

Candidate Number: 4187

Centre Number: 14739

Adam Khanzada Computer Science Coursework
Charity Shop Inventory Manager

Contents

Title Page- Page- 1

Analysis- Page- 3

Documented Design- 12

Technical Solution- 32

Testing- 63

Evaluation-82

Bibliography- 91

Analysis

Background to Problem

A charity shop must deal with thousands of items a day and must constantly monitor inventory to make sure stock is correctly accounted for. Often Items are misplaced or not accounted for due to the large scale of items hauled in by trucks daily. This makes it very difficult to keep track of inventory. Countless situations the staff find themselves spending precious time searching for items on behalf of customers only to realise long after that the specified item had been thrown away or sold weeks ago. This problem forces staff to spend countless hours a week searching for items rather than fulfilling more meaningful tasks such as tending to the till. This leads to great inefficiencies on the shop floor and puts business at a halt as staff are busy sourcing out inventory.

Furthermore, the inventory management problem extends itself to the back rooms of the shop as every morning Inventory is cycled through the shelves to make sure that newer stock is prioritized over older items. This job is made tedious and difficult due to the lack of an outdated tagging system used. Items are given sticker tags that specify general item information such as ItemID, Price, Type Code.....etc. After an Item has been tagged it is then written into the Inventory diary and prepared to be taken to the shop floor. This causes several problems as stickers often time fall off items resulting in Items without prices which increases the time taken for transactions as customers must enquire about prices which means that staff must refer to a very large diary to retrieve correct item info. This issue is both an inconvenience to the staff and customers.

The Inventory management problem causes a chain of problems to form which influence the shop negatively as general employee efficiency drops drastically and sale time (The time taken from when the customer reaches the counter to when he/she leaves the store) massively increases. This causes congestion in the store resulting in the overarching problem of less sales due to unsatisfied customers.

Who Am I solving the Problem For

I am solving this problem for the manager and general staff of the British Heart Foundation charity branch in Sutton. As a previous employee of the shop I have experienced this problem myself and have been in contact with the Manager discussing methods for reducing employee role inefficiencies and general sale times. The solution is aimed to assist this local branch of the British Heart Foundation as the stores tagging system is severely outdated thus holding the store back from its true sales potential.

Other Prospective Users

The solution to the problem has been intended for a single charity shop however the Inventory Management solution can be applied to any charity shop suffering from a poor stock management system. The solution can be replicated for other branches of the British Heart Foundation as well as being used for other Charity shops such as The Cancer Research Foundation as most of charity shops are known to suffer from sub-par stock management leading to sale time inefficiencies.

Systems Currently in Place

The current system in place is a very outdated written system that uses sticker tags to keep track of item information combined with a general item diary. This method is often confusing and tedious as tags are mostly lost and the inventory diary is too inefficient to look up item information and update stocks at the same time as changes require outdated info to be crossed out in pen.

Other Inventory management systems have been used to keep track of stocks in large scale purveyors or grocery stores; however, no current systems are in place to specialize inventory management for Charity shops.


Feedback from End Users

I have been in contact with the manager of the shop and an employee in order to get a better idea of the core problems within the current systems and understand any potential needs the users would want implemented to effectively tackle the problem.



From my discussion with the Manager it was clear that the current inventory diary was not a viable option in the long term and that this would need to be replaced by a better computer-based system that can be accessed by multiple users at a time as a single diary can only be used by one member of staff at a time. This problem can be solved by having staff being given individual user ID's to access the stores item inventory (Login system). The Manager also expressed that keeping track of employee information would be a useful feature as it is hard for the Manager to know who is still in employment due to the large numbers of volunteers working at a time; however, this information should only be accessible to senior member of staff.

From my discussion with a volunteer employee He has stressed the need for a friendly user interface as the switch from a written diary to a computer-based system can be complicated for elderly staff members to work with. Furthermore, she has placed emphasis on the need to compartmentalize items into separate categories based on type such as clothing or DVD's. This was to make it easier to place items in common together on the shop floor.

Evidence of User Needs

**Adam khazada** <khandaza1@gmail.com>
to bhfsuttonbeatr. ▾


Apr 14 ☆





Hi Beatrice

I am emailing you regarding the Inventory management project in order to improve upon the way stocks and prices are managed. I wanted to know if there were any specific user requirements as a Manager you had for the application which would best improve the efficiency of item management. Thank you.

Yours Sincerely

**Beatrice Adams**
to me ▾


Apr 14 ☆





Thank you for your email Adam. I am looking forward to implementing your proposed solution to the Inventory management problem at the store. I was wondering if it would be possible for you to implement a system in which multiple users could access the Item stock info through different accounts. This would help dearly as employees will no longer be crowding around one diary and can access information from their own work stations. Also could you add an employee management system if possible so that I may keep track of the full-time employees and volunteers efficiently since we are removing the old written system.

Yours Sincerely

...

**Adam khazada** <khandaza1@gmail.com>
to Beatrice ▾

Apr 14 ☆



Thank you, your feedback is invaluable. I will try to implement your needs into my solution.

The conversation above between the Manager of the shop and myself clearly expressed the end user needs for a Login system so that different people could access the information at the same time and that a general employee manager was needed. The employee information would have to be only accessible by senior staff.

Inventory Management Project BHF



Inbox x



Adam khazada <khandaza1@gmail.com>

Apr 14 ☆



to bhfsuttonjames ▾

Hi James

I am emailing you regarding the Inventory management project in order to improve upon the way stocks and prices are managed. I wanted to know if there were any specific user requirements as a Manager you had for the application which would best improve the efficiency of item management. Thank you.

Yours Sincerely



James McGreevey

Apr 14 ☆



to me ▾

Thanks Adam

In terms of features I would like to have implemented I think it will be useful to have:

- Item Category selection so that its easier to sort items into groups.
- Easy to use for computer novices.
- Tag printer if possible because its a bit long having to write out item info.

Yours Sincerely



Adam khazada <khandaza1@gmail.com>

Apr 14 ☆



to James ▾

Thanks James, I will make sure to take note of your suggested features

The following conversation between myself and a local volunteer at the shop shows the user needs for item compartmentalization as well as ease of use. If possible, the end user also requested for a tag printer option so that information stored on the computer could automatically print Item tags based on general information held.

Evidence of Research

I have conducted research into several already used inventory management systems that are used by many companies today such as Cin7 or Xero inventory software. These inventory managers are very advanced and complicated systems that cater for large scale(global) retail franchises and big companies. These apps use numerous functions and protocols to store data effectively and efficiently. Many of this software can manipulate the data to assist market analysts for companies in finding trends or performing economical calculations. These software are very expensive to use due to the large processing capabilities; however for a charity shop that thrives of donations and volunteer support such software is not a viable option. Furthermore, the systems are overly complicated for the needs of my users as none of these software have been tailored to the charity shop industry.

When I was researching the different coding skills and software, I encountered a language designed for managing databases known as SQL and Microsoft Access which is used to store the data. SQL stands for 'Structured Query Language' which is a database language which allows me to make changes and edit databases which will make it easy for me to store information and edit information in a database. If I do choose to use SQL I will need to install SQL Server 2016 and SQL Server Management Studio. Since I have no experience of SQL, it means I will need to research how to make queries communicate with the databases within my program. SQL is a very robust language and I will most likely use this to create my databases for my project. This led me to research into how I can communicate with SQL databases using VB.NET Forms. I found the VB Toolbox YouTube Channel which has introduced me into how to connect SQL Databases to VB.NET.

My further research due to an end user request on a recommended charity shop price calculator lead me to looking for ways of how to access data on pricing for numerous goods. The online prices could be used as a starting point to calculate fair and suitable charity shop prices. The best method of approaching this would be through an Application programming interface(API). An API is a set of functions and procedures that allow the creation of applications which can access the features or data from online services. I then investigated API's that would suit my pricing needs. I concluded that I would need to use the pricing information of eBay due to their massive number of items and stock globally. Before I can think about implementing the API I will have to learn the language JSON which is based around using classes and objects to store data.

Acceptable Limitations

Throughout the project I will have many limitations to the extent of depth my program can go into due to numerous reasons. The factor that will be playing the biggest influence on the comprehensiveness of my solution is time. Due to time constraints my solution will have to focus solely on user needs and absolute requirements. This means that I will be unable to expand this software to the extent to which it can be used by any chain of charity shops as I will only have time to tailor the software to the needs of this specific shop.

Another general limitation I cannot control is the use of hardware that I have at my disposal and the hardware that the charity shop owns. The ability of the shop computer to process changes in stock quickly is imperative to the success of this solution as the primary goal is to reduce time wasted by staff checking for items in stock. Dependent on the memory attributes of the hardware being used, the runtime of the software could severely be affected. So long as the computers specifications are relatively up to date this will not bode as a serious problem.

Finally, I will be using Microsoft Access to hold all the database tables, this is not a free software and therefore can amount to problems as the charity shop may not have the funding to support this software. All the data is to be stored locally and the program will not know where to look for the data without Microsoft Access rendering the program potentially useless without this installed

Proposed Solution

The solution I will be developing to solve this problem will be using a database of tables linked allowing for employees of the British Heart Foundation of Sutton to easily search and store Items from the shop inventory into an efficient computerized medium. The program will split items into essential groups based upon type. This will make it easier to search for items in stock and look up key information on items as the users will be able to search up items by name and gain relevant and necessary info on them. This solution will allow for staff to manage and keep track of inventory on a day to day basis as staff will be able to add, remove and update item stock using unique logins. The solution will also provide a basic employee management system to the managers request which will allow her to make changes to the employment structure and keep track of her volunteers.

Objectives

Objective 1: Be able to add items to the database and display added items

1.1: Adding items should provide useful information to the staff such as pricing or stock quantity

1.2: Items that are added should be compartmentalized into specific tables relating to the stores most commonly donated and sold item types. E.g.: Clothing, DVD's and Toys

1.3: Dependent on the Item type the database should store Item type specific information. For example, a DVD will require information on age restrictions and print quality whereas a piece of clothing requires size measurements.

Objective 2: Be able to view items in the database and update necessary changes in stock if need be.

2.1: All Information about the selected item should be able to be viewed and specific details on the item can be changed such as the item name or condition.

2.2: A refresh button will allow all changes to the Item information to be viewed in the data grids.

Objective 3: Be able to delete items from the database as stock runs out over time. The stock quantity for each item should be updateable and a quantity of zero for when the item runs out should remove the item from the database.

3.1: Before the Item is removed necessary information to do with the removal should be seen by the viewer such as the current stock quantity and the items name.

Objective 4: The software should have a login screen for the sake of security purposes as precious shop information should not be openly accessed by the public

4.1: All employees will have a Username and Password to login. The passwords will be encrypted using a hashing algorithm to add an extra layer of security

4.2: Different employees will have access to admin privileges based upon your ranking position in the store and the managers approval as admin users can appoint new administrators. The manager will be an admin by default.

Objective 5: The manager has requested that an employee management system is added to the program to make it easier keeping track of workers due to the number of volunteer staff

5.1: The Employee management can only be accessed and used by senior members of staff that have admin privileges. This is because information on employees is private and should only be seen by the manager and select co-workers of the managers choosing.

5.2: The employee management table should store necessary information about employees such as Name, hours per week and medical information.

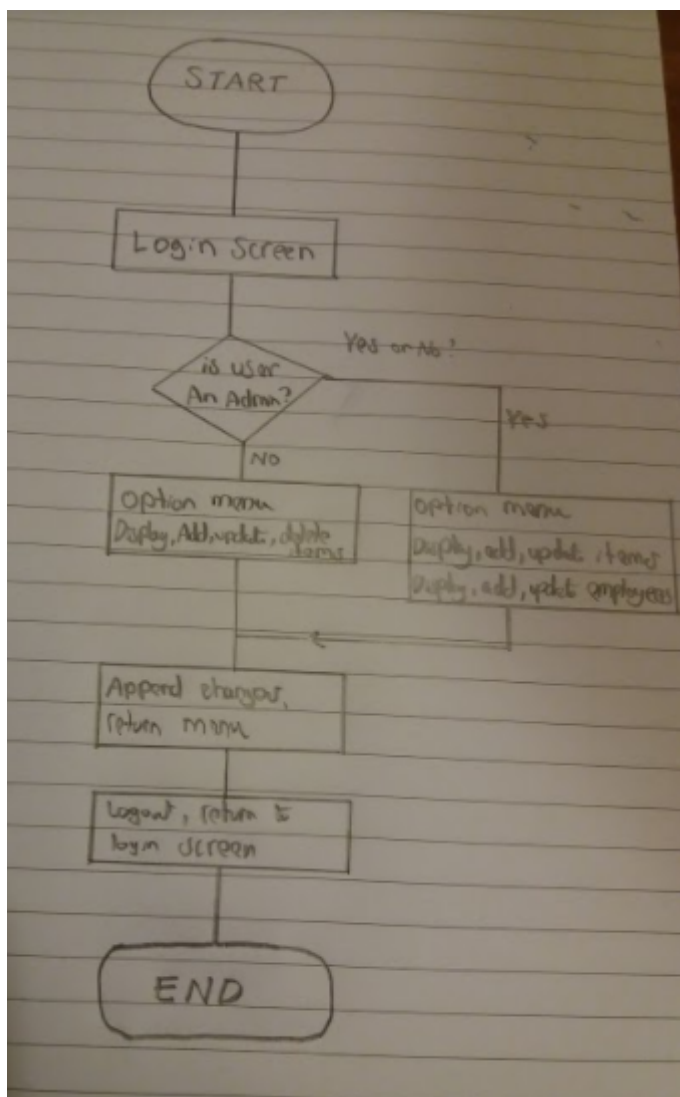
5.3: Admin staff should be able to add, deleting and updating staff information.

5.4: A refresh button in the employee management menu can be used to view changes to the employee information within the data grid view.

Objective 6: A volunteer has requested that the software should be able to take necessary information regarding an item and print a tag in order reduce time taken to manually write out tagging information. The tags should provide the Item name, ID and price.

Objective 7: All Items and employees can be searched for by name in their corresponding menus within the tables for general Items and employee management

Solution Modelling

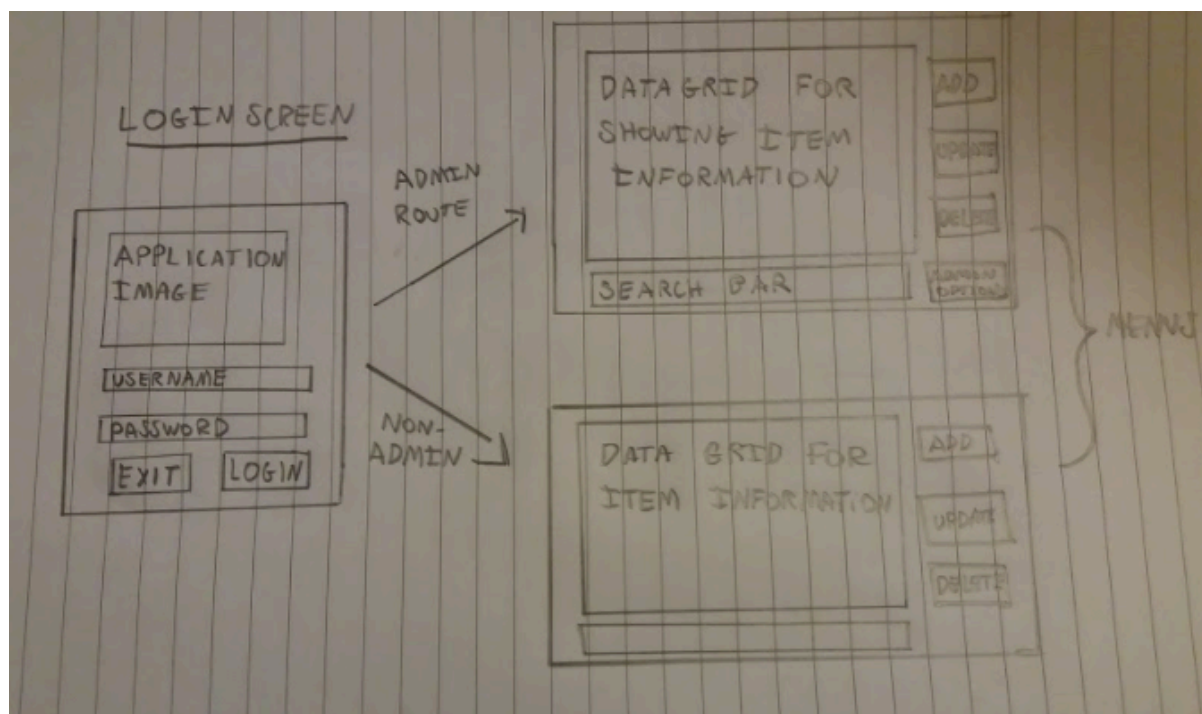


This flow chart represents the most basic form of my intended solution to the problem. The program will have a login screen for staff to offer a basic level of security.

The program will then use your login credentials to decide whether you are an admin or not. Based upon this information you will be taken to 2 slightly different menus.

The general employee menu will allow you to access information on items and prices and make changes to items; however, an admin login will give you access to employee management options.

After you are done checking or making changes to inventory based upon stock or employees, you logout of your account and end the application. The logout is so that random users cannot access the information whilst the software is not in use

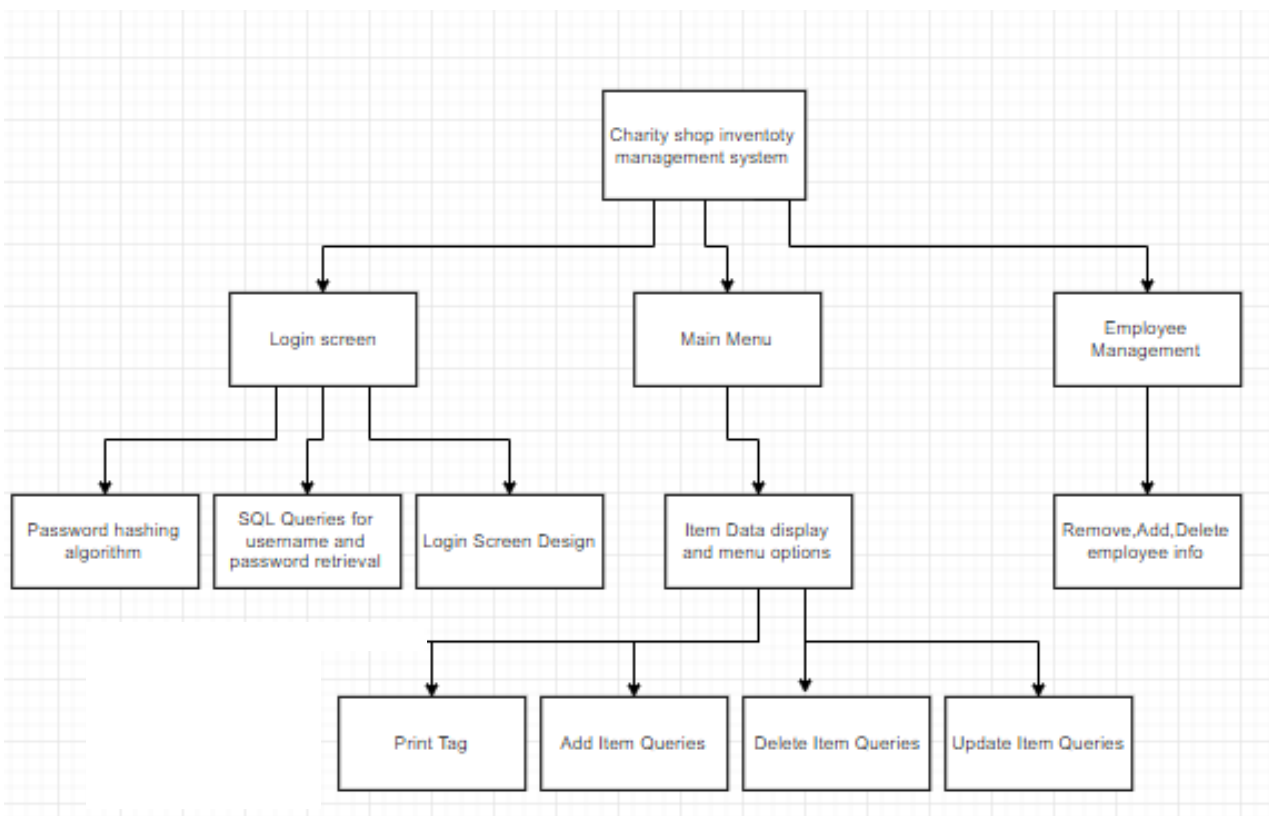


The diagram above depicts a basic idea for some of the key forms I will be creating for my solution. There will be a login screen which will lead to the main menu which differs dependent on user authorities

Documented Design

In this section of my project I will be giving an overview into the stages of technical and interface design and how the different parts of my system will work collaboratively in order to increase efficiency in Charity shops; namely the British Heart Foundation.

I have created a structured hierarchy chart in order to convey the different procedures that need to take place for my solution to the problem at hand to be developed in an efficient and methodical way. The chart below will accurately display the core procedures the solution can be narrowed down into and then break down these procedures into sub-tasks. This will allow me to manage the problem with greater perspective of the underlying tasks that need to accomplish for other parts of the process to move onwards. This system of tackling the problem will allow me to focus on one aspect of the solution at a time which makes will make it easier addressing problems at the time and applying error handling after each stage is complete.



The diagram above offers a brief outline of how I have chosen to break down the tasks to efficiently create a solution to the problem.

Login

I have chosen to first focus on the login process for my end users as this is a necessary first step in order to add a layer of security to the store information. This is also the first process a user will have to use to move forward so it seems appropriate to focus on this sector of the underlying solution to our problem.

Login: The Design

A key requirement expressed by the manager was that the program offers a user-friendly experience. This makes the design of the login screen of great importance as you want to maintain a simple design that only depicts necessary information such as Username, Password textboxes and a Login button. Moreover, the login screen needs to have an image or title reminding the user of the purpose thus reassuring the staff of a user-friendly experience.

