
            inventory_dataGrid.Items.SortDescriptions.Clear();
            foreach (var cols in inventory_dataGrid.Columns)
            {
                cols.SortDirection = null;
```

```
            }
            if (item_order_comboBox.Text == "Alphabetical")
            {
                //arrange the first column (item name) in alphabetical order
                var col = inventory_dataGrid.Columns[0];
                inventory_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

                col.SortDirection = ListSortDirection.Ascending;
            }
            else if (quantity_comboBox.Text == "Ascending")
            {
                //arrange the third column (quantity) in ascending order
                var col = inventory_dataGrid.Columns[2];
                inventory_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Ascending));

                col.SortDirection = ListSortDirection.Ascending;
            }
            else if (quantity_comboBox.Text == "Descending")
            {
                //arrange the third column (quantity) in descending order
                var col = inventory_dataGrid.Columns[2];
                inventory_dataGrid.Items.SortDescriptions.Add(new
SortDescription(col.SortMemberPath, ListSortDirection.Descending));

                col.SortDirection = ListSortDirection.Descending;
            }
            inventory_dataGrid.Items.Refresh();

        }
        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            //if the user has selected an item , then disable automatic updating
            if (inventory_dataGrid.SelectedItem == null)
            {
                updateAll();
            }
            else
            {
                editmode_label.Content = "edit mode enabled";
                Console.WriteLine("focused");
            }
        }

        private void GridMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            //deselect the datagrid, if the user clicks on the main grid
            inventory_dataGrid.UnselectAll();
        }

        private void add_button_Click(object sender, RoutedEventArgs e)
        {
            AddNewItemWindow Addwin = new AddNewItemWindow();
            Addwin.mainUsername = _MainUsername;
            Addwin.Show();
        }

        private void delete_button_Click(object sender, RoutedEventArgs e)
        {
            //Check that an item has actually been selected to be deleted
            if (inventory_dataGrid.SelectedItem != null)
```

```
            {
                //setup a list of item IDs , which have a datatype of integer
                List<int> itemIDs = new List<int>();
                //loop through the selected items
                foreach (InventoryTable item in inventory_dataGrid.SelectedItems)
                {
                    //add the item ID of each item to the itemID list
                    itemIDs.Add(item.ItemID);
                    Console.WriteLine(item.ItemID.ToString());
                }

                if (SqlLink.deleteItemsInventory(itemIDs.ToArray(),_MainUsername))
                {
                    MessageBox.Show("deleted");
                    Console.WriteLine("Updating table on delete");
                    //update the tables data to reflect the delete, also resumes auto
updating
                    inventory_dataGrid.ItemsSource =
SqlLink.getNewInventoryTable(category_comboBox.Text.Replace(" ", string.Empty),
name_textBox.Text);
                }
            }
        }



        private void submit_changes_button_Click(object sender, RoutedEventArgs e)
        {
            if (SqlLink.GetAccountType(_MainUsername) != "Staff")
            {
                SqlLink.DataContextInv.SubmitChanges();
            }
            else
            {
                MessageBox.Show("You do not have permission to change items details");
            }
        }

        private void reset_button_Click(object sender, RoutedEventArgs e)
        {
            category_comboBox.SelectedIndex = 0;
            item_order_comboBox.SelectedIndex = 0;
            quantity_comboBox.SelectedIndex = 0;
            name_textBox.Text = "";
        }

        private void transferitems_button_Click(object sender, RoutedEventArgs e)
        {
            if (SqlLink.GetAccountType(_MainUsername) != "Staff")
            {
                List<InventoryTable> ItemsToTrans = new List<InventoryTable>();//no
sell by date list
                List<InventoryTable> ItemsToTransSellByDate = new
List<InventoryTable>(); //sell by date list
                if (inventory_dataGrid.SelectedItem != null)
                {
                    if (inventory_dataGrid.SelectedItems != null)
                    {
                        //iterate through the users selection
                        foreach (InventoryTable item in
inventory_dataGrid.SelectedItems)
                        {
```

```
                            //if the item is a food or drink item , add it to the sell
by date list
                            if ((item.CategoryName.Replace(" ", "") == "Food") ||
(item.CategoryName.Replace(" ", "") == "Drink"))
                            {
                                ItemsToTransSellByDate.Add(item);
                            }
                            else
                            {
                                ItemsToTrans.Add(item);
                            }
                        }
                    }
                }
                inventory_dataGrid.UnselectAll();
                updateAll();

                if (ItemsToTransSellByDate.Count != 0)
                //if there are any items to transfer with a sell by date
                {
                    foreach (InventoryTable itemToPushWithSellByDate in
ItemsToTransSellByDate)
                    {
                        if
(perf.GetPerformanceThisMonthAsBool(itemToPushWithSellByDate.ItemName))
                        {
                            MessageBox.Show("Warning! The item " +
itemToPushWithSellByDate.ItemName.Replace(" ", "") + " Has shown to not sell well
during this month");
                        }
                        //iterate through the list and instantiate the new window ,
sending in each item as a parameter
                        transferItemWindow transitemsellby = new
transferItemWindow(itemToPushWithSellByDate);
                        transitemsellby.MainUserName = _MainUsername;
                        transitemsellby.Show();

                    }
                    //transfer the items that dont have a sell by date
                    SqlLink.TransferItemsToForSaleNoSellBy(ItemsToTrans,
_MainUsername);
                }
                else
                {
                    foreach (InventoryTable nextItem in ItemsToTransSellByDate)
                    {
                        if (perf.GetPerformanceThisMonthAsBool(nextItem.ItemName))
                        {
                            MessageBox.Show("Warning! The item " +
nextItem.ItemName.Replace(" ", "") + " Has shown to not sell well during this month");
                        }
                    }
                    //only transfer items that have no sell by date
                    SqlLink.TransferItemsToForSaleNoSellBy(ItemsToTrans,
_MainUsername);

                }
            }
            else
            {
                MessageBox.Show("You do not have permission to transfer items");
            }
```

```
        }

    }
}
```

MainPage.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.MainWindow_controls
{
    /// <summary>
    /// Interaction logic for MainPage.xaml
    /// </summary>
    public partial class MainPage : UserControl
    {
        public string MainUsername;
        SQLServerAccessor SqlLink = new SQLServerAccessor();

        public MainPage()
        {
            InitializeComponent();

        }

        private void log_button_Click(object sender, RoutedEventArgs e)
        {
            //only show the log window if the user is an admin or manager
            if (SqlLink.GetAccountType(MainUsername) == "Manager" ||
SqlLink.GetAccountType(MainUsername) == "Admin")
            {
                logWindow logwin = new logWindow();
                logwin.Show();
            }
            else
            {
                MessageBox.Show("You do not have permission to view the log window");
            }
        }
    }
}
```

