Information Processing Technology

Maximum Technology Checkout Application

By Matthew D’Agostino

Contents

[1.0 PART A - Identification Phase 4](#_Toc523361585)

[1.1 Business Description 4](#_Toc523361586)

[1.1.1 Primary Functions 4](#_Toc523361587)

[1.1.2 Customer Interactions 4](#_Toc523361588)

[1.1.3 Business Size 4](#_Toc523361589)

[1.2 Rationale 4](#_Toc523361590)

[1.3 Assumptions and Boundaries 4](#_Toc523361591)

[1.3.1 Universe of Discourse 4](#_Toc523361592)

[1.3.2 Users Access Rights 5](#_Toc523361593)

[1.3.4 Assumptions 6](#_Toc523361594)

[2.0 Conceptualization Phase 7](#_Toc523361595)

[2.1 Goals 7](#_Toc523361596)

[2.1.1 Usability 7](#_Toc523361597)

[2.1.2 Fast 7](#_Toc523361598)

[2.1.3 efficient 7](#_Toc523361599)

[2.1.4 Secure 7](#_Toc523361600)

[2.2 System Capabilities 7](#_Toc523361601)

[2.2.1 Login and Signup 7](#_Toc523361602)

[2.2.2 Inventory Management 7](#_Toc523361603)

[2.2.3 checkout 7](#_Toc523361604)

[2.2.4 Account management 7](#_Toc523361605)

[2.2.4 Invoice LOG 8](#_Toc523361606)

[2.3 Input/output 8](#_Toc523361607)

[3.0 Formalisation 9](#_Toc523361608)

[3.1 Conceptual Schema 9](#_Toc523361609)

[3.1.1 Entities 9](#_Toc523361610)

[3.1.2 Elementary Sentences 11](#_Toc523361611)

[3.1.3 CSD 13](#_Toc523361612)

[3.2 Relational Schema 14](#_Toc523361613)

[3.3 Design 15](#_Toc523361614)

[3.3 Data Flow Diagram 19](#_Toc523361615)

[4.0 Part B – Changes to implementation 20](#_Toc523361616)

[4.1 DESIGN 20](#_Toc523361617)

[4.2 Purchase form 20](#_Toc523361618)

[4.3 Database design 20](#_Toc523361619)

[4.4 Database Names 21](#_Toc523361620)

[4.5 Login Form 21](#_Toc523361621)

[4.6 Calendars 21](#_Toc523361622)

[5.0 Forms 22](#_Toc523361623)

[5.1 Login Form 22](#_Toc523361624)

[5.1.1 Design 22](#_Toc523361625)

[5.1.2 Code 22](#_Toc523361626)

[5.1.3 Testing 24](#_Toc523361627)

[5.1.4 SQL Queries 26](#_Toc523361628)

[5.1.5 Evaluation 26](#_Toc523361629)

[5.2 Signup Form 28](#_Toc523361630)

[5.2.1 Design 28](#_Toc523361631)

[5.2.2 Code 28](#_Toc523361632)

[5.2.3 Testing 30](#_Toc523361633)

[5.2.4 SQL Queries 31](#_Toc523361634)

[5.2.5 Evaluation 32](#_Toc523361635)

[5.3 Staff Menu Form 33](#_Toc523361636)

[5.3.1 Design 33](#_Toc523361637)

[5.3.2 Code 33](#_Toc523361638)

[5.3.3 Testing 34](#_Toc523361639)

[5.3.4 SQL Queries 35](#_Toc523361640)

[5.3.5 Evaluation 35](#_Toc523361641)

[5.4 Manager Settings Form 36](#_Toc523361642)

[5.4.1 Design 36](#_Toc523361643)

[5.4.2 Code 36](#_Toc523361645)

[5.4.3 Testing 41](#_Toc523361646)

[5.4.4 SQL Queries 42](#_Toc523361647)

[5.4.5 Evaluation 43](#_Toc523361648)

[5.5 New Staff Form 45](#_Toc523361649)

[5.5.1 Design 45](#_Toc523361650)

[5.5.2 CODE 45](#_Toc523361651)

[5.5.3 Testing 47](#_Toc523361652)

[5.5.4 SQL Queries 48](#_Toc523361653)

[5.5.5 Evaluation 49](#_Toc523361654)

[5.6 Purchase Form 50](#_Toc523361655)

[5.6.1 Design 50](#_Toc523361656)

[5.6.2 Code 50](#_Toc523361657)

[5.6.3 Testing 55](#_Toc523361658)

[5.6.4 SQL Queries 56](#_Toc523361659)

[5.6.5 Evaluation 57](#_Toc523361660)

[5.7 Invoice Log Form 59](#_Toc523361661)

[5.7.1 Design 59](#_Toc523361662)

[5.7.2 Code 59](#_Toc523361663)

[5.7.3 Testing 60](#_Toc523361664)

[5.7.4 SQL Queries 61](#_Toc523361665)

[5.7.5 Evaluation 61](#_Toc523361666)

[5.8 Checkout Form 62](#_Toc523361667)

[5.8.1 Design 62](#_Toc523361668)

[5.8.2 Code 62](#_Toc523361669)

[5.8.3 Testing 65](#_Toc523361670)

[5.8.4 Sql Queries 67](#_Toc523361671)

[5.8.5 Evaluation 67](#_Toc523361672)

[5.9 Inventory 69](#_Toc523361673)

[5.9.1 Design 69](#_Toc523361674)

[5.9.2 Code 69](#_Toc523361675)

[5.9.3 Testing 73](#_Toc523361676)

[5.9.4 SQL Queries 76](#_Toc523361677)

[Query #3 76](#_Toc523361678)

[5.9.5 Evaluation 76](#_Toc523361679)

[5.10 Global Variables 78](#_Toc523361680)

[5.10.1 Code 78](#_Toc523361681)

[6.0 Questionnaire Evaluation 79](#_Toc523361682)

[7.0 Final Overall Reccomendations 82](#_Toc523361683)

[8.0 Conclusion 84](#_Toc523361684)

[9.0 Appendix 85](#_Toc523361685)

[9.1 Questionnaire results 85](#_Toc523361686)

# 1.0 PART A - Identification Phase

## 1.1 Business Description

Maximum Technology is a small technology superstore established in 1974 on Roma St in the Brisbane City CBD. The business prides itself in its speedy service, fast delivery and no hassle purchases. Maximum technology has always specialised in HIFI specifically by hiring HIFI professionals in order to provide the best advice and system. In 2014 the business was sold, and the new CEO has plans to expand the already thriving business to multiple locations. The business mission statement is as follows “*At Maximum technology we want to provide you with the best experience possible, if this means lowering your prices, that is fine with us. We pride ourselves in providing the best experience possible and honour it every day. We are for the customer and are always willing to provide advice or support.”* They are a no hassles company tailored for the customer.

### Primary Functions

The primary functions of the business are to sell technology such as laptops, gaming, HIFI and more, to customers at the best prices possible catering to the customer’s needs, always willing to extend a hand. The staff are open to bargaining and have the customers best interests at heart. They provide any and all audio advice and will even provide a free setup on selected systems.

### Customer Interactions

Through the current system implanted, customers can buy and order products for pickup through both telephone and email or instore. If customers are seeking advice or information on a system/s they may also contact via email, telephone or instore as well.

### Business Size

Maximum technology has two floors on 194 Roma St in the Brisbane CBD where one floor contains all products on computers and general technology with an inventory at the back. Whilst on level two the whole floor is dedicated to HIFI. The store gets an average of 97500 people each year and an average of 300 people per day and the average order is about three items with a total price of $45. This large amount of constant traffic from customers is what is causing the CEO to want to expand the business. The store has an annual turnover of 4.3 million and a staff team of twenty-four, fifteen of which are part time.

## 1.2 Rationale

The business in the past has always used a paper system for keeping data on the inventory and orders that come in. This has been time consuming for staff and inefficient, sometimes containing errors which have caused the business problems in the past affecting business. Because of these problems, the business would like to redesign the ordering/inventory system to compensate for the growing business and outdated system that could grow with the business. This project will include a database and a client program allowing the staff member to set orders and update the inventory quickly, efficiently and accurately. This new system will improve overall staff happiness and will have a small learning curve with an easy to use interface allowing staff to become accustomed to the system quickly. The program will be designed and built to facilitate the needs of the business and will aim to provide a solution to the problem.

## 1.3 Assumptions and Boundaries

### 1.3.1 Universe of Discourse

The universe of discourse includes all the data included in the database. For Maximum Technology this includes information of user information, past login information, inventory information, and transactions information. This database will not include information regarding staff pay or hours

#### 1.3.1.1 User Information

Information regarding the user must be kept so that staff and users can operate registers. By using unique logins, if a problem does occur the problem or past transaction has the possibility of being traceable back to a specific user.

* Unique User ID
* Username
* Password
* Account Access Level
* Address
* Work Type (if applicable) [Full time, Part time, Casual]
* Position (if applicable)
* First name
* Last Name

#### 1.3.1.2 Login Log

For security reasons, all information on past logins must be kept in case there is a breach and information regarding the breach is needed. By providing a login log the information where the breach possibly happened can be found.

* Unique User ID
* Computer Name
* Computer Local Ip Address
* Login Time

#### 1.3.1.3 Inventory

Inventory information must be stored to keep track of product quantities and prices for easy transactions.

* Unique Inventory ID
* Model Number
* Name
* Description
* Price
* Discount (if applicable)

#### 1.3.1.4 Transactions

Transaction history must be kept as a means of tracking sales and proof of purchase.

* Unique Transaction ID
* Unique User ID
* Product ID
* Product Quantity
* Purchase Price

### 1.3.2 Users Access Rights

For security reasons every user will be required to login when using the program to ensure that no unknown or unwanted users access the system. There will be three levels of user access levels, Manager, HOD, Staff and Customer. Member will be assigned to normal staff member, Admin to higher level staff such a team leaders or department heads, whilst manager will go to the staff manager. These levels of access are to align with the business hierarchy which consists of the Manager, Department Heads (e.g. head of inventory), Normal Staff and customers.

#### 1.3.2.1 Customer

Any user who has the user access level of customer can place orders and view inventory and past items bought buy the user, but cannot edit inventory, view login logs, edit users, view users or see other users as this may expose private information users may not want seen.

#### 1.3.2.2 Staff

Users who have the staff access level will have access to add new items into the inventory, purchase items, view inventory, and view past transactions. Anyone with the staff access level will not be able to view any user information or edit existing inventory for security reasons.

#### 1.3.2.3 Supervisor

This access level is assigned to only head of departments and supervisors, for example, Head of Inventory. This has been applied as head of departments they are managing most products and will be aware of costing and how much a product can be discounted

#### 1.3.2.4 Manager

Manager is an access level assigned to only the manager/s. This will allow the user to edit user information. Without this access level, a user cannot edit or access user information that may be private or critical. This access level is the top access level with access to everything.

|  |  |  |
| --- | --- | --- |
| **Name (Positions)** | **Access Level** | **Permission** |
| Manager | 1 | All |
| Supervisor | 2 | Everything except editing user information |
| Staff | 3 | Everything except editing user/existing inventory information |
| Customer | No level | Can only purchase products |

### Assumptions

During the design of the program there were many assumptions that needed to be made in order to design he database and interface of the application as listed below:

It is assumed that invoices will contain more than one item with different inventory codes for the different products in the transaction, similar to the traditional store. This would result multiple items under one transaction.

It is assumed that invoices stored in the database will contain information such as items, cost, GST etc. This will be kept for proof of purchase history when customers need to return products for warranty.

It is assumed that levels of access are required to ensure the security of the system. This will create less possible threats as more user accounts have less rights, therefore, lowering the amount of possible opportunities for large scale damage to the database in the situation there is an vulnerability and the system is hacked.

It is assumed that the computer will generate invoices and not be generated manually by hand (i.e. on paper). This will save time for both the customer and the staff member as the computer generates the invoice and stores it in the database attached to the users account.

It is assumed that all payment information is not stored in the database. This is solely for the purpose of creating a more secure platform for customers, ensuring data as private as payment information (e.g. bank accounts, credit cards) would not be obtained in the event that the system is hacked or compromised. This means all payment information and payment history will have to be handled by a third party to ensure the customers security is kept.

Due to the program using a SQL database it is assumed that there is a server and local area network in place at the store or that the facilities will be organised ready for the implementation of the final software.

It is assumed that onsite at the store premises there will be computers connected to a local area network that will run the software provided or that the computers will be organised ready for the installation of the software.

It is assumed that all stores will have their own database and inventory. This is done for security reasons to ensure that customer information is not accessible through the internet as creating one database for all stores could create a vulnerability for hackers to exploit as it is only accessible over a LAN.

It is assumed that every customer will create or use their existing account logins when using the software, so invoice history may be recorded for warranty reasons as explained earlier.

# 2.0 Conceptualization Phase

## 2.1 Goals

### 2.1.1 Usability

Usability of the system will be a key goal of the system, aiming to achieve an easy to use and intuitive interface that is simple to navigate. The learning curve that is usually associated with more program will need to be low in order to make the system as user friendly as possible, allow the manager to spend less time on staff training and more time running the store.

### 2.1.2 Fast

The system needs to be fast to make sure that there are no slowdowns whilst running the program, inhibiting on staff time and disrupting the flow of work. Making sure the system is fast is a necessity as staff need to be as fast as possible in most actions, so sales are maximised.

### 2.1.3 efficient

The system will need to be efficient, making sure no extra data that does not need to be stored is stored, to ensure that the stability and efficiency of the system is up to the highest standards.

### 2.1.4 Secure

The database needs to be secure to make sure that no private data is accessible by hackers or unpermitted people. To ensure the databases security the database will need to have secure password that is not easily cracked by brute force or other methods.

## 2.2 System Capabilities

### 2.2.1 Login and Signup

From the login and signup forms users and staff will be able to:

* Login
* Signup

The use of a login will ensure that every user will be able

### 2.2.2 Inventory Management

From the inventory form staff will be able to:

* Add products to the database
* Remove products from the database
* Edit item information (Name, GST, Price, GST, Quantity etc)
* View inventory

### 2.2.3 checkout

From the checkout form staff and users will be able to:

* Search products/items
* Browse products/items
* Add products/items to cart
* Remove products/items to cart
* View cart

### 2.2.4 Account management

From the Manager Settings form, the manager will be able to:

* Edit staff information
* Make users staff
* Remove Staff

### 2.2.4 Invoice LOG

From the Invoice Log form, staff will be able to:

* View previous invoices

## 2.3 Input/output

Input data includes data such as username and password for logging into the program, data such as full name, address, phone number, access level, status, position, username, password all to be stored to their corresponding user in the database. User information can be inputted at login or when an user fills out and submits the signup form. All this data can be outputted through the manager settings form, which will be able to input and modify data as well.

Data for products will be stored and will need to be inputted by a user before they can be outputted to other users. This information will include entities such as, InventoryID, name, description, stock, price etc. which will all need to be inputted manually through the program. Furthermore, after information has been stored, via keyboard/mouse/touch input a user will be able to modify the data.

After each transaction a record of the invoice will be kept which will be stored in the database. When the user accesses the invoice log the program will output information stored in the database about the invoice. This data includes information such as, the products bought, the price, the total price, the total GST etc.

Credit card information will be able to be inputted for payment reasons during checkout, however, this data will not be stored or outputted through the program.

By clicking on different items in the DataGridView of the Purchase form the user will be able to input which items they would like to place in their virtual cart via the use of the mouse or touch interface. These items through a touch input also be removed from the cart.

# 3.0 Formalisation

## 3.1 Conceptual Schema

### 3.1.1 Entities

**Users**

|  |  |
| --- | --- |
| **Entity** | **Data** |
| UserID | Id (Primary Key) |
| Username | Name |
| Password | Text |
| HomeAddress | Address |
| FirstName | Name |
| MiddleName | Name |
| LastName | Name |
| Phone | Number |

**Staff**

|  |  |
| --- | --- |
| **Entity** | **Data** |
| UserID | Id (Primary Key) |
| Position | Name of position |
| Username | Name |
| Password | Text |
| Access Level | Number 1 - 4 |
| HomeAddress | Address |
| Status | Type (Full Time, Part Time, Casual, Volunteer) |
| FirstName | Name |
| MiddleName | Name |
| LastName | Name |
| Phone | Number |
| DOB | Date |

**Invoices**

|  |  |
| --- | --- |
| **Entity** | **Data** |
| InvoiceID | ID (Primary Key) |
| UserID | ID |
| StaffID | ID |
| Time | DateTime |
| GST | Money |
| PurchasePrice | Money |

**Orders**

|  |  |
| --- | --- |
| **Entity** | **Data** |
| OrderID | ID |
| QTY | Number |
| LineNo | Number |
| InventoryID | ID |

**Inventory**

|  |  |
| --- | --- |
| **Entity** | **Data** |
| InventoryID | ID (Primary Key) |
| Price | Money |
| Model# | Number |
| QTY | Number |
| Name | Text |
| Description | Text |
| GST | Percentage |

### 3.1.2 Elementary Sentences

**Users**

UserID (ID) 0000000001

chose / is assigned to

Username (Name) maximum

UserID (ID) 0000000001

chose / is used by

Password (text) dankpassword

UserID (ID) 0000000001

Resides at / is the residence of

HomeAddress (Address) 23 Miriwana St Kalgoolie Brisbane 4072

UserID (ID) 0000000001

uses / is assigned to

Email (Address) googlechrome@gmail.com

UserID (ID) 0000000001

goes by / is the last name of

LastName (Name) Thomas

UserID (ID) 0000000001

is called / is the name of

FirstName (Name) Christopher

UserID (ID) 0000000001

was given / is the middle name of

Middle name (Name) Patrick

UserID (ID) 0000000001

uses / is assigned to

Phone (#) 868945764

**Staff**

StaffID (ID) 0000000001

chose / is assigned to

Username (Name) spokeeym

StaffID (ID) 0000000001

chose / is used by

Password (text) aw3somep@ssword123

StaffID (ID) 0000000001

Resides at / is the residence of

HomeAddress (Address) 23 Miriwana St Kalgoolie Brisbane 4072

StaffID (ID) 0000000001

uses / is assigned to

Email (Address) spokeeym@gmail.com

StaffID (ID) 0000000001

goes by / is the last name of

LastName (Name) Spokes

StaffID (ID) 0000000001

is called / is the name of

FirstName (Name) Max

StaffID (ID) 0000000001

was given / is the middle name of

Middle name (Name) Julian

StaffID (ID) 0000000001

uses / is assigned to

Phone (#) 868945764

StaffID (ID) 0000000001

Is assigned to / is assigned to

Position (Name) Manager

StaffID (ID) 0000000001

has / is work type of

Status (Type) Full Time

StaffID (ID) 0000000001

Is assigned to / is assigned to

AccessLevel (#) 1

StaffID (ID) 0000000001

Was born on / is the birth date of

DOB (Date) 23/12/01

**Invoices**

InvoiceID (ID) 0000000022

is assigned to / is assigned to

UserID (ID) 0000000001

InvoiceID (ID) 0000000022

is assigned to / is assigned to

StaffID (ID) 0000000001

InvoiceID (ID) 0000000022

Was assigned /is gst amount for

GST ($) 0.10

InvoiceID (ID) 0000000022

Was calculated at / is price of

PurchasePrice ($) 10

InvoiceID (ID) 0000000022

Was generated on / was time of generation for

DateTime (Date) 23/06/18

**Orders**

InvoiceID (ID) 0000000022

has / was added to

InventoryID (#) 00000003

has / is amount of

QTY (#) 3

InvoiceID (ID) 0000000022

has / was added to

InventoryID (#) 00000003

has / is amount of

LineNo (#) 1

**Inventory**

InventoryID (ID) 00000003

has / belongs to

ModelNum (#) 37483274892

InventoryID (ID) 00000003

has / is amount of

QTY (#) 25

InventoryID (ID) 00000003

uses / is description for

Description (text) Easy to use! The Bop It. New from Microsoft, challenge yourself in the game spreading the nation.