Login: The SQL and Database Queries

The information used to access an account in order to login will be a Username and Password. This information will be stored in an Employee Management table within my database and the field Username will be used to identify a unique user alongside his/her password. The information in the table will be drawn from the table using SELECT queries and then compare the inputted text in the Login screen to the info collected from the table. This step will work alongside the hashing procedure in the Login screen.

Login: Hashing Passwords

The use of Hashing passwords is to add a layer of security to the data accessible. The passwords that are stored in the database are already encrypted via the hash. This means that the user input for the password will have to take the password entered and apply the hash to it. The hashed password is then compared to the one in the database table and if they match up with the appropriate username the user will be granted access to the menu. The hashing algorithm used will use a single key to decrypt and encrypt passwords set by the Manager.

Pseudocode for Password Hashing:

Function: Hash

- concatenate password and key

- convert result into binary format

- compute hash using inbuilt function and converted string as input

- convert result into string

- Return Result (encrypted string)

Main Menu

The menu will be accessible after the login process has been successfully completed. The menu will open up with several options and information regarding current shop stock. The menu will have a data grid view which will retrieve data from the database and allow the user to view relevant data based upon searches or Item type. To display data in the data grid view I am going to need to use SELECT statements to call upon relevant data in the database

Main Menu: Adding Items

A button should take you to a form which will allow you to add the new Items information such as Name, price and quantity as well as type specific information to do with the item type. For example, a piece of clothing will have information on size, gender and brand. When adding Items to the database we are going to have to add the information to 2 different tables, one for the general information and the other for the type specific info. This means we will have to use the Item ID to create a 1 to 1 link between tables. When adding the new Item, we will use two INSERT statements in which we will fill out a majority of the info and some data will be generated by itself such as ID.

Main Menu: Deleting Items

A button should take you to a form which will allow the user to delete/remove stock from the shop inventory. The item highlighted in the menu will be taken forward to the deleting form in which the Items current quantity will be displayed and the user will be allowed to make changes to the quantity. The deleting of items will work in two ways, using a DELETE query and using an UPDATE query. When changing the quantity to a value that is not zero an update statement will be executes however; when quantity is changes to 0 the Item will be removed from the database entirely. This means that the item will have to be removed from the general table and its type specific table. This will be done using the ID to reference the item across tables.

Main Menu: Updating Item Information

When an Item has been selected in the menu the user can chose to use the Update button which will take you to a form showing all information to do with the item in several textboxes. The information in the textboxes can be changed and upon clicking the update button all the changes will be applied to the item in the database. The changes are applied to the general table and the type specific table.

Print Tag:

When an item in the menu has been selected you will have the option to create a tag for the item, the tag will consist of the items Name, Price and ID. The tag will be formatted to a word document where it is able to be printed. Based on the number of tags you want to print the program should add that many tags to the word document. All tags are saved to a file location of the user's choice so that they can be re-used in the future.

Employee Management:

Based on whether the user logged in is an admin they will have the option to enter a section for employee management. The employee management form should act as a menu in which you can make changes to employee information. The user will have the option to add, remove and update employee info. Furthermore, the user should be able to view and search for certain employees and their credentials. This will work using a SELECT query.

Add Employee:

When the user selects the option to add new employee, they will be taken to a form which will allow them to enter new information and select whether this user is granted admin privileges. The new employee will be added to the database by using an INSERT query.

Remove Employee:

An employee that is selected can be deleted upon clicking the delete button. The employee's information will appear, and you will be able to remove his information from the database using a DELETE query. Error handling will prevent the number of admin employees from dropping below 1 and you will bot be able to delete yourself when logged in.

Update Employee Information:

An employee can be selected, and you will have the option to make changes to the employee's details. All the information will be placed in editable textboxes. Upon clicking the update button all changes to the employee information will be applied to the database table for employees using an UPDATE query.

IPSO table

Input	Process	Storage	Output
-------	---------	---------	--------

The user inputs a username and password to log into the main program	Username and password checked to see if they match with the ones held in database table for employees. The hash must match as well	Details are stored in the EmployeeInfo table within the database	The main form will appear with different option based upon whether the user is an admin. If login failed error message appears
User Adds a new Item to the database	User is prompted to enter Item type and information and then presses button to add item using INSERT query	The information is inserted into the ItemsGeneral table and the type table	The user is notified that the Item has been added to the stock.
User removes/deletes Items from the database	User selects an Item and is able to change the current quantity held. Use of UPDATE and DELETE queries	The new quantity is updated in the ItemsGeneral table	The user is notified that the item selected has been removed completely or just reduced in stock.
User choses to update or change an Items information	User selects an item and can alter its information and the submit the changes using an UPDATE query	The changes are stored in the table ItemsGeneral and the type specific table	The user is notified that the changes made to the item have successfully been made.
User can choose to search for an item using the search bar	User types the item name, the item is searched for and all information on Item brought up using a SELECT query	The Item found appears in a data grid view, the actual Item is stored in the database table for ItemsGeneral.	User is shown information on the Item searched for.
Admin user can Add new employees to the staff manager	The user can create new users by entering information for required fields. This is then added using an INSERT query	New employees are stored in the table EmployeeInfo within the database	User is notified that the new employee and information has been added to the database
Admin user can delete employees from the staff manager	User can select an employee and delete him from the system using a DELETE query	Employee selected is removed from the table EmployeeInfo	User is notified that the employee selected was deleted from the database
Admin user can make changes to	User selects an employee and can edit the employee	The changes made are applied to the	User is notified that the information

employee information	information using an UPDATE query	table EmployeeInfo within the database	changes were successful
All users can create a tag for an Item and save and print out to a word document	From the menu an item can be selected, and the user can save and specify how many tags he wants printed. Tag info is called upon using a SELECT query	The information used to create the tag comes from the table ItemsGeneral within the database	User receives a printout of the tag for the item selected and the tag is saved a word document in user defined location.

Database Design

In this section of my documented design I will be running through the process of creating my tables, table relationships and process of normalization.

Table: ItemsGeneral

The first table I must think about creating will be the main table which stores all of the shops items and holds general information on the item. The information that all items in the shop will require.

- Primary keys have been underlined in **bold**
- Foreign keys have been highlighted in **green**

Table: ItemsGeneral	
Primary key	<u>ItemGeneralID</u>
Fields	ItemType ItemName ItemPrice ItemQuantity ItemCondition

Table: ClothingStocks

The purpose of this table is to store the specific information on items in the store that fall under the clothing type. This information consists of clothing specific details.

Table: ClothingStocks	
Primary key	ClothingStockID
Fields	ItemGeneralID ClothingType ClothingSize ClothingBrand ClothingGender

Table: ToyStocks

The purpose of this table is to store the specific information on items in the store that fall under the Toy type. This information consists of toy specific details.

Table: ToyStocks	
Primary key	ToyStockID
Fields	ItemGeneralID ToyMake ToyAgeRecommendation ToyType ToySafety

Table: DVDorCDStocks

The purpose of this table is to store the specific information on items in the store that fall under the DVDorCD type. This information consists of disc specific details.

Table: DVDorCDStocks	
Primary key	DiscID

Fields	ItemGeneralID
	DiscCondition
	DiscPrint
	GenreorCategroy
	AgeRating

Table: EmployeeInfo

The purpose of this table is to store information on the employees in the shop so that the manager can effectively keep track of staff. The information is also necessary for logging into the program.

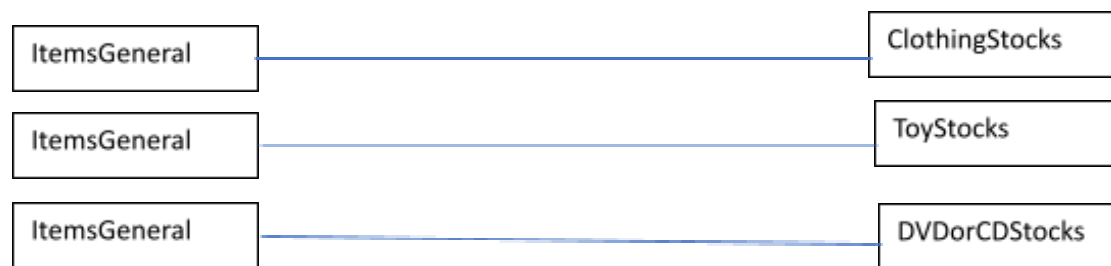
Table: EmployeeInfo	
Primary key	Username
Fields	ID FirstName LastName HoursperWeek MedicalInfo Employment Password Admin

Normalisation process

When planning out my tables and how they would relate to each other the original concept used a single table to store information on Items inventory. This table was the ItemsGeneral

table. The problem with only having one table to store items meant that information on item types would have to be held in the same table. This meant that many fields were left empty as fields which would apply to a certain type of item such as clothing could not support data for toys. This meant that lots of data points would be left unfilled which is a waste of data. This made the process of Normalisation and planning a integral part of my database design.

Each item held in ItemsGeneral will have general information stored about and additional information based upon the type. This forms a 1 to 1 relationship between the Item and its type information. The way we would best establish this connection would be using the ItemGeneralID.



The diagram above expresses the 1 to 1 connection between an Item is the ItemsGeneral table and how it will have one corresponding set of information in its type specific table.

Before the process of Normalisation all Items and information would be in a single table like the following table field diagram:

Table: ItemsGeneral	
Primary key	ItemGeneralID
Fields	ItemType ItemName ItemPrice ItemQuantity ItemCondition ClothingType ClothingSize ClothingBrand ClothingGender ToyMake ToyAgeRecommendation ToyType

	ToySafety DiscCondition DiscPrint GenreorCategroy AgeRating
--	---

This would be very inefficient in terms of data storage management as lots of fields would be empty. This lead to the process of distributing type specific information into different tables.

The new database design looked like this, for the general table and one branch of Item type, in this example clothing.

Table: ItemsGeneral	
Primary key	ItemGeneralID
Fields	ItemType ItemName ItemPrice ItemQuantity ItemCondition

Table: ClothingStocks	
Primary key	ClothingStockID

Fields	ItemGeneralID ItemName ItemPrice ClothingType ClothingSize ClothingBrand ClothingGender
--------	---

As we can see this was also inefficient and diminished the integrity of the data as fields were repeated across tables such as price and name. Information on items was repeated continuously which was a waste of space. This led to our conversion to third normal form to make our database have maximum storage efficiency. This made it necessary to establish relationships between tables using foreign keys.

Pre-Normalisation (1st Normal Form)

Tables

ItemsGeneral: (ItemGeneralID, ItemType, ItemName, ItemPrice, ItemQuantity, ItemCondition, ClothingType, ClothingSize, ClothingBrand, ClothingGender, ToyMake, ToyAgeRecommendation, ToyType, ToySafety, DiscCondition, DiscPrint, GenreorCategory, AgeRating)

EmployeeInfo: (Username, ID, FirstName, LastName, HoursperWeek, MedicalInfo, Employment, Password, Admin)

Post Normalisation (3rd Normal Form)

Tables

ItemsGeneral: (ItemGeneralID, ItemType, ItemName, ItemPrice, ItemQuantity, ItemCondition)

ClothingStocks: (ClothingStockID, ItemGeneralID, ClothingType, ClothingBrand, ClothingSize, ClothingGender)

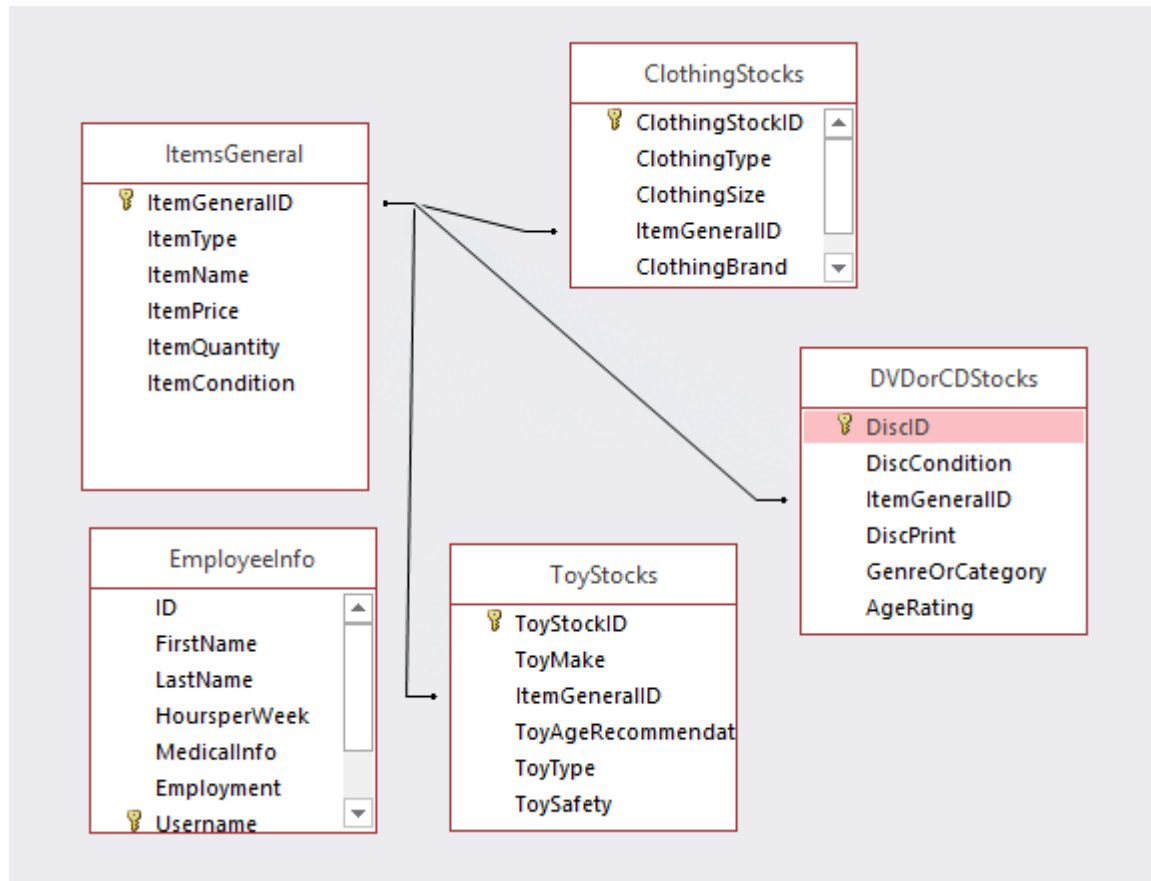
ToyStocks: (ToyStockID, ItemGeneralID, ToyMake, ToyType, ToyAgeRecommendation, ToySafety)

DVDorCDStocks: (DiscID, ItemGeneralID, DiscCondition, DiscPrint, GenreorCategory, AgeRating)

EmployeeInfo: (Username, ID, FirstName, LastName, HoursperWeek, MedicalInfo, Employment, Password, Admin)

Entity Relationship Diagram

The following diagram describes a planned design for how my tables will be linked to each other. All the connections will be in one to one relationships as one item will have one set of type specific information.



The foreign key across the tables that involve stock is the ItemGeneralID which is the primary key of the ItemsGeneral table.

SQL queries

The following will show all queries I plan to use in my solution

SELECT Queries

The following select queries are most likely going to be used to reference rows of data in the database for when I want to display information in data grid view for both items and employee information.

```
"SELECT Password, Admin, ID FROM EmployeeInfo WHERE Username = @username "
```

```
"SELECT Username, Password, Admin FROM EmployeeInfo WHERE Username LIKE @user AND Password LIKE @pass "
```

```
"SELECT * FROM ItemsGeneral ORDER BY ItemName ASC"
```

```
"SELECT * FROM ClothingStocks WHERE ItemGeneralID = " ItemGeneralID
```

```
"SELECT * FROM ToyStocks WHERE ItemGeneralID = " ItemGeneralID
```

```
"SELECT * FROM DVDorCDStocks WHERE ItemGeneralID = " ItemGeneralID
```

```
"SELECT * FROM EmployeeInfo ORDER BY ID ASC"
```

INSERT Queries

The insert queries will be used when I want to add new Items or employees to the database. When inserting Items, I will have to use two queries, one for adding to ItemsGeneral table and the other to add information to the type specific table

Items

```
"INSERT INTO ItemsGeneral  
([ItemName],[ItemPrice],[ItemCondition],[ItemQuantity],[ItemType])  
Values(@name,@price,@condition,@quantity,@typeGen)"
```

```
"INSERT INTO ClothingStocks  
([ClothingType],[ClothingSize],[ItemGeneralID],[ClothingBrand],[ClothingGender])  
Values(?,?,?,?,?)"
```

```
"INSERT INTO ToyStocks  
([ItemGeneralID],[ToyType],[ToyMake],[ToyAgeRecommendation],[ToySafety])  
Values(?,?,?,?,?)"
```

```
"SELECT ItemGeneralID FROM ItemsGeneral WHERE ItemName = ItemName """
```

Employee

```
"INSERT INTO EmployeeInfo  
([FirstName],[LastName],[HoursperWeek],[MedicalInfo],[Employment],[Username],[Password],  
[Admin]) Values(@first,@last,@hours,@medical,@Employment,@user,@password,@admin)"
```


DELETE Queries

I will only need to use delete queries seldomly as there are only two scenarios when this will occur. Scenario one will be when I am deleting an Item I will have to use two queries, one for removing the Item from the ItemsGeneral table and the other for removing the data from the type specific table. The other scenario is when an employee leaves the staff team, his information is no longer required so it is deleted.

Items

```
"DELETE FROM [ItemsGeneral] WHERE ItemName = ItemName  ""
```

```
"DELETE FROM [" & ItemType & "Stocks] WHERE ItemGeneralID = " & ItemID
```

Employee

```
"DELETE * FROM [EmployeeInfo] WHERE Username = '' EmployeeUsername  ""
```

UPDATE Queries

I will be using many update queries as they are needed in several areas of my solution. They will predominantly be used for making amends to Item information as well as employee data. On top of this an Update statement will be used in the Item removal section as changes in Item quantity which can be considered as deleting items works via updating the quantity value.

Items

```
"UPDATE [ItemsGeneral] SET  
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ  
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID"
```

```
"UPDATE [ClothingStocks] SET  
[ClothingType]=@ClothingType,[ClothingSize]=@ClothingSize,[ClothingBrand]=@ClothingBra  
nd,[ClothingGender]=@ClothingGender WHERE ItemGeneralID=@ItemGeneralID"
```

```
"UPDATE [ItemsGeneral] SET  
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ  
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID"
```

```
"UPDATE [ToyStocks] SET  
[ToyMake]=@ToyMake,[ToyAgeRecommendation]=@ToyAgeRecommendation,[ToyType]=@ToyType,[To  
ySafety]=@ToySafety WHERE ItemGeneralID=@ItemGeneralID"
```

```
"UPDATE [ItemsGeneral] SET  
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ  
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID"
```

```
"UPDATE [DVDorCDStocks] SET  
[DiscCondition]=@DiscCondition,[DiscPrint]=@DiscPrint,[GenreorCategory]=@GenreorCatego  
ry,[AgeRating]=@AgeRating WHERE ItemGeneralID=@ItemGeneralID"
```

Employee

```
"UPDATE [EmployeeInfo] SET  
[FirstName]=@FirstName,[LastName]=@LastName,[HoursperWeek]=@HoursperWeek,[MedicalInfo]  
=@MedicalInfo,[Employment]=@Employment,[Username]=@Username,[Password]=@Password,[Admin]  
=@Admin WHERE ID=@ID"
```

Aggregate Queries

I am planning on using one aggregate SQL query for the sake of error handling. This query will take a count on the number of users that are admins. This will be used to make sure that when deleting employees, the minimum number of admins on the system does not drop below one.

```
"SELECT COUNT (*) FROM EmployeeInfo WHERE Admin=true"
```

Form potential Designs

All designs were planned and constructed in visual studio 2017

Login Form



Login screen will allow the user to enter a username and password to gain access to the shops inventory management system. Two buttons are present which will log the user or exit the application

Menu Form

The 'Menu' application window features a red background and a blue title bar. It includes a 'Select Table' dropdown menu at the top left. Below this is a large grey rectangular area. To the right of this area is a vertical stack of buttons: 'Add Item', 'Delete Item', 'Update Item/Info', 'Print Tag', 'Employee Management', and 'Search'. Below the grey area is a search bar with the text 'Search for an item by name below' and a message at the bottom: 'Please make sure to highlight an item before using the options Delete, Update or Print'.

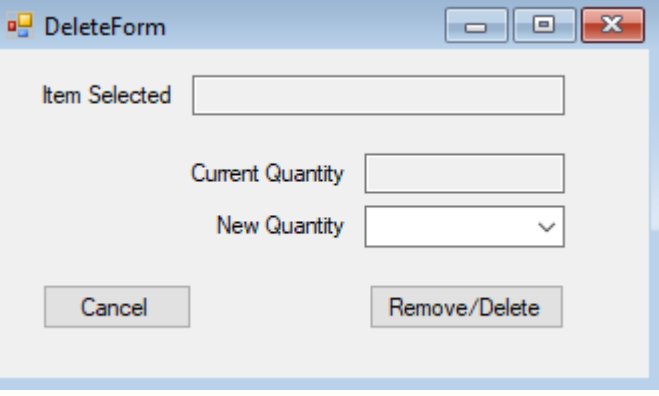
The Menu form has buttons which will allow you to Add, remove, update and create tags for items. Furthermore, you will be able to display shop inventory and search for specific items in stock. Based upon Whether you are an admin you will also have the option to access an employee management button. Also you have the option to view other tables in the database to do with stocks.

Add Items Form

The 'InsertForm' application window has a light blue title bar and a light grey background. It contains a 'Select Category' dropdown menu at the top. Below it are four text input fields arranged in two rows: 'ItemName' and 'ItemPrice' in the first row, and 'ItemCondition' and 'ItemQuantity' in the second row.