MainPerformance.xaml.cs

```
namespace InventoryManagementALEVEL.User_controls.MainWindow_controls
{
    /// <summary>
    /// Interaction logic for MainPerformance.xaml
    /// </summary>
    public partial class MainPerformance : UserControl
    {

        PeformanceAnalysis perf = new PeformanceAnalysis();
        public MainPerformance()
        {
            InitializeComponent();
            TopFive_dataGrid.ItemsSource = perf.GetTopFiveThisMonth();
            TopFiveWorst_dataGrid.ItemsSource = perf.GetTopFiveWorstThisMonth();
        }

        private void topfive_refresh_button_Click(object sender, RoutedEventArgs e)
        {
            TopFive_dataGrid.ItemsSource = perf.GetTopFiveThisMonth();
            TopFive_dataGrid.Items.Refresh();
        }

        private void topfiveworst_refresh_button_Click(object sender, RoutedEventArgs
e)
        {
            TopFiveWorst_dataGrid.ItemsSource = perf.GetTopFiveWorstThisMonth();
            TopFiveWorst_dataGrid.Items.Refresh();
        }

        private void submit_search_button_Click(object sender, RoutedEventArgs e)
        {
            if (search_textBox.Text != "Enter Item Name")
            {
                //populate the data grid with the performance records similar to the
input item name
                SearchResults_dataGrid.ItemsSource =
perf.GetSearchResults(search_textBox.Text);
                SearchResults_dataGrid.Items.Refresh();
            }
        }


        PerformanceTable perfItem = new PerformanceTable();
        private void generate_graph_button_Click(object sender, RoutedEventArgs e)
        {
            GraphDisplayWindow gdw = new GraphDisplayWindow(perfItem);
            gdw.Show();
        }

        private void selectItem_button_Click(object sender, RoutedEventArgs e)
        {

            if (SearchResults_dataGrid.SelectedItem != null)
            {
                //just take the first item
                foreach (PerformanceTable item in
SearchResults_dataGrid.SelectedItems)
```

```
                {
                    perfItem = item;
                    break;
                }
                report_textBox.Text = perf.GeneratePerformanceAnalysisText(perfItem);
                generate_graph_button.IsEnabled = true;
            }



        }

        private void search_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (search_textBox.Text == "Enter Item Name")
            {
                search_textBox.Text = "";
            }
        }
    }
}
```

## AddNewItemWindow.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for AddNewItemWindow.xaml
    /// </summary>
    public partial class AddNewItemWindow : Window
    {
        SQLServerAccessor SQLlink = new SQLServerAccessor();
        public string mainUsername;


        public AddNewItemWindow()
        {
            InitializeComponent();
            //populate the category combobox with all the categories in the category
table
            Category_comboBox.ItemsSource = SQLlink.getCategories().ToList();
            Category_comboBox.DisplayMemberPath = "CategoryName";
            Category_comboBox.SelectedValuePath = "CategoryName";
            Category_comboBox.SelectedIndex = 0;

        }
        private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            DragMove();

        }
        private void add_button_Click(object sender, RoutedEventArgs e)
        {
            Console.WriteLine(Category_comboBox.Text);
            int passedval;
            //validate the input fields, is the name field empty? is the quantity an
integer?
            if ((!string.IsNullOrWhiteSpace(ItemName_textBox.Text)) &&
(int.TryParse(Quantity_textBox.Text, out passedval)))
            {
```

```
                try
                {
                    //insert the new item
                    if (SQLlink.addNewItemInventory(ItemName_textBox.Text,
Convert.ToInt32(Quantity_textBox.Text), Category_comboBox.Text.Replace(" ",
string.Empty), mainUsername))
                    {
                        MessageBox.Show("Added new item");
                        this.Close();
                    }
                }
                catch (Exception en) { Console.WriteLine(en.ToString()); }

            }
            else if(string.IsNullOrWhiteSpace(ItemName_textBox.Text))
            {
                MessageBox.Show("You must not leave the item name field empty");
            }
            else if (!int.TryParse(Quantity_textBox.Text, out passedval))
            {
                MessageBox.Show("The quantity you have entered is not an integer");
            }

        }

        private void reset_button_Click(object sender, RoutedEventArgs e)
        {
            ItemName_textBox.Text = "Item name";
            Quantity_textBox.Text = "Quantity";
            Category_comboBox.SelectedIndex = 0;
        }

        private void cancel_button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

        private void ItemName_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (ItemName_textBox.Text == "Item name")
            {
                ItemName_textBox.Text = "";
            }
        }

        private void Quantity_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (Quantity_textBox.Text == "Quantity")
            {
                Quantity_textBox.Text = "";
            }
       }


    }
}
```

AddNewItemWindowForSale.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for AddNewItemWindowForSale.xaml
    /// </summary>
    public partial class AddNewItemWindowForSale : Window
    {

        SQLServerAccessor SQLlink = new SQLServerAccessor();
        public string mainUsername;
        public AddNewItemWindowForSale()
        {
            InitializeComponent();
            Category_comboBox.ItemsSource = SQLlink.getCategories().ToList();
            Category_comboBox.DisplayMemberPath = "CategoryName";
            Category_comboBox.SelectedValuePath = "CategoryName";
            Category_comboBox.SelectedIndex = 0;
        }

        private void Window_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            DragMove();
        }

        private void ItemName_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (ItemName_textBox.Text == "Item name")
            {
                ItemName_textBox.Text = "";
            }
        }

        private void Quantity_textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (Quantity_textBox.Text == "Quantity")
            {
                Quantity_textBox.Text = "";
            }
        }


        private void add_button_Click(object sender, RoutedEventArgs e)
        {
            int passedval;
            Console.WriteLine(Category_comboBox.Text);
            //validate the input fields, is the name field empty? is the quantity an
integer?
            if ((!string.IsNullOrWhiteSpace(ItemName_textBox.Text)) &&
(int.TryParse(Quantity_textBox.Text, out passedval)))
            {
                try
                {
                    //insert the new item
                    if (SQLlink.addNewItemForSale(ItemName_textBox.Text,
Convert.ToInt32(Quantity_textBox.Text), Category_comboBox.Text.Replace(" ",
string.Empty), SellByDate_datePicker.SelectedDate.Value , mainUsername))
                    {
                        MessageBox.Show("Added new item");
                        this.Close();
                    }
                }
```

```
                catch (Exception en) { Console.WriteLine(en.ToString()); }

            }
            else if (string.IsNullOrWhiteSpace(ItemName_textBox.Text))
            {
                MessageBox.Show("You must not leave the item name field empty");
            }
            else if (!int.TryParse(Quantity_textBox.Text, out passedval))
            {
                MessageBox.Show("The quantity you have entered is not an integer");
            }
        }

        private void reset_button_Click(object sender, RoutedEventArgs e)
        {
            ItemName_textBox.Text = "Item name";
            Quantity_textBox.Text = "Quantity";
            Category_comboBox.SelectedIndex = 0;
        }

        private void cancel_button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }



        private void Category_comboBox_DropDownClosed(object sender, EventArgs e)
        {
            Console.WriteLine(Category_comboBox.Text.Replace(" ", ""));
            //enable the date picker if they have chosen food or drink
            if (Category_comboBox.Text.Replace(" ", "") == "Food" ||
Category_comboBox.Text.Replace(" ", "") == "Drink")
            {
                SellByDate_datePicker.IsEnabled = true;
            }
        }
    }
}
```