InventoryID (ID) 00000003

Was assigned / is assigned to

GST (%) 0.10%

InventoryID (ID) 00000003

costs / is cost of

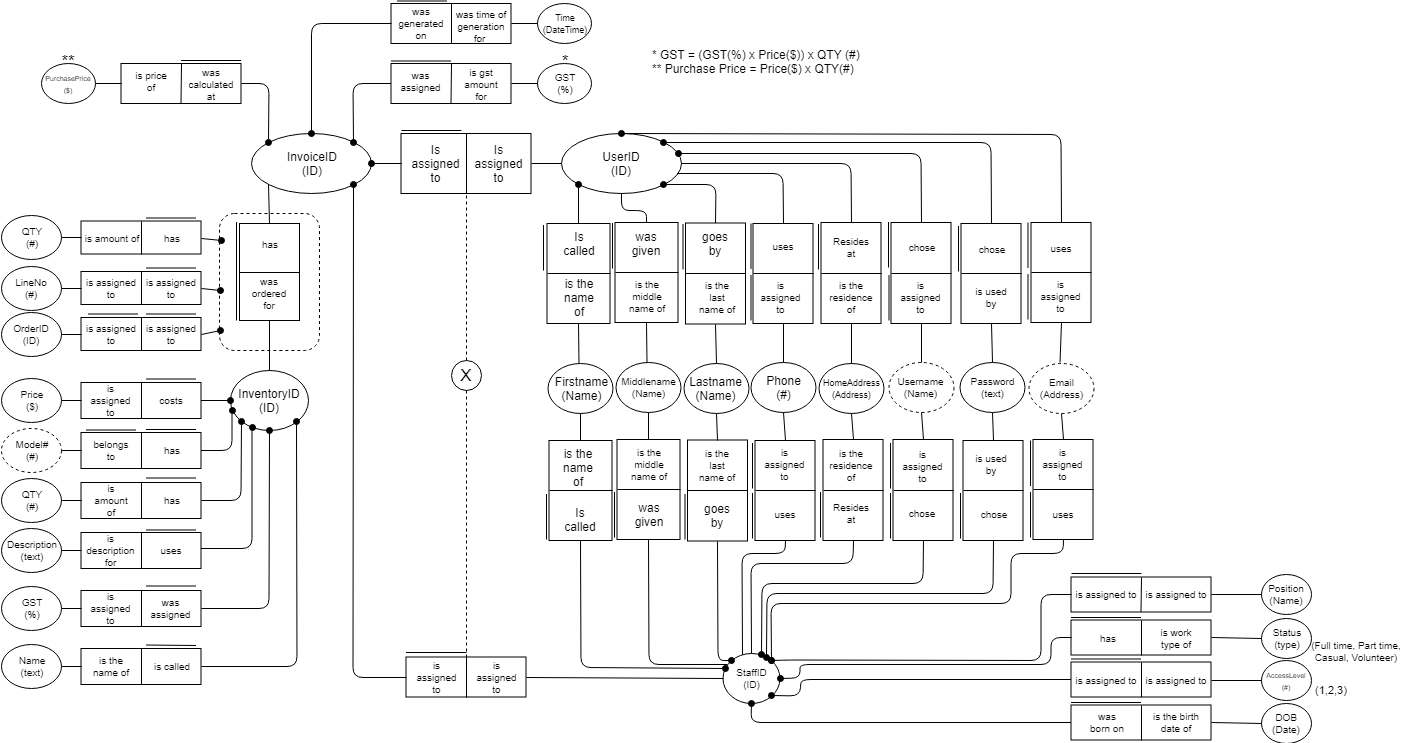
Price ($) 10

InventoryID (ID) 00000003

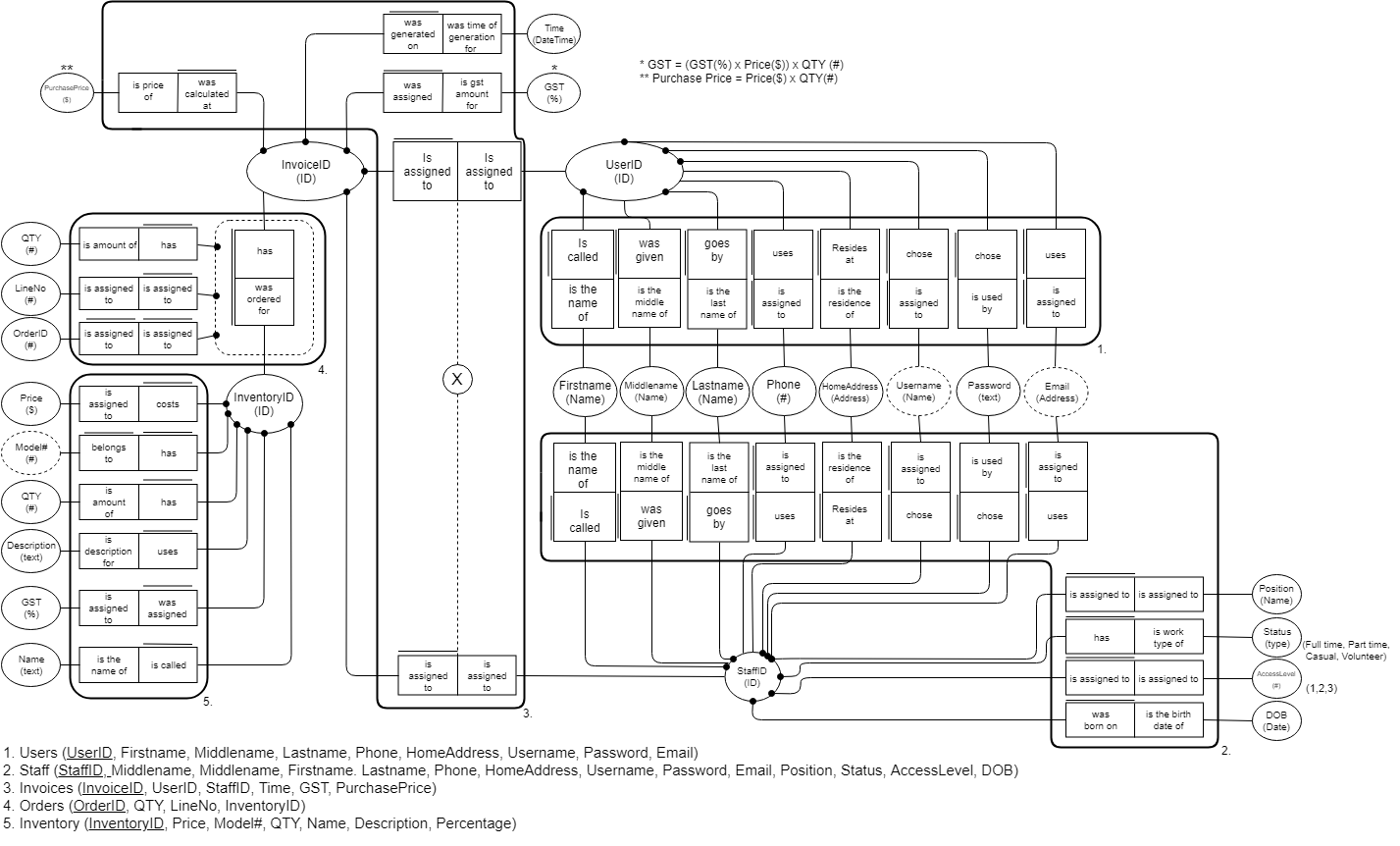
Is called / is the name of

Name (text) Bop It!

### 3.1.3 CSD



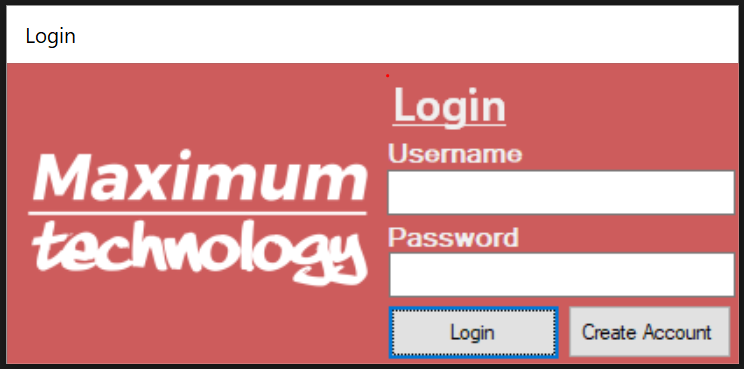
## 3.2 Relational Schema



## 3.3 Design

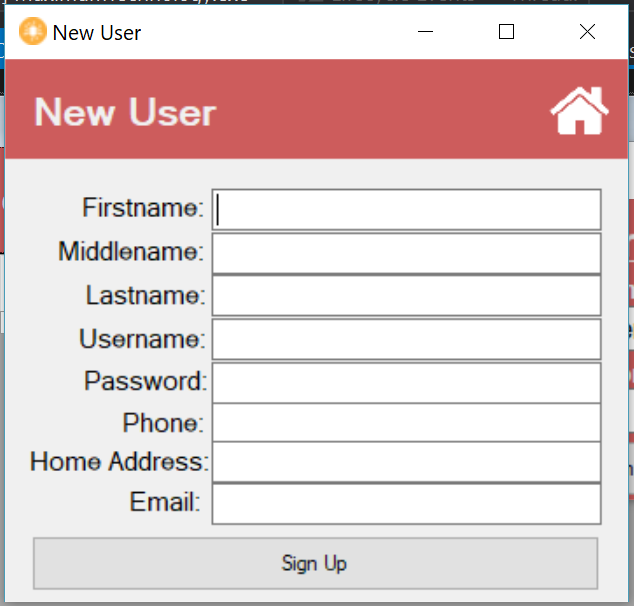
#### 3.3.1 Login

The purpose of the login form is to allow the user to login to their account which, depending weather or not they are staff will show either the purchase form for normal users or the staff menu for staff members. From the login form user should also be able to click the ‘Create Account’ button where they will be able to sign up for an account.



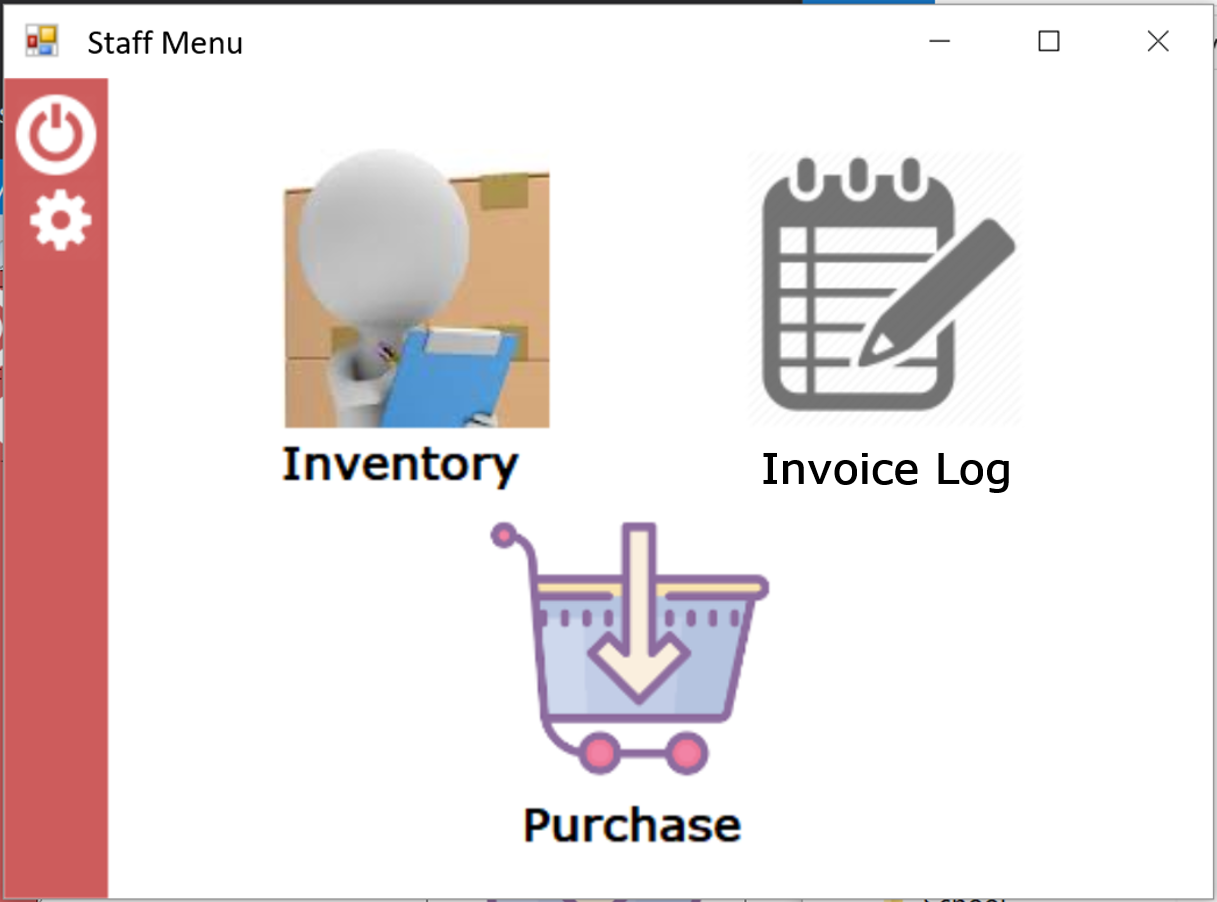
#### 3.3.2 Sign Up

The purpose of the sign-up form is to allow a customer to sign up to the system by entering their details into the corresponding boxes and creating an account that would allow them to login to the application.



#### 3.3.3 Staff Menu

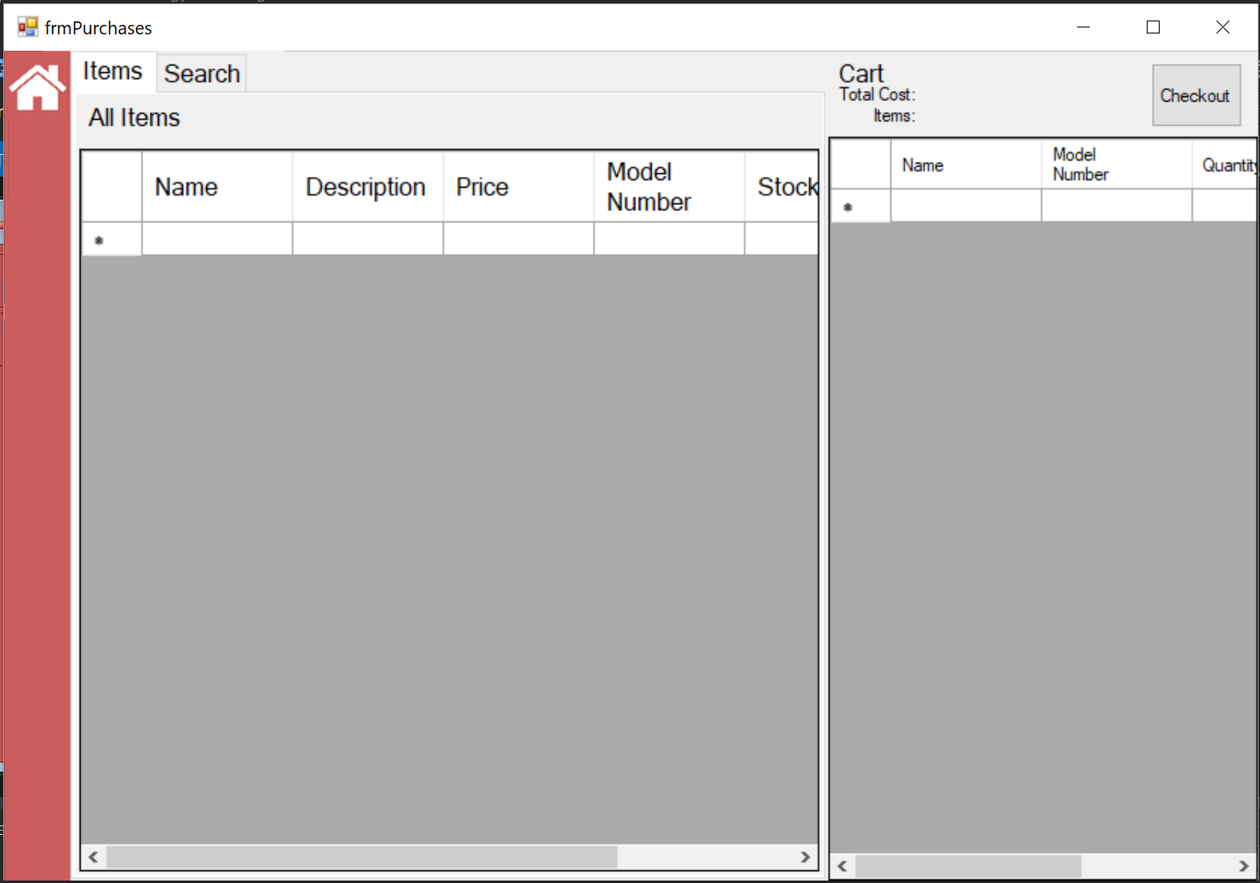
The purpose of the Staff Menu is to allow a staff member to navigate the between forms by using the pictures on the form as buttons that act as links to other forms. Furthermore, the ability to hide/show the manager settings button depending on whether a manager logs in is also necessary.



**Invoice Log**

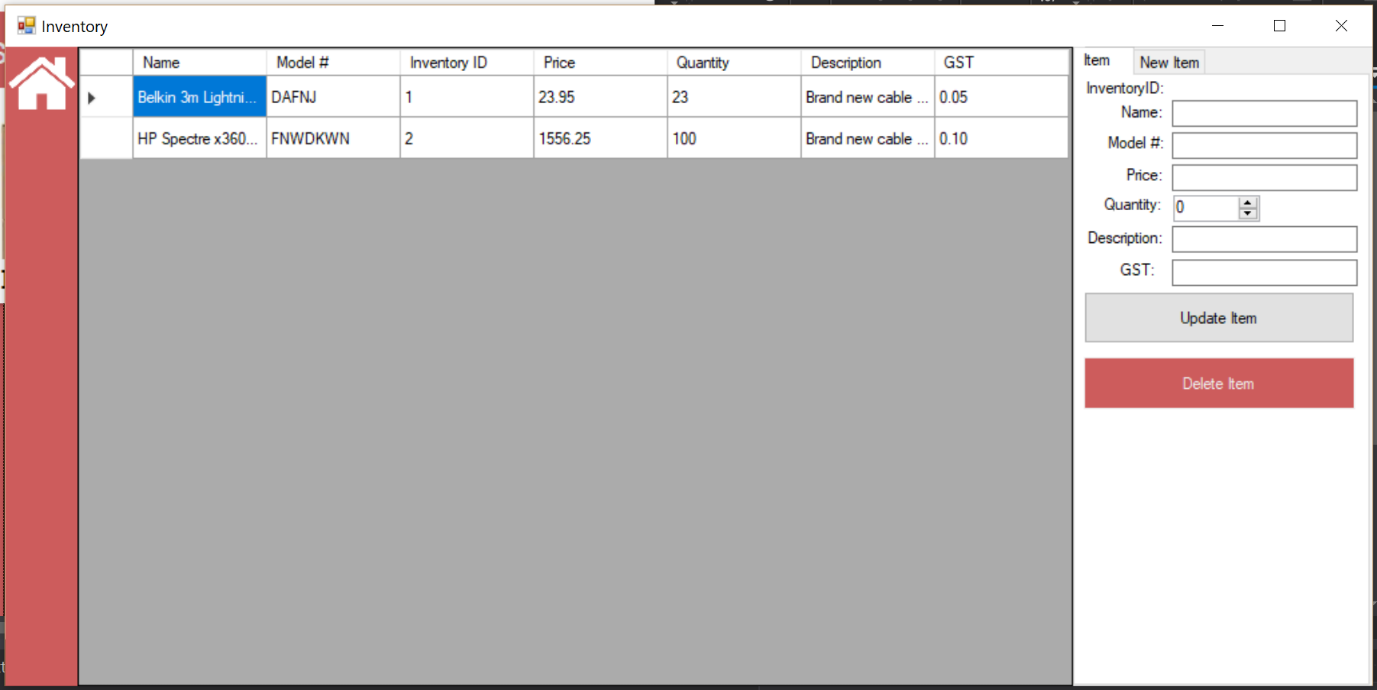
#### 3.3.4 Purchase Form

The purpose of the purchase form is to allow users the ability to be able to purchase items by adding them to their cart by either searching for an item or browsing it in the items tab. The user should have the ability to add items to the cart by clicking on the item in a DataGridView and the ability to remove them from the cart by clicking on the remove column in the cart. The form should display the total cost and the number of items in the cart. Furthermore, once the user has [memes]selected the items they would like to buy they should be able to click the checkout button to complete the transaction.



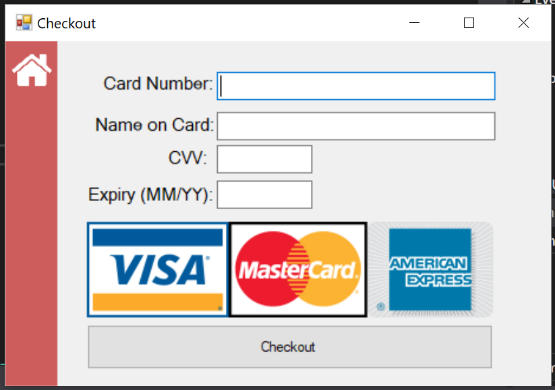
#### 3.3.5 Inventory

The purpose of the inventory form is to allow the user to edit inventory information by allowing them to click on the item they would like to edit from the inventory in the DataGridView. The user should also be able to delete the item they would like to remove from the inventory.