The Add Items form will have textboxes to enter information for the required fields. Based on the type of Item further textboxes will appear so that type specific information can be added alongside a confirmation button

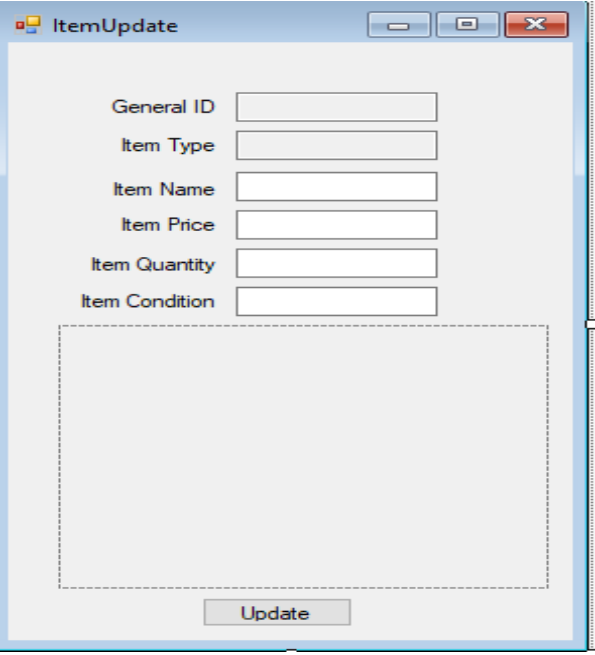
Delete Items Form



The screenshot shows a Windows-style dialog box titled "DeleteForm". It contains three input fields: "Item Selected" (a text box), "Current Quantity" (a text box), and "New Quantity" (a dropdown menu). At the bottom, there are two buttons: "Cancel" and "Remove/Delete".

The Item selected textbox will display the Item you have selected to be deleted. You will then be able to change the quantity of the item selected to a lower number and confirm the removal of the items. There is also a cancel button in case the user does not want to delete the item

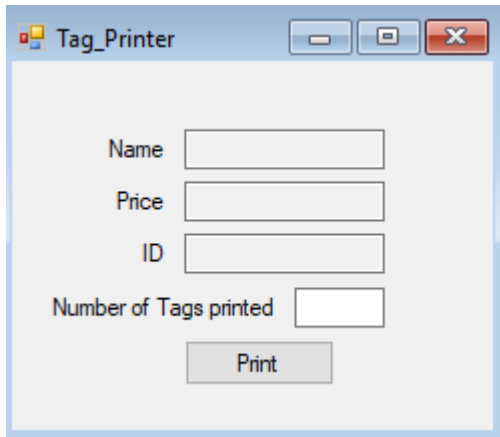
Update Items Form



The screenshot shows a Windows-style dialog box titled "ItemUpdate". It contains six input fields stacked vertically: "General ID", "Item Type", "Item Name", "Item Price", "Item Quantity", and "Item Condition". Below these fields is a large dashed rectangular box, likely for a description or image. At the bottom center is an "Update" button.


The Update item form will display information on the item selected and allow you to change the text in the boxes. More boxes will appear when the Items type has been selected. The button then confirms the changes

Tag Printer Form

The screenshot shows a window titled "Tag_Printer" with a standard Windows title bar (minimize, maximize, close buttons). Inside the window, there are four text input fields stacked vertically, labeled "Name", "Price", "ID", and "Number of Tags printed". Below these fields is a single "Print" button.

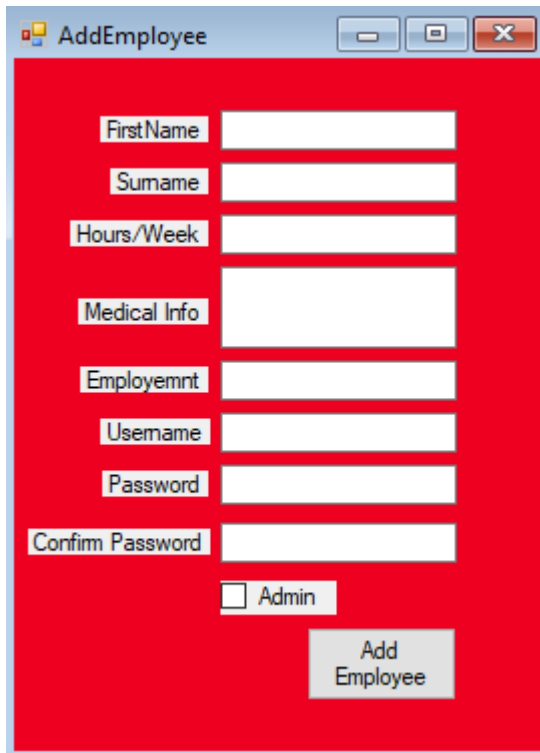
The printer tagging form will display the information that will be saved and printed to a word document. The user can type in the number of tags he wants to print and then click the button to confirm the print and select a save location

Employee Management Form

The screenshot shows a window titled "EmployeeManagement" with a standard Windows title bar. The main area of the window has a red background. At the top left, there is a tab labeled "EmployeeInfo". Below the tab is a large, empty gray rectangular area. To the right of this area, there is a circular refresh icon. Below the refresh icon are three buttons: "Add", "Update", and "Delete". At the bottom of the window, there is a search section with the text "Search for an employee by name below" followed by a text input field and a "Search" button.

If you are an admin you will have access to this form. This form allows you to add, delete, update and search for employee information through the use of the buttons on the right side of the screen.

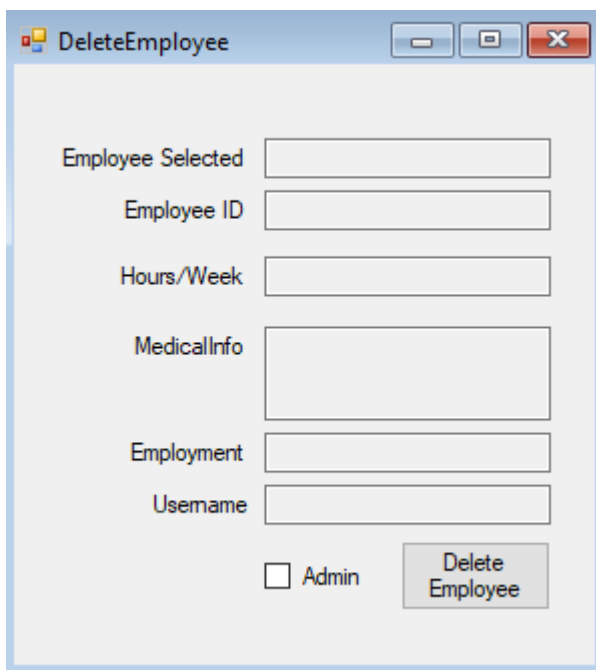
Add Employee Form



The screenshot shows a Windows-style window titled "AddEmployee". The window has a red background. It contains several text input fields with labels to their left: "FirstName", "Surname", "Hours/Week", "Medical Info", "Employemnt" (note the typo), "Username", "Password", and "Confirm Password". Below these fields is a checkbox labeled "Admin". At the bottom right of the form is a button labeled "Add Employee".

When adding an employee, you can enter details into the textboxes and choose whether the user will be an Admin or not. The button is used to confirm the changes made

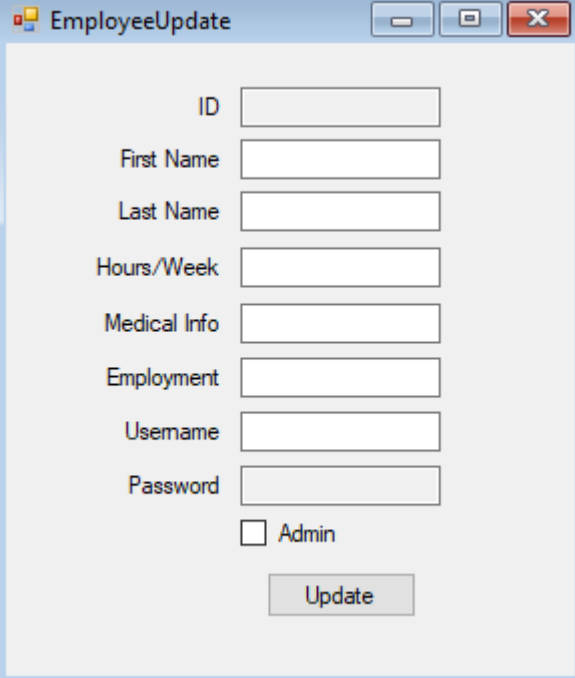
Delete Employee Form



The screenshot shows a Windows-style window titled "DeleteEmployee". The window has a light gray background. It contains several text input fields with labels to their left: "Employee Selected", "Employee ID", "Hours/Week", "MedicallInfo" (note the typo), "Employment", and "Username". Below these fields is a checkbox labeled "Admin". At the bottom right of the form is a button labeled "Delete Employee".

The delete form will allow the user to view the employee's information and then click the delete button when they finally want to remove eth employee from the database

Update Employee Form

A screenshot of a Windows-style application window titled "EmployeeUpdate". The window contains a form with the following fields: "ID", "First Name", "Last Name", "Hours/Week", "Medical Info", "Employment", "Username", and "Password". Each field is represented by a text label followed by a rectangular input box. Below the "Password" field is a checkbox labeled "Admin". At the bottom center of the form is a button labeled "Update". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

When updating an employee, you will be given the information the employee that the user selected, and you will be able to edit the data in the textboxes and the admin checkbox. The user can click the button when he wants to confirm the changes made.

Evidence of A-Level standard

- Complex SQL database design
- Writing to word files
- Aggregate SQL queries
- Admin privilege system
- Password Hashing
- SQL parametrisation to prevent sql injection

Technical Solution

In the technical solution section of my coursework I will be displaying the code I have used to create my solution alongside the form designs relevant to the pieces of code. Throughout the code I will be explain the purpose of different bits of code as well as having a written commentary in green on how the code works.

The following class was created to establish a connection between visual studio and my database of tables. This class manages my sql queries and allows me to paramterise queries as I can call upon fields from the table by referencing DBCONTROL.

DBControl.vb

```
Imports System.Data.OleDb
Public Class DBControl
    'CREATING THE DATABASE CONNECTION'
    'CONNECTION STRING IS THE LOCATION OF DATABASE IN COMPUTER FILES
    Private DbCon As New OleDb.OleDbConnection("Provider = Microsoft.Jet.
OLEDB.4.0;Data Source=C:\Users\Ajab Khanzada\Documents\NEAdbRef.mdb")
    'PREPARES DATABASE FOR COMMANDS'
    Private DBCmd As OleDbCommand
    'PREPARING DATABASE DATA'
    Public DBDA As OleDbDataAdapter
    Public DBDT As DataTable
    'QUERY PARAMETERS'
    Public Params As New List(Of OleDbParameter)
    'NECESSARY QUERY STATISTICS INVOLVED'
    Public recordcounter As Integer
    Public exception As String

    Public Sub ExecQuery(Query As String)
        'THE QUERY STATISTICS ARE BEING RESET BEFOREHAND'
        recordcounter = 0
        exception = ""

        Try
            'CONNECTION IS BEING OPENED UP'
            DbCon.Open()
            'CREATES DATABASE COMMAND'
            DBCmd = New OleDbCommand(Query, DbCon)
            'LOADS PARAMETERS INTO OUR CREATED DATABASE COMMAND
            Params.ForEach(Sub(p) DBCmd.Parameters.Add(p))
            'CLEARS THE PARAMETER LIST'
            Params.Clear()
            'EXECUTES THE APPROPRIATE COMMAND AND FILLS THE DATATABLE'
            DBDT = New DataTable
            DBDA = New OleDbDataAdapter(DBCmd)
            recordcounter = DBDA.Fill(DBDT)
        Catch ex As Exception

            MessageBox.Show(ex.Message)
        End Try
        'CLOSES THE CONNECTION'
        If DbCon.State = ConnectionState.Open Then
            DbCon.Close()
        End If
    End Sub
    'THE SUB IS USED TO INCLUDE THE QUERY AND COMMANDS PARAMETERS
```



```

Public Sub AddParam(Name As String, Value As Object)
    Dim NewParam As New OleDbParameter(Name, Value)
    Params.Add(NewParam)
End Sub

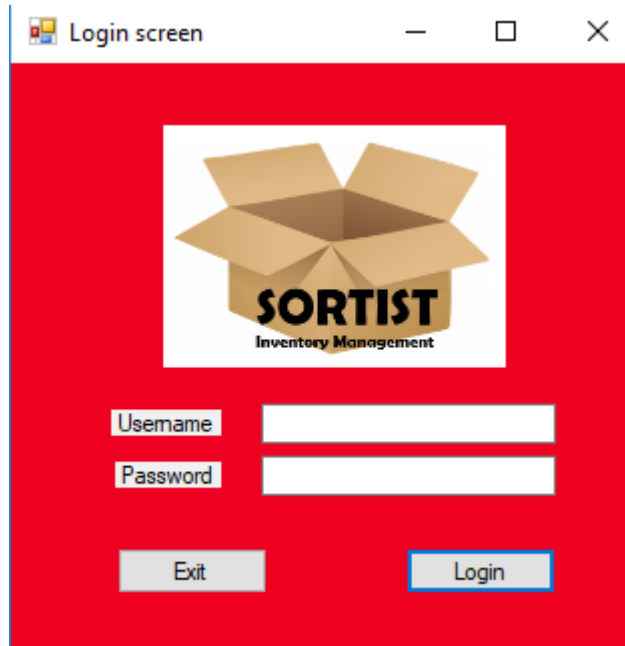
'USED TO CHECK IF THE USER IS AN ADMIN AND TAKES COUNT OF ADMIN
Public Function AdminCount()
    Using mycn As New OleDbConnection("Provider = Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Users\Ajab Khanzada\Documents\NEAdbRef.mdb")

        mycn.Open()
        Dim checkloginquery As String = "SELECT COUNT(*) FROM EmployeeInfo WHERE
Admin=true"
        'AGGREGATE SQL QUERY
        Dim sqlcmd1 As New OleDbCommand(checkloginquery, mycn)
        'STORES HOW MANY RESUKTS ARE FOUND FOR ADMIN ACCOUNTS BASED UPON INFO IN
DATABASE TABLE FOR EMPLOYEES
        Dim count As Integer = Convert.ToInt32(sqlcmd1.ExecuteScalar())
        'THE COUNT FOR ADMINS MAKES SURE THAT THE NUMBER OD ADMINS CANNOT DROP
BELOW 1
        If count <= 1 Then
            'THE FUNCTION RETURNS TRUE OR FALSE BASED UPON THE COUNT OF THE NUMBER
OF ADMINS
            Return False
        Else
            Return True
        End If
    End Using
End Function
End Class

```

Login.vb

The login form is in charge of adding a layer of security to the solution as the database holds information that should only be accessible to employees. The use of an administrative system is used here as users will have admin privileges based upon login credentials entered. Furthermore, the password entered will be encrypted and compared to the one held under the user in the employee management table.



```
'CLASS FOR THE FORM USED TO LOGIN TO APPLICATION
Public Class Login
    'THE VARIABLE ACCESS IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Private access As New DBControl
    'ESTABLISHING VARIABLES THAT ARE NEEDED TO AUTHENTICATE WHETHER A USER IS AN ADMIN
    AND ENTERING CORRECT DETAILS
    Dim password As String
    'ADMINISTRATORS WILL HAVE PRIVILEGES SET TO TRUE UPON DATABASE EXAMINATION
    Public privileges As Boolean
    Public currentUserID As Integer
    'SUB ROUTINE THAT MANAGES THE PROCESSES OCCURRING WHEN THE LOGIN BUTTON IS PRESSED
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles LoginBtn.Click
        'MAKES SURE THAT YOU ENTER TEXT INTO THE TWO TEXTBOXES PROVIDED
        If String.IsNullOrEmpty(UsernameLogin.Text) Then MsgBox("Please Enter a
username") : Exit Sub
        If String.IsNullOrEmpty(TextBox2.Text) Then MsgBox("Please Enter a password")
: Exit Sub
        'PARAMETER USERNAME IS BEING ASSIGNED THE TEXT HELD IN THE USERNAME BOX
        access.AddParam("@username", UsernameLogin.Text)
        'QUERY DRAWS INFORMATION ON THE PASSWORD, ADMIN PRIVILEGES AND ID BASED UPON
        THE USERNAME ENTERED
        access.ExecQuery("SELECT Password, Admin, ID FROM EmployeeInfo WHERE Username
= @username ")
        'THE PASSWORD IN THE DATABASE THAT HAS BEEN SELECTED IS BEING SET TO THE
        VARIABLE PASSWORD
        'BASED UPON THE ADMIN CHECKBOX IN THE DATABASE TABLE THE USER IS BEING GIVEN
        APPROPRIATE PRIVILEGES IF THEY ARE AN ADMIN
```

```

For Each row As DataRow In access.DBDT.Rows
    password = row("Password").ToString
    privileges = row("Admin").ToString
    currentUserID = Convert.ToInt32(row("ID"))
Next

If password <> Nothing Then
    'THE PASSWORD IN THE TABLE RELATING TO YOUR ID IS BEING DECRYPTED AND
    COMPARED TO THE PASSWORD ENTERED
    If New PasswordHashing().Decrypt(password) = TextBox2.Text Then
        'MAIN FORM OPENS
        FormMenu.Show()
        Me.Hide()
        'APPROPRIATE ERROR MESSAGES FOR INCORRECT DETAILS BEING ENTERED
    Else
        MessageBox.Show("Username or Password Incorrect")
    End If
Else
    MessageBox.Show("Username or Password Incorrect")
End If
End Sub

Function GetUsers(x As String, y As String)
    access.AddParam("@user", "%" & x & "%")
    access.AddParam("@pass", "%" & y & "%")
    access.ExecQuery("SELECT Username, Password, Admin FROM EmployeeInfo WHERE
Username LIKE @user AND Password LIKE @pass ")
End Function

Function GetUser(A As String)
    access.AddParam("@user", "%" & A & "%")
    access.ExecQuery("SELECT Username FROM EmployeeInfo WHERE Username")
End Function
'BUTTON ONLY USED FOR TESTING AS MAKES LOGGING IN WITH ADMIN AUTHORITIES QUICKER
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    privileges = True
    FormMenu.Show()
    Me.Hide()
End Sub

End Class

```

Hashing Class

I have used a basic hashing algorithm for my project which revolves around the use of a single key to encrypt text entered. The text that will be encrypted will be user passwords. The class is used to both encrypt and decrypt the password dependent on the scenario. When trying to create a new user the password entered will be encrypted and entered into the table. On the other hand a decryption can be used to check if passwords in the database match one entered in the login screen

```
Imports System.Security.Cryptography
Imports System.Text
'CLASS USED FOR HASHING PASSWORDS
Public Class PasswordHashing
    'ESTABLISHING VARIABLES AND KEY
    Const KEY = "xucphra"
    Dim serviceProvider As New TripleDESCryptoServiceProvider()
    Dim hashMD5Provider As MD5CryptoServiceProvider = New MD5CryptoServiceProvider()
    Dim decryptedStr As String
    Dim encryptedStr As String
    'THE FOLLOWING FUNCTION ENCRYPTS THE PASSWORDS YOU ENTER USING A PRESET KEY
    Public Function Encrypt(Pword As String) As String

        'THE KEY IS CONVERTED INTO BINARY
        Dim byteHash = hashMD5Provider.ComputeHash(Encoding.UTF8.GetBytes(KEY))
        serviceProvider.Key = byteHash
        serviceProvider.Mode = CipherMode.ECB

        'THE ECRYPTOR IS CREATED USING BYTEHASH FROM THE BINARY KEY
        Dim encryptor = serviceProvider.CreateEncryptor()

        'THE STRING OF PASSWORD THE USER ENTERS IS ENCRYPTED
        Dim byteBuff = Encoding.UTF8.GetBytes(Pword)
        'USES THE KEY AND PASSWORD TO ENCRYPT TO A BASE 64 STRING
        encryptedStr = Convert.ToBase64String(encryptor.TransformFinalBlock(byteBuff,
0, byteBuff.Length))
        'THE ENCRYPTED STRING IS RETURNED
        Return encryptedStr
    End Function
    'THIS IS THE FUNCTION USED TO DECRYPT PASSWORDS
    Public Function Decrypt(Pword As String) As String
        'THE KEY IS CONVERTED TO A BINARY FORM
        Dim byteHash = hashMD5Provider.ComputeHash(Encoding.UTF8.GetBytes(KEY))
        serviceProvider.Key = byteHash
        serviceProvider.Mode = CipherMode.ECB
        'A DECRYPTOR IS CREATED USING BYTEHASH FROM THE BINARY KEY
        Dim decryptor = serviceProvider.CreateDecryptor()

        'THE BASE 64 STRING IS THEN CONVERTED TO BINARY
        Dim byteBuff = Convert.FromBase64String(Pword)
        'THE PASSWORD IS DECODED TO UTF8 FROM THE BASE 64 STRING BY THE KEY WE
        CONVERTED TO BINARY
        decryptedStr = Encoding.UTF8.GetString(decryptor.TransformFinalBlock(byteBuff,
0, byteBuff.Length))
        'THE DECRYPTED TEXT IS RETURNED
        Return decryptedStr
    End Function
End Class
```