## GraphDisplayWindow.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for GraphDisplayWindow.xaml
    /// </summary>
    public partial class GraphDisplayWindow : Window
    {
        public GraphDisplayWindow(PerformanceTable perfData)
        {
            InitializeComponent();
            //links the x axis (month) to the transfers for that month ( y axis)
            ((LineSeries)Perf_Chart.Series[0]).ItemsSource =
            new KeyValuePair<string, int>[]{
            new KeyValuePair<string, int>("January", perfData.January.Value),
            new KeyValuePair<string, int>("February", perfData.February.Value),
            new KeyValuePair<string, int>("March", perfData.March.Value),
            new KeyValuePair<string, int>("April", perfData.April.Value),
            new KeyValuePair<string, int>("May", perfData.May.Value),
```

```
            new KeyValuePair<string, int>("June", perfData.June.Value),
            new KeyValuePair<string, int>("July", perfData.July.Value),
            new KeyValuePair<string, int>("August", perfData.August.Value),
            new KeyValuePair<string, int>("September", perfData.September.Value),
            new KeyValuePair<string, int>("October", perfData.October.Value),
            new KeyValuePair<string, int>("November", perfData.November.Value),
            new KeyValuePair<string, int>("December", perfData.December.Value),};

        }
    }
}
```

## Login.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for Login.xaml
    /// </summary>
    public partial class Login : Window
    {
            /*Global var instances for each usercontrol*/
        LoginMainCtrl loginMainIns = new LoginMainCtrl();
        LoginCreateAccount loginCreateAccountIns = new LoginCreateAccount();
        LoginForgotPassword loginForgotPasswordIns = new LoginForgotPassword();
        LoginAbout loginAboutIns = new LoginAbout();
        LoginHelp loginHelpIns = new LoginHelp();

        public Login()
        {
            InitializeComponent();
            /*Set the contents of the page to the login main page user control*/
            this.mainContentControl.Content = loginMainIns;
        }
        private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            /*Allows the user to drag the screen around the window*/
            DragMove();
        }

        private void forgotten_password_button_Click(object sender, RoutedEventArgs e)
        {
            /*Set the contents of the page to the forgotten password user control
             When the user clicks on the forgotten password button*/
            this.mainContentControl.Content = loginForgotPasswordIns;
        }

        private void home_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = loginMainIns;
        }

        private void create_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = loginCreateAccountIns;
        }

        private void about_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = loginAboutIns;
```

```
        }

        private void help_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = loginHelpIns;
        }
    }
}
```

logWindow.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for logWindow.xaml
    /// </summary>
    public partial class logWindow : Window
    {
        SQLServerAccessor SqlLink = new SQLServerAccessor();

        public logWindow()
        {
            InitializeComponent();

            log_dataGrid.ItemsSource = SqlLink.DataContextInv.EventLogTables.ToList();
        }

        private void Window_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            DragMove();
        }

        private void close_button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

        private void log_DatePicker_CalendarClosed(object sender, RoutedEventArgs e)
        {
            if (log_DatePicker.SelectedDate != null)
            {
                log_dataGrid.ItemsSource =
SqlLink.GetLogEntriesAfterDate(log_DatePicker.SelectedDate.Value);
                log_dataGrid.Items.Refresh();
            }
        }
    }
}
```

MainWindow.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        MainPage mainPageIns = new MainPage();
        MainInventory mainInventoryIns = new MainInventory();
        MainForSale mainForSaleIns = new MainForSale();
        MainPerformance mainPerformanceIns = new MainPerformance();
        SQLServerAccessor SqlLink = new SQLServerAccessor();


        public MainWindow(string _username)
        {
            InitializeComponent();
            //Set the content control to the main page instance when the page loads
            this.mainContentControl.Content = mainPageIns;
            //sets the maximum width and heigh to the size of the users screen when
maximising.
            this.MaxHeight = SystemParameters.MaximizedPrimaryScreenHeight;
            this.MaxWidth = SystemParameters.MaximizedPrimaryScreenWidth;
            mainInventoryIns._MainUsername = _username;
            mainPageIns.MainUsername = _username;
            mainForSaleIns._MainUsername = _username;
            MessageBox.Show("Welcome " + _username);
            windowtitle.Text += " Welcome " + _username;
            string AccountType = SqlLink.GetAccountType(_username).Replace(" ", "");
            //disable the performance button if the user doesnt have permission to
access it
            if (AccountType == "Staff" || AccountType == "Senior Staff")
            {
                Performance_button.IsEnabled = false;
            }

        }
        private void close_button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
        private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            DragMove();
        }

        private void Main_Menu_button_Click(object sender, RoutedEventArgs e)
        {
            //set the content control the main page instance
            this.mainContentControl.Content = mainPageIns;
        }

        private void Inventory_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = mainInventoryIns;
        }

        private void For_Sale_button_Click(object sender, RoutedEventArgs e)
        {
```

```
            this.mainContentControl.Content = mainForSaleIns;
        }

        private void Performance_button_Click(object sender, RoutedEventArgs e)
        {
            this.mainContentControl.Content = mainPerformanceIns;
            PeformanceAnalysis s = new PeformanceAnalysis();
        }

        private void maximize_button_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Maximized;

        }

        private void minimize_button_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Minimized;
        }

        private void topbar_Rectangle_MouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
        {
            //restores the window back to its dimensions before it was maximized
            this.WindowState = WindowState.Normal;
        }
    }
}
```