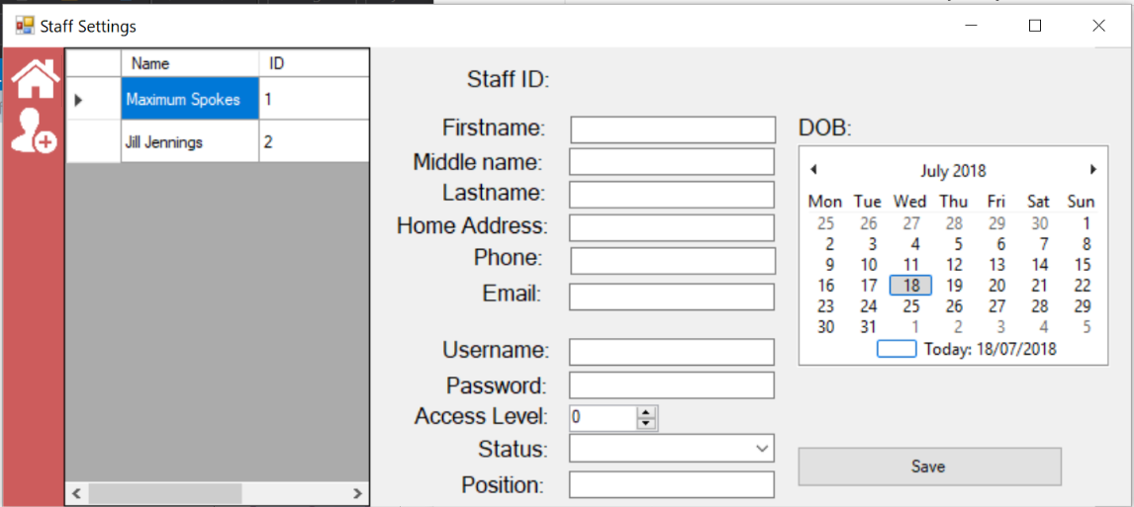
#### 3.3.6 Checkout

The purpose of the checkout form is to allow the user to be able to enter in their card number, name, CVV, and expiry of their credit/debit card to purchase the items they selected and added to cart.



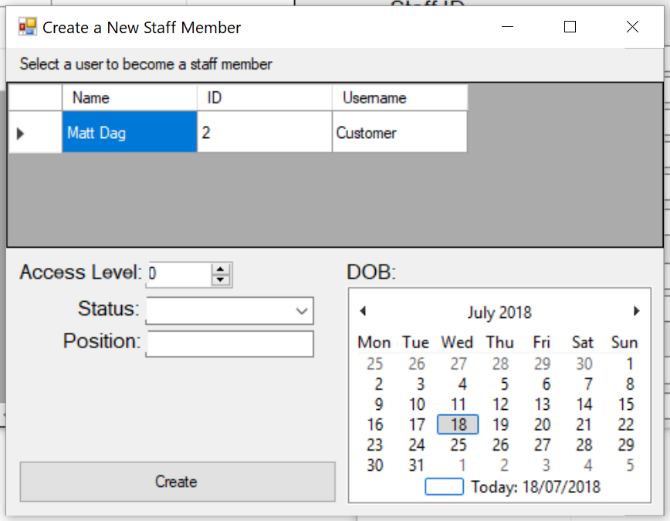
#### 3.3.7 Staff Settings

The purpose of the staff settings form is to allow the manager the ability to select a staff member from the data grid view and edit their details. The manager should also be able to click the add staff member button in the side bar where the New Staff Member form will open.



#### 3.3.8 New Staff Member

The purpose of the New Staff Member form is to allow a user to select another user which is not currently a member of staff and assign them a staff id, access level, date of birth, status and position.

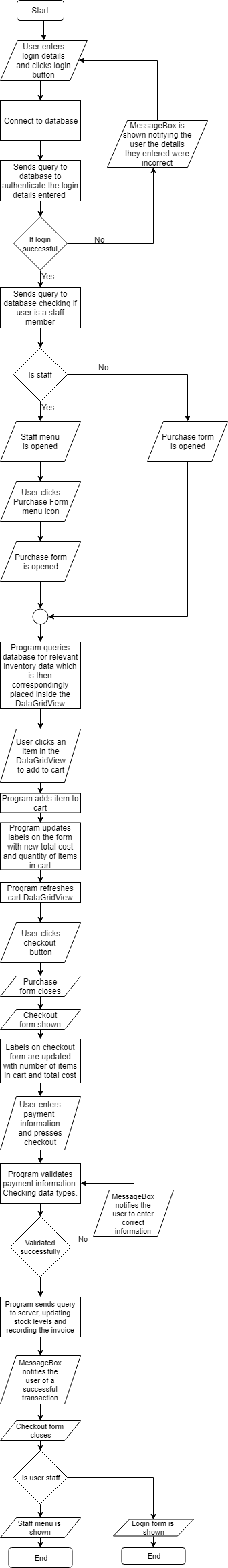


#### 3.3.9 Invoice Log

The purpose of the invoice log form is to allow the user to view past invoices and information stored in the database assigned to each invoice.

#### 

## 3.3 Data Flow Diagram



The dataflow diagram situated to the left consists of three main parts, including login, purchase and checkout as the user moves between these main forms.

The diagram starts with the user entering their login details where the details they enter will be authenticated and checked against database records. If the user does not enter in the correct details a MessageBox will be promptly shown notifying the user that the login details entered are wrong. During login the program will also check if the user logging in is a member of staff so the corresponding form can be shown. Normal customers will directly see the purchase form whilst staff will be shown the staff menu where they can select the purchase form.

From the purchase form the user can then add a product to the cart by clicking on the item row in the DataGridView. Relevant cart information will then be updated and displayed on the form such as total cost and item quantity in cart. Once the item has been added to the cart, the user can then click the checkout button where the checkout form will be displayed, and the purchase form will disappear.

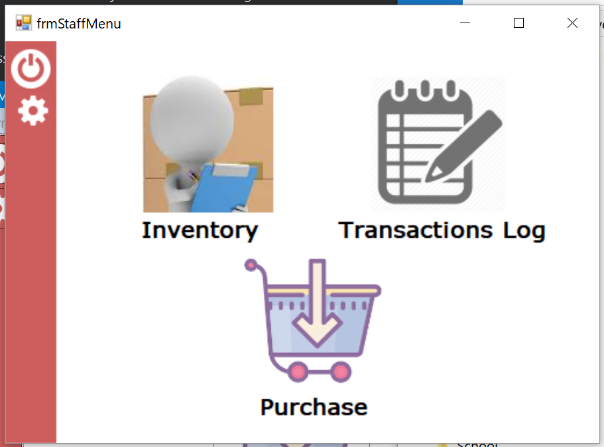
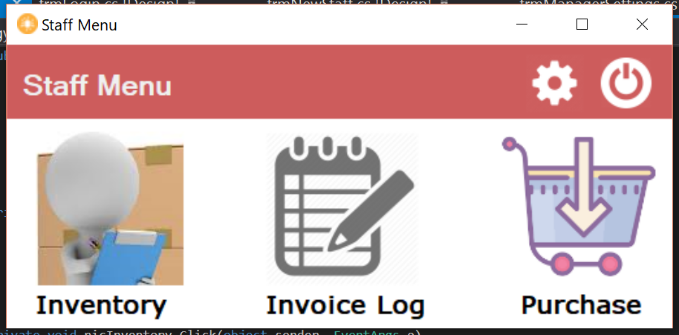
Once the checkout form has opened, the user must then enter in their payment details and click the checkout button. The program will then check if the payment information entered is valid. If the information is not valid a MessageBox will be shown notifying the user, the information is not correct. If the information is correct the program will update the stock of products in the inventory and record the invoice information into the database. Following this success, a MessageBox will be shown notifying the user of a successful transaction. At this point the program will then check if the user is a staff member or customer and the checkout form will close. If the user is a staff member and the staff menu will display, whilst, if the user is a customer, the login form will be shown.

# 4.0 Part B – Changes to implementation

### 4.1 DESIGN

The overall design of all forms was changed to better suit the program. The old design provided little space on the side of the program where buttons and text could be placed making the design inconvenient. With the new design, text is now able to sit in the header of the form, meaning instructions or the menu name could be placed in the header, allowing better communication and overall design. All major forms that used the old sidebar design now use the new header design.

**Old Design** **New Design**



### 4.2 Purchase form

Due how most POS checkout machines generally operate the quantity column in the cart of the purchase form have been removed. In most situations at a normal POS machine, when more than one of the same item is scanned in, the product is added like a separate product and is not grouped together. This change is to make sure the machine falls into the same standards required and set by other POS systems. This will ensure, when customer use the POS machine, they are not confronted by different operation standards.

### 4.3 Database design

During the implementation of the program, it was decided that design of the database should change to make the programming and SQL queries simpler to ensure that data is not lost due to the design being over complicated. This in turn has also made the implementation quicker to ensure that the product is delivered completed and has meant that more time was able to be dedicated into fixing bugs and improving other forms before the due date. During the implementation the user and staff tables were changed as many records were being duplicated across both tables. Not to mention the difficulty of the login form as the form would need to check records from two separate table at login for matching usernames and passwords. Therefore, to ensure that the final product was at the highest standards and bug free the tables were redesigned so staff id became a entity of the users table and the staff table contained information for staff only. This change means that every staff is a user but not every user is a staff as not all users will be assigned a staff id.

Another change that was made to the databases was the change from using separate order and invoice tables as this would have created too many problems and extended programming time that was not available. So to ensure that the program was finished on time, the tables were redesigned so all entities part of the orders table become part of the invoices table (Except for the OrderID entity). With this change, it would mean that the LineNo and InvoiceID would form the primary key of the table as the InvoiceID would be repeated multiple times. This change meant that a GROUP BY statement could be used in the SQL queries speeding up implementation times rather than using separate queries on multiple tables.

### 4.4 Database Names

Due to the nature of Microsoft SQL Studio, when tables were deleted and recreated, occasionally some names were not able to be reused due to an error claiming that the database name was already in use. Because of this some database names have changed including the ‘Staff’ database to ‘AllStaff’, ‘Users’ to ‘AllUsers’

### 4.5 Login Form

Because the software is intended to be used as a POS machine, if the program is closed, this could create problems for customers as they cannot buy items creating an inconvenience to the business. To solve this, the login form was changed by disabling the login form X button to make sure that users cannot close the form. Furthermore, the disabling of alt + f4 has been implemented to ensure that users cannot close the form through hot keys. Instead to close the program, the user must open task manager and end the task. With this design change, the business can expect less program downtime as it becomes more difficult for the user to close the program.

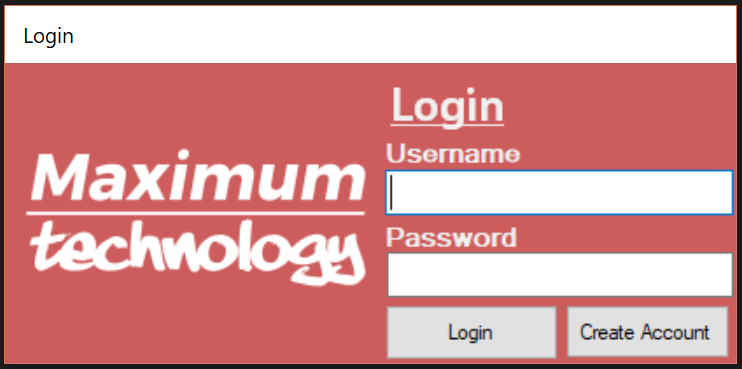
### 4.6 Calendars

All calendars that were used in the design were removed from the forms as they created errors during the implementation stage as the selected date in the calendar could not be changed programmatically to reflect data in the database. However, to ensure that the functionality that was instilled by the calendar is not lost, ‘DateTime Pickers’ are now being used as they provide a graphical interface similar to the calendar (making the user experience incredibly easy) for selecting a date on a calendar and through code the selected value can be changed. This means that the solution is a zero-compromise solution as all functionality is still present.

# 5.0 Forms

## 5.1 Login Form

### 5.1.1 Design



### 5.1.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System[memes].Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmLogin : Form

{

public frmLogin()

{

InitializeComponent();

this.KeyPreview = true;

List<Form> openForms = new List<Form>();

foreach (Form f in Application.OpenForms)

openForms.Add(f);

foreach (Form f in openForms)

{

if (f.Name != "frmLogin")

f.Close();

}

}

private void testfunction(object sender, EventArgs e)

{

string connectionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

string username = txtUsername.Text;

string password = txtPassword.Text;

sql = "SELECT AllUsers.UserID, AllUsers.Username, AllUsers.Password, AllUsers.Firstname, AllUsers.Lastname FROM AllUsers WHERE Username='" + username + "' AND Password= '" + password + "';";

connection = new SqlConnection(connectionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

if (dataReader.HasRows == false)

{

MessageBox.Show("Login failed. Please try again.");

}

else

{

while (dataReader.Read())

{

if (txtUsername.Text == dataReader.GetValue(1).ToString() && txtPassword.Text == dataReader.GetValue(2).ToString())

{

User.ID = (long)Convert.ToDouble(dataReader.GetValue(0).ToString());

User.Username = dataReader.GetValue(1).ToString();

User.Firstname = dataReader.GetValue(3).ToString();

User.Lastname = dataReader.GetValue(4).ToString();

sql = "SELECT AllStaff.StaffID, AllStaff.AccessLevel FROM AllStaff WHERE ID='" + User.ID + "';";

SqlConnection connection2 = new SqlConnection(connectionString);

connection2.Open();

SqlCommand command2 = new SqlCommand(sql, connection2);

SqlDataReader dataReader2 = command2.ExecuteReader();

if (dataReader2.HasRows == true)

{

try

{

while (dataReader2.Read())

{

User.AccessLevel = Convert.ToInt32(dataReader2.GetValue(1).ToString());

User.isStaff = true;

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

this.Hide();

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error trying to connect to the server. Error: " + ex);

}

}

else

{

User.isStaff = false;

frmPurchase frm = new frmPurchase();

frm.Show();

this.Hide();

}

}

else

{

MessageBox.Show("Login unsuccessful. Please try again.");

break;

}

}

dataReader.Close();

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection to server. Error: " + ex);

}

}

private void frmLogin\_FormClosing(object sender, FormClosingEventArgs e)

{

frmLogin frm = new frmLogin();

frm.Show();

}

private void frmLogin\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Alt && e.KeyCode == Keys.F4)

{

e.Handled = true;

}

}

private void btnSignUp\_Click(object sender, EventArgs e)

{

frmSignup frm = new frmSignup();

frm.Show();

}

private void frmLogin\_FormClosed(object sender, FormClosedEventArgs e)

{

Environment.Exit(0);

}

}

}

### 5.1.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User clicks on login button after entering in login credentials | User logs in and the corresponding form appears | Program crashes with error, no such entity exists | UserID entity name was spelt wrong | Replaced incorrect entity name with correct spelling | Yes |
| 2 | User clicks on login button after entering in login credentials | User logs in and the corresponding form appears | Program crashes with error, index outside the bounds of the array | Entity name in WHERE condition was spelt with capital letters | Replaced incorrect casing with correct casing. | Yes |
| 3 | User clicks on login button after entering in login credentials | User logs in and the corresponding form appears | Program crashes with error, index outside the bounds of the array | If statement checking credentials against sql data just received was calling function txtPassword not txtPassword.Text comparing objects not values | Changed if statement to check txtPassword to txtPassword.Text | Yes |
| 4 | User clicks on login button after entering in login credentials | User logs in and the purchase form appears | Program crashes with an error attempting to locate and connect to the database | Server roles were not set for sql server user used, meaning the account had no permissions to certain databases | Changed server roles for user in MS SQL Server Management Studio to allow the user to edit and view the DB | Yes |
|  | User clicks on login button after entering in login credentials | User logs in and the purchase form appears | User logs in and the purchase form appears | N/A | N/A | No |
| 5 | Login with staff account | User logs in and staff menu appears | Login was successful and staff menu appeared | N/A | N/A | No |
| 6 | Customer enters in their login details and clicks login button to login | Customer logs in and the purchase menu appears | Login is unsuccessful as no info is returned from the database | Join statement in SQL statement used for login algorithm joins staff table with the users table and causes the server to not return any data as the statement does not work with the algorithm | Move staff id checking outside while loop and create a new connection and query after if statement checking if user has a staff id to get staff information | Yes |
| 7 | Customer enters in their login details and clicks login button to login | User logs in and purchase menu appears | Login was successful and purchase menu appeared | N/A | N/A | No |
| 8 | User clicks the X button in the top right-hand program | Nothing | Program closes | Code was missing | Disabled X button in form properties | Yes |
| 9 | User clicks the X button in the top right-hand program | Nothing | Nothing | N/A | N/A | No |
| 9 | User enters correct username and password in wrong case | MessageBox is shown with text "Login Unsucessfull" | MessageBox is shown with text "Login Unsucessfull" | N/A | N/A | No |
| 9 | User enters in a password not associated with username | MessageBox is shown with text "Login Unsucessfull" | MessageBox is shown with text "Login Unsucessfull" | N/A | N/A | No |

### 5.1.4 SQL Queries

#### Query #1

Used to check if username and password matchup in database together when user is logging in by checking if a username and password are part of the same record.

"SELECT AllUsers.UserID, AllUsers.Username, AllUsers.Password, AllUsers.Firstname, AllUsers.Lastname FROM AllUsers

WHERE Username='" + username + "' AND Password= '" + password + "';";

#### Query #2

Used to see if the user that just logged in is a staff member or not and grabbing the associated staff access level by checking if their UserID exists in the staff database. Any users that are staff will have their UserID in the staff table so by querying the table for a specific UserID will show if the user is a staff member. If no results are returned they are not staff, if results are returned they are.

sql = "SELECT AllStaff.StaffID, AllStaff.AccessLevel

FROM AllStaff WHERE ID='" + User.ID + "';";

### 5.1.5 Evaluation

#### 5.1.5.1 Testing and Programming

Through much testing and bug fixing the program was eventually programmatically successful as all bugs were fixed. One of the main problems that made this form difficult was trying to find a method to login staff and users. Time was limited, and an advanced complicated algorithm could be too long and have a higher chance of failure or bugs due to more complicated code. So, it was decided to change the database design as stated in section 4.3. This change to instead assign users a staff id and not completely rely on separate ids made the login system more manageable to code meaning the number of possible bugs or errors was lowered. This, in turn, makes the final improved program improved as the chances for bugs are lowered creating a more stable program.

As seen in early records of the testing above, even with the database changes, the login algorithm was still complicated due to the need to check if the user logging in was a staff member. This created many problems where code needed to be moved and the algorithm needed to be modified as seen in test id 6. By looking through the testing logs, it is also visible that overtime the algorithm still needed to be modified to fit changing demands. This is evident in test id 5 where the staff login was working, however, in test id 6 the initial query for general login does not work. However, after much testing, the algorithm now works with no currently visible problems.

In future, it is recommended that SQL objects be reused across multiple queries to increase program efficiency and reduce the number of possible bugs. With the current code, each new query is assigned new SQL objects, however, this is inefficient as SQL objects could be reused across queries. This would allow the program to run more efficiently on lower end hardware and would declutter code making it more readable for future programmers that wish to update the program. This would benefit the business as the self-service machine hardware selected could be of lower performance generally meaning lower cost saving the business money, therefore benefiting the business.

Overall the login was successful as the algorithm was able to check if the username and password were correct and show the corresponding form depending if the user is a staff member or not. It would be recommended in the future to make the system more secure, using encryption or hashing is an example that if implemented could make the system more versatile and secure. Another future possibility could be to use a third-party solution specifically designed as a login system that could be implemented into the program rather than using a proprietary solution that would not be to the same standards as specific login solution.

#### 5.1.5.2 Design

The design of the form was straight forward yet effective with a simple design to reflect the simple nature and procedure of the form. The login form uses a simple red background with contrasting white and grey colours, as well as a logo to the left side of the form. With two simple buttons its hard to find the login form difficult or confusing to navigate making the form a good greeting for customers when they go to checkout their goods.

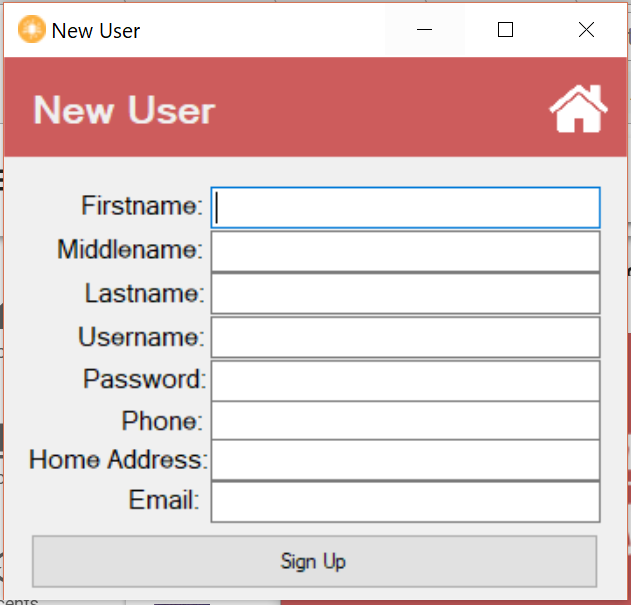
The first recommendation that could be made would be to place text inside the username and password text boxes that are lightly shaded indicating the input for the box. This text would disappear when the box has been clicked. This would in general, make the form slightly easier to navigate but is not necessary. This would only be recommended as a cosmetic addition and should not be considered a major issue.