FormMenu.vb

The menu has been made accessible once a user logs in with the correct credentials. Based upon the username and password the user will have different options in the menu due to admin privileges. Also the menu is in charge of displaying all of the shops item information and will allow you to search for Items across all tables.



```
'CLASS USED FOR THE FORM MAIN MENU ONE LOGGED IN WITH ACCEPTED USERNAME AND PASSWORD
Public Class FormMenu
    'ESTABLISHING VARIABLES NEEDED TO ORDER THE WAY ITEM INFO IS DOCKED INTO THE DATA
    GRID BASED ON TABLE BEING VIEWED
    Dim combostring As String
    Dim combosort As String
    'THE VARIABLE ACCESS IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Private access As New DBControl
    'DATA THAT IS PRE-EMPTIVELY LOADED WITH THE MENU
    Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'QUERY IS GATHERING ALL DATA FROM THE TABLE ITEMSGENERAL AND ORDERING
        APPROPRIATELY
        access.ExecQuery("SELECT * FROM ItemsGeneral ORDER BY ItemName ASC")
        If Not String.IsNullOrEmpty(access.exception) Then MsgBox(access.exception) :
    Exit Sub
        'TABLE INFO IS BEING PUT INTO DATA GRID VIEW
        DgvData.DataSource = access.DBDT
        'ESTABLISHING THE DEFAULT TABLE AS ITEMSGENERAL IN THE COMBOBOX
        combostring = "ItemsGeneral"
        'CALLS UPON INFO FROM THE LOGIN TO CHECK WHETHER USER IS AN ADMIN, IF USER IS
        AN ADMIN THEN THEY HAVE ACCESS TO EMPLOYEE MANAGEMENT
        If Login.priveleges = True Then
            EmployeeManageBtn.Show()
        End If
    End Sub

    'HANDLES INFORMATION IN COMBOBOX TO DECIDE WHICH TABLE TO DISPLAY AND WHAT ORDER
    INFO IS DISPLAYED IN
```

```

    Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBoxMENU.SelectedIndexChanged
        combostring = ComboBoxMENU.Text
        If combostring = "ClothingStocks" Or combostring = "DVDorCDStocks" Or
combostring = "ToyStocks" Then
            combosort = "ItemGeneralID"
        ElseIf combostring = "ItemsGeneral" Then
            combosort = "ItemName"
        End If
        'QUERY USED TO SELECT ALL ITEMS FROM WHICHEVER TABLE IS IN THE COMBOBOX AND
DISPLAY IN A SPECIFIC ORDER
        access.ExecQuery("SELECT * FROM " & combostring & " ORDER BY " & combosort & "
ASC")
        If Not String.IsNullOrEmpty(access.exception) Then MsgBox(access.exception) :
Exit Sub
        DgvData.DataSource = access.DBDT
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
AddItemBtn.Click
        'UPON CLICKING ON THE BUTTON IT OPENS A FORM TO ADD NEW ITEMS
        InsertForm.Show()
    End Sub
    'BUTTON USED FOR INITAITING A SEARCH FOR AN ITEM IN THE TABLE
    Private Sub Button4_Click(sender As Object, e As EventArgs) Handles
SearchItemBtn.Click
        'THE VARIABLE ITEMROW IS BEING USED TO REPRESENT THE NAME OF THE SEARCHED ITEM
AS IT DISPLAYS THE ROWS INFO ON THE ITEM
        Dim itemRow As String
        'IF THE TABLE BEING DISPLAYED ISNT ITEMSGENERLA THEN THE ITEMROW VARIABLE IS
ASSIGNED THE GENERALID
        If combostring = "ClothingStocks" Or combostring = "DVDorCDStocks" Or
combostring = "ToyStocks" Then
            itemRow = "ItemGeneralID"
        'OTHERWISE THE ITEMROW IS BEING USED TO REFER TO THE NAME BEING SEARCHED
        ElseIf combostring = "ItemsGeneral" Then
            itemRow = "ItemName"
        'ITEMS WILL BE ORDERED BY NAME
            combosort = "ItemName"
        End If
        'THE QUERY IS BEING USED TO DISPLAY THE ROW OF THE SEARCHED ITEM USING THE
INFORMATION ITEMROW AND COMBOSORT WHICH SPECIFIES THE ORDER
        access.ExecQuery("SELECT * FROM " & combostring & " WHERE " & itemRow & " = '"
& SearchBar.Text & "' ORDER BY " & combosort & " ASC")
        If Not String.IsNullOrEmpty(access.exception) Then MsgBox(access.exception) :
Exit Sub
        'THE DATA SLECTED BASED UPON THE SEARCH IS DISPLAYED IN THE DATA GRID VIEW
        DgvData.DataSource = access.DBDT

    End Sub
    'UPON CLICKING THE DELETE ITEM BUTTON THE FORM FOR DELETING ITEMS WILL SHOW UP
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
DeleteItemBtn.Click
        DeleteForm.Show()
    End Sub
    'UPON CLICKING THE EMPLOYEE MANAGEMENT BUTTON WHICH IS EXCLUSIVE TO ADMINS THE
EMPLOYEE MANAGEMENT FORM WILL SHOW UP
    Private Sub EmployeeManageBtn_Click(sender As Object, e As EventArgs) Handles
EmployeeManageBtn.Click
        EmployeeManagement.Show()
    End Sub

```

```

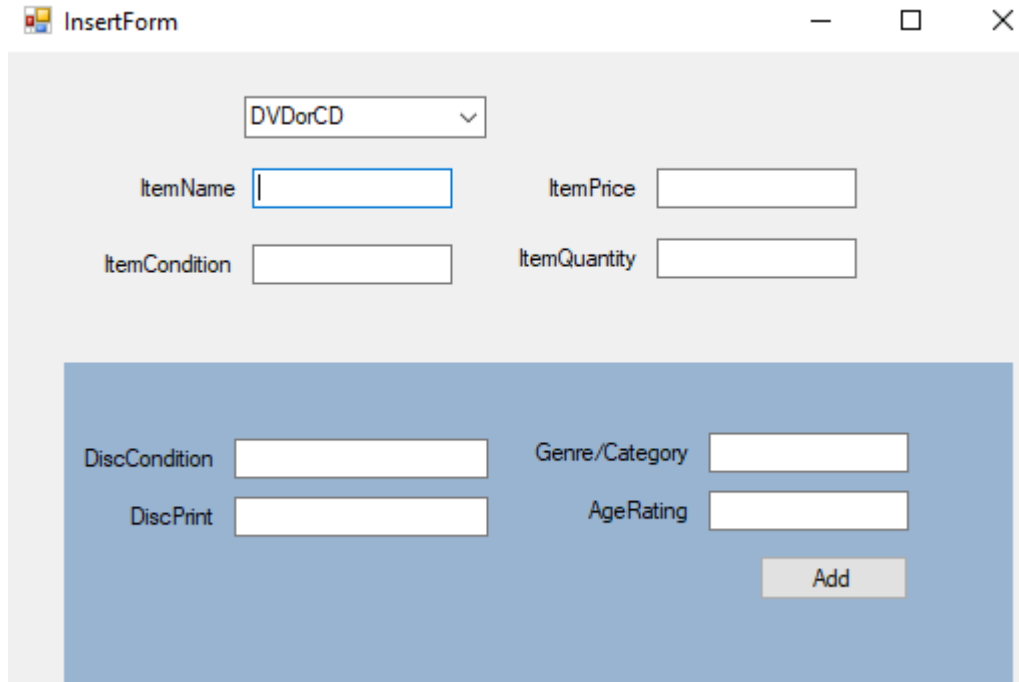
    'THE REFRESH BUTTON IS USED TO SEE UPDATED INFORMATION IN THE TABLES AS IT RELOADS
    THE ITEMSGENERAL TABLE
    Private Sub Button6_Click(sender As Object, e As EventArgs) Handles
MenuRefresh.Click
    'THE TABLE IS RELOADED EASILY BY CHANGING THE TEXT IN THE COMBOBOX AS A
    DIFFERENT TABLE IS LOADED QUICKLY
        ComboBoxmMENU.Text = "ToyStocks"
        ComboBoxmMENU.Text = "ItemsGeneral"

    End Sub
    'UPON CLICKING THE UPDATE ITEM BUTTON THE FORM FOR UPDATING ITEM INFORMATION WILL
    SHOW UP
    Private Sub UpdateItemBtn_Click(sender As Object, e As EventArgs) Handles
UpdateItemBtn.Click
        ItemUpdate.Show()
    End Sub
    'UPON CLICKING THE BUTTON FOR TAG PRINTING THE FORM FOR TAG PRINTING WILL SHOW UP
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles
TagButton.Click
        Tag_Printer.Show()
    End Sub
End Class

```

InsertForm.vb

The Insert form will allow a user to add new items to the database. It will display relevant textboxes based upon the Item type selected to add and prompt you to fill out all the information. All the data will be added to 2 tables, the Items General table and the type specific table depending on the item type.



```
'THE CLASS USED FOR THE FORM REQUIRED TO ADD NEW ITEMS
Public Class InsertForm
    'THE VARIABLE ACCESS IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Dim db As New DBControl
    Dim ID As Integer
    'THE COMBOBOX IS BEING USED TO SELECT THE TYPE OF ITEM YOU WANT TO ADD
    Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles TypeGenCB.SelectedIndexChanged
        For Each c As Control In Panel_CategoryBackground.Controls
            If (TypeOf c Is Panel) Then
                c.Hide()
            End If
        Next
        'BASED UPON THE TYPE IN THE COMBOBOX THE PANEL ASSOCIATED WITH THAT TYPE WILL
        APPEAR
        'THE PANELS FOR THE TYPES ALLOW YOU TO ENTER TYPE SPECIFIC INFORMATION
        If TypeGenCB.SelectedItem = "Clothing" Then
            'DISPLAYS CLOTHING PANEL AND DOCKS IT INTO THE MOTHER PANEL
            PanelClothes.Show()
            PanelClothes.Dock = DockStyle.Fill
        ElseIf TypeGenCB.SelectedItem = "DVDorCD" Then
            'DISPLAYS THE DVD PANEL AND DOCKS IT INTO THE MOTHER PANEL
            PanelDisc.Show()
            PanelDisc.Dock = DockStyle.Fill
        ElseIf TypeGenCB.SelectedItem = "Toy" Then
            'DISPLAYS THE TOY PANEL AND DOCKS IT INTO THE MOTHER PANEL
            PanelToy.Show()
        End If
    End Sub
End Class
```



```

        PanelToy.Dock = DockStyle.Fill
    End If
End Sub
'SUBROUTINE IN CHARGE OF ADDING THE INFORMATION TO THE TABLE ITEMS GENERAL
Sub AddGeneral()
    Try
        'PARAMETERISATION
        'TEXTBOXES ARE BEING ASSIGNED TO THE DIFFERENT PARAMETERS
        db.AddParameter("@name", ItemGenName.Text)
        db.AddParameter("@price", ItemGenPrice.Text)
        db.AddParameter("@condition", ItemGenCondition.Text)
        db.AddParameter("@quantity", ItemGenQuantity.Text)
        db.AddParameter("@typeGen", TypeGenCB.Text)
        'THE QUERY IS INSERTING THE INFORMATION FROM THE TEXTBOXES INTO THE TABLE
ITEMSGENERAL
        db.ExecQuery("INSERT INTO ItemsGeneral
([ItemName],[ItemPrice],[ItemCondition],[ItemQuantity],[ItemType])
Values(@name,@price,@condition,@quantity,@typeGen)")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Exit Sub
End Try
End Sub
'SUBROUTINE USED TO ADD THE SPECIFIC CLOTHING INFORMATION TO THE REQUIRED TABLE
Private Sub Addclothing()
    Try
        db.AddParameter("@type", ctypetxt.Text)
        db.AddParameter("@size", csize.txt.Text)
        db.AddParameter("@itemID", ID)
        db.AddParameter("@brand", cbrand.txt.Text)
        db.AddParameter("@gender", cgendertxt.Text)
        db.ExecQuery("INSERT INTO ClothingStocks
([ClothingType],[ClothingSize],[ItemGeneralID],[ClothingBrand],[ClothingGender])
Values(?,?,?,?,?)")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Exit Sub
End Try
    MsgBox("item added")
End Sub
'SUBROUTINE USED TO ADD THE SPECIFIC TOY INFORMATION TO THE REQUIRED TABLE
Private Sub AddToy()
    Try
        db.AddParameter("@itemID", ID)
        db.AddParameter("@type", ToyStckType.Text)
        db.AddParameter("@make", ToyStckMake.Text)
        db.AddParameter("@age", ToyStckAge.Text)
        db.AddParameter("@safety", ToyStckSafety.Text)
        db.ExecQuery("INSERT INTO ToyStocks
([ItemGeneralID],[ToyType],[ToyMake],[ToyAgeRecommendation],[ToySafety])
Values(?,?,?,?,?)")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Exit Sub
End Try
    MsgBox("Item added")
End Sub
'SUBROUTINE USED TO ADD SPECIFIC DVD INFORMATION TO REQUIRED TABLE
Private Sub AddDVD()
    Try
        db.AddParameter("@itemID", ID)
        db.AddParameter("@print", DiscStkPrint.Text)
    
```

```

        db.AddParameter("@genre", DiscStkGenre.Text)
        db.AddParameter("@age", DiscStkAge.Text)
        db.AddParameter("@condition", DiscStkCondition.Text)
        db.ExecQuery("INSERT INTO DVDorCDStocks
([ItemGeneralID],[DiscPrint],[GenreOrCategory],[AgeRating],[DiscCondition])
Values(?,?,?,?,?)")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Exit Sub
End Try
MsgBox("Item added")
End Sub

'SUBROUTINE USED TO STORE THE ITEM ID OF THE ITEM BEING ADDED TO THE TABLE
ITEMSGENERAL
'THIS ALLOWS US TO ADD THE ADDITIONAL TYPE INFO WHERE THE IDs MATCH UP
Public Sub SummonID()
    'QUERY SELECTS THE GENERLAID FROM THE GENERAL ITEMS TABLE FOR THE NAME WE
    ENETERED OF OUR NEWLY ADDED ITEM
    db.ExecQuery("SELECT ItemGeneralID FROM ItemsGeneral WHERE ItemName = '" &
    ItemGenName.Text & "'")

    ID = CType(db.DBDT(0)("ItemGeneralID"), Integer)
    'DEFAULT VALUE GIVEN TO THE VARIABLE ITEM
    Dim item As Integer = 23

End Sub
'CODE REFERS TO THE BUTTON TO ADD NEW ITEM INFO FOR CLOTHING TYPES
Private Sub AddBtn_Click(sender As Object, e As EventArgs) Handles AddBtn.Click
    If ValidatAllTxbx() Then
        If Not Validatetxt(PanelClothes) Then
            AddGeneral()
            SummonID()
            Addclothing()
        Else
            MessageBox.Show("Please fill out all fields")
        End If
    Else
        MessageBox.Show("Please fill out all fields")
    End If
End Sub
'CODE REFERS TO THE BUTTON TO ADD NEW ITEM INFO FOR TOY TYPES
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
AddToyBtn.Click
    If ValidatAllTxbx() Then
        If Not Validatetxt(PanelToy) Then
            AddGeneral()
            SummonID()
            AddToy()
        Else
            MessageBox.Show("Please fill out all fields")
        End If
    Else
        MessageBox.Show("Please fill out all fields")
    End If

End Sub
'CODE REFERS TO THE BUTTON TO ADD NEW ITEM INFO FOR DISC TYPES
Private Sub AddDiscBtn_Click(sender As Object, e As EventArgs) Handles
AddDiscBtn.Click
    'CHECKS IF THE DATA HAS BEEN FILLED OUT FOR THE TEXTBOXES ON THE FORM
    If ValidatAllTxbx() Then

```

```

        'CHECKS IF THE TEXTBOXES HAVE BEEN FILLED OUT FOR THE PANEL OF THE ITEM
TYPE
        If Not Validatetxt(PanelDisc) Then
            'PROCEEDS TO CALL UPON SUB ROUTINES TO ADD ITEM TO TABLE
            AddGeneral()
            SummonID()
            AddDVD()
            'DISPLAYS APPROPRIATE ERROR MESSAGES
        Else
            MessageBox.Show("Please fill out all fields")
        End If
    Else
        MessageBox.Show("Please fill out all fields")
    End If

End Sub
'THE FUNCTION IS USED FOR ERROR HANDLING, ITE ENSURES THAT NO FIELDS ARE LEFT
EMPTY
'ALL YTEXTBOXES ARE FILLED OUT OR ERROR MESSAGE APPEARS
Public Function ValidatAllTxtbx()
    Dim ctrl As Control
    Dim result As Boolean

    result = True        'set this to false if a textbox fails
    For Each ctrl In Me.Controls
        If TypeOf ctrl Is TextBox Then
            'USES THE FACT THAT A TEXTBOX IS EMPTY WHEN IT HAS NO CHARACTERS TO
TELL WHETHER A TEXTBOX HAS BEEN LEFT OUT
            If Len(ctrl.Text) = 0 Then
                result = False
                Exit For 'EXIT ON THE FIRST FAILURE OR APPEARANCE OF A TEXTBOX
BEING EMPTY
            End If
        End If
    Next ctrl
    'USED TO RETURN THE RESUKT OF THE TEXTBOX FILLOUT CHECK
    ValidatAllTxtbx = result
End Function
'FUNCTION CHECKS IF THE TEXT IN THE BOXES WITHIN THE PANELS HAVE BEEN FILLED OUT
Public Function Validatetxt(ByVal p As Panel) As Boolean
    For Each t As TextBox In p.Controls.OfType(Of TextBox)
        If t.Text = vbNullString Then
            Return True
        End If
    Next
    Return False
End Function
End Class

```

ItemUpdate.vb

When trying to update an Items information the Item highlighted in the menu will have its information drawn from the tables and displayed in the correct textboxes. The text in the boxes can be altered and upon the button press the changes to items are applied. This means that the information in the general table and the type specific table will have to be updated through unique sql queries.

General ID 51

Item Type DVDorCD

Item Name Jurassic World: Dire

Item Price \$9.99

Item Quantity 4

Item Condition Used

DiscID 9

Disc Condition Mint

Disc Print DVD

Genre/Category Sci-Fi

Age Rating 15

Update

```
'CLASS FOR THE FORM USED TO UPDATE ITEM INFORMATION
Public Class ItemUpdate
    'THE VARIABLE ACCESS IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Dim access As New DBControl
    Public Function ValidatAllTxbx()
        Dim ctrl As Control
        Dim result As Boolean

        result = True 'set this to false if a textbox fails
        For Each ctrl In Me.Controls
            If TypeOf ctrl Is TextBox Then
                If Len(ctrl.Text) = 0 Then
                    result = False
                    Exit For 'bail on the first failure
                End If
            End If
        Next ctrl
        ValidatAllTxbx = result
    End Function
```

```

Public Function Validatetxt(ByVal p As Panel) As Boolean
    For Each t As TextBox In p.Controls.OfType(Of TextBox)
        If t.Text = vbNullString Then
            Return True
        End If
    Next
    Return False
End Function

'SUBROUTINE THAT LOADS UP SPECIFIC INFORMATION ALONGSIDE THE FORM
Private Sub ItemUpdate_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'CHECKS THAT THE NUMBER OF HIGHLIGHTED VALUES IN THE DATAGRID IN THE MENU IS
    MORE THAN ONE
    If FormMenu.DgvData.SelectedRows.Count > 0 Then
        'DISPLAYS THE TEXT IN THE DATAGRIDVIEW FOR HIGHLIGHTED ITEMS IN THE
        APPROPRIATE TEXTBOX
        'DIFFERENT DATA IS ASSIGNED TO A DIFFERENT TEXTBOX WITH A LABEL
        ItemUpGenID.Text =
FormMenu.DgvData.SelectedRows(0).Cells(0).Value.ToString
        ItemUpType.Text = FormMenu.DgvData.SelectedRows(0).Cells(1).Value.ToString
        ItemUpName.Text = FormMenu.DgvData.SelectedRows(0).Cells(2).Value.ToString
        ItemUpPrice.Text =
FormMenu.DgvData.SelectedRows(0).Cells(3).Value.ToString
        ItemUpQuantity.Text =
FormMenu.DgvData.SelectedRows(0).Cells(4).Value.ToString
        ItemUpCondition.Text =
FormMenu.DgvData.SelectedRows(0).Cells(5).Value.ToString
        'THE INFORMATION HIGHLIGHTED ONLY BRINGS UP ITEMGENERAL DATA
        'THE CORRECT PANEL IS THEN BROUGHT UP DEPENDING ON THE ITEM TYPE
        For Each c As Control In MotherUpPanel.Controls
            If (TypeOf c Is Panel) Then
                c.Hide()
            End If
        Next
        'BRINGS UP CLOTHING PANEL AND DOCKS IT
        If ItemUpType.Text = "Clothing" Then
            ClothingUpPanel.Show()
            ClothingUpPanel.Dock = DockStyle.Fill
            'BRINGS UP TOY PANEL AND DOCKS IT
        ElseIf ItemUpType.Text = "Toy" Then
            ToyUpPanel.Show()
            ToyUpPanel.Dock = DockStyle.Fill
            'BRINGS UP DVD PABNEL AND DOCKS IT
        ElseIf ItemUpType.Text = "DVDorCD" Then
            DVDorCDUpPanel.Show()
            DVDorCDUpPanel.Dock = DockStyle.Fill
        End If

        'ITEN GENERAL IFO IS DISPLAYED DUE TO THE CODE ABOVE
        'THE FOLLOWING CODE DISPLAYS THE EXTRA INFO ON ITEMS BASED UPON TYPE
        If ItemUpType.Text = "Clothing" Then
            'THE QUERY SELECTS ALL THE INFO FROM THE CLOTHING TABLE WHERE THE ID
            MATCHES TO THAT INSERTED INTO THE ID TEXTBOX
            access.ExecQuery("SELECT * FROM ClothingStocks WHERE ItemGeneralID = "
& ItemUpGenID.Text)
            'THE INFO SELECTED IS THEN PLACED INTO THE CORRECT TEXTBOXES
            For Each r As DataRow In access.DBDT.Rows
                ItemUpClothID.Text = r(0)
                ItemUpClothType.Text = r(1)
                ItemUpClothSize.Text = r(2)
                ItemUpClothBrand.Text = r(4)
                ItemUpClothGender.Text = r(5)
            End For
        End If
    End If
End Sub