## PeformanceAnalysis.cs

```
namespace InventoryManagementALEVEL
{
    class PeformanceAnalysis
    {
        public class PerformanceObj
        {
            public string ItemName { get; set; }
            public int Transfers { get; set; }
        }


        SQLServerAccessor SqlLink = new SQLServerAccessor();

        public string GeneratePerformanceAnalysisText(PerformanceTable PerfData)
        {
            //create a an array where index 0 corresponds to january , 1 to February
and so on
            int[] TransfersMonths = new int[12];
            TransfersMonths[0] = PerfData.January.Value;
            TransfersMonths[1] = PerfData.February.Value;
            TransfersMonths[2] = PerfData.March.Value;
            TransfersMonths[3] = PerfData.April.Value;
            TransfersMonths[4] = PerfData.May.Value;
            TransfersMonths[5] = PerfData.June.Value;
            TransfersMonths[6] = PerfData.July.Value;
            TransfersMonths[7] = PerfData.August.Value;
            TransfersMonths[8] = PerfData.September.Value;
            TransfersMonths[9] = PerfData.October.Value;
            TransfersMonths[10] = PerfData.November.Value;
            TransfersMonths[11] = PerfData.December.Value;
            int lastHighestVal = 0;
```

```
            int lastHighestIndex = 0;
            int lastLowestVal = 0;
            int lastLowestIndex = 0;
            string worstMonth = "";
            string bestMonth = "";
            //iterate through the array
            for (int pos = 0; pos < TransfersMonths.Length; pos++)
            {
                //if its the first iteration then setup the variables
                if (pos == 0)
                {
                    lastHighestIndex = pos;
                    lastHighestVal = TransfersMonths[pos];
                    lastLowestIndex = pos;
                    lastLowestVal = TransfersMonths[pos];
                }
                //if the month being iterated through has had more transfers than last
highest month
                else if(TransfersMonths[pos] > lastHighestVal)
                {
                    //update the last highest values and index
                    lastHighestVal = TransfersMonths[pos];
                    lastHighestIndex = pos;
                }
                //if the month being iterated through has had less transfers than last
highest month
                else if (TransfersMonths[pos] < lastLowestVal)
                {
                    //update the last lowest values and index
                    lastLowestVal = TransfersMonths[pos];
                    lastLowestIndex = pos;
                }
            }

            if (lastHighestIndex == 0 )
            {
                bestMonth = "January";
            }
            if (lastHighestIndex == 1 )
            {
                bestMonth = "February";
            }
            if (lastHighestIndex == 2 )
            {
                bestMonth = "March";
            }
            if (lastHighestIndex == 3 )
            {
                bestMonth = "April";
            }
            if (lastHighestIndex == 4 )
            {
                bestMonth = "May";
            }
            if (lastHighestIndex == 5 )
            {
                bestMonth = "June";
            }
            if (lastHighestIndex == 6 )
            {
                bestMonth = "July";
            }
```

```
if (lastHighestIndex == 7 )
{
    bestMonth = "August";
}
if (lastHighestIndex == 8 )
{
    bestMonth = "September";
}
if (lastHighestIndex == 9 )
{
    bestMonth = "October";
}
if (lastHighestIndex == 10 )
{
    bestMonth = "November";
}
if (lastHighestIndex == 11 )
{
    bestMonth = "December";
}

if (lastLowestIndex == 0)
{
    worstMonth = "January";
}
if (lastLowestIndex == 1)
{
    worstMonth = "February";
}
if (lastLowestIndex == 2)
{
    worstMonth = "March";
}
if (lastLowestIndex == 3)
{
    worstMonth = "April";
}
if (lastLowestIndex == 4)
{
    worstMonth = "May";
}
if (lastLowestIndex == 5)
{
    worstMonth = "June";
}
if (lastLowestIndex == 6)
{
    worstMonth = "July";
}
if (lastLowestIndex == 7)
{
    worstMonth = "August";
}
if (lastLowestIndex == 8)
{
    worstMonth = "September";
}
if (lastLowestIndex == 9)
{
    worstMonth = "October";
}
if (lastLowestIndex == 10)
```

```
            {
                worstMonth = "November";
            }
            if (lastLowestIndex == 11)
            {
                worstMonth = "December";
            }


            return (PerfData.ItemName.Replace(" ","") + "  has shown to sell best in
the month " + bestMonth + " and sells worst in the month " + worstMonth );



        }

        public List<PerformanceTable> GetSearchResults(string name)
        {
            return SqlLink.GetPerformanceItemsByLIKEName(name);
        }

        public PeformanceAnalysis()
        {

        }

        public bool GetPerformanceThisMonthAsBool(string ItemName)
        {
            //returns false is item sells good in current month , and true if not
            //if a performance record for the
            if (SqlLink.GetPerformanceItemsByName(ItemName).Count != 0)
            {
                //get the actual performance record
                var PerfData = SqlLink.DataContextInv.PerformanceTables.Single(x =>
x.ItemName == ItemName);
                //gets the current month
                string currentMonth = DateTime.Now.ToString("MMMM");
                int[] TransfersMonths = new int[12];
                TransfersMonths[0] = PerfData.January.Value;
                TransfersMonths[1] = PerfData.February.Value;
                TransfersMonths[2] = PerfData.March.Value;
                TransfersMonths[3] = PerfData.April.Value;
                TransfersMonths[4] = PerfData.May.Value;
                TransfersMonths[5] = PerfData.June.Value;
                TransfersMonths[6] = PerfData.July.Value;
                TransfersMonths[7] = PerfData.August.Value;
                TransfersMonths[8] = PerfData.September.Value;
                TransfersMonths[9] = PerfData.October.Value;
                TransfersMonths[10] = PerfData.November.Value;
                TransfersMonths[11] = PerfData.December.Value;

                if (currentMonth == "January")
                {
                    for (int pos = 0; pos < TransfersMonths.Length; pos++)
                    {
                        //if any other month had more sales than the current month ,
then its not likely to sell best in this month
                        if (TransfersMonths[0] < TransfersMonths[pos])
                        {
                            return true;
                        }
                        else
```

```
                {
                    return false;
                }
            }
        }
        if (currentMonth == "February")
        {
            for (int pos = 0; pos < TransfersMonths.Length; pos++)
            {
                if (TransfersMonths[1] < TransfersMonths[pos])
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        if (currentMonth == "March")
        {
            for (int pos = 0; pos < TransfersMonths.Length; pos++)
            {
                if (TransfersMonths[2] < TransfersMonths[pos])
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        if (currentMonth == "April")
        {
            for (int pos = 0; pos < TransfersMonths.Length; pos++)
            {
                if (TransfersMonths[3] < TransfersMonths[pos])
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        if (currentMonth == "May")
        {
            for (int pos = 0; pos < TransfersMonths.Length; pos++)
            {
                if (TransfersMonths[4] < TransfersMonths[pos])
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        if (currentMonth == "June")
```

```
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[5] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
if (currentMonth == "July ")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[6] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
if (currentMonth == "August")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[7] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
if (currentMonth == "September")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[8] < TransfersMonths[pos])
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
if (currentMonth == "October")
{
    for (int pos = 0; pos < TransfersMonths.Length; pos++)
    {
        if (TransfersMonths[9] < TransfersMonths[pos])
        {
            return true;
```

```
                        }
                        else
                        {
                            return false;
                        }
                    }
                }
                if (currentMonth == "November")
                {
                    for (int pos = 0; pos < TransfersMonths.Length; pos++)
                    {
                        if (TransfersMonths[10] < TransfersMonths[pos])
                        {
                            return true;
                        }
                        else
                        {
                            return false;
                        }
                    }
                }
                if (currentMonth == "December")