The only other recommendation that could be considered would be to give the option to customers and staff the opportunity of being issued a key card that when scanned at the login form at a self-service machine would allow the user to login to their account without needing to enter any details using the computer. This would greatly increase the overall efficiency of the machines as a singular person would most likely take less time at the machine as they would not be required to enter their login details. This would in turn be beneficial to the business as more people would be able to pass through the self-service checkouts in a given time frame, possibly boosting sales and increasing customer satisfaction, in turn incentivising them to return to the store again.

Overall this form due to its simple and limited feature set will leave the users with a good impression of the program leaving them satisfied as the navigation of the form is simple and quick. Only one recommendation could be made, though as stated previously, the first recommendation should not be considered major as it is purely a design tweak, however, thought should be considered into the second recommendation as issuing key cards could drastically help the business in terms of customer satisfaction, experience and business efficiency.

## 5.2 Signup Form

### 5.2.1 Design



### 5.2.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmSignup : Form

{

public frmSignup()

{

InitializeComponent();

}

private void btnSignUp\_Click(object sender, EventArgs e)

{

if (txtFirstname.Text == "" || txtLastname.Text == "" || txtUsername.Text == "" || txtPassword.Text == "")

{

MessageBox.Show("There are missing feilds. Please try again");

}

else

{

string connectionString = null;

string sql = null;

connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

SqlConnection connectionUsername;

SqlCommand command = new SqlCommand();

SqlDataReader dataReaderUsername;

connectionUsername = new SqlConnection(connectionString);

sql = "SELECT \* FROM AllUsers WHERE Username='" + txtUsername.Text + "';";

try

{

connectionUsername.Open();

command = new SqlCommand(sql, connectionUsername);

dataReaderUsername = command.ExecuteReader();

if (dataReaderUsername.HasRows == true)

{

MessageBox.Show("That username already exists. Please choose a different username.");

}

else

{

dataReaderUsername.Close();

SqlCommand command2;

SqlDataReader dataReader;

SqlConnection connection = new SqlConnection(connectionString);

sql = "INSERT INTO AllUsers VALUES ('" + txtUsername.Text + "', '" + txtPassword.Text + "', '" + txtFirstname.Text + "', ";

if (txtMiddlename.Text == "")

sql += "null, '";

else

sql += "'" + txtMiddlename.Text + "', '";

sql += txtLastname.Text + "', ";

if (txtPhone.Text == "")

sql += "null, ";

else

sql += "'" + txtPhone.Text + "', ";

if (txtHomeAddress.Text == "")

sql += "null, '";

else

sql += "'" + txtHomeAddress.Text + "', '";

sql += txtEmail.Text + "');";

try

{

connection.Open();

command2 = new SqlCommand(sql, connection);

dataReader = command2.ExecuteReader();

dataReader.Close();

MessageBox.Show("You have successfully signed up.");

this.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection to server. Error: " + ex);

}

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error checking if the username already exists. Please try again. \n\n Error: " + ex);

}

}

}

private void picLogout\_Click(object sender, EventArgs e)

{

this.Close();

}

private void frmSignup\_FormClosed(object sender, FormClosedEventArgs e)

{

frmLogin frm = new frmLogin();

frm.Show();

}

}

}

### 5.2.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User enters username that already exists | MessageBox is shown notifying the user 'That username already exists. Please choose a different username.' | New row is inserted into database with duplicated username | Sql does not automatically stop duplicates | Create new code that checks if username already exists by quering the database with the users input | Yes |
| 2 | User enters username that already exists | MessageBox is shown notifying the user 'That username already exists. Please choose a different username.' | MessageBox is shown notifying the user 'That username already exists. Please choose a different username.' | N/A | N/A | No |
| 3 | User does not enter a phone number | null is inserted into the database | String 'null' is inserted into the database rather than a null value | String modification statement is adding ''null'' instead of 'null' | Change string modification if statement to add 'null' not "null" | Yes |
| 4 | User does not enter a phone number | null is inserted into the database | null is inserted into the database | N/A | N/A | No |
| 5 | User does not enter a email address | null is inserted into the database | null is inserted into the database | N/A | N/A | No |
| 6 | User does not enter an address | null is inserted into the database | null is inserted into the database | N/A | N/A | No |
| 7 | User clicks on home button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 8 | User clicks on X button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |

### 5.2.4 SQL Queries

#### Query #1

Used to check if the username entered during signup is already exits in the database by sending it the username entered and using a where statement. If no results are returned, then the username is not currently in usage. If results are returned the Username is in use.

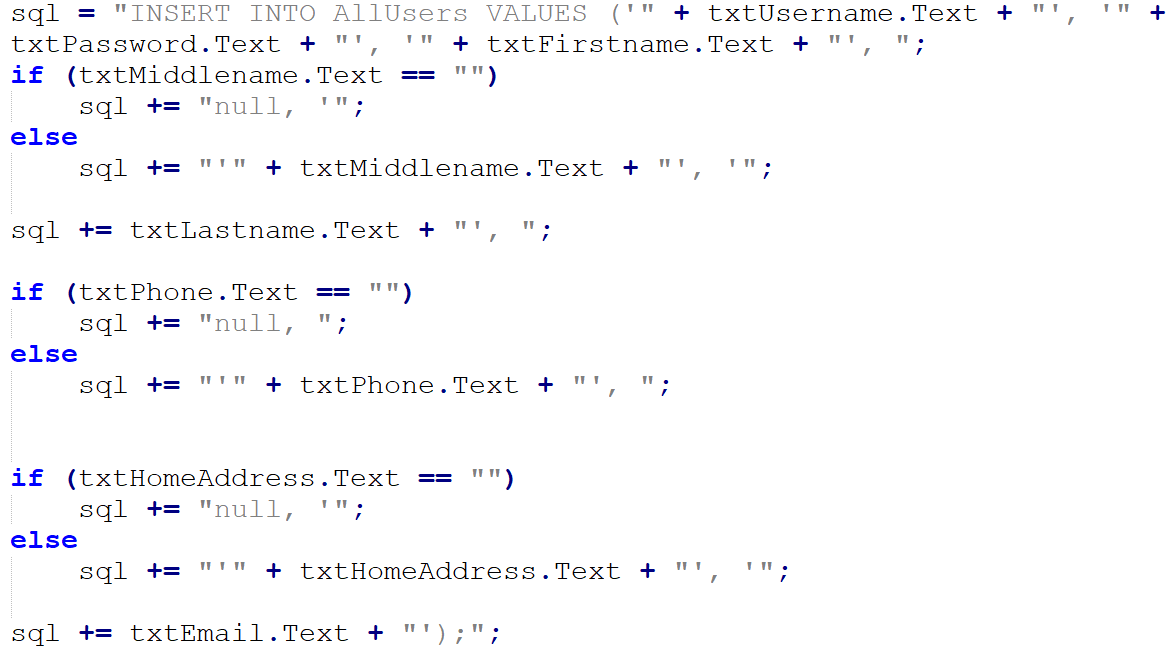
sql = "SELECT \*

FROM AllUsers

WHERE Username='" + txtUsername.Text + "';";

#### Query #2

Used to insert a new user into the database. Inserts all fields and checks to see if middle name, password and home address were entered. Each field is checked individual and if the field was filled in the value entered is inserted, if the field was left blank, ‘null’ was inserted.



### 5.2.5 Evaluation

#### 5.2.5.1 Testing and Programming

#### This form was one of the more challenging forms due to the need to handle null values. Before the code to the form was written much time went into researching methods of handling null values as it is not a compulsory requirement that a user enters their address, phone number and middle name as some people may not have these. However, it was crucial that it works for the final product and, as hoped, on the first attempt the code worked. If this didn’t work problems could have occurred where addresses, phone numbers, and middle names would have had to been taken out of the database as it cannot be guaranteed that everyone had one of these, as keeping these fields null with no way to test if they are null could have led to many problems. However, after much testing, the form works well and to standards without any noticeable problems causing the program to be unusable.

#### It is recommended though in future that an email and phone verification system be implemented to ensure that unknowing peoples email addresses or phone numbers are signed up to the system. This would just act as an extra layer of security to also ensure that fake email addresses are not entered into the system. This checking of credentials could be beneficial to the business when the correct user details are needed for a legal process such as warranty or other issues.

#### Another recommendation that could be made would be to utilise a third-party web API to check to see if the address entered by each user is a valid address. This could be a major issue in the future as the user may accidentally enter the wrong address causing their delivery to subsequently not be deliverable as the address may not exist. This could create a backlash or legal problems with customers as customers may their rights as a consumer and report the business. It is strongly recommended this be implemented in the future.

#### It is also recommended in future that less SQL related objects be created in the form code as some SQL objects (such as both SQLConnection objects) in the form could have had one object used twice rather than recreating new instances of them for each use. This would ensure a better and more efficient program that would run smoother. This in total could then increase the total user experience of the program.

#### Overall the form performs what its intended use was as users are able to register their account and log in straight afterward meeting the initial premise of the form. There are currently no bugs in the form, but many recommendations could be considered for future updates/releases.

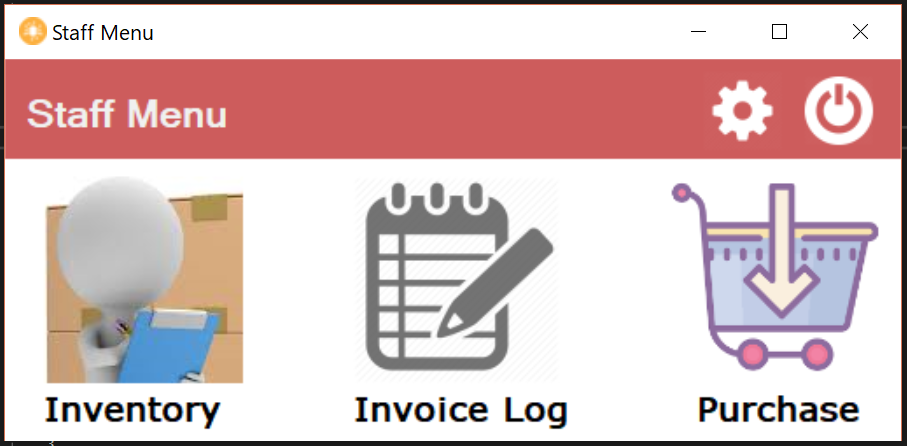
#### 5.2.5.2 Design

Since this will be one of the first forms a user will see when using the checkout for the first time, it is important that the process is understandable and intuitive. If the form is too difficult or frustrating customers may be inclined to just leave the store and shop elsewhere, so it is crucial that this form is easy to use and practical. However, this has been achieved as the form is simple yet effective, with everything ordered from start to finish, input boxes and their corresponding labels exist and a submit button exists at the bottom where the user can create their account once they have finished entering their login details. Furthermore, if for some reason, the user needs to go back there is a home button located at the top of the form that the user can click to go back to the previous form, or they can use the X located in the top right-hand corner.

This form is simple and easy to use, and the design allows the user to easily understand the flow of procedure. By the user starting at the top of the form and making their way down the corresponding elements the user can easily complete and process the form without major difficulty.

## 5.3 Staff Menu Form

### 5.3.1 Design



### 5.3.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmStaffMenu : Form

{

public frmStaffMenu()

{

InitializeComponent();

if (User.AccessLevel != 1)

{

picSettings.Hide();

}

}

private void picSettings\_Click(object sender, EventArgs e)

{

frmManagerSettings frm = new frmManagerSettings();

frm.Show();

this.Hide();

}

private void picInventory\_Click(object sender, EventArgs e)

{

frmInventory frm = new frmInventory();

frm.Show();

this.Hide();

}

private void picTransactions\_Click(object sender, EventArgs e)

{

frmInvoiceLog frm = new frmInvoiceLog();

frm.Show();

this.Hide();

}

private void frmStaffMenu\_FormClosed(object sender, FormClosedEventArgs e)

{

Process.Start(Environment.CurrentDirectory.ToString() + @"\MaximumTechnology.exe");

Environment.Exit(0);

}

private void picPurchase\_Click(object sender, EventArgs e)

{

frmPurchase frm = new frmPurchase();

frm.Show();

this.Hide();

}

private void picLogout\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

### 5.3.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User closes the form using the X in the right-hand corner or the logout button | User closes the form, they are signed out and the login form appears | Form does not close, and the login form is opened multiple times infinitely | Form close event can cause some code repeatedly to execute if error handling is not implemented | Changed from using the FormClosing event to FormClosed event | Yes |
| 2 | User closes the form using the X in the right-hand corner or the logout button | User closes the form, they are signed out and the login form appears | User closes the form, they are signed out and the login form appears | N/A | N/A | No |
| 3 | Manager logs in and settings button is shown | Manager logs in and settings button is shown | Manager logs in and settings button is shown | N/A | N/A | No |
| 4 | Staff member other than manager logs in and settings button is hidden | Staff member other than manager logs in and settings button is hidden | Staff member other than manager logs in and settings button is hidden | N/A | N/A | No |
| 5 | Manager clicks staff settings button | Staff settings form is displayed | Staff settings form is displayed | N/A | N/A | No |

### 5.3.4 SQL Queries

There were no SQL queries used for this form.

### 5.3.5 Evaluation

#### 5.3.5.1 Testing and Programming

The programming and testing of this form were quite simple compared to other forms due to the simplicity and usage. The form acts only as a menu, using buttons to open other forms, making this form incredibly easy and quick to code. Because of coding feasibility, the form is currently bug-free and stable.

The only problem that was encountered in the programming of this form was in test id 1, in which, when the form closed the login form would open multiple instances infinitely until the program was ended in task manager. To fix this error, research into the event handler being used was conducted and it was deduced that the FormClosed event would work better than the FormClosing event. If this error was not fixed, it could have created many critical errors that if included in the final program could create major problems for the business and customers. Instead, now the menu functions normally without problems.

Once you assess the problems that have been solved and the demands of the form it is evident that the form does what it was designed to do and that it is a functional menu. All buttons work with no bugs or known issues and the manager settings button is shown depending on the access level of the staff member logged in achieving the goals of this form.

#### 5.3.5.2 Design

As the main form for navigation around the program as a staff member, it was crucial that the design of this form was easy enough that minimal training for staff would be required, but instead that the form was intuitive enough that most people would be able to use the program with little experience or training. However, the form does manage to provide an easy user face as it integrates simple pictures that act as buttons/links to other forms with corresponding text that identify the form.

Due to the simplicity of the form and its limited feature set, there is no recommendations or suggested improvements for this form.

This form is simple and easy to use comprising of only six buttons and acting as only a menu to other forms making the design of the form simple and intuitive. Because the form is so simple, the business will benefit from the lowered staff training needed to navigate the form. This form will when implemented serve well as a menu form thanks to simplicity which in turn will benefit the business.

## 5.4 Manager Settings Form

### 5.4.1 Design

### 

### 5.4.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmManagerSettings : Form

{

public long id = 0;

public frmManagerSettings()

{

InitializeComponent();

}

private void frmManagerSettings\_Load(object sender, EventArgs e)

{

refreshDGV();

}

private void dgvStaff\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

var row = dgvStaff.Rows[e.RowIndex];

string staffid = row.Cells[1].Value.ToString();

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT \* FROM AllUsers JOIN AllStaff ON AllStaff.ID = AllUsers.UserID WHERE AllStaff.StaffID='" + staffid + "';";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

lblStaffID.Text = dataReader.GetByte(dataReader.GetOrdinal("StaffID")).ToString();

txtFirstname.Text = dataReader.GetString(dataReader.GetOrdinal("Firstname"));

try

{

txtMiddlename.Text = dataReader.GetString(dataReader.GetOrdinal("Middlename"));

}

catch { }

txtLastname.Text = dataReader.GetString(dataReader.GetOrdinal("Lastname"));

try

{

txtAddress.Text = dataReader.GetString(dataReader.GetOrdinal("HomeAddress"));

}

catch { }

try

{

txtPhone.Text = dataReader.GetString(dataReader.GetOrdinal("Phone"));

}

catch { }

txtEmail.Text = dataReader.GetString(dataReader.GetOrdinal("Email"));

txtUsername.Text = dataReader.GetString(dataReader.GetOrdinal("Username"));

txtPassword.Text = dataReader.GetString(dataReader.GetOrdinal("Password"));

numAccessLevel.Value = dataReader.GetByte(dataReader.GetOrdinal("AccessLevel"));

comStatus.SelectedItem = dataReader.GetString(dataReader.GetOrdinal("Status"));

txtPosition.Text = dataReader.GetString(dataReader.GetOrdinal("Position"));

id = dataReader.GetInt64(dataReader.GetOrdinal("UserID"));

DateTime DOB = dataReader.GetDateTime(dataReader.GetOrdinal("DOB"));

calDOB.Value = DOB;

}

dataReader.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection to server. Error: " + ex);

}

}

}

private void btnSave\_Click(object sender, EventArgs e)

{

if (id != 0)

{

long value;

if (long.TryParse(txtPhone.Text, out value) || txtPhone.Text == "")

{

if (txtUsername.Text.Contains(' ') == false || txtPassword.Text.Contains(' ') == false || txtEmail.Text.Contains(' ') == false)

{

if (txtFirstname.Text == "" || txtLastname.Text == "" || txtUsername.Text == "" || txtPassword.Text == "" || txtPosition.Text == "" || comStatus.Text == "" || numAccessLevel.Text == "")

{

MessageBox.Show("There are missing feilds. Please try again.");

}

else

{

try

{

string sql = "UPDATE AllUsers SET"

+ " Firstname ='" + txtFirstname.Text + "', " + " Middlename =";

if (txtMiddlename.Text == "")

sql += "null";

else

sql += " '" + txtMiddlename.Text + "'";

sql += ", Lastname ='" + txtLastname.Text + "', HomeAddress =";

if (txtAddress.Text == "")

sql += "null, ";

else

sql += " '" + txtAddress.Text + "', ";

sql += "Phone = ";

if (txtPhone.Text == "")

sql += "null";

else

sql += " '" + txtPhone.Text + "'";

sql += ", Email ='" + txtEmail.Text + "', "

+ " Username ='" + txtUsername.Text + "', "

+ " Password ='" + txtPassword.Text + "' WHERE UserID = '" + id + "';";

string connectionString = null;

SqlConnection connection;

SqlCommand command;

connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connectionString);

connection.Open();

command = new SqlCommand(sql, connection);

command.ExecuteReader();

connection.Close();

DateTime selectedDate = Convert.ToDateTime(calDOB.Value.Date);

if (selectedDate >= DateTime.Now.AddDays(-7))

{

MessageBox.Show("Date selected was invalid.");

}

else

{

sql = "UPDATE AllStaff SET"

+ " Position ='" + txtPosition.Text + "', "

+ " Status ='" + comStatus.Text + "', "

+ " AccessLevel ='" + Convert.ToByte(numAccessLevel.Value) + "', "

+ " DOB ='" + selectedDate.ToString("yyyy-MM-dd") + "' WHERE StaffID = '" + Convert.ToInt32(lblStaffID.Text) + "';";

SqlConnection connectionStaff = new SqlConnection(connectionString);

connection.Open();

SqlCommand commandStaff = new SqlCommand(sql, connection);

commandStaff.ExecuteReader();

MessageBox.Show("User information was updated successfully");

refreshDGV();

}

}

catch (Exception ex)

{

MessageBox.Show("There was a problem updating user information. Error: " + ex);

}

}

}

else

{

MessageBox.Show("Make sure your Username, Password and Email does not contain any spaces");

}

}

else

{

MessageBox.Show("Make sure your phone number only contains numbers");

}

}

else

{

MessageBox.Show("Please select a user");

}

}

private void refreshDGV()

{

dgvStaff.Rows.Clear();

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT AllUsers.Firstname, AllUsers.Lastname, AllStaff.StaffID, AllStaff.Position FROM AllUsers JOIN AllStaff ON AllStaff.ID = AllUsers.UserID;";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

this.dgvStaff.Rows.Add(dataReader.GetValue(0).ToString() + " " + dataReader.GetValue(1).ToString(), dataReader.GetValue(2).ToString(), dataReader.GetValue(3).ToString());

}

dataReader.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection to server. Error: " + ex);

}

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

frmNewStaff frm = new frmNewStaff();

frm.Show();

}

private void frmManagerSettings\_FormClosed(object sender, FormClosedEventArgs e)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

}

private void picLogout\_Click(object sender, EventArgs e)

{

this.Close();

}

private void btnRemoveMember\_Click(object sender, EventArgs e)

{

if (lblStaffID.Text != "")

{

try

{

if (numAccessLevel.Value != 1)

{

string connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

SqlConnection connection = new SqlConnection(connectionString);

connection.Open();

string sql = "DELETE FROM AllStaff WHERE StaffID= " + lblStaffID.Text + ";";

SqlCommand command = new SqlCommand(sql, connection);

command.ExecuteReader();

connection.Close();

refreshDGV();

}

else

{

MessageBox.Show("You cannot delete the manager");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

else

{

MessageBox.Show("Please select a user");