```

```

        Next
    ElseIf ItemUpType.Text = "Toy" Then
        'THE QUERY SELECTS ALL THE INFO FROM THE TOYSTOCKS TABLE WHERE THE ID
MATCHES TO THAT INSERTED INTO THE ID TEXTBOX
        access.ExecQuery("SELECT * FROM ToyStocks WHERE ItemGeneralID = " &
ItemUpGenID.Text)
        'THE INFO SELECTED IS THEN PLACED INTO THE CORRECT TEXTBOXES
        For Each r As DataRow In access.DBDT.Rows
            ItemUpToyID.Text = r(0)
            ItemUpToyMake.Text = r(1)
            ItemUpToyAgeRec.Text = r(3)
            ItemUpToyType.Text = r(4)
            ItemUpToySafety.Text = r(5)
        Next
    ElseIf ItemUpType.Text = "DVDorCD" Then
        'THE QUERY SELECTS ALL THE INFO FROM THE DVDORCD TABLE WHERE THE ID
MATCHES TO THAT INSERTED INTO THE ID TEXTBOX
        access.ExecQuery("SELECT * FROM DVDorCDStocks WHERE ItemGeneralID = "
& ItemUpGenID.Text)
        'THE INFO SELECTED IS THEN PLACED INTO THE CORRECT TEXTBOXES
        For Each r As DataRow In access.DBDT.Rows
            ItemUpDiscID.Text = r(0)
            ItemUpDiscCondition.Text = r(1)
            ItemUpDiscPrint.Text = r(3)
            ItemUpDiscCategory.Text = r(4)
            ItemUpDiscAgeRating.Text = r(5)
        Next
    End If
End If
End Sub
'THE FOLLOWING SUB APPLIES THE ERROR CHECKING METHOD FOR EMPTY FIELDS TO THE
TEXTBOXES
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If ValidatAllTxbx() Then
        'CHECKS IF THE TEXTBOXES ARE ALL FILLED OUT WITHIN THE CLOTHING PANEL
        If ItemUpType.Text = "Clothing" Then
            If Not Validatetxt(ClothingUpPanel) Then
                'IF THE TEXTBOX CHECK PASSES THEN THE ITEM INFO IS UPADTED AS THE
CORRECT SUBROUTINE IS CALLED
                UpdateClothing()
            Else
                MsgBox("please fill all required fields for Clothing")
            End If
            'CHECKS IF THE TEXTBOXES ARE ALL FILLED OUT WITHIN THE DVDORCD PANEL
        ElseIf ItemUpType.Text = "DVDorCD" Then
            If Not Validatetxt(DVDorCDUpPanel) Then
                'IF THE TEXTBOX CHECK PASSES THEN THE ITEM INFO IS UPADTED AS THE
CORRECT SUBROUTINE IS CALLED
                UpdatedVD()
            Else
                MsgBox("please fill all required fields for DVD's/CD's")
            End If
            'CHECKS IF THE TEXTBOXES ARE ALL FILLED OUT WITHIN THE TOY PANEL
        ElseIf ItemUpType.Text = "Toy" Then
            If Not Validatetxt(ToyUpPanel) Then
                'IF THE TEXTBOX CHECK PASSES THEN THE ITEM INFO IS UPADTED AS THE
CORRECT SUBROUTINE IS CALLED
                UpdateToys()
            Else
                MsgBox("please fill all required fields for toys")
            End If
        End If
    End If

```

```

Else
    MsgBox("please fill all required fields")
End If

End Sub

'SUBROUTINE USED TO UPDATE CLOTHING TYPES
Sub UpdateClothing()
    Try
        'ASSIGNING PARAMETERS TO TEXTBOXES ON THE FORM
        access.AddParam("@ItemType", ItemUpType.Text)
        access.AddParam("@ItemName", ItemUpName.Text)
        access.AddParam("@ItemPrice", ItemUpPrice.Text)
        access.AddParam("@ItemQuantity", ItemUpQuantity.Text)
        access.AddParam("@ItemCondition", ItemUpCondition.Text)
        access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
        access.ExecQuery("UPDATE [ItemsGeneral] SET
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID")
        'ASSIGNING PARAMETERS TO TEXTBOXES ON THE ITEM SPECIFIC TYPE PANEL
        access.AddParam("@ClothingType", ItemUpClothType.Text)
        access.AddParam("@ClothingSize", ItemUpClothSize.Text)
        access.AddParam("@ClothingBrand", ItemUpClothBrand.Text)
        access.AddParam("@ClothingGender", ItemUpClothGender.Text)
        access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
        access.ExecQuery("UPDATE [ClothingStocks] SET
[ClothingType]=@ClothingType,[ClothingSize]=@ClothingSize,[ClothingBrand]=@ClothingBra
nd,[ClothingGender]=@ClothingGender WHERE ItemGeneralID=@ItemGeneralID")
        MessageBox.Show("Update successful")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub

'SUBROUTINE USED TO UPDATE TOY TYPES
Sub UpdateToys()
    Try
        'ASSIGNING PARAMETERS TO TEXTBOXES ON THE FORM
        access.AddParam("@ItemType", ItemUpType.Text)
        access.AddParam("@ItemName", ItemUpName.Text)
        access.AddParam("@ItemPrice", ItemUpPrice.Text)
        access.AddParam("@ItemQuantity", ItemUpQuantity.Text)
        access.AddParam("@ItemCondition", ItemUpCondition.Text)
        access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
        access.ExecQuery("UPDATE [ItemsGeneral] SET
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID")
        'ASSIGNING PARAMETERS TO TEXTBOXES ON THE ITEM SPECIFIC TYPE PANEL
        access.AddParam("@ToyMake", ItemUpToyMake.Text)
        access.AddParam("@ToyAgeRecommendation", ItemUpToyAgeRec.Text)
        access.AddParam("@ToyType", ItemUpToyType.Text)
        access.AddParam("@ToySafety", ItemUpToySafety.Text)
        access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
        access.ExecQuery("UPDATE [ToyStocks] SET
[ToyMake]=@ToyMake,[ToyAgeRecommendation]=@ToyAgeRecommendation,[ToyType]=@ToyType,[To
ySafety]=@ToySafety WHERE ItemGeneralID=@ItemGeneralID")
        MessageBox.Show("Update Successful")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub

'SUBROUTINE USED TO UPDATE DVD TYPES
Sub UpdateDVD()

```

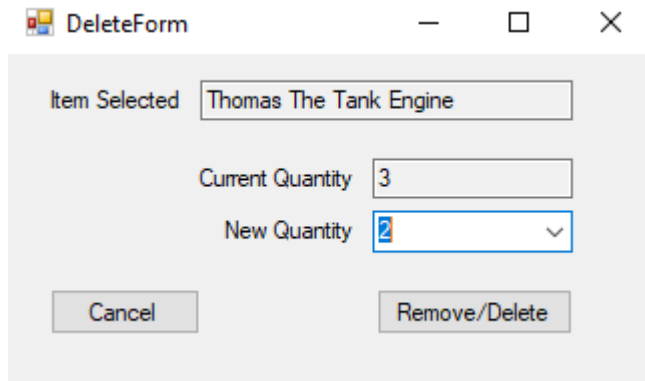
```

Try
    'ASSIGNING PARAMETERS TO TEXTBOXES ON THE FORM
    access.AddParam("@ItemType", ItemUpType.Text)
    access.AddParam("@ItemName", ItemUpName.Text)
    access.AddParam("@ItemPrice", ItemUpPrice.Text)
    access.AddParam("@ItemQuantity", ItemUpQuantity.Text)
    access.AddParam("@ItemCondition", ItemUpCondition.Text)
    access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
    access.ExecQuery("UPDATE [ItemsGeneral] SET
[ItemType]=@ItemType,[ItemName]=@ItemName,[ItemPrice]=@ItemPrice,[ItemQuantity]=@ItemQ
uantity,[ItemCondition]=@ItemCondition WHERE ItemGeneralID=@ItemGeneralID")
    'ASSIGNING PARAMETERS TO TEXTBOXES ON THE ITEM SPECIFIC TYPE PANEL
    access.AddParam("@DiscCondition", ItemUpDiscCondition.Text)
    access.AddParam("@DiscPrint", ItemUpDiscPrint.Text)
    access.AddParam("@GenreorCategory", ItemUpDiscCategory.Text)
    access.AddParam("@AgeRating", ItemUpDiscAgeRating.Text)
    access.AddParam("@ItemGeneralID", ItemUpGenID.Text)
    access.ExecQuery("UPDATE [DVDorCDStocks] SET
[DiscCondition]=@DiscCondition,[DiscPrint]=@DiscPrint,[GenreorCategory]=@GenreorCatego
ry,[AgeRating]=@AgeRating WHERE ItemGeneralID=@ItemGeneralID")
    MessageBox.Show("Update Successful")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
End Sub
End Class

```


DeleteForm.vb

The selected Item will have its current quantity and name on display and you will be allowed to change the quantity value. Based upon whether your quantity is zero or not a different sql query will be used. When the stock drops to zero a DELETE statement will be used to remove the item from its 2 tables however when the stock value simply changes an UPDATE query will be used.



```
'CLASS FOR THE FORM USED TO DELTE ITEMS FROM THE DATABASE
Public Class DeleteForm
    'THE VARIABLE DB IS BEING USED TO DRAW ARGUEMNETS FROM THE CLASS DBCONTROL
    Dim db As New DBControl
    'ITEMTYPE AND ID ARE BEING USED TO STORE STRING
    Dim ItemType As String
    Dim ItemID As Integer
    'SUB USED FOR A BUTTON TO CLOSE THE FORM
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles CLOSEBTN.Click
        Me.Close()
    End Sub
    'SUB USED FOR THE BUTTON CLICK TO DELETE AND ITEM FROM THE DATABASE
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
DeleteBtn.Click
        'DEPENDANT ON THE NEW QUANTITY OF ITEM YOU SELECT IT EITHER UPDATES OR DELETES
AN ITEM
        'WHEN THE QUANTITY IS CHANGED TO 0 WE CALL UPON THE DELETEDDELETE SUB
        If ItemQuantityCB.Text = "0" Then
            DeleteDelete()
            MessageBox.Show("Items Deleted")
        Else
            'IF THE QUANTITY IS GREATER THAN 0 THE SUB UPDATEDDELETE IS CALLED UPON
            UpdateDelete()
            MessageBox.Show("Items Updated. " & ItemQuantityGen.Text -
ItemQuantityCB.Text & " items removed")
        End If
        'Form IS CLOSED
        Me.Close()
    End Sub
    'SUB USED TO UPDATE THE QUANTITY OF AN ITEM
    Sub UpdateDelete()
        'PARAMATERISATION OF ITENAME AND QUANTITY
        db.AddParam("@name", ItemNameGen.Text)
        db.AddParam("@quantity", ItemQuantityGen)
        'COMBOBOX IS BEING INPUTTED WITH THE CURRENT QUANTITY
        Dim cbString As String
        cbString = ItemQuantityCB.Text
```

```

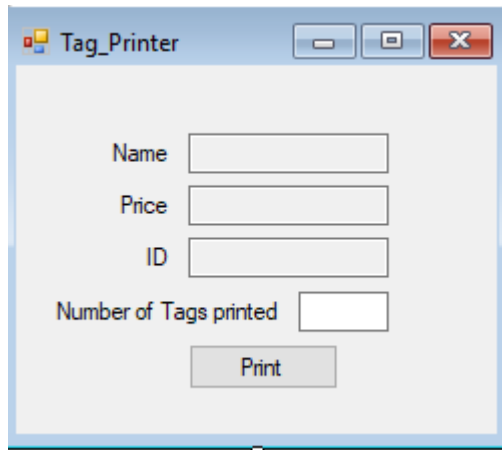
        'UPDATE STATEMENT WHICH CHANGES THE QUANTITY OF AN ITEM TO WHTEVER THE NEW
        VALUE IN THE COMBOBOX IS
        'THE UPDATE IS APPLIED TO ITEMSGENERAL WHERE THE ITEMNAME IS USED TO SOURCE
        THE CORRECT ITEM ROW TO EDIT
        db.ExecQuery("UPDATE ItemsGeneral SET ItemQuantity = '" & ItemQuantityCB.Text
& "' WHERE ItemName = '" & ItemNameGen.Text & "'")
    End Sub
    'SUB USED TO DELETE AN ITEM ENTIRRELY FROM THE DATABASE WHHEN QUANTITY REACHES 0
    Sub DeleteDelete()
        'THE ITEM WHOS NAME IS IN THE TEXTBOX ItemNameGen IS DELETED FROM THE
        ITEMSGENERAL TABLE
        db.ExecQuery("DELETE FROM [ItemsGeneral] WHERE ItemName = '" &
ItemNameGen.Text & "'")
        'THE ITEM IS ALSO DELETED FROM ITS TYPE SPECIFIC TABLE USING ITS ID
        db.ExecQuery("DELETE FROM [" & ItemType & "Stocks] WHERE ItemGeneralID = " &
ItemID)
    End Sub
    'SUB USED TO LOAD IN DATA ALONGSIDE THE DELETE ITEM FORM
    Private Sub DeleteForm_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'i WILL BE USED TO REPRESENT AN INTEGER VALUE FOR ITEM QUANTITY - 1
        Dim i As Integer
        'LOAD DATA IS THE COMBOBOX IN THE MENUFORM IS ON ITEMSGENERAL
        If FormMenu.ComboBoxmMENU.Text = "ItemsGeneral" Then
            'MAKE SURE THAT THE NUMBER OF HIGHLIGHTED ROWS ON DATA GRID VIEW IS MORE
            THAN ONE
            If FormMenu.DgvData.SelectedRows.Count > 0 Then
                'EQAUTING HIGHLIGHTED VALUES TO STRING IN THE FOLLOWING TEXTBOXES
                ItemNameGen.Text =
FormMenu.DgvData.SelectedRows(0).Cells(2).Value.ToString
                ItemQuantityGen.Text =
FormMenu.DgvData.SelectedRows(0).Cells(4).Value.ToString
                ItemType = FormMenu.DgvData.SelectedRows(0).Cells(1).Value.ToString
                ItemID = FormMenu.DgvData.SelectedRows(0).Cells(0).Value.ToString

                'COMBOBOX DISPLAYS NEW POTENTIAL QUANTITIES IN INCRIMENTS OF -1 UNTIL
                0
                For i = ItemQuantityGen.Text - 1 To 0 Step -1
                    ItemQuantityCB.Items.Add(i)
                Next
            End If
        Else
            'ERROR MESSAGE TO SHOW THAT ITEMS CAN ONLY BE DELETD ONCE SELECTED FROM
            ITEMSGENERAL TABLE
            MessageBox.Show("You can only delete from the table ItemsGeneral")
            Me.Close()
        End If
    End Sub
End Class

```

TagPrinter.vb

The tag printer form is in charge of formatting a tag into a word document, saving the document and finally printing a specified number of tags for the chosen item. This is done by establishing a connection between visual studio and Microsoft word. This works in a very similar way to reading and writing to files.



```
'IMPORTS FUNCTIONS AND ESTABLISHES A CONNECTION BETWEEN VS AND WORD
Imports word = Microsoft.Office.Interop.Word
'CLASS USED FOR THE FORM IN ORDER TO PRINT AND SAVE ITEM TAGS FOR EMPLOYEES
Public Class Tag_Printer
    'SUB THAT LOADS DATA IN WITH THE FORM
    Private Sub Tag_Printer_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'WHEN THE NUMBER OF SELETEDC ROWS IS GREATER THAN 1 IT FETCHES DATA FROM THE
MENU
        If FormMenu.DgvData.SelectedRows.Count > 0 Then
            'THE FOLLOWING DATA IN THE MENU IS APPENDED TO THESES TEXTBOXES
            PrintID.Text = FormMenu.DgvData.SelectedRows(0).Cells(0).Value.ToString()
            PrintName.Text =
FormMenu.DgvData.SelectedRows(0).Cells(2).Value.ToString()
            PrintPrice.Text =
FormMenu.DgvData.SelectedRows(0).Cells(3).Value.ToString()
        End If
    End Sub
    'SUBROUTINE FOR PRINTING AND SAVING TAGS
    Sub AddtoWord()
        'CREATING THE WORD DOCUMENT
        Dim wordapp As word.Application = New word.Application
        Dim worddoc As word.Document = wordapp.Documents.Add()
        Dim para As word.Paragraph = worddoc.Paragraphs.Add()
        For i = 0 To NoPrinted.Text - 1
            'THIS WILL WRITE OUT INFORMATION ON AN ITEM AND THEN LEAVE 2 LINES AFTER
            para.Range.Text = ("Item Name: " & PrintName.Text)
            para.Range.InsertParagraphAfter()
            para.Range.Text = ("Item Price: " & PrintPrice.Text)
            para.Range.InsertParagraphAfter()
            para.Range.Text = ("Item ID: " & PrintID.Text)
            para.Range.InsertParagraphAfter()
            para.Range.InsertParagraphAfter()
        Next
        'GETS THE FILE LOCATION THE USER WANTS TO STORE THE TAGS TO
        Dim fd As FolderBrowserDialog = New FolderBrowserDialog
```

```

If (fd.ShowDialog() = DialogResult.OK) Then
    'THE USER CAN CHOOSE WHERE THE FILE IS STORED
    worddoc.SaveAs(fd.SelectedPath & "\tags" & PrintID.Text & ".docx")
End If
'THIS PRINTS THE WORD DOCUMENT TO THE DEFAULT PRINTER
worddoc.PrintOut()
Dim savechanges As Object = False
'FINALLY CHANGES ARE SAVED AND THE DOCUMENTS ARE CLOSED
worddoc.Close(savechanges)
wordapp.Quit(savechanges)
MessageBox.Show("Tags saved and being printed")
End Sub
'THE BUTTON USED TO PRINT AND STORE INFO TO WORD
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
PRINTNSAVE.Click
'CALLS UPON THE SUBROUTINE WHICH PRINTS AND SAVES TAGS WHEN BUTTON IS PRESSED
    AddtoWord()
End Sub
End Class

```

EmployeeManagement.vb

If you are an admin you will have access to employee management. The employee management menu will allow you to search for and view employee information. The menu will also have options to Add, remove or update employee information.

ID	FirstName	LastName	HoursperWeek	MedicalInfo
19	Adam	Khanzada	5	N/A
29	z	z	z	z
32	a	a	a	a
37	b	b	b	b

```
'CLASS REFFERING TO THE FORM IN CHARGE OF DISAPLAYING EMPLOYEE MANAGEMENT OPTIONS
Public Class EmployeeManagement
    'THE VARIBALE DB IS BEING USED TO DRAW ARGUEMNETS FROM THE CLASS DBCONTROL
    Dim access As New DBControl
    'VARIABLE BEING USED TO SORT THE ODER OF INFORMATION DISPLAYED
    Dim EmployeeSort As String
    'SUB FOR THE REFRESH BUTTON
    Private Sub EmployeeManagement_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        RefreshEmployee()
        'SORTS THE EMPLOYEES OUT BY FIRST NAME
        EmployeeSort = "FirstName"
    End Sub
    'SUB FOR THE BUTTON TO SEARCH FOR EMPLOYEES
    Private Sub EmployeeSearchBtn_Click(sender As Object, e As EventArgs) Handles
EmployeeSearchBtn.Click
        Dim itemRow As String
        'STIPULATING THE ODER THE SEARCHED DATA IS VIEWED IN
        If EmployeeInfoTB.Text = "EmployeeInfo" Then
            itemRow = "FirstName"
            EmployeeSort = "FirstName"
        End If
        'QUERY IS USED TO BEING UP ALL INFORMATION ON THE EMPLOYEE BY THE FIRST NAME
        SEARCHED IN THE TEXT BOX
        access.ExecQuery("SELECT * FROM " & EmployeeInfoTB.Text & " WHERE " & itemRow
& " = '" & EmployeeSearchTB.Text & "' ORDER BY " & EmployeeSort & " ASC")
        If Not String.IsNullOrEmpty(access.exception) Then MsgBox(access.exception) :
Exit Sub
End Sub
```

```

        'THE DATA FOUND IS DISPLAYED IN THE DATA GRID VIEW ON THE FORM
        DgvEmployee.DataSource = access.DBDT
    End Sub
    'SUB USED TO REFRESH EMPLOYEE INFORMATION ONCE CHANGES HAVE BEEN MADE
    Sub RefreshEmployee()
        'QUERY SIMPLY RELOADS THE DATA ON EMPLOYEES IN THE DATA GRID VIEW
        access.ExecQuery("SELECT * FROM EmployeeInfo ORDER BY ID ASC")
        If Not String.IsNullOrEmpty(access.exception) Then MsgBox(access.exception) :
Exit Sub
        'DATA IS IVIEWD IN DATA GRID VIEW
        DgvEmployee.DataSource = access.DBDT
    End Sub
    Private Sub AddEmployeeBtn_Click(sender As Object, e As EventArgs) Handles
AddEmployeeBtn.Click
        'DISPLAYS THE FORM FOR ADDING NEW EMPLOYEES
        AddEmployee.Show()
    End Sub

    Private Sub DeleteEmployeeBtn_Click(sender As Object, e As EventArgs) Handles
DeleteEmployeeBtn.Click
        'DISPLAYS THE FORM FOR DELETING EMPLOYEES
        DeleteEmployee.Show()
    End Sub

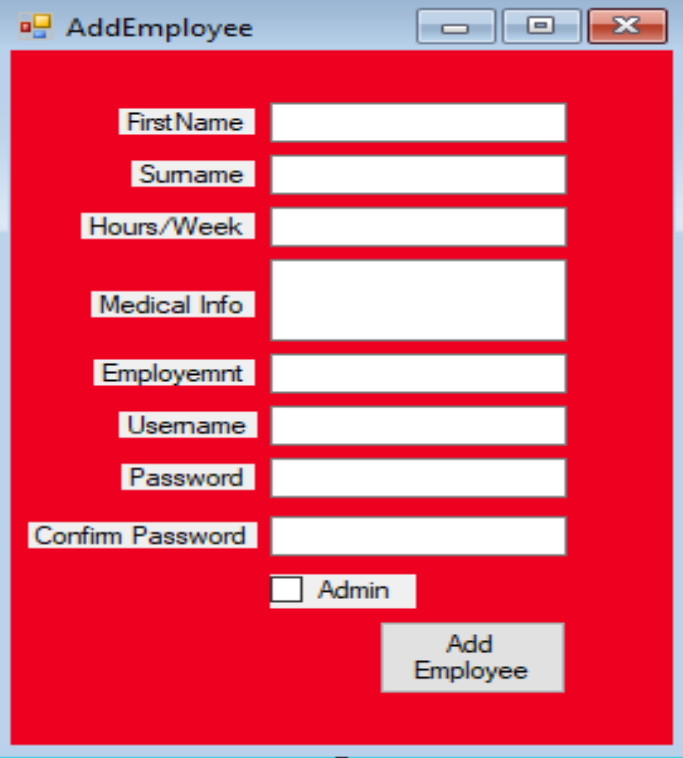
    Private Sub Refresh_Click(sender As Object, e As EventArgs) Handles Refresh.Click
        'UPON PRESSING BUTTON THE REFRESH EMPLOYEE INFO SUB IS CALLED UPON
        RefreshEmployee()
    End Sub

    Private Sub UpdateEmployeeBtn_Click(sender As Object, e As EventArgs) Handles
UpdateEmployeeBtn.Click
        'DISPLAYS THE FORM IN CHARGE OF UPDATING EMPLOYEE INFO ON BUTTON CLICK
        EmployeeUpdate.Show()
    End Sub
End Class