                {
                    for (int pos = 0; pos < TransfersMonths.Length; pos++)
                    {
                        if (TransfersMonths[11] < TransfersMonths[pos])
                        {
                            return true;
                        }
                        else
                        {
                            return false;
                        }
                    }
                }
            }
            else { return false; }
            return false;
        }

        public List<PerformanceObj> GetTopFiveThisMonth()
        {
            //get every performance record
            List<PerformanceTable> AllPerformanceItems =
SqlLink.GetPerformanceItems();
            //the top five items
            List<PerformanceTable> TopFivePerformance = new List<PerformanceTable>();
            //top five items as performance objects
            List<PerformanceObj> TopFivePerformanceObjects = new
List<PerformanceObj>();

            //get the current month
            string currentMonth = DateTime.Now.ToString("MMMM");

            if (currentMonth == "January")
            {
                //orders the list in descending order for the January column
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.January).ToList();
            }
            if (currentMonth == "February")
            {
```

```
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.February).ToList();
            }
            if (currentMonth == "March")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.March).ToList();
            }
            if (currentMonth == "April")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.April).ToList();
            }
            if (currentMonth == "May")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.May).ToList();
            }
            if (currentMonth == "June")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.June).ToList();
            }
            if (currentMonth == "July ")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.July).ToList();
            }
            if (currentMonth == "August")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.August).ToList();
            }
            if (currentMonth == "September")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.September).ToList();
            }
            if (currentMonth == "October")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.October).ToList();
            }
            if (currentMonth == "November")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.November).ToList();
            }
            if (currentMonth == "December")
            {
                AllPerformanceItems = AllPerformanceItems.OrderByDescending(o =>
o.December).ToList();
            }
            int x = 1;
            //add the first five items to the top five performance list
            foreach (PerformanceTable perfElement in AllPerformanceItems)
            {
                if (x <= 5)
                {
                    TopFivePerformance.Add(perfElement);
                    x++;
                }
```

```
        }
        foreach (PerformanceTable conv in TopFivePerformance)
        {
            PerformanceObj nextobj = new PerformanceObj();
            //setup the performance object
            nextobj.ItemName = conv.ItemName;
            if (currentMonth == "January")
            {
                nextobj.Transfers = conv.January.Value;
            }
            if (currentMonth == "February")
            {
                nextobj.Transfers = conv.February.Value;
            }
            if (currentMonth == "March")
            {
                nextobj.Transfers = conv.March.Value;
            }
            if (currentMonth == "April")
            {
                nextobj.Transfers = conv.April.Value;
            }
            if (currentMonth == "May")
            {
                nextobj.Transfers = conv.May.Value;
            }
            if (currentMonth == "June")
            {
                nextobj.Transfers = conv.June.Value;
            }
            if (currentMonth == "July ")
            {
                nextobj.Transfers = conv.July.Value;
            }
            if (currentMonth == "August")
            {
                nextobj.Transfers = conv.August.Value;
            }
            if (currentMonth == "September")
            {
                nextobj.Transfers = conv.September.Value;
            }
            if (currentMonth == "October")
            {
                nextobj.Transfers = conv.October.Value;
            }
            if (currentMonth == "November")
            {
                nextobj.Transfers = conv.November.Value;
            }
            if (currentMonth == "December")
            {
                nextobj.Transfers = conv.December.Value;
            }
            TopFivePerformanceObjects.Add(nextobj);
        }
        return TopFivePerformanceObjects;
    }

    public List<PerformanceObj> GetTopFiveWorstThisMonth()
    {
```

```
            List<PerformanceTable> AllPerformanceItems =
SqlLink.GetPerformanceItems();
            List<PerformanceTable> TopFiveWorstPerformance = new
List<PerformanceTable>();
            List<PerformanceObj> TopFivePerformanceObjectsWorst = new
List<PerformanceObj>();
            string currentMonth = DateTime.Now.ToString("MMMM");

            if (currentMonth == "January")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.January).ToList();
            }
            if (currentMonth == "February")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.February).ToList();
            }
            if (currentMonth == "March")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.March).ToList();
            }
            if (currentMonth == "April")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.April).ToList();
            }
            if (currentMonth == "May")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.May).ToList();
            }
            if (currentMonth == "June")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.June).ToList();
            }
            if (currentMonth == "July ")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.July).ToList();
            }
            if (currentMonth == "August")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.August).ToList();
            }
            if (currentMonth == "September")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.September).ToList();
            }
            if (currentMonth == "October")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.October).ToList();
            }
            if (currentMonth == "November")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.November).ToList();
```

```
            }
            if (currentMonth == "December")
            {
                AllPerformanceItems = AllPerformanceItems.OrderBy(o =>
o.December).ToList();
            }
            int x = 1;
            foreach (PerformanceTable perfElement in AllPerformanceItems)
            {
                if (x <= 5)
                {
                    TopFiveWorstPerformance.Add(perfElement);
                    x++;
                }
            }
            foreach (PerformanceTable conv in TopFiveWorstPerformance)
            {
                PerformanceObj nextobj = new PerformanceObj();
                nextobj.ItemName = conv.ItemName;
                if (currentMonth == "January")
                {
                    nextobj.Transfers = conv.January.Value;
                }
                if (currentMonth == "February")
                {
                    nextobj.Transfers = conv.February.Value;
                }
                if (currentMonth == "March")
                {
                    nextobj.Transfers = conv.March.Value;
                }
                if (currentMonth == "April")
                {
                    nextobj.Transfers = conv.April.Value;
                }
                if (currentMonth == "May")
                {
                    nextobj.Transfers = conv.May.Value;
                }
                if (currentMonth == "June")
                {
                    nextobj.Transfers = conv.June.Value;
                }
                if (currentMonth == "July ")
                {
                    nextobj.Transfers = conv.July.Value;
                }
                if (currentMonth == "August")
                {
                    nextobj.Transfers = conv.August.Value;
                }
                if (currentMonth == "September")
                {
                    nextobj.Transfers = conv.September.Value;
                }
                if (currentMonth == "October")
                {
                    nextobj.Transfers = conv.October.Value;
                }
                if (currentMonth == "November")
                {
                    nextobj.Transfers = conv.November.Value;
```

```
                }
                if (currentMonth == "December")
                {
                    nextobj.Transfers = conv.December.Value;
                }
                TopFivePerformanceObjectsWorst.Add(nextobj);
            }
            return TopFivePerformanceObjectsWorst;
        }



    }
}
```