}

}

}

}

### 5.4.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | Form is opened | Form opens, DataGridView is filled in | Program crashes with SQL Error Exception | Column names were invalid | Recorrect column 'userID' to 'UserID' | Yes |
| 2 | Form is opened | Form opens, DataGridView is filled in | Program crashes with invalid cast exception | Invalid cast exception trying use function getByte instead of getString | Change getByte to getString | Yes |
| 3 | Form is opened | Form opens, DataGridView is filled in | Form opens, DataGridView is filled in | N/A | N/A | No |
| 4 | User clicks on the last row of DataGridView | Nothing | Program crashes | Last row is a row that allows users to add rows by default | Set property to allow users adding rows to false in DataGridView properties | Yes |
| 5 | User clicks on the last row of DataGridView | User can’t click it because it no longer exists | User can’t click it because it no longer exists | N/A | N/A | No |
| 6 | User clicks on row in datagridview | Input fields are filled with corresponding user data that was clicked | Input fields are filled with corresponding user data that was clicked | N/A | N/A | No |
| 7 | User clicks on header row in DataGridView | Rows are sorted correspondingly | Rows are sorted correspondingly | N/A | N/A | No |
| 8 | User clicks save button after editing user details | User information is updated in the database | User information is updated in the database | N/A | N/A | No |
| 9 | User clicks save button where no user has been selected for editing in the DataGridView | User information is updated in the database | User information is updated in the database | N/A | N/A | No |
| 10 | User clicks on home button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 11 | User clicks on X button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 12 | Manager clicks remove staff member button after selecting a user | Staff member is removed and the DataGridView is refreshed to show changes | Staff member is removed and the DataGridView is refreshed to show changes | N/A | N/A | No |
| 13 | Manager clicks remove staff member button when a user hasn’t been selected | MessageBox is shown stating “Please select a user” | MessageBox is shown stating “Please select a user” | N/A | N/A | No |
| 14 | Manager clicks remove staff member button when the manager has been selected | MessageBox is shown stating “You cannot delete the manager” | MessageBox is shown stating “You cannot delete the manager” | N/A | N/A | No |

### 5.4.4 SQL Queries

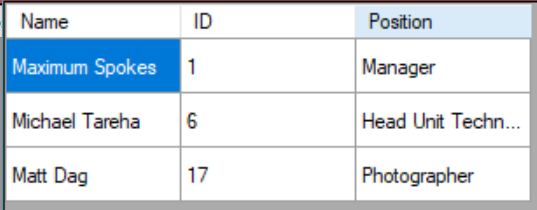
#### Query #1

Used to select all users that are staff members registered in the database. The statement also uses a JOIN statement so information from both databases may be accessed simultaneously.

sql = "SELECT AllUsers.Firstname, AllUsers.Lastname, AllStaff.StaffID, AllStaff.Position

FROM AllUsers

JOIN AllStaff ON AllStaff.ID = AllUsers.UserID;";



#### Query #2

Used to select a specific users’ information when a user from the DataGridView is clicked. Data from query is shown in the corresponding fields where the user may then edit the data.

sql = "SELECT \*

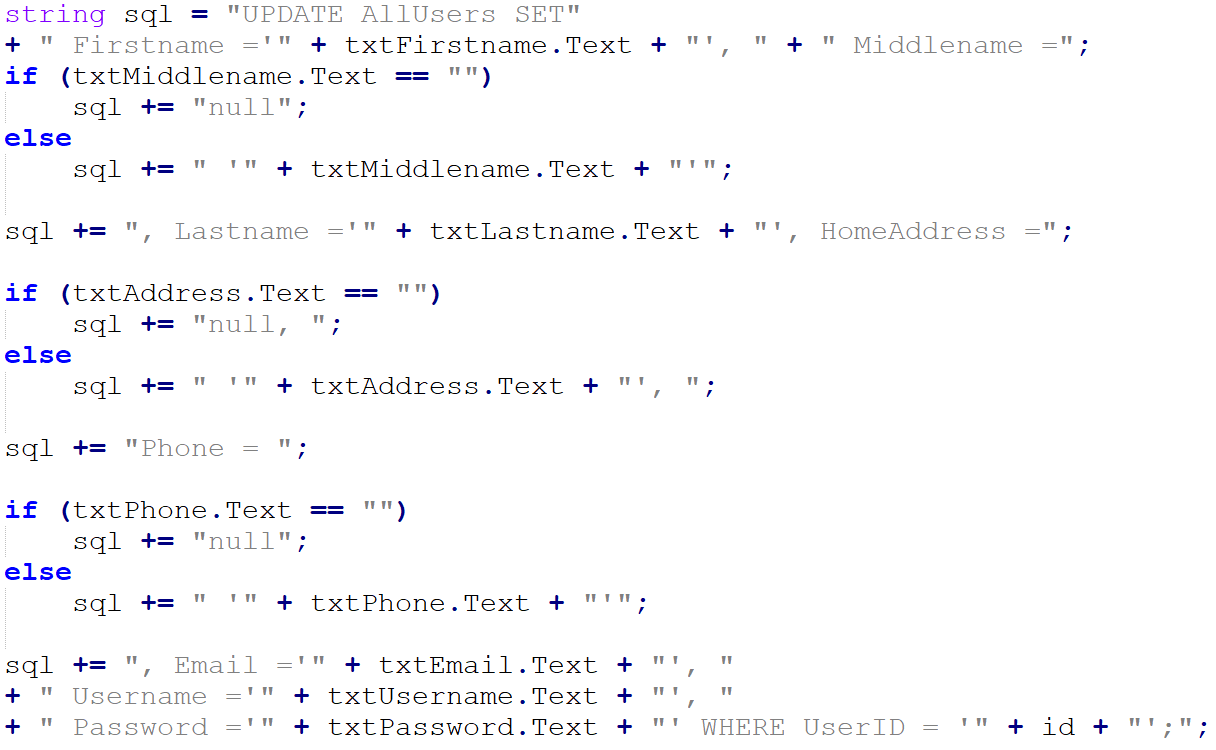
FROM AllUsers

JOIN AllStaff ON AllStaff.ID = AllUsers.UserID

WHERE AllStaff.StaffID='" + staffid +"';";

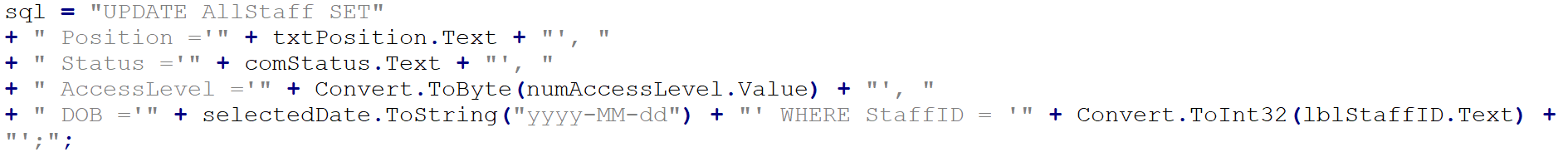
#### Query #3

Used to update a specific staff member generally user information once the manager has finished editing the corresponding fields. The query is constructed using if statements to check to see if the user fields have been filled out, so that the values entered into the fields, or null is inserted into the database.



#### Query #4

Used for updating staff information once the Manager has edited any fields necessary. The statement below uses a WHERE statement to update information of a specific user.



### 5.4.5 Evaluation

#### 5.4.5.1 Testing and Programming

Due to the complexity of this form, the amount of time invested in this form was much greater than other forms. The increased complexity is due to the number of SQL queries that all needed to work together to produce results, not to mention the length of them. Multiple problems were encountered trying to use a calendar as described in section 4.6 which were eventually addressed (also described in section 4.6) and overcome complementing the success of the form programmatically. If the aforementioned problems with the calendar were not overcome the current database and program may not have included a DOB section due to the complexity of correctly implementing dates with SQL server. This intern would have then negatively affected the programs ability to store staff details, a crucial feature of the program. However, after much bug fixing and tweaking the form now works correctly and up to standards, with all functions working and null values being handled properly.

However, in the future, it is recommended that less duplicate SQL Objects be used in future updates/releases as this decreases the efficiency of the program. If the same object was instead used multiple times rather than a new object for every use, the total efficiency of the program could increase due to the less ram and CPU being utilised creating more objects.

It is also recommended that more functions are used in future as current methods implemented are long and could be broken into corresponding functions that could be used multiple times in many different situations or parts of the form. By implementing a more function-oriented code design, the code would become easier to read and duplicates of code would become limited. This would lead to overall a more robust program and an easier update process in the future. The usage of a DataGridView refresh function (refreshDGV), for refreshing the DataGridView with new data is a good example and one instance where using separate functions was beneficial.

Overall the form performs what it was intended for, which is mainly the editing of staff members. Though there are improvements to be made, the form performs adequately without any currently known bugs.

#### 5.4.5.2 Design

The manager settings form was luckily one of the better designed forms due to the ease of use by just clicking on the user you wish to edit. Furthermore, the input of the information was also well thought out when referring to using a DateTime picker rather than a manual keyboard entry.

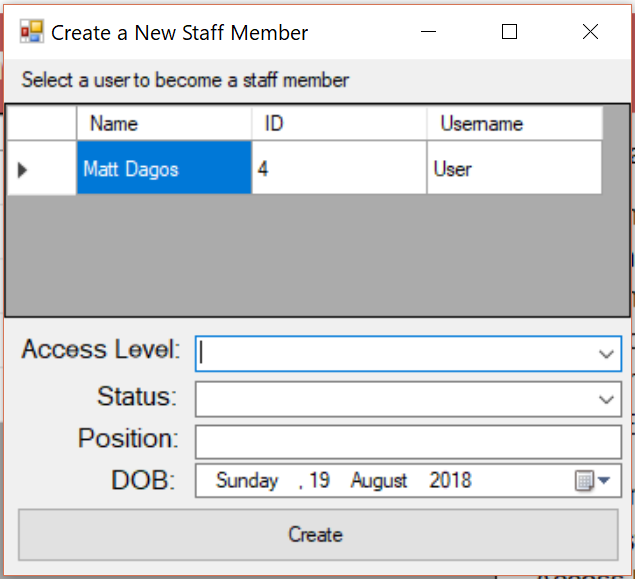
However, in future it is recommended more specific address entry methods are used as no specific testing is occurring to the address information being entered, however, if the form next time was designed to accommodate a more specific address entry the possibility for the wrong format or incorrect information would be lowered. An example could be to split the address into different form elements such as house number, street name, suburb, city and postal code.

Another recommendation that could be considered in later updates would be to add a search feature where users would be able to search for staff rather than manually browsing through the DataGridView which can be tedious and a waste of time the business may not have. Implementing a search bar will be beneficial to the business as the time that would have been wasted searching users manually from the DataGridView can be instead invested into other tasks around the office/business increasing the total efficiency of the business. Not to mention, including a search bar the navigation of the form is improved drastically leaving the user with a better overall experience.

If these recommendations were considered and acted upon the forms functionality could be improved significantly, saving time for the business which could be spent elsewhere. Overall the form was well thought out with its linear based procedural design (to enter data go in a straight line from the top).

## 5.5 New Staff Form

### 5.5.1 DESIGN



### 5.5.2 CODE

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmNewStaff : Form

{

public int id = -1;

public frmNewStaff()

{

InitializeComponent();

}

private void frmNewStaff\_Load(object sender, EventArgs e)

{

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = [memes]null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT AllUsers.Firstname, AllUsers.Lastname, AllUsers.UserID, AllUsers.Username FROM AllUsers WHERE UserID != ALL (SELECT ID FROM AllStaff);";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

this.dgvCustomers.Rows.Add(dataReader.GetValue(0).ToString() + " " + dataReader.GetValue(1).ToString(), dataReader.GetValue(2).ToString(), dataReader.GetValue(3).ToString());

}

dataReader.Close();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection to server. Error: " + ex);

}

}

private void dgvCustomers\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

id = Convert.ToInt32(dgvCustomers.Rows[e.RowIndex].Cells[1].Value.ToString());

}

}

private void btnCreateStaff\_Click(object sender, EventArgs e)

{

if (id != -1)

{

if (numAccessLevel.Text != "" && txtPosition.Text != "" && comStatus.Text != "")

{

try

{

string sql = "INSERT INTO AllStaff (DOB, Position, Status, AccessLevel, ID) VALUES ('" + calDOB.Value.Date.ToString("yyyy/MM/dd") + "', '" + txtPosition.Text + "', '" + comStatus.Text + "', " + numAccessLevel.Text + ", " + id + ");";

string connectionString = null;

SqlConnection connection;

SqlCommand command;

connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connectionString);

connection.Open();

command = new SqlCommand(sql, connection);

command.ExecuteReader();

connection.Close();

MessageBox.Show("New staff member successfully created");

this.Close();

}

catch (Exception ex)

{

MessageBox.Show("We ran into a problem... Error: " + ex);

}

}

else

{

MessageBox.Show("Please ensure all fields are filled ina");

}

}

else

{

MessageBox.Show("Please select a user");

}

}

}

}

### 5.5.3 Testing

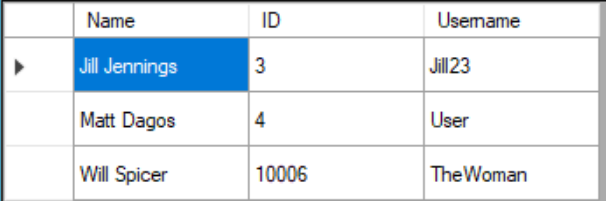
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User clicks create button once all fields have been field in and a user has been selected | Messagebox is shown notifying the user that the user was added to the database and the form closes | SQL error is returned "he conversion utility data type varchar to datetime produced a value out of range" | calDOB.Value.ToShortDateString was being used to grab the date from calDOB. This means date is not formatted to SQL server standards | Change old code to calDOB.Value.Date.ToString("yyyy/MM/dd") | Yes |
| 2 | User clicks create button once all fields have been field in and a user has been selected | Messagebox is shown notifying the user that the user was added to the database and the form closes | Messagebox is shown notifying the user that the user was added to the database and the form closes | N/A | N/A | No |
| 3 | User clicks create button once when all fields are not filled in but a user has been selected | Messagebox is shown notifying the user that not all fields were filled in | Messagebox is shown notifying the user that not all fields were filled in | N/A | N/A | No |
| 4 | User clicks create button once when no user has been selected | Messagebox is shown notifying the user that a user has not been selected | Messagebox is shown notifying the user that a user has not been selected | N/A | N/A | No |
| 5 | User clicks header row in DataGridView | Datagridview rows are ordered by that column header clicked | Datagridview rows are ordered by that column header clicked | N/A | N/A | No |
| 6 | User clicks on home button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 7 | User clicks on X button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |

### 5.5.4 SQL Queries

#### Query #1

Used for updating the DataGridView with users that are not staff. This query is using a subquery and an ALL statement to request only members who are not staff.

sql = "SELECT AllUsers.Firstname, AllUsers.Lastname, AllUsers.UserID, AllUsers.Username FROM AllUsers WHERE UserID != ALL (SELECT ID FROM AllStaff);";



#### Query #2

Used for converting a general user into a staff member account, by creating a new record that includes the users UserID and assigning them a StaffID automatically due to it being a primary key. The query also inserts new information for the user such as, date of birth, position, status and access level.

string sql = "INSERT INTO AllStaff (DOB, Position, Status, AccessLevel, ID) VALUES ('" + calDOB.Value.Date.ToString("yyyy/MM/dd") + "', '" + txtPosition.Text + "', '" + comStatus.Text + "', " + numAccessLevel.Value + ", " + id + ");";

### 5.5.5 Evaluation

#### 5.5.5.1 Testing and Programming

This form presented itself as a difficult form with the calendar presenting the most problems due to the formatting of the date and time with the same problems as the Manager Settings form. These problems are evident between test id 1 and test id 2 where the problem was solved. If this problem was not solved, problems with not being able to store the DOB of users could be detrimental to the business as it is a legal requirement that businesses know the age of their staff, so they can pay them accordingly.

The SQL in this form was applied well as subqueries were used to find information rather than using separate queries that would decrease the efficiency of the program. This way the efficiency of the form is increased as less SQL C# objects are needed in the physical program code. This decision will benefit the program positively as the hardware requirements needed are lower allowing the program to perform better.

It is recommended (similar to other forms) that in future, fewer SQL objects are used in favour for reusing the SQL objects that are reusable. As previously discussed, this would ensure that the form has the maximum efficiency possible creating a more reliable and easier to run program.

To summarise, the form performs what it is intended to do with all SQL queries working correctly and all known bugs being solved. Through the lengthy testing and programming stages, the programming has been refined with every iteration lowering the number of possible bugs or errors that could arise or exist. In future, if the recommendation to reuse SQL objects is implemented the form could become more efficient and provide a better overall experience to the form.

#### 5.5.5.2 Design

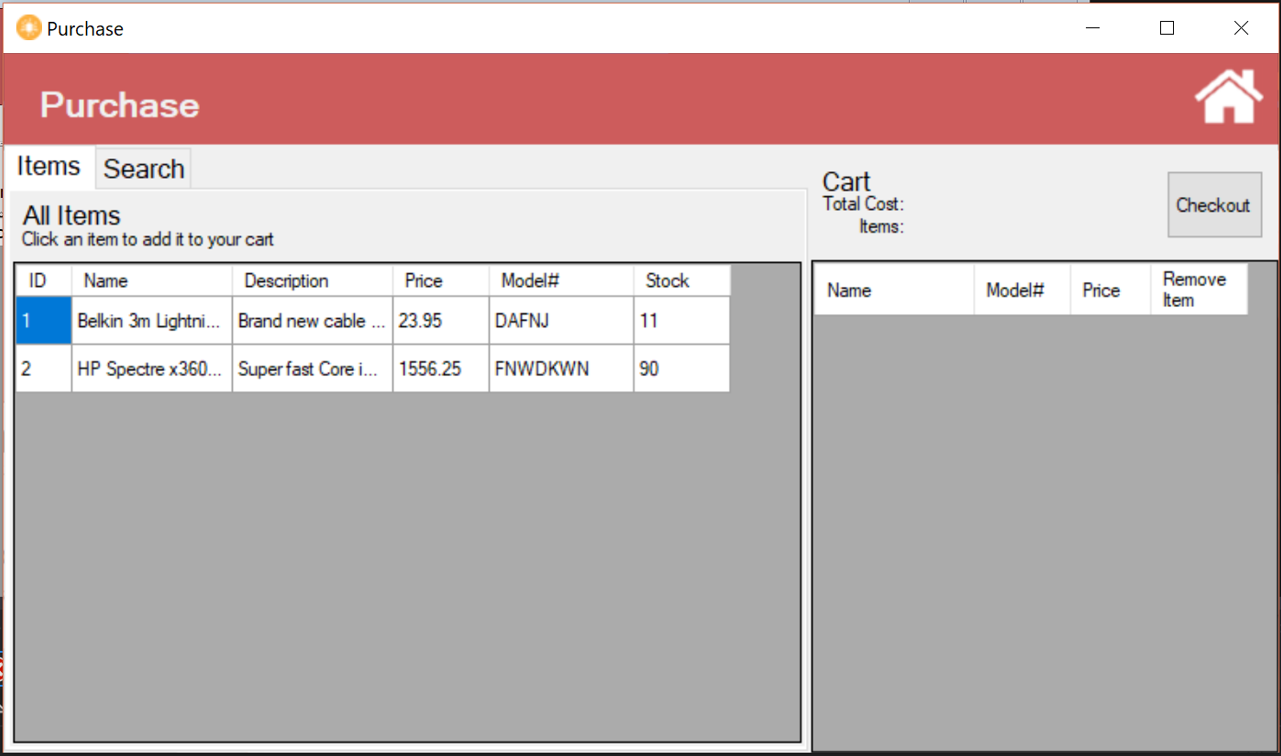
The overall design for this form is good as it utilises elements such as a DateTime picker to enhance the overall experience and make the software easier to use with little experience. Furthermore, the design of the form makes it incredibly easy to use as the order of steps starts at the top of the form (selecting a user) and finishes at the bottom of the form (submitting changes).