```

AddEmployee.vb

When adding a new employee, you will be taken to the following form. This will allow you to enter necessary employee information as requested by the manager. When the button is pressed the password entered is hashed and then an INSERT statement is used to store the data in the employee management table



```
'CLASS WHICH REFERS TO THE FORM THAT ADDS EMPLOYEES
Public Class AddEmployee
    'THE VARIABLE DB IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Dim db As New DBControl
    'THE VARIABLE CHECK IS A BOOLEAN THAT ALLOWS US TO CHECK IF USERS ARE ADMINS
    Dim check As Boolean
    'SUB USED TO INSERT INFORMATION TYPED INTO THE EMPLOYEE MANAGEMENT TABLE
    Sub AddEmployee()
        'PASSWORDS THAT ARE ADDED NEED TO BE ENCRYPTED
        Dim password As String = New PasswordHashing().Encrypt(AddEmpPasswordC.Text)
        'PARAMETERISING ARGUMENTS
        db.AddParameter("@first", AddEmpFirst.Text)
        db.AddParameter("@last", AddEmpLast.Text)
        db.AddParameter("@hours", AddEmpHours.Text)
        db.AddParameter("@medical", AddEmpMed.Text)
        db.AddParameter("@Employment", AddEmpEmp.Text)
        db.AddParameter("@user", AddEmpUser.Text)
        db.AddParameter("@password", password)
        db.AddParameter("@admin", AddEmpAdminCheck.CheckState)
```

```

        'THE QUERY BELOW ADDS INFO ON EMPLOYEES BASED UPON WHATS WRITTEN IN THE
SELECTED TEXTBOXES
        db.ExecQuery("INSERT INTO EmployeeInfo
([FirstName],[LastName],[HoursperWeek],[MedicalInfo],[Employment],[Username],[Password
],[Admin]) Values(@first,@last,@hours,@medical,@Employment,@user,@password,@admin)")
        Me.Close()
    End Sub
    'SUB FOR THE BUTTON WHICH ADDS THE EMPLOYEE INFO TO THE APPROPRIATE TABLE
    Private Sub AddEmpBtn_Click(sender As Object, e As EventArgs) Handles
AddEmpBtn.Click
        'ERROR HANDLING AS ALL INFO MUST BE FILLED OUT
        If ValidatAllTxtbx() Then
            'THE PASSWORD AND CONFIRMATION NEED TO MATCH UP
            If AddEmpPassword.Text = AddEmpPasswordC.Text Then
                'CALLS UPON THE SUB TO ADD EMPLOYEE INFO
                AddEmployee()
                MessageBox.Show("Employee Added")
            Else
                MessageBox.Show("Please make sure your passwords match")
            End If
        Else
            MessageBox.Show("Please fill out all fields")
        End If
    End Sub
    'FUNCTION USED TO MAKE SURE ALL INFO IS FILLED OUT
    Public Function ValidatAllTxtbx()
        Dim ctrl As Control
        Dim result As Boolean
        'RESULT IS SET TO FALSE IF A TEXTBOX IS FOUND TO BE EMPTY
        result = True
        For Each ctrl In Me.Controls
            If TypeOf ctrl Is TextBox Then
                'USES LENGTH OF TEXT TO CHECK IF THE TEXTBOX IS EMPTY
                If Len(ctrl.Text) = 0 Then
                    result = False
                    Exit For
                End If
            End If
        Next ctrl
        ValidatAllTxtbx = result
    End Function
End Class

```


DeleteEmployee.vb

An employee can be selected in the menu and his information will be taken to the delete form. An employees info can be viewed and deleted upon the button click. This is done by the use of a DELETE query. Several uses of error handling are present as an employee cannot delete himself and the minimum number of admins on the system must be kept to at least 1.

```
Public Class DeleteEmployee
    'THE VARIABLE DB IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Dim db As New DBControl
    'SUB WHICH PRELOADS INFO ALONGSIDE THE FORM FOR DELETING EMPLOYEES
    Private Sub DeleteEmployee_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'A ROW NEEDS TO BE SELECTED FOR AN EMPLOYEE TO BE DELETED
        If EmployeeManagement.DgvEmployee.SelectedRows.Count > 0 Then
            'PRELOADS HIGHLIGHTED INFORMATION INTO TEXTBOXES FROM THE DATA GRID VIEW
            DeleteEmpSelected.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(1).Value.ToString & " " &
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(2).Value.ToString
            DeleteEmpID.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(0).Value.ToString
            DeleteEmpHours.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(3).Value.ToString
            DeleteEmpMed.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(4).Value.ToString
            DeleteEmpEmp.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(5).Value.ToString
            DeleteEmpUser.Text =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(6).Value.ToString
            DeleteEmpCheck.Checked =
                EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(8).Value
            'CHECKING IF THE USER YOU ARE TRYING TO DELETE IS YOURSELF
            If DeleteEmpID.Text = Login.currentUserID Then
                'ERROR HANDLING AS YOU CANT DELETE YOURSELF
            End If
        End If
    End Sub
End Class
```

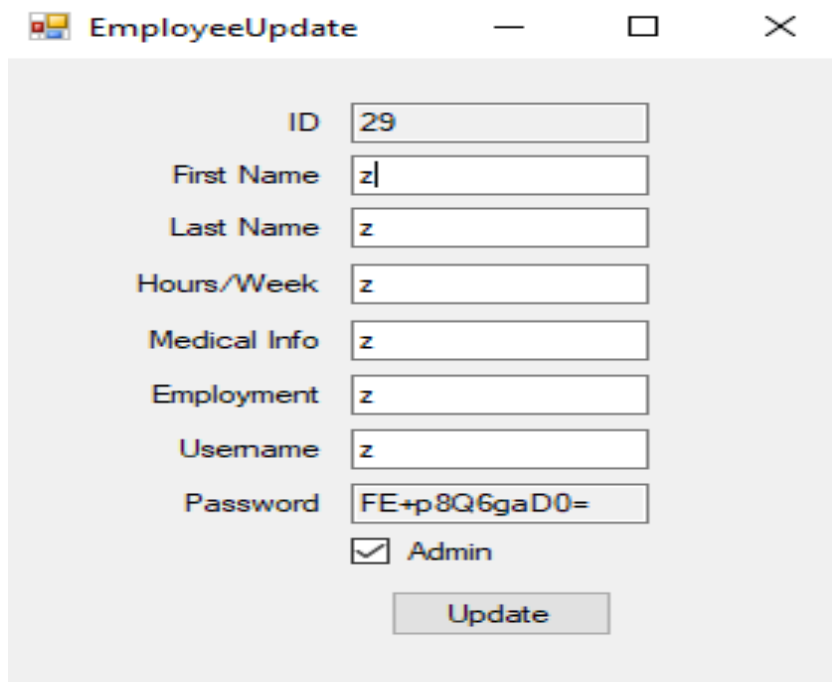
```

        MessageBox.Show("You cannot delete yourself, you silly goat")
        Me.Close()
    End If
Else
    MessageBox.Show("you must select a row first")
    Me.Close()
End If
End Sub
'SUB USED TO DELETE EMPLOYEE FROM THE TABLE
Sub DeleteEmp()
    'THE QUERY DELETES THE EMPLOYEE FROM THE TABLE WHERE THE USERNAME IS THE ONE
    IN THE TEXTBOX
    db.ExecQuery("DELETE * FROM [EmployeeInfo] WHERE Username = '" &
DeleteEmpUser.Text & "'")
End Sub
'THE SUB USED TO DESCRIBE PROCESSES OCCURING WHEN YOU CLICK IT
Private Sub DeleteEmpBtn_Click(sender As Object, e As EventArgs) Handles
DeleteEmpBtn.Click
    'ERROR HANDLING TO MAKE SURE THERES ALWAYS ATLEAST ON ADMIN
    If New DBControl().AdminCount() = False And DeleteEmpCheck.Checked = True Then
        MsgBox("only 1 admin in system")
    Else
        'THE SUB TO DELETE THE SELECTED EMPLOYEE IS BEING CALLED UPON
        DeleteEmp()
        MessageBox.Show("Employee Deleted")
        Me.Close()
    End If
End Sub
End Class

```

EmployeeUpdate.vb

When an employee is selected in the menu his information can be updated in the following form. The employee's information is drawn from the tables in the database and displayed here. The user can then re-enter any changes to the info and click the button to apply the changes. This is done through the use of an UPDATE query



```
Public Class EmployeeUpdate
    'THE VARIABLE ACCESS IS BEING USED TO DRAW ARGUMENTS FROM THE CLASS DBCONTROL
    Dim access As New DBControl
    'RECURRING FUNCTION USED TO CHECK IF ALL FIELDS HAVE BEEN FILLED OUT
    Public Function ValidatAllTxtbx()
        'VARIABLES USED TO RETURN A RESULT
        Dim ctrl As Control
        Dim result As Boolean
        'RESULT IS CHANGED TO FALSE IF THE TEST FAILS
        result = True
        For Each ctrl In Me.Controls
            If TypeOf ctrl Is TextBox Then
                'CHECK IF THE LENGTH OF STRING IN THE TEXTBOX IS ZERO
                'IF THE LENGTH IS ZERO THE TEXTBOX IS EMPTY
                If Len(ctrl.Text) = 0 Then
                    result = False
                    'AT THE FIRST OCCURRENCE OF A FAILURE THE RESULT CHANGES
                    Exit For
                End If
            End If
        Next ctrl
        'RETURNS A RESULT
        ValidatAllTxtbx = result
    End Function
End Class
```

```

End Function
'SUB USED TO PRELOAD DATA ALONGSIDE THE FORM
Private Sub EmployeeUpdate_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    'MAKES SURE A MEMEBER OF STAFF HAS BEEN SELECTED
    If EmployeeManagement.DgvEmployee.SelectedRows.Count > 0 Then
        'BASED ON THE ROW SELECTED THE EMPLOYEE INFO IS FILLED INTO THE
APPROPRIATE TEXTBOXES
        EmpUpID.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(0).Value.ToString
        EmpUpFName.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(1).Value.ToString
        EmpUpLName.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(2).Value.ToString
        EmpUpHours.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(3).Value.ToString
        EmpUpMeds.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(4).Value.ToString
        EmpUpEmp.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(5).Value.ToString
        EmpUpUser.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(6).Value.ToString
        EmpUpPass.Text =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(7).Value.ToString
        EmpUpAdmin.Checked =
EmployeeManagement.DgvEmployee.SelectedRows(0).Cells(8).Value
    End If
End Sub
'SUB FOR UPDATING THE CHANGED EMPLOYEE INFORMATION
Sub UpdateEmp()
    Try
        'PARAMTERISATION OF EMPLOYEE ARGUEMENTS
        access.AddParameter("@FirstName", EmpUpFName.Text)
        access.AddParameter("@LastName", EmpUpLName.Text)
        access.AddParameter("@HoursperWeek", EmpUpHours.Text)
        access.AddParameter("@MedicalInfo", EmpUpMeds.Text)
        access.AddParameter("@Employment", EmpUpEmp.Text)
        access.AddParameter("@Username", EmpUpUser.Text)
        access.AddParameter("@Password", EmpUpPass.Text)
        access.AddParameter("@Admin", EmpUpAdmin.Checked)
        access.AddParameter("@ID", EmpUpID.Text)
        'THE QUERY UPDATES THE EMPLOYEE INFORMATION SO THAT THE NEWLY WRITTEN INFO
IN THE TEXBOXES ARE UPDATED TO THE DATABASE FO RTHE USER WHERE THE ID MATCHES
        access.ExecuteNonQuery("UPDATE [EmployeeInfo] SET
[FirstName]=@FirstName,[LastName]=@LastName,[HoursperWeek]=@HoursperWeek,[MedicalInfo]
=@MedicalInfo,[Employment]=@Employment,[Username]=@Username,[Password]=@Password,[Admi
n]=@Admin WHERE ID=@ID")
        'CONFIRMATION MESSAGE
        MessageBox.Show("Employee details updated")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub
'SUB USED TO HOLD THE PROCCESSES WHEN THE BUTTON HAS BEEN CLICKED
Private Sub EmpUpBtn_Click(sender As Object, e As EventArgs) Handles
EmpUpBtn.Click
    'RMAKES SURE ALL FILEDS ARE FILLED OUT
    If ValidatAllTxtbx() Then
        'CALLS UPON SUB TO UPDATE EMPLOYEE INFO
        UpdateEmp()
    Else
        'ERROR MESSAGE

```

```
        MessageBox.Show("Please make sure all fields are filled in")
    End If
End Sub
End Class
```

Testing

In the testing phase of my project I will be checking for any potential errors in my software as well as testing for whether I have successfully met all of my objectives and met the specific requirements of the Manager and volunteering staff.

Testing Overview

Test Name	Test Procedure	Expected Result	Actual Result
Test 1	Login with correct details as a regular member of staff	Taken to the main menu screen without an employee management button	Success
Test 2	Login with correct details as an admin user	Taken to menu with an employee management button	Success
Test 3	Login with incorrect details	Error message eluding to incorrect details	Success
Test 4	Be able to add an Item to stock for any item type	Add an Item and have it appear in Items General and its type specific table	Success
Test 5	Be able to delete an Item from stock for any item type. (changing quantities)	Select an item and alter its quantity, if quantity goes to zero the Item should be completely removed.	Success
Test 5.1	When quantity of an Item stock goes to zero it should be removed from the database entirely	Selected Item has been deleted from the general table alongside the appropriate type table.	Success
Test 6	Be able to update an Items info for any Item type	Change any selected Items information	Success
Test 7	Be able to search for an Item by name	Item appears if name is typed correctly	Success

Test 7.1	Search for an Item that does not exist in the database by Name	No information should be returned as the name entered cannot be matched to any known in the tables	Success
Test 8	Add an employee to employee management (regular)	New employee should appear in employee table	Success
Test 8.1	Add an employee to employee management (Admin	New employee should appear in employee table with an admin tick	Success
Test 9	Update employee information	Be able to change a selected employees information	Success
Test 10	Remove an employee from the system	Removes a selected employee	Success
Test 10.1	Total number of Admin accounts should not be able to drop below 1.	Error message when Admin count tries to drop below 1	Success
Test 10.2	An employee should not be able to delete himself when logged in. We can test this by trying to delete ourselves as an admin account	An error message should appear stopping you from deleting yourself	Success
Test 11	Attempt to save and print an item tag for the item selected in the menu	Tag is saved in a word document in specified location by the user and user can also print the newly created tags	Success


All my various tests are being conducted to check on my programs ability to meet its objects stipulated in the Analysis section of the project as well as ensure error handling is functioning effectively to prevent fatal errors/crashing.

Test 1

From Test 1 we are trying to validate the success of our login and our admin privileges system as you should be able to login with correct regular user details and be taken to the menu without an employee management button.

Test Username: Test1 (Typical data)

Test Password: Test1 (Typical data)



Result:



	ItemGeneralID	ItemType	ItemName	ItemPrice	ItemC
▶	58	Clothing	dad	\$100	231
	55	Toy	fd	20	23
	51	DVDorCD	Jurassic World: D...	\$9.99	4
	54	DVDorCD	Reality and Retri...	\$5.00	1
	59	Clothing	TEST2	\$12.00	17
	61	DVDorCD	Testing fields	\$12	10
	52	Clothing	testings	213	1

As we can see the test was a success as the employee management button is not present

Test 2

From Test2 we are checking to see if you receive Admin privileges by typing in correct admin information. This will be seen by a successful login taking you to the menu and the appearance of an employee management button

Test Username: Test2 (Typical data)

Test Password: Test2 (Typical data)



Result:



As we can see the test was a success as the employee management button is present

Test 3

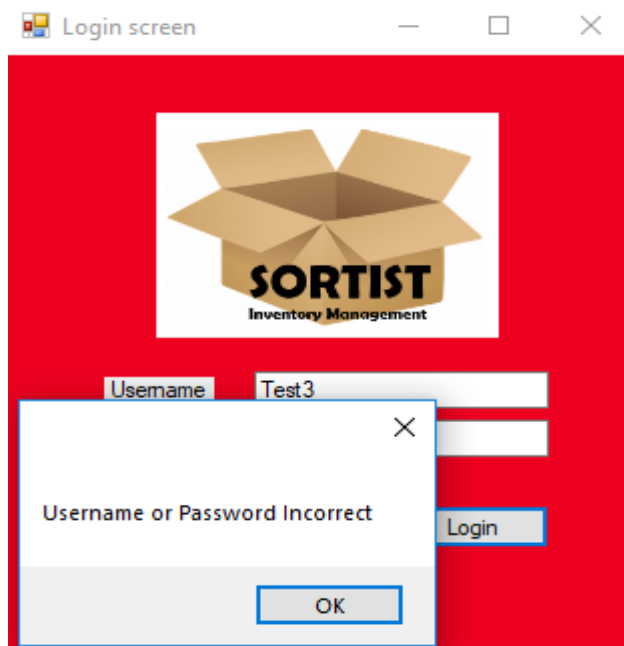
From Test3 we are hoping to see that an error message pops up when incorrect employee details are entered, this is to make sure random customers can't access this information

Test Username: Test3 (erroneous data)

Test Password: Test3 (erroneous data)



Result:



As we can see the test was a success as a random username and password does not admit access to the Inventory Manager.

Test 4

This test will be making sure that a user can add any Item from the different types into the table relating to its type and the general items table.

Test Item Name: Test4 (typical data)

Test Type: Clothing (typical data)

The 'InsertForm' window displays the following fields:

- Item Type: Clothing (dropdown)
- ItemName: Test4
- ItemPrice: \$50.00
- ItemCondition: New
- ItemQuantity: 10
- Clothing Type: Shoes
- ClothingBrand: Nike
- ClothingSize: 9
- Gender: Male

An 'Add' button is visible. A confirmation dialog box titled 'NEA Refined' is open, showing 'item added' and an 'OK' button.

Results:

The 'Menu' window displays a table of items with the following data:

ItemGeneralID	Item Type	ItemName	ItemPrice	ItemC
55	Toy	fd	20	23
51	DVDorCD	Jurassic World: D...	\$9.99	4
54	DVDorCD	Reality and Retri...	\$5.00	1
59	Clothing	TEST2	\$12.00	17
62	Clothing	Test4	\$50.00	10
61	DVDorCD	Testing fields	\$12	10
52	Clothing	testings	213	1

The table has a search bar at the bottom: 'Search for an item by name below'. A sidebar on the right contains buttons: 'Add Item', 'Delete Item', 'Update Item/info', 'Print Tag', 'Employee Management', and 'Search'. A footer message states: 'Please make sure to highlight an item before using the options Delete, Update or Print.'

ItemGeneralID	ClothingStockID	ClothingType	ClothingSize	ClothingBrand
52	20	Jacket	L	Louis Vuitton
58	22	zsf	I	asdfsdf
59	23	Shoes	L	Gucci
62	26	Shoes	9	Nike

Search for an item by name below

Please make sure to highlight an item before using the options Delete, Update or Print

The test was a success as the new Item was added to both tables with correct info

Test 5

This test is to make sure that when an item is selected you can remove or delete stock from it and if quantity reaches zero the item is completely gone from the tables.

Test Item Name: Test5 (typical data)

Test Quantity: 15 (typical integer data)

New Test Quantity: 5 (typical integer data)

Item Selected: Test5

Current Quantity: 15

New Quantity: 5

Buttons: Cancel, Remove/Delete

Results:

Items Updated. 10 items removed

OK

Menu

ItemsGeneral

	Item Type	ItemName	ItemPrice	ItemQuantity	Item
	DVDorCD	Reality and Retri...	\$5.00	1	New
	Clothing	TEST2	\$12.00	17	TES
	Clothing	Test4	\$50.00	10	New
▶	Toy	Test5	\$12.00	5	New
	DVDorCD	Testing fields	\$12	10	New
	Clothing	testings	213	1	Test
	Toy	Thomas The Tan...	\$3.00	3	Use

Search for an item by name below

Please make sure to highlight an item before using the options Delete, Update or Print

Add Item
 Delete Item
 Update Item/Info
 Print Tag
 Employee Management
 Search

The test was as success as the quantity changed from 15 to 5 as intended.

Test 5.1

This test is a continuation of the previous test in which we will take our Item with its newly quantity of 5 and change it to 0. The change to zero should remove the item from the database entirely.