SplashScreen.xaml.cs

```
namespace InventoryManagementALEVEL
{
    /// <summary>
    /// Interaction logic for SplashScreen.xaml
    /// </summary>
    public partial class SplashScreen : Window
    {

        DispatcherTimer timer;
        public SplashScreen()
        {
            InitializeComponent();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromSeconds(5);
            timer.Tick += timer_Tick;
            timer.Start();
        }

        private void WindowMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            DragMove();
        }

        void timer_Tick(object sender, EventArgs e)
        {
            //instantiate the login window
            Login lg = new Login();
            //show the login window
            lg.Show();
            //stop the timer for effeciency
            timer.Stop();
            this.Close();
        }
    }
}
```

SQLServerAccessor.cs

```
namespace InventoryManagementALEVEL
{
    class SQLServerAccessor
    {
        public InventoryManagementLINKQtoSQLClassDataContext DataContextInv
            = new
InventoryManagementLINKQtoSQLClassDataContext(Properties.Settings.Default.InventoryMan
agementDBConnectionString1);


        public bool deleteItemsInventory(int[] itemIDs, string username)
        {
            //deletes a list of items from inventory
            try
            {
                //loop through the list of itemIDs
                for (int x = 0; x < itemIDs.Length; x++)
                {
                    // This uses a lambda expression query to create a inventory table
object that corresponds to the current item id
                    InventoryTable deleteitem =
DataContextInv.InventoryTables.FirstOrDefault(e => e.ItemID.Equals(itemIDs[x]));
                    //delete the item
                    DataContextInv.InventoryTables.DeleteOnSubmit(deleteitem);
                    //generate a log message
                    string message = username + " has removed " + deleteitem.ItemName
+ " from inventory ";
                    AddLogEntry(username, message);
                    DataContextInv.SubmitChanges();
                }
                return true;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
                return false;
            }
        }

        public List<EventLogTable> GetLogEntriesAfterDate(DateTime ShowAferDate)
        {
            //A list of the event logs
            List<EventLogTable> eventLog = DataContextInv.EventLogTables.ToList();
            //A list of all the event logs that are after the given date , initially
it is empty
            List<EventLogTable> sortedLog = new List<EventLogTable>();
            //iterate through eventlog
            foreach (EventLogTable Log in eventLog)
            {
                Console.WriteLine(Log.TimeOfAction.Date);
                //if the date of the log is after the given date
                if (Log.TimeOfAction.Date > ShowAferDate.Date)
                {
                    //add it to the sorted log
                    sortedLog.Add(Log);
                }
            }
            return sortedLog;
        }
```

```
        public void AddLogEntry(string username, string actionMessage)
        {
            EventLogTable newlog = new EventLogTable();
            //replaces any white spaces in the message
            actionMessage =
System.Text.RegularExpressions.Regex.Replace(actionMessage, @"\s+", " ");
            //get the accounts table object for the user passed
            var GetUserData = DataContextInv.AccountsTables.Single(x => x.Username ==
username);
            //setup the new event log instance
            newlog.Username = GetUserData.Username;
            newlog.FirstName = GetUserData.FirstName;
            newlog.Role = GetUserData.Role;
            newlog.Action = actionMessage;
            newlog.TimeOfAction = DateTime.Now;
            DataContextInv.EventLogTables.InsertOnSubmit(newlog);
            DataContextInv.SubmitChanges();
        }

        public bool deleteItemsForSale(int[] itemIDs, string username)
        {
            try
            {
                for (int x = 0; x < itemIDs.Length; x++)
                {
                    ForSaleTable deleteitem =
DataContextInv.ForSaleTables.FirstOrDefault(e => e.ItemID.Equals(itemIDs[x]));
                    string message = username + " has removed " + deleteitem.ItemName
+ " from the for sale table ";
                    AddLogEntry(username, message);
                    DataContextInv.ForSaleTables.DeleteOnSubmit(deleteitem);
                    DataContextInv.SubmitChanges();


                }
                return true;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
                return false;
            }
        }
        public List<InventoryTable> getNewInventoryTable(string category, string name)
        {
            DataContextInv.Dispose();
            DataContextInv = new InventoryManagementLINKQtoSQLClassDataContext();
            if (name != "" && category == "Any")
            {
                //returns the items which are similar or exactly the same as the input
name but can come from any category
                return (from x in DataContextInv.InventoryTables
                        where x.ItemName.Contains(name)
                        select x).ToList();
            }
            else if (name != "" && category != "Any")
            {
                //returns the items which are similar or exactly the same as the input
name and are in the input category
                return (from x in DataContextInv.InventoryTables
                        where x.ItemName.Contains(name) &&
```

```
                        x.CategoryName.Equals(category)
                        select x).ToList();
            }
            else if (name == "" && category != "Any")
            {
                //returns items in the input category
                return (from x in DataContextInv.InventoryTables
                        where x.CategoryName.Equals(category)
                        select x).ToList();
            }

            else
            {
                //no filter , return everything
                return (from x in DataContextInv.InventoryTables
                        select x).ToList();
            }

        }

        public List<ForSaleTable> getNewForSaleTable(string category, string name)
        {
            DataContextInv.Dispose();
            DataContextInv = new InventoryManagementLINKQtoSQLClassDataContext();

            if (name != "" && category == "Any")
            {
                //returns the items which are similar or exactly the same as the input
name but can come from any category
                return (from x in DataContextInv.ForSaleTables
                        where x.ItemName.Contains(name)
                        select x).ToList();
            }
            else if (name != "" && category != "Any")
            {
                //returns the items which are similar or exactly the same as the input
name and are in the input category
                return (from x in DataContextInv.ForSaleTables
                        where x.ItemName.Contains(name) &&
                        x.CategoryName.Equals(category)
                        select x).ToList();
            }
            else if (name == "" && category != "Any")
            {
                //returns items in the input category
                return (from x in DataContextInv.ForSaleTables
                        where x.CategoryName.Equals(category)
                        select x).ToList();
            }
            else
            {
                //no filter , return everything
                return (from x in DataContextInv.ForSaleTables
                        select x).ToList();
            }

        }



        public List<InventoryTable> SearchForItemInventory(string name)
```

```
        {
            //returns a list of all items with the item name the same as the parameter
            return (from x in DataContextInv.InventoryTables
                    where x.ItemName.Equals(name)
                    select x).ToList();
        }

        public List<ForSaleTable> SearchForItemForSale(string name)
        {
            return (from x in DataContextInv.ForSaleTables
                    where x.ItemName.Contains(name)
                    select x).ToList();
        }

        public List<PerformanceTable> GetPerformanceItems()
        {
            return (from x in DataContextInv.PerformanceTables
                    select x).ToList();
        }
        public List<PerformanceTable> GetPerformanceItemsByName(string itemName)
        {
            return (from x in DataContextInv.PerformanceTables
                    where x.ItemName == itemName
                    select x).ToList();
        }
        public List<PerformanceTable> GetPerformanceItemsByLIKEName(string itemName)
        {
            return (from x in DataContextInv.PerformanceTables
                    where x.ItemName.Contains(itemName)
                    select x).ToList();
        }

        public List<InventoryTable> PopulateSelectedItemsWithSearchResults(int[]
itemIDs)
        {

            List<InventoryTable> nitems = new List<InventoryTable>();

            for (int x = 0; x < itemIDs.Length; x++)
            {
                nitems.Add(DataContextInv.InventoryTables.FirstOrDefault(e =>
e.ItemID.Equals(itemIDs[x])));

            }
            return nitems;


        }

        public bool AuthenticateLogin(string Username, string Password)
        {
            //the method returns true if the login was authenticated
            Console.WriteLine("authenticating login");
            //Query the accounts table for a username with the entered username and
the entered password
            var accQuery = from u in DataContextInv.AccountsTables
                           where u.Username.Equals(Username) &&
                           u.Password.Equals(Password)
                           select u;
            if (accQuery.Count() != 0)
                //if the query returns nothing , the username and password does not
exist together in a record
```

```
                //so the username or password was incorrect
            {
                Console.WriteLine("Authenticated");
                return true;
            }
            else
            {
                Console.WriteLine("Authentication failed");
                return false;
            }
        }

        public bool CreateNewAccount(string username, string firstname, string
lastname, string password, string role)
        {
            //search for an existing user with the same username
            var accQuery = from u in DataContextInv.AccountsTables
                           where u.Username.Equals(username)
                           select u;
            if (accQuery.Count() == 0)  //if the query returns more than 0 , the
username is taken
            {
                //create a new Accounts table instance and fill it with the passed
values
                AccountsTable newuser = new AccountsTable();
                newuser.Username = username;
                newuser.FirstName = firstname;
                newuser.LastName = lastname;
                newuser.Password = password;
                newuser.Role = role;
                DataContextInv.AccountsTables.InsertOnSubmit(newuser);
                DataContextInv.SubmitChanges();
                string message = "Created a new account for the user " + username;
                AddLogEntry(username, message);
                return true;
            }
            else //the username was taken so return false
            {
                return false;
            }

        }