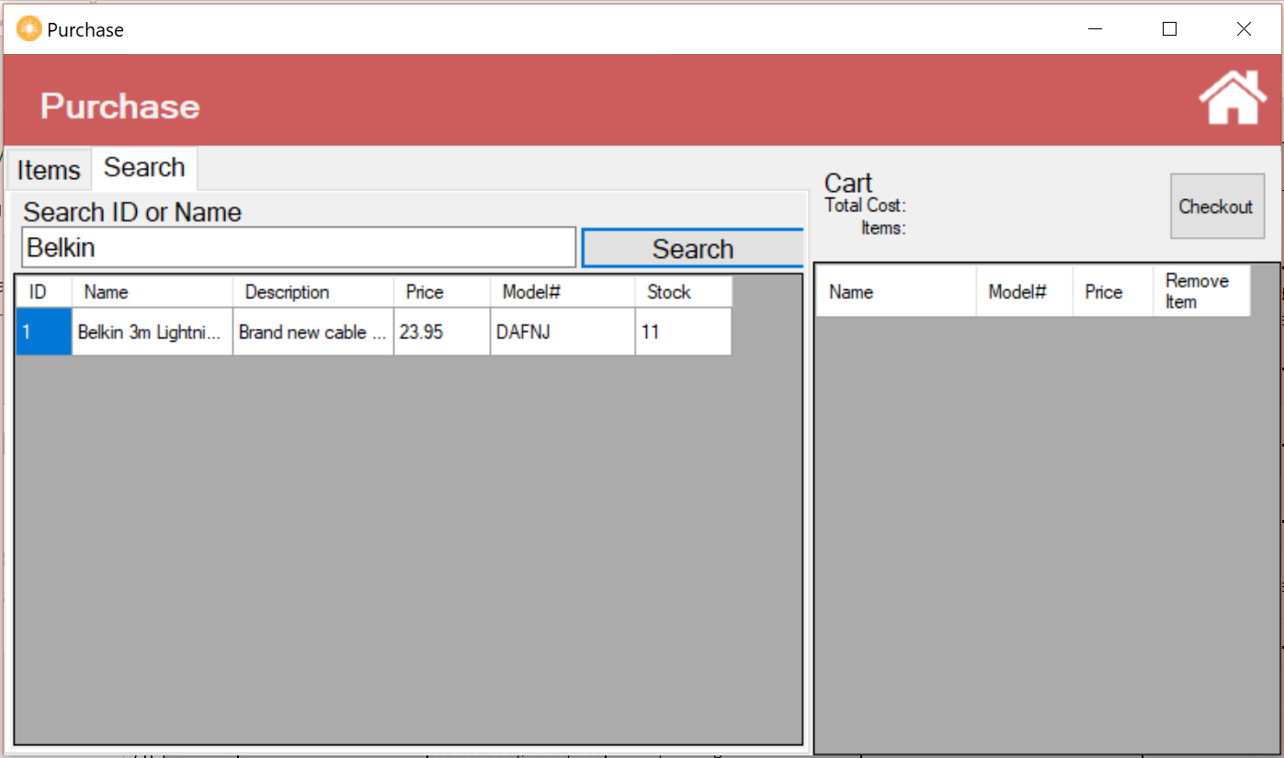
However, it is recommended next time that a search box is included somewhere in order to search for members. Because this form does not have a way of manually filtering results, once users start registering, finding a specific user in the database could be tedious as the user would be required to scroll through the DataGridView looking for the specific user. If this recommendation was implemented, the businesses time could be spent better leading to a more efficient program and better business outcomes.

In general, the forms design is well rounded as it is simple yet effective, however, it is critical that the recommendation mentioned is implemented as the amount of time saved the feature would create would be highly beneficial to the business.

## 5.6 Purchase Form

### 5.6.1 Design





### 5.6.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmPurchase : Form

{

public int itemsAmount = 0;

public frmPurchase()

{

InitializeComponent();

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT InventoryID, Name, Description, Price, Model#, QTY FROM Inventory";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

dgvInventory.Rows.Add(dataReader.GetValue(0), dataReader.GetValue(1), dataReader.GetValue(2), dataReader.GetValue(3), dataReader.GetValue(4), dataReader.GetValue(5));

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch

{

MessageBox.Show("Cannot open connection! ");

}

}

private void frmPurchase\_FormClosed(object sender, FormClosedEventArgs e)

{

exit();

}

private void picLogout\_Click(object sender, EventArgs e)

{

this.Hide();

exit();

}

public void exit()

{

GlobalVariables.cart.Clear();

GlobalVariables.cartPrice = 0;

if (User.isStaff == true)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

}

else

{

Process.Start(Environment.CurrentDirectory.ToString() + @"\MaximumTechnology.exe");

Environment.Exit(0);

}

}

private void dgvInventory\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (Convert.ToInt32(dgvInventory.Rows[e.RowIndex].Cells[5].Value) > 0)

{

if (e.RowIndex >= 0)

{

string sql = "SELECT InventoryID, GST FROM Inventory WHERE InventoryID='" + dgvInventory.Rows[e.RowIndex].Cells[0].Value.ToString() + "';";

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

string connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

stCart item = new stCart();

item.ModelNo = dgvInventory.Rows[e.RowIndex].Cells[4].Value.ToString();

item.Name = dgvInventory.Rows[e.RowIndex].Cells[1].Value.ToString();

item.Price = Convert.ToDecimal(dgvInventory.Rows[e.RowIndex].Cells[3].Value);

item.invID = Convert.ToInt32(dataReader.GetValue(0).ToString());

item.dGST = Convert.ToDecimal(dataReader.GetValue(1).ToString()) \* Convert.ToDecimal(dgvInventory.Rows[e.RowIndex].Cells[3].Value);

GlobalVariables.cart.Add(item);

refreshCart();

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Cannot open connection! \n\n Error: " + ex.ToString());

}

}

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error getting your data. \n\n Error: " + ex);

}

}

public void refreshCart()

{

GlobalVariables.cartPrice = 0;

dgvCart.Rows.Clear();

itemsAmount = 0;

foreach (stCart i in GlobalVariables.cart)

{

dgvCart.Rows.Add(i.Name, i.ModelNo, i.Price, "Remove");

GlobalVariables.cartPrice += i.Price;

itemsAmount += 1;

}

lblCost.Text = "Total Cost: $" + GlobalVariables.cartPrice;

lblItems.Text = "Items: " + itemsAmount;

GlobalVariables.iItems = itemsAmount;

}

private void dgvCart\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

if(e.ColumnIndex == 3)

{

GlobalVariables.cart.RemoveAt(e.RowIndex);

refreshCart();

itemsAmount -= 1;

}

}

}

private void btnCheckout\_Click(object sender, EventArgs e)

{

if(GlobalVariables.cart.Count > 0)

{

frmCheckout frm = new frmCheckout();

frm.Show();

this.Hide();

lblCost.Text = "Total Cost:";

lblItems.Text = "Items: ";

GlobalVariables.iItems = itemsAmount;

}

else

{

MessageBox.Show("You have not added any items to the cart");

}

}

private void btnSearch\_Click(object sender, EventArgs e)

{

if (txtSearch.Text != " ")

{

string sql = "SELECT InventoryID, Name, Description, Price, Model#, QTY FROM Inventory WHERE Name LIKE '%" + txtSearch.Text + "%' OR InventoryID LIKE '%" + txtSearch.Text + "%';";

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

string connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

dgvSearch.Rows.Add(dataReader.GetValue(0), dataReader.GetValue(1), dataReader.GetValue(2), dataReader.GetValue(3), dataReader.GetValue(4), dataReader.GetValue(5));

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

}

private void dgvSearch\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (Convert.ToInt32(dgvInventory.Rows[e.RowIndex].Cells[5].Value) > 0)

{

if (e.RowIndex >= 0)

{

string sql = "SELECT InventoryID, GST FROM Inventory WHERE InventoryID='" + dgvInventory.Rows[e.RowIndex].Cells[0].Value.ToString() + "';";

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

string connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

stCart item = new stCart();

item.ModelNo = dgvInventory.Rows[e.RowIndex].Cells[4].Value.ToString();

item.Name = dgvInventory.Rows[e.RowIndex].Cells[1].Value.ToString();

item.Price = Convert.ToDecimal(dgvInventory.Rows[e.RowIndex].Cells[3].Value);

item.invID = Convert.ToInt32(dataReader.GetValue(0).ToString());

item.dGST = Convert.ToDecimal(dataReader.GetValue(1).ToString()) \* Convert.ToDecimal(dgvInventory.Rows[e.RowIndex].Cells[3].Value);

GlobalVariables.cart.Add(item);

refreshCart();

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Cannot open connection! \n\n Error: " + ex.ToString());

}

}

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error getting your data. \n\n Error: " + ex);

}

}

}

}

### 5.6.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | Staff closes the purchase form | Form closes, and staff menu opens | Staff menu opens, then promptly closes followed by the login form opening | Was using this.Close() instead of this.Hide() | Changed this.close() to this.hide() | Yes |
| 2 | Staff closes the purchase form | Form closes, and staff menu opens | Form closes, and staff menu opens | N/A | N/A | No |
| 3 | Staff clicks on an item in the inventory to add it to the cart | Item is added to cart and appear in the cart DataGridView | Form crashes with error "System.FormatException: 'Input string was not in a correct format.' " | The wrong column index was being used when grabbing data from the inventory DataGridView | Change the column index from 3 to 2. | Yes |
| 4 | Staff clicks on an item in the inventory to add it to the cart | Item is added to cart and appear in the cart DataGridView | The DataGridView is not cleared and all the information stays inside the DataGridView with the cart being written again | DataGridView needs to be cleared before adding data to it again | Clear the DataGridView before writing the cart to the DataGridView | Yes |
| 5 | Staff clicks on an item in the inventory to add it to the cart | Item is added to cart and appear in the cart DataGridView | Price and number of items keep getting added onto themselves | Variables storing such data is not being reset before calculations | Reset variables | Yes |
| 6 | Staff clicks on an item in the inventory to add it to the cart | Item is added to cart and appear in the cart DataGridView | Item is added to cart and appear in the cart DataGridView | N/A | N/A | No |
| 7 | Form is loaded | All products appear in the inventory DataGridView with all columns and their corresponding values | All columns are loaded apart from GST column is not loaded | No code was specified to load GST column with data from datareader | Add dataReader.GetValue(6) to row add function | Yes |
| 8 | Form is loaded | All products appear in the inventory DataGridView with all columns and their corresponding values | All products appear in the inventory DataGridView with all columns and their corresponding values | N/A | N/A | No |
| 9 | User clicks on home button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 10 | User clicks on X button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |

### 5.6.4 SQL Queries

#### Query #1

Used for updating product information in database once user has changed product information. Sets values from corresponding input fields on the form.

string sql = "UPDATE Inventory SET Price =" + dPrice + ", Model# = '" + sModel + "', QTY = " + iQuantity + ", Description = '" + sDescription + "', GST =" + dGST + ", Name ='" + sName + "' WHERE InventoryID =" + id + ";";

#### Query #2

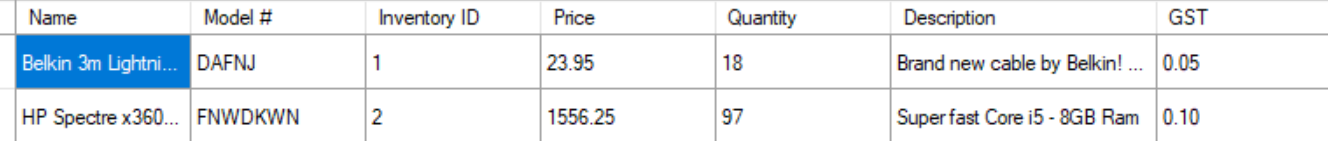
Used for deleting a specific product from inventory table selected by the user. Triggered when delete button is pressed.

string sql = "DELETE FROM Inventory WHERE InventoryID=" + id + ";";

#### Query #3

Used for selecting all data from the inventory table and populating the main DataGridView on the form that shows all products in the inventory.

sql = "SELECT \* FROM Inventory";



### 5.6.5 Evaluation

#### 5.6.5.1 Testing and Programming

This form was to some extents well thought out and in others, really shows where it need improvement. The form performs what it was set out to do, but from a programming and efficiency point of view, there is room for improvement. Due to the limited time frame on the whole project, some portions of code have been rushed in order for the final program to be completed on time with documentation. The inventory form is an example where corners were cut in order to complete the final product, both in terms of design and programming.

An example of corners being cut during the development in this form is (similar to other forms) when functions should have most definitely been created and used for specific tasks and SQL Objects reused, rather than used once. In the development of this form, the amount of code that was copied and pasted from other sections of code instead of just creating a function that could handle multiple parameters and scenarios was too high. When reading through the code it is evident how much of the form could have been turned into separate functions rather than keeping them as separate functions that leave the code unreadable and inefficient. Moreover, as mentioned before, this form utilises the same impractical technique of creating a new SQL Object for every new query. As mentioned in earlier sections, this is inefficient and a waste of CPU power and RAM. With these considerations taken into account, it is clearly evident that the programming for this form should have been neater and more conscientious when considering the readability of the code as this could affect its ability to be updated in the future by a third-party company.

Conversely, the SQL queries for this form were simple and yet effective enough to easily adhere to the requirements of the system. All queries worked as expected as shown in testing above and the desired outputs were produced. No queries needed to be subqueries as each function required only one query with no extra information from the database that wasn’t already stored in memory from past queries.

During testing, there were major problems between test id’s three to six where multiple problems occurred when a cell was clicked in the DataGirdView to add the selected product to the cart. Many different tests and fixes were implemented in hopes of fixing the problem as it remains one of the critical functions of the program and the method of adding products to the cart. If this section of the program did not work the whole program could have run into problems as the main feature of the program would not have worked correctly. However, after much research, the problem was solved and is now working correctly.

Generally speaking, the form is an adequate solution to the problem set forward as it addressed the need to edit the inventory, delete products from the inventory and add products to the inventory. This leads to a solution to which the user will be able to use without problems. After comprehensive testing, no bugs have been recorded to still exist creating a more robust program that will deliver a better experience to the user. However, it is strongly recommended that the recommendations above are taken into consideration as the code needs improving, which could help with future updatability.

#### 5.6.5.2 Design

Though highly functional and efficient, the design for this form will be the main detractor for this program due to its complexity at first glance. With very little visual cues it can be difficult for a first-time user to become experienced with the form. In future, it is recommended that an easier or more efficient solution be found possibly using a barcode scanner and a search function as a backup. This method would greatly reduce the time spent at the counter browsing or manually searching for items. This wastage of time could hurt the business sales wise or turn away customers who find the program too confusing to navigate.

However, from an efficiency point of view the form, once understood and learned, can be very efficient due to its search function that provides back detailed results on the query sent. Though it may not look visually appealing the program does have visual buttons and some cues that do aid the user, however, this still does not exempt the form from being complex at first.

Nevertheless, the form still does follow similar design philosophy as the other forms in this program as it continues to follow the convention of using a bar at the top of the form that indicates what form is the user is currently using and allows access to the staff menu by clicking the home button. Once again as well, the form uses the same methodology of clicking on a cell in a DataGridView to perform an action which is a testament to the form as it less time is needed learning how to add products to the cart as it is a process already used in the program.

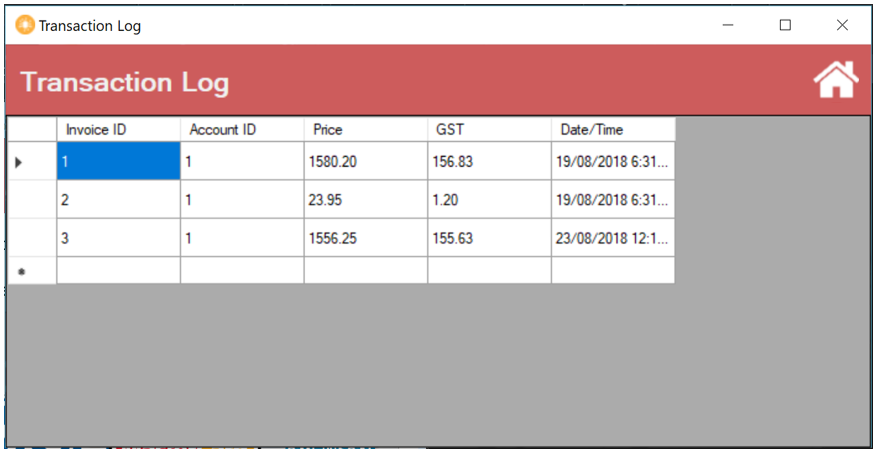
A recommendation that could be made for the form would be to use only one DataGridView instead of two for product selection and browsing. Currently, there are two DataGridViews separated by a tab control which can be confusing as is used for searching for a specific item and the other to browse the items available. If this was combined into one DataGridView the ease of the form could be increased as users would not have to move between the two DataGirdViews and tabs. This recommendation should be considered for future updates. Though not crucial to the success of the program, the amount of time a user spends at a self-service machine could be decreased possibly increasing the number of sales as more customers can pass through the checkout during a given time frame. Not to mention the less time a user spends dealing with the machine the more likely the customer satisfaction is to be higher increasing the likelihood of the person wanting to shop with the business again.

Secondly, a recommendation that would is highly recommended would be to allow the user the option to scan their desired product they may be holding at the checkout rather than having to manually search for the item. This would decrease the amount of time customer spends at a checkout, meaning (a discussed previously) higher customer satisfaction, increased sales and increased likelihood of the user shopping with the business are to be expected. By utilising a function like this the business benefits greatly and should seriously consider implementing it in the future.

On the whole, though the form has many recommendations that should be implemented to create a more user-friendly experience that does not turn away customers, the forms design is efficient and does allow the user much control. However, due to the complexity of the forms, it may be too difficult for others leading to an unpleasant user experience and a stain on the business.

## 5.7 Invoice Log Form

### 5.7.1 Design



### 5.7.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmInvoiceLog : Form

{

public frmInvoiceLog()

{

InitializeComponent();

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT InvoiceID, UserID, SUM(PurchasePrice) AS 'Price', SUM(GST) AS 'GST', [DateTime] FROM [Maximum Technology].[dbo].[Invoices] GROUP BY InvoiceID, UserID, DateTime;";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

dgvTransactions.Rows.Add(dataReader.GetValue(0), dataReader.GetValue(1), dataReader.GetValue(2), dataReader.GetValue(3), dataReader.GetValue(4));

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection ! \n\n Error: " + ex.ToString());

}

}

private void picLogout\_Click(object sender, EventArgs e)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

this.Hide();

}

private void frmTransactionLog\_FormClosed(object sender, FormClosedEventArgs e)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

}

}

}

### 5.7.3 Testing

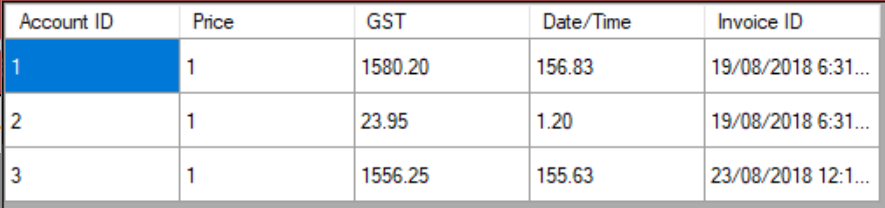
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User clicks transaction log button in staff menu | Transaction log form is shown with DataGridView filled out | Index was outside the bounds of the array | SQL statement was not correct when compared tested in SQL server | Fix SQL statement by fixing name spelling error | Yes |
| 2 | User clicks transaction log button in staff menu | Transaction log form is shown with DataGridView filled out | Index was outside the bounds of the array | Function was trying to read value from a row that did not exist in DataReader | Remove code trying to access null column 5 | Yes |
| 3 | User clicks transaction log button in staff menu | Transaction log form is shown with DataGridView filled out | Transaction log form is shown with DataGridView filled out | N/A | N/A | No |
| 4 | User clicks exit button | Form is closed, and staff form is shown | Form is closed, and staff form is shown | N/A | N/A | No |
| 5 | User clicks X button | Form is closed, and staff form is shown | Form is closed, and staff form is shown | N/A | N/A | No |
| 6 | User clicks header of DataGridView | Rows are ordered by the column selected | Rows are ordered by the column selected | N/A | N/A | No |

### 5.7.4 SQL Queries

#### Query #1

Used for selecting invoice history. Sums total cost of the invoice and displays it as ‘Price’ and uses the group by command to group together the summed records.

sql = "SELECT InvoiceID, UserID, SUM(PurchasePrice) AS 'Price', SUM(GST) AS 'GST', [DateTime] FROM [Maximum Technology].[dbo].[Invoices] GROUP BY InvoiceID, UserID, DateTime;";



### 5.7.5 Evaluation

#### 5.7.5.1 Testing and Programming

The purpose of this form was to allow the user to view past transaction details in a way that was easily readable and conveyed necessary information about the invoice. This was achieved as the form works correctly with no bugs known as of yet.

Due to the forms simple code, the code has been designed efficiently with not duplicates or extra functions/items that are not needed. This goes hand in hand with the design of the SQL query which to uses a SUM command and GROUP BY to organise the data in the table. This usage was a good idea as it drastically reduced the amount of code that would be needed in order to do the same process using C#, thus ensuring that more time could be spent on other problems.

There are no programming recommendations for this form due to its simple design.

Overall the form does not have any problems and works correctly organising invoice information into easy to read information. No programmatic recommendations are needed for this form.

#### 5.7.5.2 Design

The design of this form is rather simple which is great in some regards as the form remains simple to use. On the hand, however, the form lacks much functionality which should have been considered during the design phase of the project. The form does not have any method of searching records or viewing more specific invoice information such as the items purchased. ‘

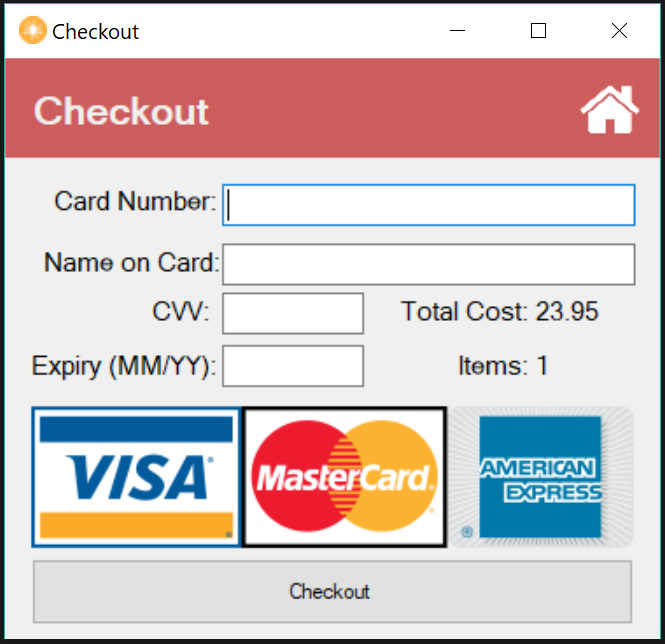
It is recommended in future that these issues are solved. An idea that could be considered for implementation would be to allow the user to click on a row and trigger an event where a form would be displayed, showing the items purchased and a detailed breakdown of the invoice. This would help in the process of warranty or proof of purchase situations where information on the invoice is needed to be known greatly reducing any business frustration as these records would be easily accessible.