Test Item Name: Test5 (typical data)

Test Quantity: 5 (typical integer data)

New Test Quantity: 0 (typical integer data)

DeleteForm

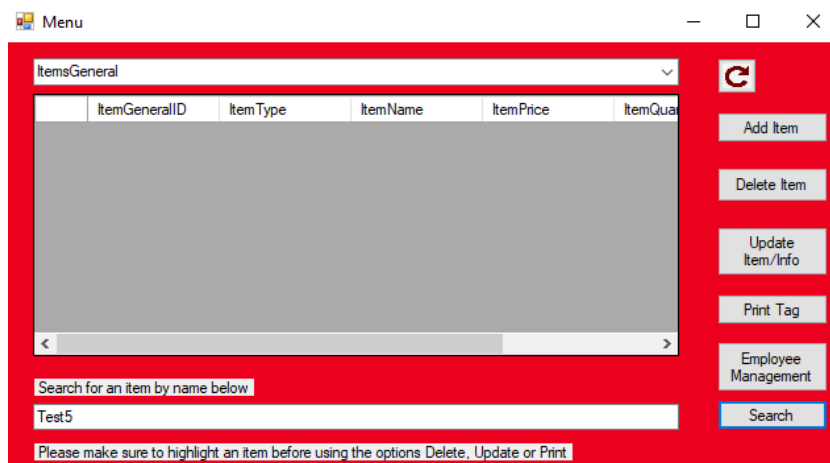
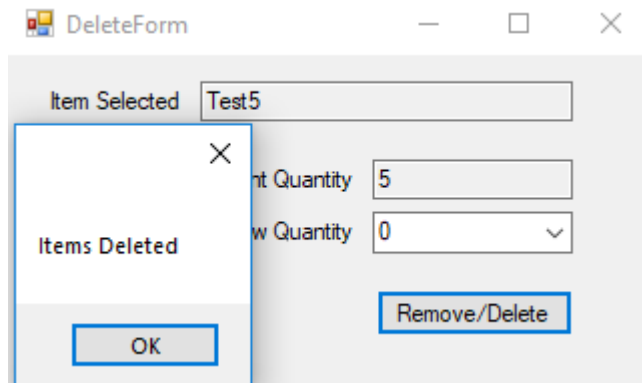
Item Selected: Test5

Current Quantity: 5

New Quantity: 0

Cancel
 Remove/Delete

Results:



The test was a success as the search for an Item with the name Test5 returns nothing.

Test 6

This test is carried out to make sure that an Items information can be changed once added to the database using the ItemUpdate form.

Test Item Name: Test6 (typical data)

Test Item Price: \$5.00 (typical data)

Test Item New Name: Success (typical data)

Test Item New Price: \$10.00 (typical data)

ItemUpdate

General ID: 65

Item Type: Toy

Item Name: Success

Item Price: \$10.00

Item Quantity: 12

Item Condition: New

ToyID: 37

Toy Make: Windu

Age Recommendation: 5+

Toy Type: Car

Safety Notice: N/A

Update

Result:

Menu

ItemsGeneral

	ItemGeneralID	Item Type	ItemName	ItemPrice	ItemC
	64	Toy	dff	\$12	100
	55	Toy	fd	20	23
	51	DVDorCD	Jurassic World: D...	\$9.99	4
	54	DVDorCD	Reality and Retri...	\$5.00	1
▶	65	Toy	Success	\$10.00	12
	59	Clothing	TEST2	\$12.00	17
	62	Clothing	Test4	\$50.00	10

Search for an item by name below

Please make sure to highlight an item before using the options Delete, Update or Print

Add Item

Delete Item

Update Item/Info

Print Tag

Search

ItemUpdate

General ID: 65

Item Type: Toy

Item Name: Success

Item Price: \$10.00

Item Quantity: 12

Item Condition: New

ToyID: 37

Toy Make: Windu

Age Recommendation: 5+

Toy Type: Car

Safety Notice: N/A

Update Successful

OK

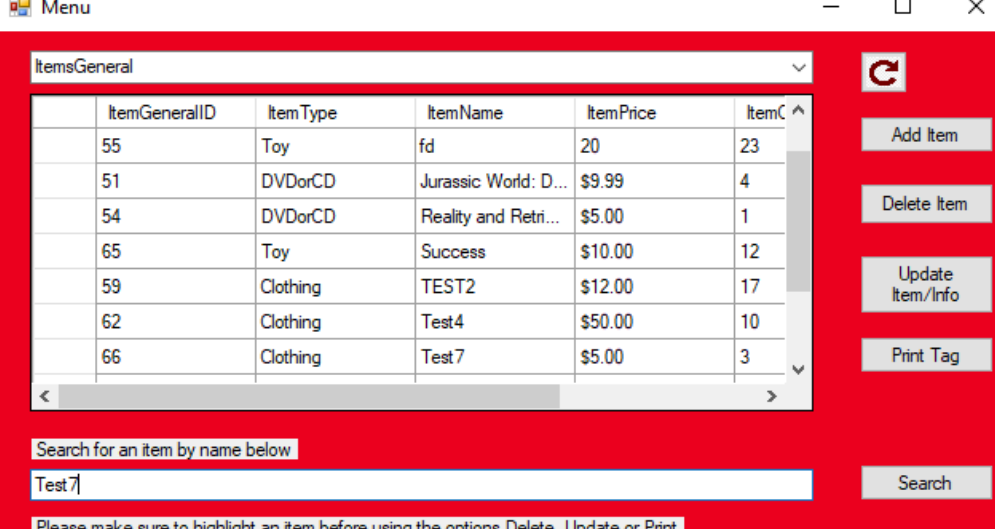
The test was a success as the search for an Item with the name 'Success' returns a toy with the updated information as the price has changed to \$10 and the item was renamed to success.

Test 7

This test is being carried out to ensure that an Item can be searched for by name in the menu and that the relevant Item information is displayed when the name matches with one from the database.

Test Item Name: Test7 (typical data)

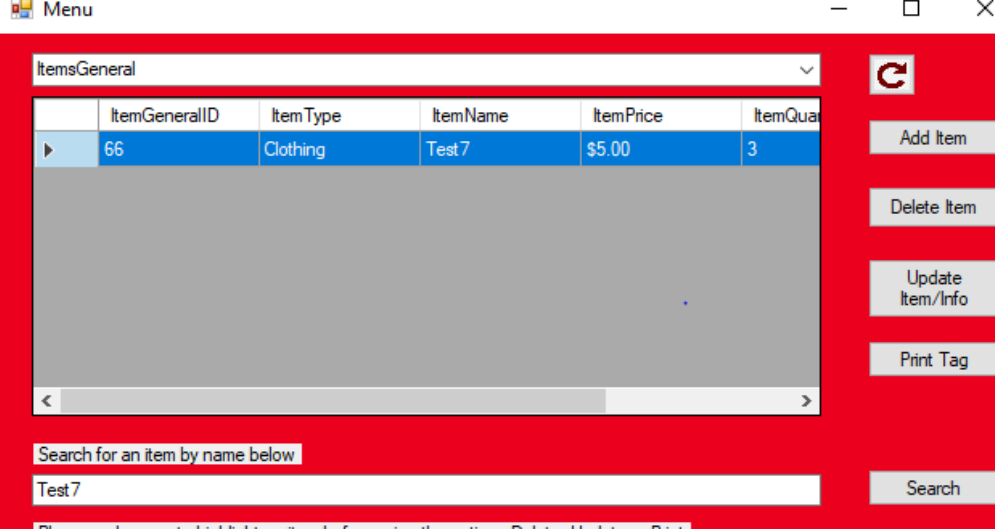
Test Item Name Searched: Test7 (typical data)



The screenshot shows a window titled 'Menu' with a table of items. The table has columns: ItemGeneralID, Item Type, ItemName, ItemPrice, and ItemC. The item 'Test7' is highlighted in the table. To the right of the table are buttons: Add Item, Delete Item, Update Item/Info, and Print Tag. Below the table is a search bar with the text 'Test7' and a 'Search' button. A message at the bottom says 'Please make sure to highlight an item before using the options Delete, Update or Print'.

	ItemGeneralID	Item Type	ItemName	ItemPrice	ItemC
	55	Toy	fd	20	23
	51	DVDorCD	Jurassic World: D...	\$9.99	4
	54	DVDorCD	Reality and Retri...	\$5.00	1
	65	Toy	Success	\$10.00	12
	59	Clothing	TEST2	\$12.00	17
	62	Clothing	Test4	\$50.00	10
	66	Clothing	Test7	\$5.00	3

Result:



The screenshot shows the same 'Menu' window, but now the item 'Test7' is highlighted in the table. The search bar still contains 'Test7' and the 'Search' button is visible. The message at the bottom remains the same.

	ItemGeneralID	Item Type	ItemName	ItemPrice	ItemQua
▶	66	Clothing	Test7	\$5.00	3

The test was successful as the search for an Item with a name 'Test7' returned relevant information on the piece of clothing.

Test 7.1

This test is done to make sure that nothing is returned from the database when an Item that does not exist is searched for.

Test Item Searched Name: Test7.1 (erroneous data)

The screenshot shows a window titled 'Menu' with a red background. At the top, there is a dropdown menu labeled 'ItemsGeneral'. Below it is a table with the following columns: ItemGeneralID, Item Type, ItemName, ItemPrice, and ItemC. The table contains several rows of data, including items like 'Toy', 'DVDorCD', and 'Clothing'. To the right of the table are buttons for 'Add Item', 'Delete Item', 'Update Item/Info', and 'Print Tag'. Below the table is a search bar with the text 'Test 7.1' entered. A message at the bottom says 'Please make sure to highlight an item before using the options Delete, Update or Print'.

ItemGeneralID	Item Type	ItemName	ItemPrice	ItemC
55	Toy	fd	20	23
51	DVDorCD	Jurassic World: D...	\$9.99	4
54	DVDorCD	Reality and Retri...	\$5.00	1
65	Toy	Success	\$10.00	12
59	Clothing	TEST2	\$12.00	17
62	Clothing	Test4	\$50.00	10
66	Clothing	Test7	\$5.00	3

Result:

The screenshot shows the same 'Menu' application window. The search bar now contains 'Test 7.1'. The table below it is empty, indicating that no items were found matching the search criteria. The buttons and messages remain the same as in the previous screenshot.

The test was a success as no Item information has been returned by the search as no item in the database had a name matching to the one searched for.

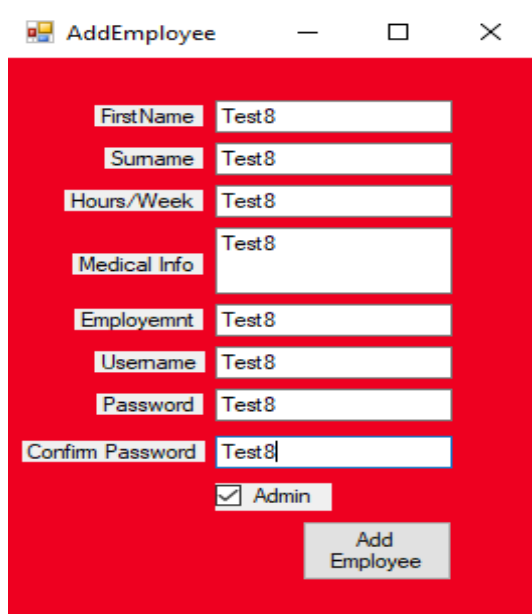
Test 8

This test is used to make sure that an Admin user can add a new Employee into the Employee management system. The new user will be an Admin account

Test Employee First Name: Test8 (typical data)

Test Employee Last Name: Test8 (typical data)

Test Admin Authorities: Checked (typical data)



First Name: Test8

Surname: Test8

Hours/Week: Test8

Medical Info: Test8

Employmnt: Test8

Username: Test8

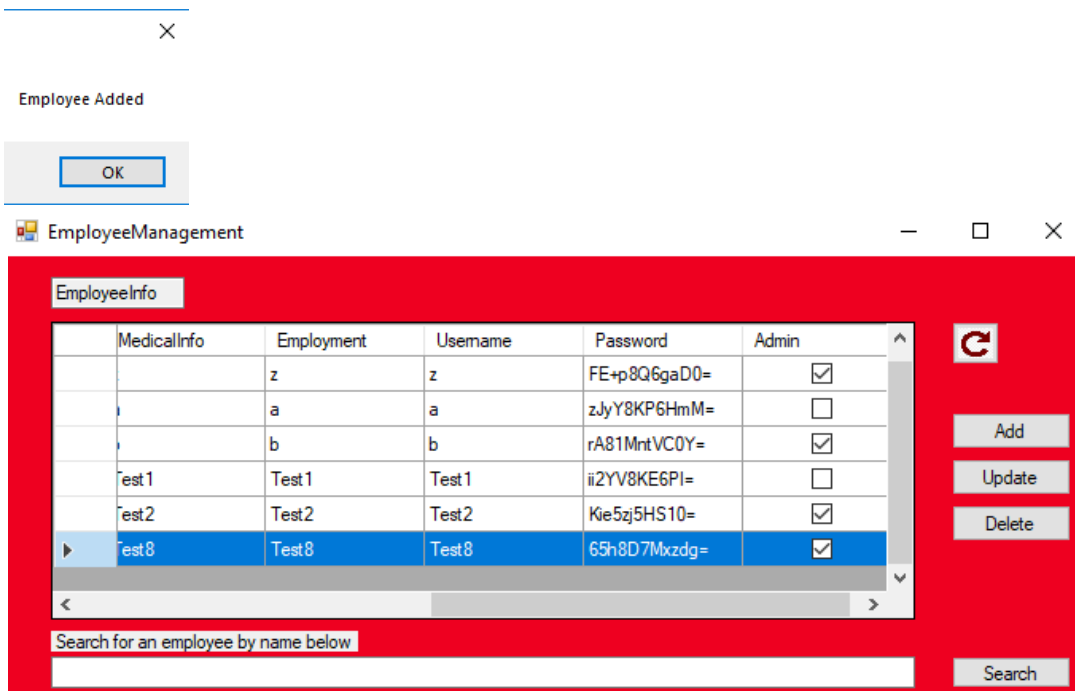
Password: Test8

Confirm Password: Test8

☒ Admin

Add Employee

Results:



Employee Added

OK

EmployeeManagement

MedicalInfo	Employment	Username	Password	Admin
	z	z	FE+p8Q6gaD0=	<input checked="" type="checkbox"/>
	a	a	zJyY8KP6HmM=	<input type="checkbox"/>
	b	b	rA81MntVC0Y=	<input checked="" type="checkbox"/>
est1	Test1	Test1	ii2YV8KE6PI=	<input type="checkbox"/>
est2	Test2	Test2	Kie5zj5HS10=	<input checked="" type="checkbox"/>
est8	Test8	Test8	65h8D7Mxzdg=	<input checked="" type="checkbox"/>

Search for an employee by name below

Search

Add

Update

Delete

The test was successful as an employee was added with admin privileges ticked.

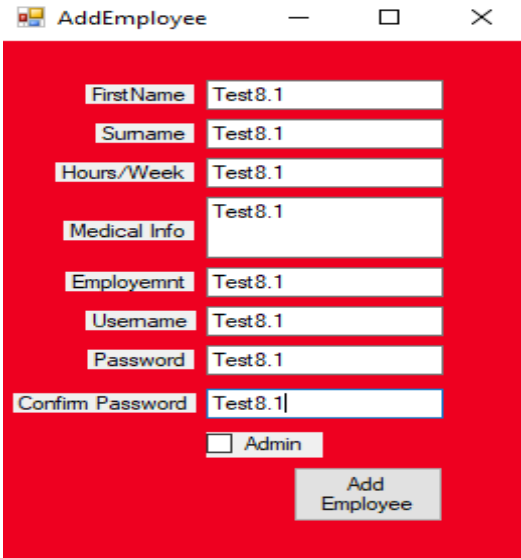
Test 8.1

This test is used to make sure that an Admin user is able to add a new Employee into the Employee management system. The new user will be an Regular staff account with no admin authorities

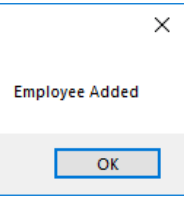
Test Employee First Name: Test8.1 (typical data)

Test Employee Last Name: Test8.1 (typical data)

Test Admin Authorities: Unchecked (typical data)

A screenshot of a web application window titled "AddEmployee". The window has a red background. It contains several text input fields with labels: "FirstName", "Surname", "Hours/Week", "Medical Info", "Employemnt", "Username", "Password", and "Confirm Password". All these fields contain the text "Test8.1". Below the "Confirm Password" field, there is a checkbox labeled "Admin" which is unchecked. At the bottom right of the form, there is a grey button labeled "Add Employee".

Results:

A screenshot of a small dialog box with a white background. It has a title bar with a close button (X). The text "Employee Added" is displayed in the center. At the bottom, there is a grey button labeled "OK".

EmployeeManagement

EmployeeInfo

	MedicalInfo	Employment	Username	Password	Admin
	a	a	a	zJyY8KP6HmM=	<input type="checkbox"/>
	b	b	b	rA81MntVC0Y=	<input checked="" type="checkbox"/>
	Test1	Test1	Test1	ii2YV8KE6Pl=	<input type="checkbox"/>
	Test2	Test2	Test2	Kie5zj5HS10=	<input checked="" type="checkbox"/>
	Test8	Test8	Test8	65h8D7Mxzdg=	<input checked="" type="checkbox"/>
▶	Test8.1	Test8.1	Test8.1	hOtqKrvx6no=	<input type="checkbox"/>

Search for an employee by name below

Search

Add

Update

Delete

The test was successful as the new employee is not an admin as the box is unchecked

Test 9

This test will be carried out to make sure that an Admin member of staff is able to update and make changes to existing employee information.

Test Employee First Name: Test9 (typical data)

Test Employee Hours per week: 10 (typical data)

Test New Employee Hours per week: 20 (typical data)

Test Employee Medical Info: Hay fever (typical data)

Test New Employee Medical Info: Not applicable (typical data)

EmployeeUpdate

ID: 42

First Name: Test9

Last Name: Test9

Hours/Week: 10

Medical Info: Hayfever

Employment: Test

Username: Test9

Password: UH3GgNvC/ws=

☐ Admin

Update

Result:

EmployeeUpdate

ID: 42

First Name: Test9

Last Name: Test9

Hours/Week: 20

Medical Info: Hayfever

Employment: Test

EmployeeManagement

EmployeeInfo

	ID	First Name	Last Name	HoursperWeek	MedicalInfo
	37	b	b	b	b
	38	Test1	Test1	5	Test1
	39	Test2	Test2	5	Test2
	40	Test8	Test8	Test8	Test8

Search

Add

Update

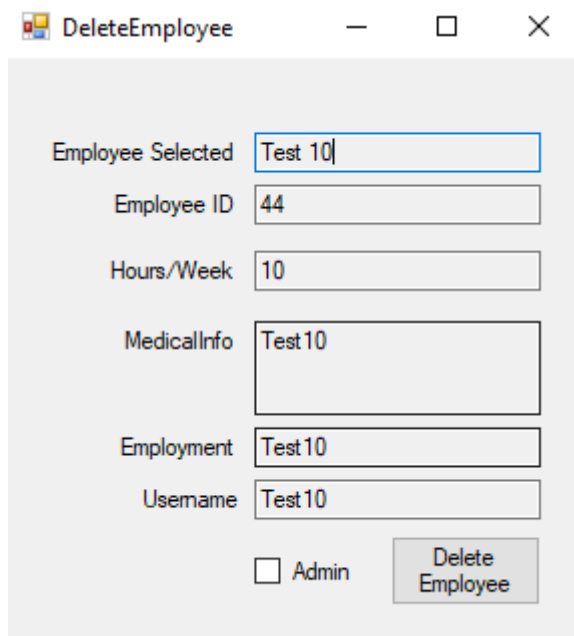
The test was successful as the employees medical info and hours were changed from the previous data.

Test 10

This test is being conducted to see whether an Admin member of staff can remove/delete other users from the employee database.

Test Employee First Name: Test10 (typical data)

The following employee will be deleted from the database (typical data)



The screenshot shows a window titled "DeleteEmployee" with a standard Windows title bar (minimize, maximize, close buttons). The window contains several text input fields and a checkbox. The fields are labeled "Employee Selected", "Employee ID", "Hours/Week", "MedicalInfo", "Employment", and "Username". Each field contains the text "Test 10". Below the fields is a checkbox labeled "Admin" which is unchecked. To the right of the checkbox is a button labeled "Delete Employee".

Result:



The screenshot shows a window titled "EmployeeManagement" with a standard Windows title bar. The window has a red background. At the top left, there is a tab labeled "EmployeeInfo". Below the tab is a table with the following columns: "ID", "FirstName", "LastName", "HoursperWeek", and "MedicalInfo". The table is currently empty. To the right of the table is a large grey rectangular area. Below the table is a search bar with the text "Search for an employee by name below" and a "Search" button. The search bar contains the text "Test10". To the right of the search bar are three buttons: "Add", "Update", and "Delete".

The test was successful as the search for an employee with the name 'Test10', did not return any data from the table therefore the user must have been successfully deleted

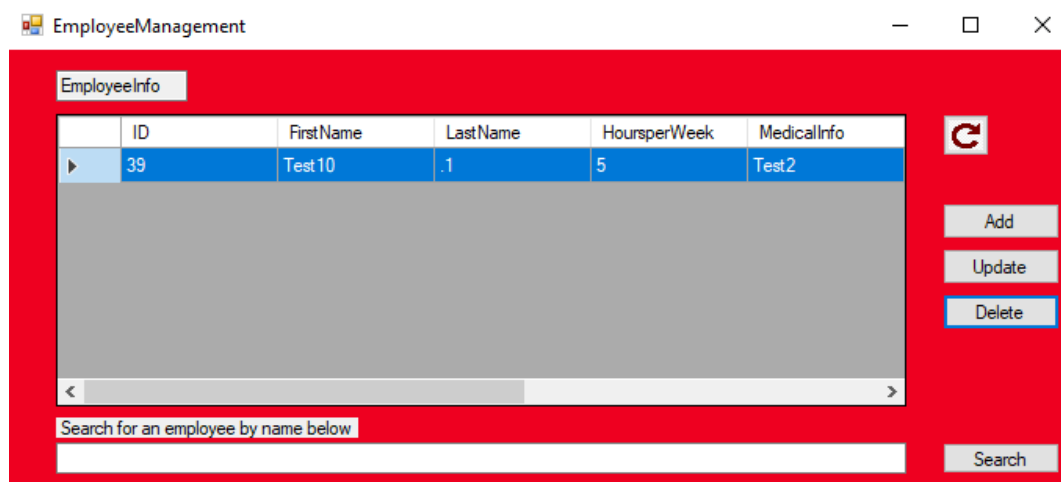
Test 10.1

This test is required as the Employee management system should always have at least one Administrative member of staff always. This is because without at least one Admin it would be impossible to view or amend employee information as this is a privilege only offered to senior members of staff.