        public IEnumerable<CategoryTable> getCategories()
        {
            return from s in DataContextInv.CategoryTables
                   select s;

        }
        public bool addNewItemInventory(string _itemName, int _quantity, string
Category, string username)
        {
            try
            {
                InventoryTable newItem = new InventoryTable();
                newItem.ItemName = _itemName;
                newItem.Quantity = _quantity;
                newItem.CategoryName = Category;
                DataContextInv.InventoryTables.InsertOnSubmit(newItem);
                DataContextInv.SubmitChanges();
                string message = username + " Has added " + _itemName + " to
inventory," + " quantity " + _quantity + " category " + Category;
                AddLogEntry(username, message);
```

```
                return true;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
                return false;
            }

        }
        public bool addNewItemForSale(string _itemName, int _quantity, string
Category, DateTime SellByDate, string username)
        {
            try
            {
                ForSaleTable newItem = new ForSaleTable();
                newItem.ItemName = _itemName;
                newItem.Quantity = _quantity;
                newItem.CategoryName = Category;
                newItem.SellByDate = SellByDate;
                DataContextInv.ForSaleTables.InsertOnSubmit(newItem);
                DataContextInv.SubmitChanges();
                string message = username + " Has added " + _itemName + " to the for
sale table," + " quantity " + _quantity + " category " + Category;
                AddLogEntry(username, message);
                return true;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
                return false;
            }

        }

        public void GeneratePerformanceLog(ForSaleTable ActItem)
        {

            var PerfQuery = from u in DataContextInv.PerformanceTables
                            where u.ItemName == ActItem.ItemName
                           select u;
            if (PerfQuery.Count() == 0) // then a perf record for the item doesnt
exist
            {
                PerformanceTable PerformanceLog = new PerformanceTable();
                PerformanceLog.ItemName = ActItem.ItemName;
                PerformanceLog.January = 0;
                PerformanceLog.February = 0;
                PerformanceLog.March = 0;
                PerformanceLog.April = 0;
                PerformanceLog.May = 0;
                PerformanceLog.June = 0;
                PerformanceLog.July = 0;
                PerformanceLog.August = 0;
                PerformanceLog.September = 0;
                PerformanceLog.October = 0;
                PerformanceLog.November = 0;
                PerformanceLog.December = 0;
                //returns the current month , e.g. January
                string currentMonth = DateTime.Now.ToString("MMMM");

                Console.WriteLine("The quantity is "  + ActItem.Quantity);
```

```
                if (currentMonth == "January")
                {
                    PerformanceLog.January += ActItem.Quantity.Value;
                }
                if (currentMonth == "February")
                {
                    PerformanceLog.February = ActItem.Quantity.Value;
                }
                if (currentMonth == "March")
                {
                    PerformanceLog.March = ActItem.Quantity.Value;
                }
                if (currentMonth == "April")
                {
                    PerformanceLog.April = ActItem.Quantity.Value;
                }
                if (currentMonth == "May")
                {
                    PerformanceLog.May = ActItem.Quantity.Value;
                }
                if (currentMonth == "June")
                {
                    PerformanceLog.June = ActItem.Quantity.Value;
                }
                if (currentMonth == "July ")
                {
                    PerformanceLog.July = ActItem.Quantity.Value;
                }
                if (currentMonth == "August")
                {
                    PerformanceLog.August = ActItem.Quantity.Value;
                }
                if (currentMonth == "September")
                {
                    PerformanceLog.September = ActItem.Quantity.Value;
                }
                if (currentMonth == "October")
                {
                    PerformanceLog.October = ActItem.Quantity.Value;
                }
                if (currentMonth == "November")
                {
                    PerformanceLog.November = ActItem.Quantity.Value;
                }
                if (currentMonth == "December")
                {
                    PerformanceLog.December = ActItem.Quantity.Value;
                }
                DataContextInv.PerformanceTables.InsertOnSubmit(PerformanceLog);
                DataContextInv.SubmitChanges();
            }
            else
            {
                //get the actual record in the performance table
                var PerfData = DataContextInv.PerformanceTables.Single(x => x.ItemName
== ActItem.ItemName);
                string currentMonth = DateTime.Now.ToString("MMMM");
                if (currentMonth == "January")
                {
                    //increment the value by the quantity of item
                    PerfData.January += ActItem.Quantity.Value;
                }
```

```
                if (currentMonth == "February")
                {
                    PerfData.February += ActItem.Quantity.Value;
                }
                if (currentMonth == "March")
                {
                    PerfData.March += ActItem.Quantity.Value;
                }
                if (currentMonth == "April")
                {
                    PerfData.April  += ActItem.Quantity.Value;
                }
                if (currentMonth == "May")
                {
                    PerfData.May += ActItem.Quantity.Value;
                }
                if (currentMonth == "June")
                {
                    PerfData.June += ActItem.Quantity.Value;
                }
                if (currentMonth == "July ")
                {
                    PerfData.July += ActItem.Quantity.Value;
                }
                if (currentMonth == "August")
                {
                    PerfData.August += ActItem.Quantity.Value;
                }
                if (currentMonth == "September")
                {
                    PerfData.September += ActItem.Quantity.Value;
                }
                if (currentMonth == "October")
                {
                    PerfData.October += ActItem.Quantity.Value;
                }
                if (currentMonth == "November")
                {
                    PerfData.November += ActItem.Quantity.Value;
                }
                if (currentMonth == "December")
                {
                    PerfData.December += ActItem.Quantity.Value;
                }
                DataContextInv.SubmitChanges();
            }

        }
        public void TransferItemsToForWITHsellByDate(InventoryTable itemToTransfer,
DateTime sellbydate, string username)
        {
            //get the transferred item from inventory first
            var oldItem = DataContextInv.InventoryTables.Single(x => x.ItemID ==
itemToTransfer.ItemID);

            Console.WriteLine("is " + itemToTransfer.Quantity + " < " +
oldItem.Quantity);
            if (itemToTransfer.Quantity < oldItem.Quantity)
            {
                //update the quantity of the item in inventory
                oldItem.Quantity = oldItem.Quantity - itemToTransfer.Quantity;
                ForSaleTable transitem = new ForSaleTable();
```

```
                transitem.ItemName = itemToTransfer.ItemName;
                transitem.Quantity = itemToTransfer.Quantity;
                transitem.SellByDate = sellbydate;
                transitem.CategoryName = itemToTransfer.CategoryName;
                DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
                DataContextInv.SubmitChanges();
                GeneratePerformanceLog(transitem);
                string message = username + " has made " + transitem.Quantity + " " +
transitem.ItemName + " available for sale and added the sell by date of " +
transitem.SellByDate;
                AddLogEntry(username, message);
            }
            else
            {
                ForSaleTable transitem = new ForSaleTable();
                transitem.ItemName = itemToTransfer.ItemName;
                transitem.Quantity = itemToTransfer.Quantity;
                transitem.SellByDate = sellbydate;
                transitem.CategoryName = itemToTransfer.CategoryName;
                DataContextInv.InventoryTables.DeleteOnSubmit(oldItem);
                DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
                DataContextInv.SubmitChanges();
                GeneratePerformanceLog(transitem);
                string message = username + " has made " + transitem.ItemName + "
available for sale and added the sell by date of " + transitem.SellByDate;
                AddLogEntry(username, message);
            }
        }
        public void TransferItemsToForSaleNoSellBy(List<InventoryTable>
ListOfItemsToTransfer, string username)
        {
            foreach (InventoryTable nextItem in ListOfItemsToTransfer)
            {
                var oldItem = DataContextInv.InventoryTables.Single(x => x.ItemID ==
nextItem.ItemID);
                Console.WriteLine("is " + nextItem.Quantity + " < " +
oldItem.Quantity);
                if (nextItem.Quantity < oldItem.Quantity)
                {
                    oldItem.Quantity = oldItem.Quantity - nextItem.Quantity;
                    ForSaleTable transitem = new ForSaleTable();
                    transitem.ItemName = nextItem.ItemName;
                    transitem.Quantity = nextItem.Quantity;
                    DateTime? newdate = null;
                    transitem.SellByDate = newdate;
                    transitem.CategoryName = nextItem.CategoryName;
                    DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
                    DataContextInv.SubmitChanges();
                    GeneratePerformanceLog(transitem);
                    string message = username + " has made " + transitem.Quantity + "
" + transitem.ItemName + " available for sale";
                    AddLogEntry(username, message);
                }
                else
                {
                    ForSaleTable transitem = new ForSaleTable();
                    transitem.ItemName = nextItem.ItemName;
                    transitem.Quantity = nextItem.Quantity;
                    DateTime? newdate = null;
                    transitem.SellByDate = newdate;
                    transitem.CategoryName = nextItem.CategoryName;
                    DataContextInv.InventoryTables.DeleteOnSubmit(oldItem);
```

```
                    DataContextInv.ForSaleTables.InsertOnSubmit(transitem);
                    DataContextInv.SubmitChanges();
                    GeneratePerformanceLog(transitem);
                    string message = username + " has made " + transitem.ItemName + "
available for sale";
                    AddLogEntry(username, message);
                }
            }

        }

        public string GetAccountType(string username)
        {
            //returns the user group of the passed username
            var UserData = DataContextInv.AccountsTables.Single(x => x.Username ==
username);
            //remove any whitespace in the returned query
            return UserData.Role.Replace(" ","");
        }

    }
}
```