In future, it is also recommended that a method of searching for items via a search box or checkbox filters be implemented that would allow the user to refine their search results. Though the DataGridView does have the ability to order rows by columns, it does not allow the excluding of certain items. This means that when the form is loaded, all past invoices are loaded into the form. This could create problems in the future where returned results from the database could be too large creating problems when handling the data, more specifically prolonged form loading times. Not to mention the time wasted by staff as they would need to manually search through all invoices to find a specific invoice.

Overall, though the design is easy to understand, it does not contain enough elements that make the form more complete as an invoice indexing tool. The suggestions above should be highly considered during the next updates as time is an asset to businesses and the time wasted in this form will be high once more users start to perform transactions due to the limited functionality.

## 5.8 Checkout Form

### 5.8.1 Design



### 5.8.2 Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmCheckout : Form

{

public frmCheckout()

{

InitializeComponent();

}

private void Checkout\_Load(object sender, EventArgs e)

{

lblItems.Text = "Items: " + GlobalVariables.iItems;

lblPrice.Text = "Total Cost: " + GlobalVariables.cartPrice;

}

private void btnCheckout\_Click(object sender, EventArgs e)

{

if (txtCardNumber.Text == null || txtCVV.Text == null || txtExpiry.Text == null || txtName.Text == null)

{

MessageBox.Show("Please fill all fields");

}

else

{

try

{

string connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

SqlConnection updateConnection = new SqlConnection(connectionString);

SqlCommand updateCommand;

SqlDataReader updateDataReader;

updateConnection.Open();

foreach (stCart i in GlobalVariables.cart)

{

string sqlUpdate = "UPDATE Inventory SET QTY = (SELECT QTY FROM Inventory WHERE InventoryID=" + i.invID + ")-1 WHERE InventoryID= " + i.invID + ";";

updateCommand = new SqlCommand(sqlUpdate, updateConnection);

updateDataReader = updateCommand.ExecuteReader();

updateCommand.Dispose();

updateDataReader.Close();

}

updateConnection.Close();

try

{

long cardNumber = Convert.ToInt64(txtCardNumber.Text);

int cvv = Convert.ToInt32(txtCVV.Text);

string expirey = txtExpiry.Text;

string name = txtName.Text;

string sql = "INSERT INTO Invoices (InvoiceID, InventoryID, UserID, Line, GST, OrderComplete, DateTime, PurchasePrice) VALUES ";

int x = 0;

foreach (var i in GlobalVariables.cart)

{

x += 1;

if (GlobalVariables.cart.Count == x)

{

sql += "(ISNULL((SELECT TOP 1 InvoiceID FROM Invoices ORDER BY InvoiceID DESC)+1, 1), " + i.invID + ", " + User.ID + ", " + x + ", " + i.dGST + ", " + 1 + ", '" + DateTime.Now.ToString("yyyy-MM-dd HH:mm") + "', " + i.Price + ");";

}

else

{

sql += "(ISNULL((SELECT TOP 1 InvoiceID FROM Invoices ORDER BY InvoiceID DESC)+1, 1), " + i.invID + ", " + User.ID + ", " + x + ", " + i.dGST + ", " + 1 + ", '" + DateTime.Now.ToString("yyyy-MM-dd HH:mm") + "', " + i.Price + "),";

}

}

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

connection = new SqlConnection(connectionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

dataReader.Close();

command.Dispose();

connection.Close();

MessageBox.Show("Thanks for shopping with Maximum Technology!");

this.Hide();

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

GlobalVariables.cart.Clear();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection ! " + ex);

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error trying to process your request. Please try again later. Error: " + ex);

}

}

catch (Exception ex)

{

MessageBox.Show("Please enter valid inputs. Error: " + ex.ToString());

}

}

}

private void frmCheckout\_FormClosed(object sender, FormClosedEventArgs e)

{

GlobalVariables.cart.Clear();

if (User.isStaff == true)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

}

else

{

Process.Start(Environment.CurrentDirectory.ToString() + @"\MaximumTechnology.exe");

Environment.Exit(0);

}

}

private void picLogout\_Click(object sender, EventArgs e)

{

GlobalVariables.cart.Clear();

if (User.isStaff == true)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

this.Close();

}

else

{

Process.Start(Environment.CurrentDirectory.ToString() + @"\MaximumTechnology.exe");

Environment.Exit(0);

}

}

}

}

### 5.8.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | User clicks checkout button | Transaction history is logged into the database and messagebox is shown | Sql Exception Error | DateTime format being inserted with query was not supported by SQL Server | Change DateTime formatting | Yes |
| 2 | User clicks checkout button | Transaction history is logged into the database and messagebox is shown | Sql Exception Error. Syntax error in query | SQL command LIMIT is not supported by SQL Server | Using Top command instead of Limit | Yes |
| 3 | User clicks checkout button | Transaction history is logged into the database and messagebox is shown | Transaction history is logged into the database and messagebox is shown | N/A | N/A | No |
| 4 | User clicks checkout button with a credit card number longer than 10 digits | Transaction history is logged into the database and messagebox is shown | MessageBox is shown saying "Please enter valid inputs." | Cannot store a number larger than 10 digits in a integer | Change from storing credit card number in an integer to using a long | Yes |
| 5 | User clicks checkout button with a credit card number longer than 10 digits | Transaction history is logged into the database and messagebox is shown | Transaction history is logged into the database and messagebox is shown | N/A | N/A | No |
| 6 | User checkouts/purchases items when there have been no previous transactions | Invoice history is logged into database with an invoice id of 1. Form closes. | MessageBox is shown with error "Cannot insert value NULL into column InvoiceID" | SQL sub-query is being used to get the last id used in the Invoices table. However, because there are no previous transactions the value returned is null | Using an ISNUL expression to check if the query returns a null value. If not the last id +1 will be used otherwise 1 will be used. | Yes |
| 7 | User checkouts/purchases items when there have been no previous transactions | Invoice history is logged into database with an invoice id of 1. Form closes. | Invoice history is logged into database with an invoice id of 1. Form closes. | N/A | N/A | No |
| 8 | User clicks on home button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |
| 9 | User clicks on X button in right hand corner | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | Form closes, and staff menu opens if user is a staff member, if user is not a staff member form closes and login opens | N/A | N/A | No |

### 5.8.4 Sql Queries

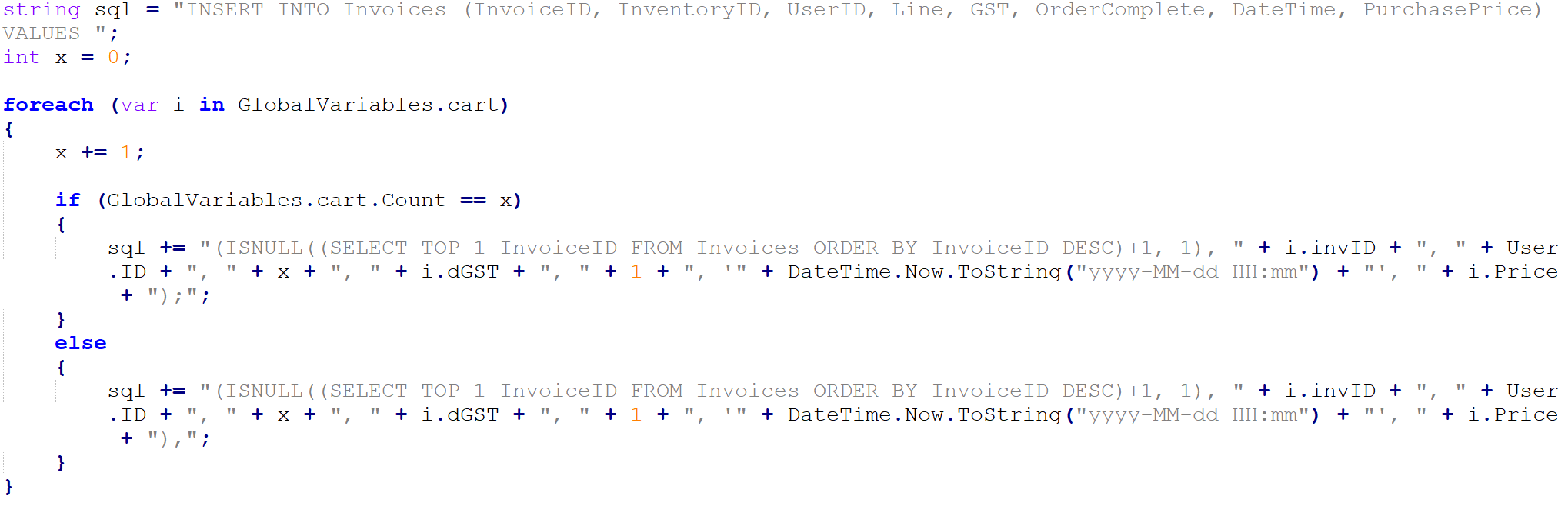
#### Query #1

Used for updating inventory information after user has purchased items. Uses a subquery to get the qty of the product before the purchase and subtracts 1 to calculate the stock of the product after the transaction. Using an update statement, the sock information of the product is then updated.

string sqlUpdate = "UPDATE Inventory SET QTY = (SELECT QTY FROM Inventory WHERE InventoryID=" + i.invID + ")-1 WHERE InventoryID= " + i.invID + ";";

#### Query #2

Used for creating a new invoice record once an invoice has been processed. This is done by using a foreach loop that iterates through all products in the cart to construct the SQL statement to insert multiple products from the cart at once into the database in the same query. Each iteration inserts a new row of values.



### 5.8.5 Evaluation

#### 5.8.5.1 Programming and Testing

This form was successful as it was able to fulfil its purpose of being able to record transactions and check out items from the inventory. The form has no currently known problems or bugs, as shown in the testing above as all errors listed above, were eventually fixed. However, though the code is considered bug free and working correctly the code itself still definitely has improvement that should be considered for future updates.

Due to the length of the methods in the code, in future, it is recommended that certain sections of the code be organised into separate functions that can be reused. For example, the process of updating a SQL table was repeated multiple times in the code, however, this could have been reduced if a function was introduced where the query could just be passed and sent to the server using one function rather than creating multiple unneeded, inefficient SQL objects. Furthermore, by organising the code into separate functions the readability of the code is also increased making future programmers’ jobs easier and quicker when updates are needed for the program.

The usage of using ISNULL in the SQL invoice table update query was a good idea that paid off as time was saved that was eventually utilised in improving other aspects of the application. Though research was needed before using the command, the amount of time that would have been needed to perform the same process without using the ISNULL command would have been too large. Not to mention the usage of the ISNULL command ensure that SQL subqueries could be used. If the ISNULL command was not used the subquery would need to be run separately and the results would need to be parsed using C#, however, this method saves time, code and computing resources as less code is needed to run on the client side.

Overall, though there is room for improvement in the form, the form was still highly successful as it is able to perform its required purpose with no currently known problems. It should be noted, however, that if the form is to be updated in the future, it is recommended that the code is separated into functions in order to make future updates easier.

#### 5.8.5.2 Design

The design of the form was well thought out with very little to complain about as it follows an easy to follow design pattern. Everything on the form is laid out vertically showing the procedure of completing the form, from top to bottom. The form is not overcrowded and shows the accepted credit/debit cards through the use of a PictureBox. This positively helps the business as it allows them to effectively communicate to the customer the supported card types saving customers and stuff money.

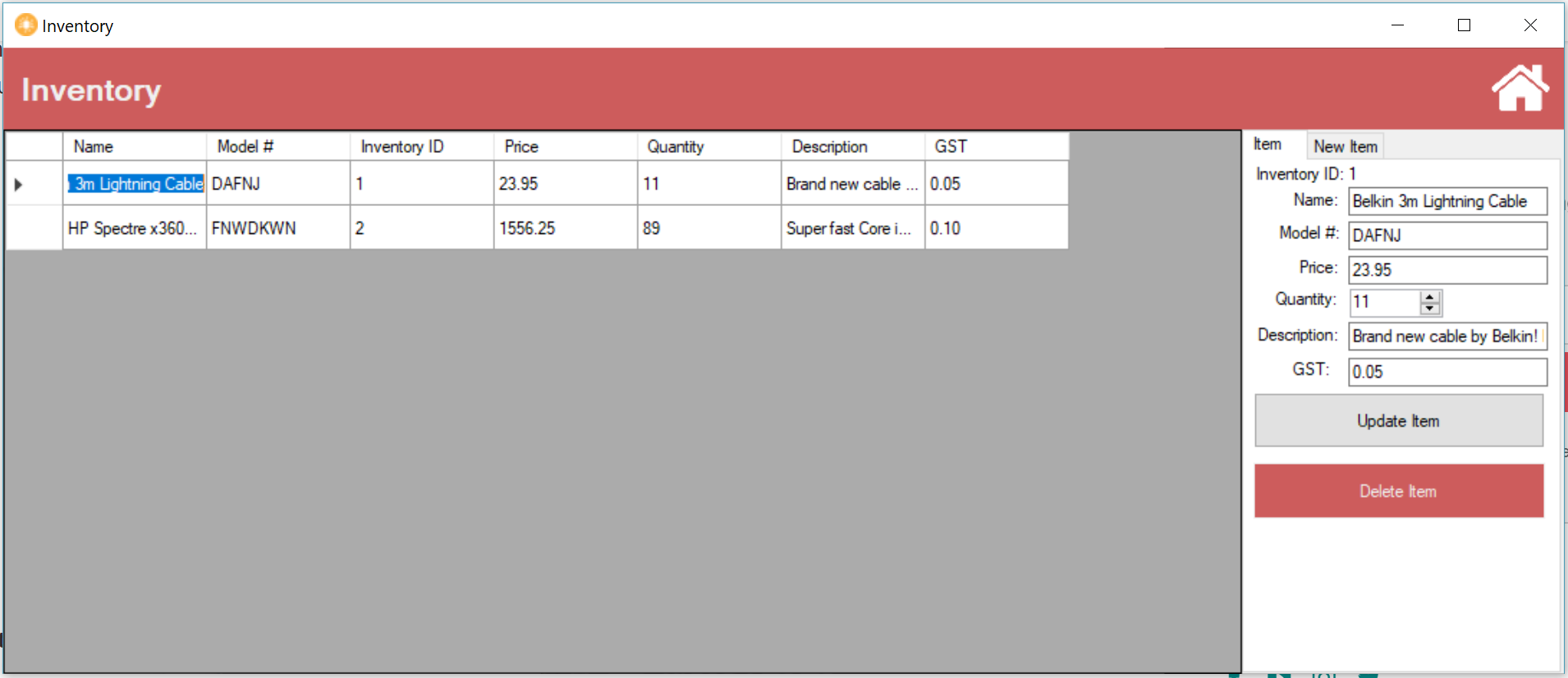
However, it is recommended in future that entry method of entering the credit card expiry date should use more than just a text box for entry as it does not have a specific entry format meaning different users may enter the date differently causing problems in the system. In future, it is recommended that something like a date time picker or drop-down lists be used where a user can select a date using the mouse or touch meaning all formatting will be handled by the computer resulting in a unified date format for card expiries.

It is also recommended in the future that a system for handling cash be implemented and designed for the system as the system is currently limited to card only limiting the available user base that can use the system. By implementing a cash system, a larger user base may be reached benefitting the business by possibly increasing sales. This recommendation should be strongly considered as this feature could have an impact on the business's earnings.

To summarise the design of the form was well implemented, however, there are still improvements that should be considered. There are of course still many other ideas that could be implemented but the recommendations above outline some key points that should be considered for later updates. By considering or implementing the recommendations above the program and system efficiency could be improved leading to a better customer experience and satisfied customers willing to shop with the store again.

## 5.9 Inventory

### 5.9.1 Design



### 5.9.2 Code

using System;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace MaximumTechnology

{

public partial class frmInventory : Form

{

int id;

public frmInventory()

{

InitializeComponent();

if (User.AccessLevel > 2)

btnDelete.Hide();

else

btnDelete.Show();

}

private void frmInventory\_Load(object sender, EventArgs e)

{

refreshData();

}

private void dgvInventory\_CellClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (e.RowIndex >= 0)

{

id = Convert.ToInt16(dgvInventory.Rows[e.RowIndex].Cells[2].Value);

lblID.Text = "Inventory ID: " + dgvInventory.Rows[e.RowIndex].Cells[2].Value.ToString();

txtName.Text = dgvInventory.Rows[e.RowIndex].Cells[0].Value.ToString();

txtModel.Text = dgvInventory.Rows[e.RowIndex].Cells[1].Value.ToString();

txtPrice.Text = dgvInventory.Rows[e.RowIndex].Cells[3].Value.ToString();

numQuantity.Value = Convert.ToInt32(dgvInventory.Rows[e.RowIndex].Cells[4].Value.ToString());

txtDescription.Text = dgvInventory.Rows[e.RowIndex].Cells[5].Value.ToString();

txtGST.Text = dgvInventory.Rows[e.RowIndex].Cells[6].Value.ToString();

}

}

catch (Exception ex)

{

MessageBox.Show("There was an error grabbing item information. \n\n Error: " + ex);

}

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

if (User.AccessLevel <= 2)

{

if (txtNewName.Text == "" || txtNewModel.Text == "" || txtNewPrice.Text == "" || numNewQuantity.Value == 0 || txtDescription.Text == "" || txtGST.Text == "")

{

if (id == 0)

{

MessageBox.Show("You have not selected an item");

}

else

{

string sName = txtName.Text;

string sModel = txtModel.Text;

string sDescription = txtDescription.Text;

int iQuantity = Convert.ToInt32(numQuantity.Value.ToString());

decimal dPrice;

decimal dGST;

try

{

dPrice = Convert.ToDecimal(txtPrice.Text);

try

{

dGST = Convert.ToDecimal(txtGST.Text);

string sql = "UPDATE Inventory SET Price =" + dPrice + ", Model# = '" + sModel + "', QTY = " + iQuantity + ", Description = '" + sDescription + "', GST =" + dGST + ", Name ='" + sName + "' WHERE InventoryID =" + id + ";";

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

string connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

command.Dispose();

dataReader.Close();

connection.Close();

refreshData();

}

catch (Exception ex)

{

MessageBox.Show("There was an error updating inventory. \n\n + Error: " + ex);

}

}

catch

{

MessageBox.Show("GST entered is not valid.");

}

}

catch

{

MessageBox.Show("Price entered is not valid.");

}

}

}

else

{

MessageBox.Show("All feilds must be filled in.");

}

}

else

{

MessageBox.Show("You do not have sufficient permissions to perform this task.");

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

if (id == 0)

{

MessageBox.Show("You have not selected an item.");

}

else

{

string sql = "DELETE FROM Inventory WHERE InventoryID=" + id + ";";

SqlConnection connection;

SqlCommand command;

SqlDataReader dataReader;

string connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

command.Dispose();

dataReader.Close();

connection.Close();

refreshData();

}

catch (Exception ex)

{

MessageBox.Show("There was an error updating inventory. \n\n + Error: " + ex);

}

}

}

private void btnNewItem\_Click(object sender, EventArgs e)

{

if (txtNewName.Text == "" || txtNewModel.Text == "" || txtNewPrice.Text == "" || numNewQuantity.Value == 0 || txtNewDescription.Text == "" || txtNewGST.Text == "")

{

MessageBox.Show("All fields must filled in.");

}

else

{

decimal dPrice;

decimal dGST;

try

{

dPrice = Convert.ToDecimal(txtNewPrice.Text);

try

{

dGST = Convert.ToDecimal(txtNewGST.Text);

string sql = "INSERT INTO Inventory VALUES(" + dPrice + ", '" + txtNewModel.Text + "', " + numNewQuantity.Value + ", '" + txtDescription.Text + "', " + dGST + ", '" + txtNewName.Text + "');";

SqlConnection connection;

SqlCommand command;

SqlDataReader datareader;

string connectionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

connection = new SqlConnection(connectionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

datareader = command.ExecuteReader();

command.Dispose();

connection.Dispose();

datareader.Close();

refreshData();

}

catch (Exception ex)

{

MessageBox.Show("There was en error accessing the database. \n\n Error: " + ex.ToString());

}

}

catch

{

MessageBox.Show("The GST amount you entered in invalid");

}

}

catch

{

MessageBox.Show("The price you entered is not valid");

}

}

}

public void refreshData()

{

dgvInventory.Rows.Clear();

string connetionString = null;