Test Employee First Name: Test10.1 (typical data)

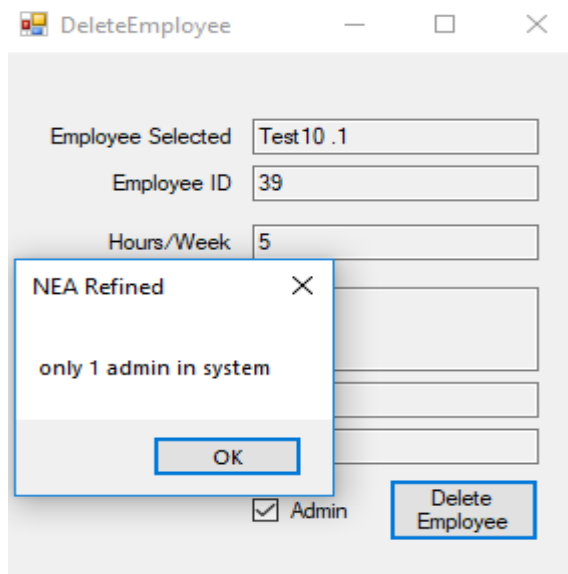
Admin Authorities: Checked (typical data)

This user will have to be the last case of an Admin. To test the solution, I will attempt to delete the user and we should be prompted with an error message relating to the minimal number of admin accounts



As you can see there is only one employee admin left in the system.

Result:



As you can see the test was a success as we receive a warning message when we try to delete the only admin left in the database.

Test 10.2

This test is being conducted to make sure that a user cannot delete himself whilst he/she is logged in with the specified account that is trying to be deleted.

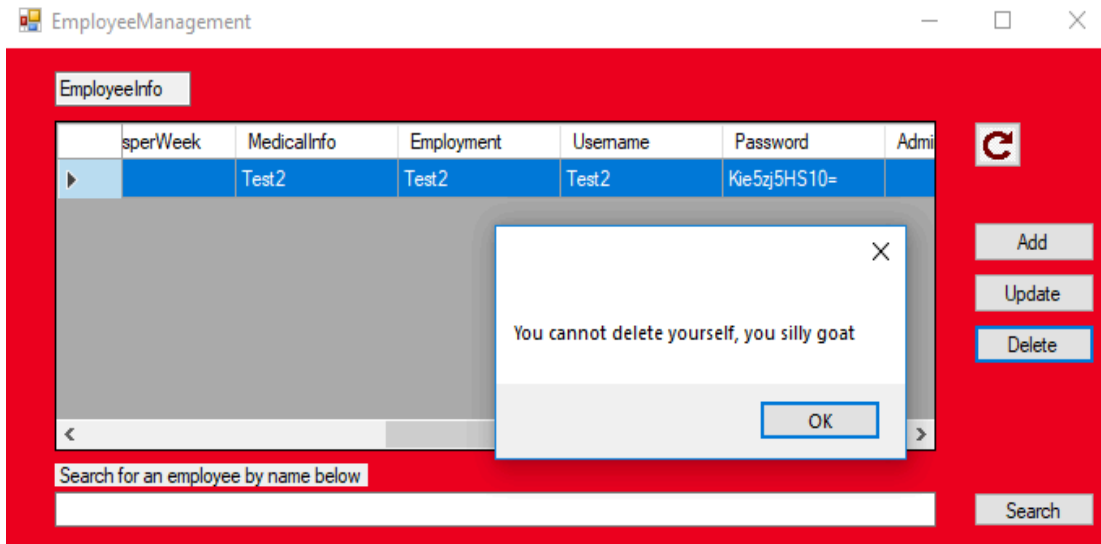
Test Employee Username: Test2 (typical data)

Test Employee Password: Test2 (typical data)



As you can see above I am logging in with the user Test2.

Result:



As you can see the test was a success as an error message appears which ensures that when you are logged in you cannot delete yourself from the employee management system.

Test 11

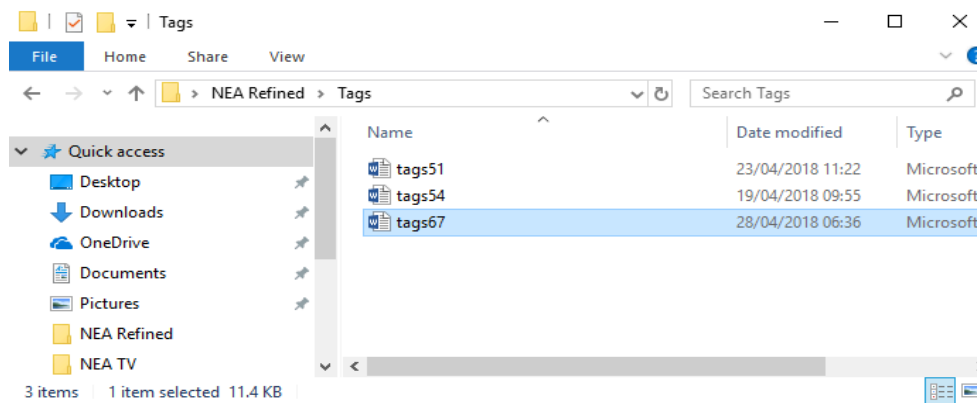
This test is being conducted to make sure that the Tag printing and saving feature as requested by a volunteer works correctly. For the sake of the test a selected item will be converted to a tag and displayed in a word document format.

Item Name: Test11 (typical data)

Number of Tags Printed: 3 (typical data)

The screenshot shows a window titled "Tag_Printer" with a light gray background. It contains a form with the following fields: "Name" with the value "Test11", "Price" with the value "\$12.00", "ID" with the value "67", and "Number of Tags printed" with the value "3". There is a "Print" button at the bottom.

Result:



The tag was saved to a word document with an ID of 67 which is correct for our test value

Item Name: Test11
Item Price: \$12.00
Item ID: 67

Item Name: Test11
Item Price: \$12.00
Item ID: 67

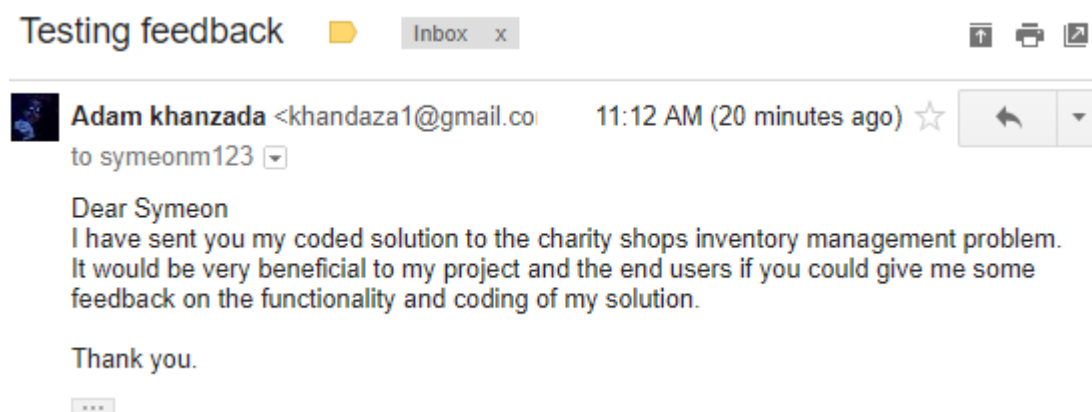
Item Name: Test11
Item Price: \$12.00
Item ID: 67

3 tags were formatted and available to print on the word document therefore the test was a success.

Whitebox Testing

My Whitebox tester will be Symeon Mehmet. Symeon previously worked in a charity shop and is an experienced coder. He will be receiving the coded solution as well as the finished user interface. His feedback will be invaluable.

Message:



Reply:

Testing feedback



Inbox x



Adam khanzada Dear Symeon I have sent you my coded solution to t... 11:12 AM (19 minutes ago) ☆



Symeon Mehmet

11:28 AM (3 minutes ago) ☆



to me ▾

Dear Mr Khanzada

I have found using your program to be delightful. I conducted a thorough inspection of the software by using it to store several hundred items and I am yet to come across an error due to your in-depth and thorough error handling. The code was very well commented on and neatly written in a modular fashion. Your main form sub-routines are very well designed as they are not over clogged with repeat code and you have clearly kept global variables to a minimum. I have made several attempts to manipulate your tables using SQL injection and all my attempts have failed due to your consistent use of paramaterisation. Furthermore, your tables have clearly been normalised well which is a positive sign for the integrity of your data.

Finally I must add that your employee management system through the use of admin accounts has been exception handled very well.

Yours Sincerely

As we can clearly see from Symeon's thorough testing and response my solution has been written and designed to a high standard. Error checking was implemented along every step of the way to minimize the risk of crashing. Furthermore, he mentioned my use of parametrisation in sql queries in order to prevent sql injection. Finally, he appreciated the integrity of my data in the tables as everything had been designed to 3rd normal form. Symeon's feedback has outlined my programs vast error handling and its success in preventing crashes. His feedback was very helpful as I can conclude that my code is of a high quality.

Evaluation

In my evaluation I will be analyzing the overall success of my solution to the charity shops problem. This will revolve around my ability to cater to the needs of my end user while meeting the objectives I set in my initial analysis around the problem.

Objective	Completion?	Comments
-----------	-------------	----------

<p>1.1: Adding items should provide useful information to the staff such as pricing or stock quantity</p>	<p>Yes</p>	<p>INSERT sql statements have been used throughout to make sure that items can be added with the relevant level of information to the shop. Price of items has been included along with various other general info necessary to the staff</p>
<p>1.2: Items that are added should be compartmentalized into specific tables relating to the stores most commonly donated and sold item types. E.g.: Clothing, DVD's and Toys</p>	<p>Yes</p>	<p>As a part of the normalization process the fields within tables had to be carefully figured out to prevent empty data points or the repetition of information. Items have been stored in a main table and then further information follows in the type specific tables</p>
<p>1.3: Dependent on the Item type the database should store Item type specific information. For example, a DVD will require information on age restrictions and print quality whereas a piece of clothing requires size measurements.</p>	<p>Yes</p>	<p>Leading on from the previous objective the item type information that is unique to the table will hold information that only that type will deem appropriate as a DVD cannot have a size or gender. This was done to limit the use of redundant table space. An item type table will only hold necessary info to do with the item it is.</p>

<p>2.1: All Information about the selected item should be able to be viewed and specific details on the item can be changed such as the item name or condition.</p>	<p>Yes</p>	<p>The code I have created has used UPDATE queries for the user to be able to amend aspects of the stock. All necessary stock info prone to change is editable such as quantity or price. The Item General information may be updated alongside the info unique to the type of item being dealt with.</p>
<p>3.1: Be able to delete items from the database as stock runs out over time. The stock quantity for each item should be updateable and a quantity of zero for when the item runs out should remove the item from the database.</p>	<p>Yes</p>	<p>DELETE queries have been essential in making sure that stock management stays up to date as items run out of stock daily. Any highlighted item can be deleted entirely from the database or any quantity removed through the use of an UPDATE statement changing the integer number quantity.</p>

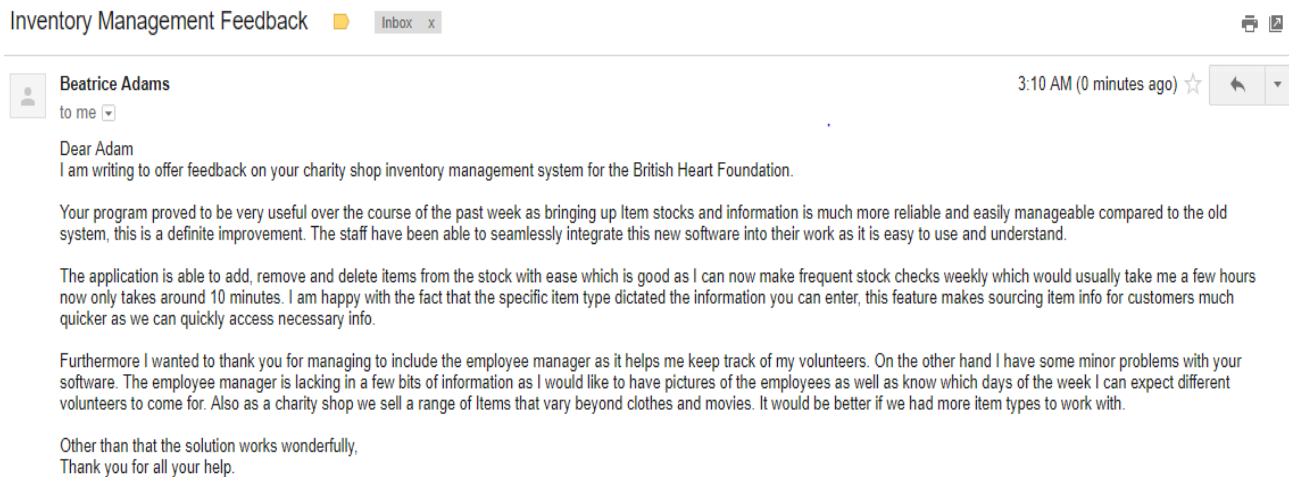
<p>3.2: Before the Item is removed necessary information to do with the removal should be seen by the viewer such as the current stock quantity and the items name.</p>	<p>Yes</p>	<p>When an Item is highlighted and chosen to be deleted I felt as though it would necessary to include the name of the item being tended to and the relevant information when trying to edit quantity or remove the item entirely. This information would refer to the current quantity.</p>
<p>4.1: The software should have a login screen for the sake of security purposes as precious shop information should not be openly accessed by the public</p>	<p>Yes</p>	<p>All staff can have a username and login set by the manager or a senior member of staff. The login prevents any unknown strangers from accessing the shop info as they will be unable to login without the correct credentials.</p>
<p>4.2: All employees will have a Username and Password to login. The passwords will be encrypted using a hashing algorithm to add an extra layer of security</p>	<p>Yes</p>	<p>I have used a hashing algorithm based around a key to encrypt all passwords for staff when an account is created. When logging in the password entered is encrypted and compared to the encrypted text in the database. This can help to prevent access to sensitive employee info.</p>

<p>4.3: Different employees will have access to admin privileges based upon your ranking position in the store and the managers approval as admin users can appoint new administrators. The manager will be an admin by default.</p>	<p>Yes</p>	<p>The manager and other senior member of staff can have admin privileges attached to their accounts. This means that they will retain access to employee information and be able to make changes to employee info. All non-admin users will not receive an option to overlook employee details</p>
<p>5.1: The manager has requested that an employee management system is added to the program to make it easier keeping track of workers due to the number of volunteer staff</p>	<p>Yes</p>	<p>The admin system allows access to the employee management system upon request of the manager. This allows the manager to make changes to employee details and keep track of current volunteers and hours.</p>
<p>5.2: The Employee management can only be accessed and used by senior members of staff that have admin privileges. This is because information on employees is private and should only be seen by the manager and select co-workers of the managers choosing.</p>	<p>Yes</p>	<p>Admin users have an extra button on the menu screen which allows them access to employee info.</p>

<p>5.3: The employee management table should store necessary information about employees such as Name, hours per week and medical information.</p>	<p>Yes</p>	<p>The database tables have been carefully designed to hold the necessary information on employees working within the shop. This information covers requirements by law such as medical information as well as employment hours for the managers convenience.</p>
<p>5.4: Admin staff should can add, delete and update staff information.</p>	<p>Yes</p>	<p>A separate database table is being used to be able to add, remove and update employee info. This is done using many sql queries such as INSERT, UPDATE and DELETE allowing admin users to keep track of employees effectively.</p>
<p>6.1: A volunteer has requested that the software should be able to take necessary information regarding an item and print a tag in order reduce time taken to manually write out tagging information. The tags should provide the Item name, ID and price.</p>	<p>Yes</p>	<p>A feature was implemented into the menu so that a worker may highlight an item and print tags for the item through a word document. This was done by establishing a connection between the word document and the code to append writing to the document and then print. The user can also save the tags to a specified location of choice.</p>

End User Feedback (Blackbox Testing)

This is an email received from the manager upon asking for feedback for my solution to the problem after a brief few days of general testing.



As we can see from the email sent by the local manager many features of the solution have been praised for meeting the shops requirements. The overall effectiveness of my solution has been denoted as the time taken when sourcing information for customers has dropped on top of the time taken to manually write stock into a diary. The manager made it clear that the software serves its purpose well and no points on crashing or software issues were risen therefore it is fair to conclude that my program was well exception handled.

My solution has obviously met the needs of the user as the owner expressed the ability to seamlessly add, remove or update new Item info into the stock. Furthermore, the Manager mentioned how the staff were able to easily integrate my software into their work routines over a short period of time showing my software's friendly user interface and simplicity in use for computer beginners.

In terms of negatives the Manager expressed some weak areas that could be worked upon to improve the solution further. The Employee management aspect was added at the request of the manager and she felt that it need to hold more information on the employees such as a picture and specific working days and hours. Also, she expressed the need for a system of adding in custom item types as the charity does sell more than just clothing, DVD's and toys.

Other than these minor flaws the solution meets the needs of the end user and has proven to be successful in decreasing inefficiency in work in the charity shop.

Potential for expansion/ Improvement

Based upon the feedback received from the Manager some areas of weakness have been highlighted which leave room for potential upgrading and improvement.

Improvements

Employee Manager

The employee management system has room to improve as brought up by the manager as the level of information held in the system on employees is the bare minimum and therefore does not help the manager to a significant degree. If features such as employee time tables, latest salaries and photo ID could be implemented this would make the management of workers much smoother.

More Item Types

Another feedback point received was that the shop sells a wide variety of items besides clothes, toys and DVDs so the software couldn't cater towards all inventory. A system needed to be added which allows the user to create more item types and thus create new tables outlining fields for item specific information. An example of this would be gardening/ outdoor supplies. The user would have to be able to choose what fields of information to include and a new table based upon this would have to be constructed.

Potential

API price recommender

Often charity shops struggle to correctly price items due to the staff having a lack of knowledge or experience in that market. This leads to charity shops selling goods at significantly lower prices than the typical charity standards thus decreasing potential charity revenue. This could be fixed by using an API that can search for prices of goods and recommend a reasonable 2nd hand price.

Outward Scaling

Due to time constraints the software can only cater for a single shop. The solution has the potential to branch out into the other British Heart shops which could be done by using a unique shop code before the login stage to identify the specific stores stocks.

Statistical calculator

Finally, a Manager can find it useful having more statistics to do with sales, this allows them to better set prices and advertise products of larger demand.

Activity Log

<u>Date</u>	<u>Task Commenced</u>	<u>Description</u>
11/11/17	Database design and connection	In this step of the process I was learning how databases work and also attempting to establish a link between my database and my visual studio project
20/11/17	Linking database tables together	This took great planning as I needed to devise a way to display information about a product and have further details on it based upon its type without having large empty data spaces
05/12/17	Designing the basic user interface	This would require me to do some research into how I can best display information to a user. Involved lots of concept designs
22/12/17	Login screen code and design	Started with the basic template for a login screen. I then implemented the code needed for it to give user access based on validated text
24/12/17	Main menu code and design	The menu had to have all the main options alongside a way to clearly view all the data tables
05/02/18	Update Item design and code, Add Item design and code.	The design aspect would allow you to either view or fill out fields to do with the item based upon type. You could then update info on that item by highlighting it

10/02/18	Delete Item design and code	Deleting items used two basic sql queries. One updated the quantity if the stock value was greater than zero and the other removed the item if the quantity went to zero
15/02/18	Password hashing algorithm code	When a password is entered the key is used to encrypt the password and later decrypt for security and successful logins
06/03/18	Employee management and admin setup	Based upon whether the database ticked you as an admin the button for employee management was revealed or hidden to users
06/03/18	Add Employee design, Remove employee design	To add an employee works similarly to items; however, the data is only appended to one table making it easier
13/03/18	Update Employee code and design	When updating employee info all current employee data is displayed and you can change that information and hit the update button to confirm changes
20/03/18	Tag printing and connection to word document	The connection to the word document allows me to save tags and print them out for employees. This involved formatting the word document and calling upon necessary information.
04/04/18	Error handling and testing based on white box tester checking of code	Made sure that the program does not crash unexpectedly and makes sure all fields are filled out when necessary

Bibliography

Websites and sources used for research and development

Purpose of source	URL or source	Date Accessed
Implementation of a functional login system	http://www.visual-basic-tutorials.com/form/LoginT.htm	10/01/18
Creating a database and linking tables together	https://www.youtube.com/watch?v=nszRT3nRUMU	29/12/17
Establishing a connection between Visual Studio and Microsoft Access database	https://www.youtube.com/watch?v=7Z4BGEHD-JQ	06/01/18
Learning SQL queries such as INSERT, DELETE and UPDATE	AQA AS and A LEVEL Computer Science by Rob and Pat Heathcote, PG ONLINE	20/01/18
Further learning of SQL queries. (JOIN)	https://www.youtube.com/watch?v=aDaEHHzryAY	30/01/18
Hashing algorithm for passwords using a key.	https://support.microsoft.com/en-gb/help/301053/how-to-compute-and-compare-hash-values-by-using-visual-basic-net-or-vi	15/02/18
Developing a login screen using usernames and passwords.	https://www.youtube.com/watch?v=2s9BLuhLVY4	18/11/17
Establishing a connection between Microsoft word and visual studio to save and print files.	https://www.dotnetperls.com/word-vbnet	15/11/17