# References

Anonymous, 2016. *Acctivate Inventory Management Software.* [Online]
Available at: http://www.softwareadvice.com/uk/inventory-management/acctivate-profile/
[Accessed 21 September 2016].

Anonymous, 2016. *inFlow Inventory Software.* [Online]
Available at: http://www.softwareadvice.com/uk/inventory-management/inflow-inventory-profile/
[Accessed 21 September 2016].

Azure, M., 2016. *SQL Database.* [Online]
Available at: https://azure.microsoft.com/en-gb/services/sql-database/
[Accessed 4 January 2017].

ERS, 2016. *ERS.* [Online]
Available at: https://www.ers-online.co.uk/o5977/ers-stockcount-kit1-ers-stockcount-barcode-stock-taking-solution
[Accessed 27 November 2016].

Fletcher, M., 2016. *Introduction to Network Programming.* [Online]
Available at: http://www.webbasedprogramming.com/Java-Unleashed-Second-Edition/f23-1.gif
[Accessed 21 September 2016].

Microsoft, 2015. *Windows 10.* [Online]
Available at: https://www.microsoft.com/en-gb/windows/windows-10-specifications
[Accessed 2 November 2016].

Microsoft, 2016. *Supported Query Types (Visual Database Tools).* [Online]
Available at: https://technet.microsoft.com/en-us/library/ms188316(v=sql.110).aspx
[Accessed 12 December 2016].

Microsoft, n.d. *Office Word,* s.l.: s.n.

Templates, A., 2017. *Access Inventory Management Control Templates,* s.l.: s.n.

# Table of figures

Figure 1 ..... 11
Figure 2 ..... 12
Figure 3 ..... 37
Figure 4 ..... 37
Figure 5 ..... 38
Figure 6 ..... 39
Figure 7 ..... 39
Figure 8 ..... 40
Figure 9 ..... 41
Figure 10 ..... 42
Figure 11 ..... 43
Figure 12 ..... 44