SqlConnection connection;

SqlCommand command;

string sql = null;

SqlDataReader dataReader;

connetionString = @"Server=localhost\sqlexpress; Initial Catalog = Maximum Technology; User ID = MaximumTech; Password = password";

sql = "SELECT \* FROM Inventory";

connection = new SqlConnection(connetionString);

try

{

connection.Open();

command = new SqlCommand(sql, connection);

dataReader = command.ExecuteReader();

while (dataReader.Read())

{

dgvInventory.Rows.Add(dataReader.GetValue(6), dataReader.GetValue(2), dataReader.GetValue(0), dataReader.GetValue(1), dataReader.GetValue(3), dataReader.GetValue(4), dataReader.GetValue(5));

}

dataReader.Close();

command.Dispose();

connection.Close();

}

catch

{

MessageBox.Show("Can not open connection ! ");

}

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

this.Close();

}

private void frmInventory\_FormClosed(object sender, FormClosedEventArgs e)

{

frmStaffMenu frm = new frmStaffMenu();

frm.Show();

}

}

}

### 5.9.3 Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Scenario** | **Expected result** | **Actual Result** | **Cause of error** | **Fix** | **Retest required?** |
| 1 | Staff member clicks on a cell in datagridview | Data in sidebar changes to represent row selected | Oncy some cells work meaning some when clicked causes data in sidebar to change some some don’t | Event handler CellContentClick was being used instead of CellClick | Change function to be called on event handler Cell Click | Yes |
| 2 | Staff member clicks on a cell in datagridview | Data in sidebar changes to represent row selected | Data in sidebar changes to represent row selected | N/A | N/A | No |
| 3 | Staff member clicks on header bar at the top of datagridview | Rows are ordered by column selected | Program crashes with error: System.ArgumentOutOfRangeException: 'Index was out of range. Must be non-negative and less than the size of the collection. Parameter name: index' | CellClick event handler fucntion uses row index clicked, however, when header is clicked a negative or null value is returned. | Surround if statement around CellClick function to check if value is greater than 0 | Yes |
| 4 | Staff member clicks on header bar at the top of datagridview | Rows are ordered by column selected | Rows are ordered by column selected | N/A | N/A | No |
| 5 | User clicks on row in datagridview | Data in sidebar changes to represent row selected | When first data row is clicked sidebar is not updated, but all other rows work | Using greater than 0 condition in if statement meant that the first data row in the datagridview is skipped due to its index being 0 | Change condition in if statement to greater than or equal to 0 | Yes |
| 6 | User clicks on row in datagridview | Data in sidebar changes to represent row selected | Data in sidebar changes to represent row selected | N/A | N/A | No |
| 7 | User clicks update button when all text boxes are filled in | Sql data for that primary key is updated | All rows in table are updated | Where statement needed in sql statement | Add where condition into sql statement to update only primary key | Yes |
| 8 | User clicks update button when all text boxes are filled in | Sql data for that primary key is updated | Sql data for that primary key is updated | N/A | N/A | No |
| 9 | User clicks update button when all text boxes are filled in | Sql data for that primary key is updated and the datagridview is refreshed with new data | Sql data for that primary key is updated and datagrid view refreshes, but duplicate copies of the elements appear in the the datagridview | Datagridview is not being emptied before being filled again | Clear all rows at the start of the refreshdata function | Yes |
| 10 | User clicks update button when all text boxes are filled in | Sql data for that primary key is updated and the datagridview is refreshed with new data | Sql data for that primary key is updated and datagrid view refreshes, but duplicate copies of the elements appear in the the datagridview | Datagridview is not being emptied before being filled again | Clear all rows at the start of the refreshdata function | Yes |
| 11 | User clicks update button when all text boxes are filled in | Sql data for that primary key is updated and the datagridview is refreshed with new data | Sql data for that primary key is updated and the datagridview is refreshed with new data | N/A | N/A | No |
| 12 | User clicks new item box with all boxes filled | New item is created and added to databse and datagridview refreshes | New item is created, however, name is replaced with "System.Windows.Forms.TextBox, Text:" and description is not present | txtName was being used instead of txtName.Text, so the code was getting the object name and not the text in the textbox. Code was getting description from updatetab txtDescription textbox | Change name from txtName to txtName.Text so it gets the text in the box rather than the name of the object. Change txtDescription to txtNewDescription, the text box on the new item page | Yes |
| 13 | User clicks new item box with all boxes filled | New item is created and added to databse and datagridview refreshes | New item is created and added to databse and datagridview refreshes | N/A | N/A | No |
| 14 | User clicks new item box with all boxes filled but and a decimal price tag | New item is created and added to databse and datagridview refreshes | New item is created and added to databse and datagridview refreshes | MessageBox is displayed saying "Price entered is not valid" | Price was being converted to an integer instead of a decimal. This was changed so the string was converted to a decimal | Yes |
| 15 | User clicks new item box with all boxes filled but and a decimal price tag | New item is created and added to databse and datagridview refreshes | New item is created and added to databse and datagridview refreshes | N/A | N/A | No |

### 5.9.4 SQL Queries

#### Query #1

Used for updating product information in database once user has changed product information. Sets values from corresponding input fields on the form.

string sql = "UPDATE Inventory SET Price =" + dPrice + ", Model# = '" + sModel + "', QTY = " + iQuantity + ", Description = '" + sDescription + "', GST =" + dGST + ", Name ='" + sName + "' WHERE InventoryID =" + id + ";";

#### Query #2

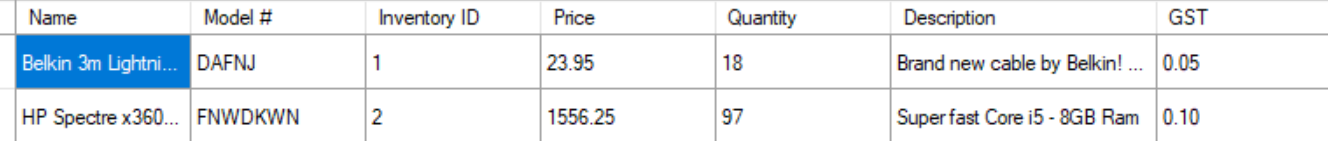
Used for deleting a specific product from inventory table selected by the user. Triggered when delete button is pressed.

string sql = "DELETE FROM Inventory WHERE InventoryID=" + id + ";";

### Query #3

Used for selecting all data from the inventory table and populating the main DataGridView on the form that shows all products in the inventory.

sql = "SELECT \* FROM Inventory";



### 5.9.5 Evaluation

#### 5.9.5.1 Testing and Programming

Being one of the more difficult forms, more time was needed to code the form, therefore, some corners were cut during the process. All known bugs that were discovered and recorded were fixed so a better user experience can be expected. Having all known bugs will provide a better user and business experience as the business will not need to hire a third party to fix the program.

The functions in this form were too long and could be improved upon by splitting up the functions into separate smaller functions. This was done to an extent as the refreshData function was created to ensure that code is not repeated. This implementation was a good idea as it increased the code readability ensuring that updating the program in the future will be easier and less expensive for the business. However, there is still much code which could have been separated into separate smaller functions such as the multiple update queries that occur. In the future, it would be recommended that these are split to increase readability and updatability.

Similar to other forms, it would also be recommended that SQL objects be reused instead of creating a new object for every new function/query. This would save on RAM and CPU usage and increase efficiency. Many SQL objects used in the form were used only once when they could have been used across more than one operation. This would be an improvement to be made in future updates.

During testing, a major error occurred associated with the DataGridView’s in which when a column header was selected the form would crash with a system exception error as seen in test id three. If this bug was not rectified the project could have been derailed, as the form relies heavily on the column headers as a method for organising the data in the DataGridViews

The usage of a List<Struct> and struct object was beneficial to the project as much time was saved using this method. This method made it easy to ‘simulate’ a ‘multidimensional’ list, meaning every object in the list contained columns. This made it easy to add objects to the cart, read from the cart and remove objects from the cart as normal functions that would generally be used in a list such as List.Remove() and List.Add(), were able to be applied making programming easier and reducing the amount of time needed to finish the final program. This was beneficial to the business as more time was able to be used working on other parts of the program and fixing bugs.

#### 5.9.5.2 Design

Compared to forms such as the purchase form, the form design was tidier and more orderly as the form consists of only one DataGridView and a panel to the right side where a tab control is located allowing access to edit selected product information and add new products. Furthermore, all error dialogue boxes store relevant information to the user and have been implemented at the correct times. For example, if a user tries to edit product information without selecting an item a MessageBox will notify the user stating, “You much first select a product”. This increases the overall usability and ease of the form, as relevant MessageBoxes ensure that the user knows how to navigate and use the form.

However, in the future, it is recommended that a search element is included that allows the user to search for the item they would like to edit as the only way of editing a product as of now is by manually searching for the product in the DataGridView and selecting it. By including a search function, more time could be gained as less time is used for selecting products allowing a staff time to be used of on other tasks increasing business efficiency.

Overall the design of the form was well rounded and easy to understand as the same method of selecting an item from the DataGirdView is used the same as forms such as the purchase form and the new staff form. This creates continuity through the program and decreases the learning curve, decreasing overall training time and increasing business efficiency making design choice beneficial to the business. If the search recommendation was included in future updates, the forms overall ease and business efficiency are expected to increase.

## 5.10 Global Variables

### 5.10.1 Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MaximumTechnology

{

public class GlobalVariables

{

public static decimal cartPrice;

public static List<stCart> cart = new List<stCart>();

public static int iItems;

}

public static class User

{

public static long ID;

public static string Username;

public static string Firstname;

public static string Lastname;

public static int AccessLevel;

public static bool isStaff;

}

public struct stCart

{

public string ModelNo;

public string Name;

public decimal Price;

public int invID;

public decimal dGST;

public stCart(string modelno, string name, int qty, decimal price, int invid, decimal dgst)

{

ModelNo = modelno;

Name = name;

Price = price;

invID = invid;

dGST = dgst;

}

}

}

# 6.0 Questionnaire Evaluation

To further evaluate the success of the program a questionnaire was produced and distributed to several random customers in the store. The questionnaire consisted of 10 questions ranging from yes/no questions to sentence responses to ensure that enough data was collated (See appendix 9.1 for results). The survey was sent to users through the google forms platform.

The questions were:

1. Are the forms user friendly?
2. Would you recommend this application to family and friends?
3. How would you rate the design of the interface?
4. During testing did you discover any bugs or problems with the application?
5. Did all the queries return the correct data? If not please mention the form
6. How would you rate the ease of the interface?
7. What parts of the software was the most confusing or could use improvement?
8. How would you rate the ease of the process to purchasing an item?
9. Did you find it difficult at times to understand how to use a form to complete a process (e.g. purchasing an item or creating a user account). If so, which form?
10. Would you agree that more visual prompts should be included, explaining how to use the form?
11. Extra comments

By analysing the trends in the data, it is evident that the purchase form is the most confusing form and is causing the most amount of disruption among the users with 57.1% of users voting in question seven that the form could use improvement, compared to 28.6% of users or two users who said that no forms need improvement and 14.3% or one user saying that the manager form needs improvement. This data shows how many people are rating the purchase form to be too confusing and difficult to understand which could negatively affect the business in the future. These results can be synthesised in question eight with five out of seven users responding that the purchase form was too confusing rating the form between four to two. If people find the form too confusing customers may be turned away from the business and shop elsewhere affecting sales and negatively affecting the business. It is recommended that the design for this form be reconsidered and changed in future updates to ensure that the customer base finds the program easy to navigate and doesn’t negatively affect the business (previously mentioned in section 5.6.5.2).

To evaluate whether the purchase form is the cause of most confusion, the extra comments question responses where six out of seven users responded can be analysed. By analysing the data, it can be seen that the two users who voted in question seven that no forms needed improvement are the two responses whom they had previous cashier experience stating, “I have worked in many cashier settings in the past and so using this program was not difficult compared to other programs I have traditionally used.”. However, this group is the minority is most situations as the majority of people who will be purchasing items at the store may never have worked as a cashier previously. Furthermore, by looking deeper it can be seen that three out of the six responses stated something about the difficulty of the purchase form stating similar comments such as “I was really confronted by the purchase form interface and found it incredibly difficult to use without some assistance. All other forms I found easy apart from the purchase form” or “Overall, in the past, I have been generally quite adept at using computers, however, I feel a less experienced person could struggle to use the purchase form.”. These quotes directly from the questionnaire responses show how people struggled with the use of the purchase forms which reinforces the prediction earlier that the purchase form is creating much confusion among the userbase. As stated earlier, an action is recommended to rectify this problem.

Most users scored the user-friendliness of the project highly with question one having one user scoring the user-friendliness at three, three users scoring the user-friendliness at four, and three users scoring the user-friendliness at five. These results in question two can be synthesised with the results in question three as both partially based on the user interface. As seen in question two, all but one user (who rated the design a three) scored the design a four or five. These shows how most of the forms are generally quite user-friendly and intuitive making the program easier to navigate and increasing business efficiency. It can also be seen that people enjoy the design scheme of the forms which can make the business more appealing to shop with again. The reason for the lower ratings is most likely due to the purchase form, as discussed earlier which was reviewed by users to be too confusing. It is recommended in future that more attention to detail be taken into consideration when designing forms as the design will be one of the main causes that will decide if the user will want to shop at the store again. The sample size for this testing was small with only several responses meaning, results are expected to change in a real setting.

In question two, many users decided that if they had/wanted to, they would recommend the application to other friends with four out of seven users rating the form at five, two out of seven at four and one out of seven at three. These results show that the form has been designed well enough that the user would be either be likely happy to recommend the application or heavily consider recommending it. It also means that if the user is happy to recommend the application, they are probably also happy to shop with the business again. This is good for business sales and growth as customers (from a software point of view) would be happy to recommend the business to friends/family increasing the total amount of customers in the store. However, the customer satisfaction with staff and service quality are still factors that could make or break this prediction.

Furthermore, it can be observed that though many are ready to recommend the application, many would still not call the experience completely easy. This is easily seen in question six where three out of seven users responded with three’s, two out of seven with fours and two out of sevens with fives. Though the majority is still in the higher bands (4 and 5) there are still quite a large majority nearly half (42.9%) of users complaining the forms are generally not easy to use. This could create problems as efficiency and time needs to be a factor when these machines are used. Because they can be used as self-service checkout machines, they must be easy enough for the user to be able to quickly and with little difficulty checkout their desired products. However, if forms such as the purchase form (the form all customers will be using), a form which has already been rated low, is too difficult to use, time spent at a self-service checkout machine could be increased due to the interface is less intuitive, increasing line/queue times which could cause customers to not shop with the business in the future. As mentioned earlier in the questionnaire evaluation and in section 5.6.5.2, it is highly recommended that the design to the purchase form be reconsidered as this could be a hindrance to business efficiency and growth due to the complexity and difficulty users have interacting and using the form.

On the other hand, all SQL queries, according to users worked correctly 100% of the time with all seven survey participants responding yes to question five which asked “Did all the queries return the correct data? If not please mention the form”. This is crucial as the queries make up the fundamentals of the program, acting as the ‘middleman’ between the program and database when retrieving, editing or storing data in the databases. If the queries were faulty or the returned results were incorrect the program would need to be either sent back to receive patches or a third party would need to step into to solve the problems, however, at the current point in time and in trails no issued with the queries have been found which is beneficial to the business.

These results are the same for question four as all seven survey participants of 100% of participants responding that no bugs were discovered during the trials/testing of the program. It is crucial that this is the case as bugs in the code could cause data in the database to become incorrect or corrupt negatively affecting the business. Furthermore, if errors were found and the system was deployed, customers could run into the issues which could have caused users to not want to shop with the business again. Bugs or problems in the code/program could have created major problems for the business as money would need to be spent patching the program for usage, which would be anti-beneficial to the business.

However, though the program is bug-free, it was discovered in question ten that the majority of users (four out of seven or 57.1% of survey participants) thought that more visual prompts should have been included in the design of the forms. Including visual prompts for the user ensures that the user can follow along with the form and its procedure increasing efficiency at each machine as the required staff assistance would be lessened. Though there are some visual prompts in the form, most notably the new staff form and purchase form, it can be recommended that in subsequent updates more visual prompts or access to visual prompts/instructions be available directly on the form for the user to ensure they can navigate the form more quickly. One method of approaching this could be to use a button that when trigged displays temporary popup messages around the screen indicating the forms procedure and what each element/button does. A method such as this would increase time efficiency as staff members do not need to help customers to the same extent allowing staff to spend their time on more important tasks. Moreover, by adding visual prompts of some sort the user satisfaction would likely increase as users would not feel the frustration of needing to ask a staff member for help and would be more comforted shopping with the business again as they have the comfort of viewing the visual prompts again.

# 7.0 Final Overall Reccomendations

To summarise the points made in the individual evaluations, an overall final recommendations section was written to provide an overview of the recommendations made.

Firstly, before going any further, the first recommendation to be made would be to create another survey that consisted of a wider demographic and more participants, so more results accurate and representative results can be collected. Only seven people were surveyed in the survey conducted above decreasing the accuracy of the data represents the general public, meaning in a real-life setting the responses could be completely different to the results found in the survey. If more representative data wanted to be collected there would need to be a minimum 100 participants take part in the survey and trailing the program to ensure that a wide demographic is surveyed.

Secondly, the usage of more functions is highly recommended as code became difficult to read due to the length and complexity of forms. Not to mention, much of the code in the program was repeated multiple times when it could have instead been placed into functions where the functions could be used multiple times on different queries making the code more readable and increasing efficiency as fewer SQL objects would be initialized at once. Because the readability is increased by using functions, if the program was updated to use more functions, the updatability of the program is increased as developers can better read the code. Furthermore, the benefit of increased efficiency means that the program will run faster, smoother and have a lower chance of failure which in a customer interface that deals with money is crucial. By updating the program to split more code across smaller functions the business will reap the long-term benefits in years to come when the program needs to be updated to support newer versions of windows.

Thirdly, it is highly recommended that the purchase form is redesigned to create a friendlier user interface that is easier to understand as it was discovered during the questionnaire that many users found the purchase form too difficult and confusing to use. By redesigning the form, it is expected that users would find the form easier to understand making the checkout time per customer lower as the form becomes more intuitive and the procedure becomes easier to understand. Furthermore, the redesign of the purchase form also means that customers are more likely to leave satisfied rather than frustrated with the current purchase form. Though the form to many experienced cashiers is easy to use, to the average middle-aged or senior person the form is likely too confusing. It would be in the business's best interest to redesign the form to ensure a retained satisfied customer base.

Fourthly, it is recommended that a barcode/key card system be implemented for products and customers/staff to easily scan to login or scan products to edit in the inventory form or purchase in the purchase form. By implementing a barcode/key card system the amount of time spent during checkout would be lowered as the user would not be required to enter manually enter in each set of details for login or product information but rather just scan their key card/product barcode. This would benefit the business as more customers would be able to move through the checkouts during a set time frame due to checkout times per customer is lower.

Fifthly it is hugely recommended that more search bars be included in the overall design of the form as many forms lack this crucial feature causing many users to resort to searching the DataGridViews manually. This is a waste of both staff and customer time where time could be better dedicated to other tasks increasing business efficiency and possibly boosting sales of the business. A recommendation such as this could be highly beneficial to the business and should be highly considered.

Lastly, it is recommended that could drastically increase the efficiency, security and features of the program would be to utilise third-party services. By using third-party services, code can be implemented that is specifically designed for a purpose. This can increase security in some situations such as logins where data needs to be secured and private to ensure that a data leak does not occur. Implementing third-party solutions also means that the solutions being implemented have been specifically designed for the purpose they are incorporated for meaning more time has been spent on those specific features improving them and ensuring they are the working correctly unlike the solutions incorporated in this current version where time has had to be split across multiple sections of the program. By using a third-party solution, it can be predicted that the overall experience of the program would be more efficient and smoother as elements incorporated specifically designed for the features they are being used for.

Overall the recommendations that have been listed here are the summaries of recommendations made during the evaluation process that has been picked out as important or for the business to highly consider. These are only recommendations and are not necessary for the running of the program but should be thought should be placed into the recommendations made. There are still many other actions or recommendations that could be made to better the program, however, these are just recommendations that were considered of higher performance.

# 8.0 Conclusion

To conclude this project, the program that was eventually developed managed to fit the requirements of the business as it met the goals and the requirements of each form set out in the design of the project. The program was successful during testing with no bugs found as of yet however, this cannot be guaranteed. In this project, the database and initial design of the program were planned and during part B of the project the design was implemented, changes were made, and these changes were mentioned. Furthermore, an evaluation and survey were performed to determine the success of the project. After everything that has been considered, the project can be deemed successful as it fulfils the businesses needs and goals initially laid out during the design, not to mention the program works successfully with all queries returning the correct results. However, there are recommendations that should be addressed in future updates that would increase the quality of the program and customer satisfaction. Though all functions currently wok, if the recommendations were addressed the program would be a great solution to the problem initially set put forth (replace a paper inventory and invoice system) as it would be more user-friendly and appealing to customers. Overall the project managed to address the original issue and was successfully created by the deadline leaving the project with a positive outcome.

# 9.0 Appendix

### 9.1 Questionnaire results

