Project Milestone 1

PRG 281

Project

Lecturer: Sbusiso Mhlabane

Students:

1. Ofentse Mapheto 2.
2. Nthabiseng Motshwane
3. Retema Sedibu Abel
4. Sindisiwe Mnukwa

Table of Contents

1. Introduction

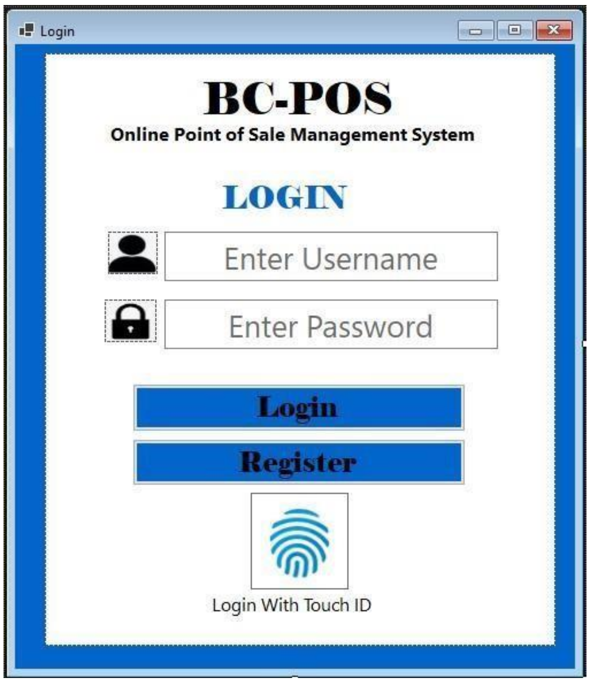
This document contains the planning that group 8 did for the milestone one of the PRG 2x1 project. It also contains an explanation of the different learning outcomes that will be used and why and how they will be used.

Key words:

- POS – Point of Sale

2. Planning of pseudocode and form layout.

2.1 Layout of the login form.



The Login Form gets displayed 1st to the user. The user has 2 options, Login or Register. If the user picks Login 1st the program checks for a directory to read his login details. If the directory isn't found it means the user doesn't exist yet, then the user gets forced to register first. Alternatively, if the directory is found the program reads the login details and if the login details in directory match what the user typed into the textboxes, the user gets logged in, else it tells the user login details are wrong. Alternatively, if the user picks Register 1st, the Register form gets opened and the Login form is closed.

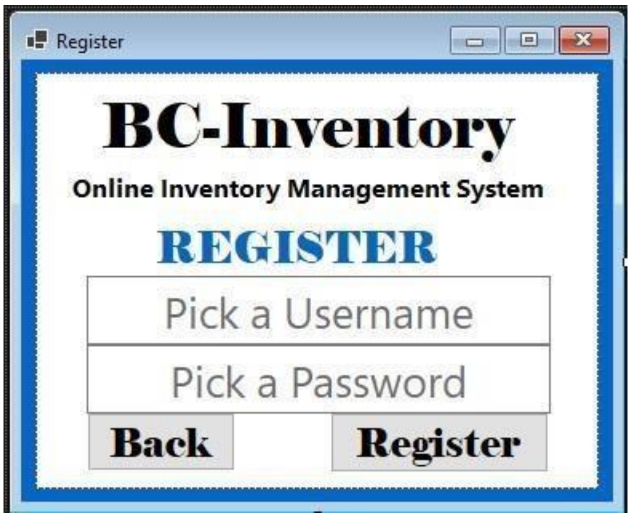
2.2Pseudocode for login form.

```
Declare public string username, password
Private Void When Login Button is Clicked first()
```

```
{
   TRY
      {
Sr = new StreamReader that reads from folder called (Username Textbox + "\\login.ID"+); username
= 1st Line Sr reads; password = 2nd Line Sr reads;
Close Sr;

IF ((username = Username-TextBox) & (password = Password-TextBox ))
   { Show Message ("You are now successfully logged in");
       Close Login-Form;
       POS-Form f1 = new Instance of POS-Form;
       Show POS-Form f1;
   }
ELSE
{
       Show Message ("Username or Password is wrong");
   }
}
  CATCH (System.IO Could Not Find Directory Exception)
       {
           Show Message ("The user does not exist yet, please register first")
           Disable Username-TextBox;
           Disable Password-TextBox;
       }
}
Private Void When Register Button is Clicked first()
{
    Hide Login-From f2();
    Register-Form f3 = new Instance of Register-From();
    Show Register-From f3();
}
```

2.3Layout of the register form.

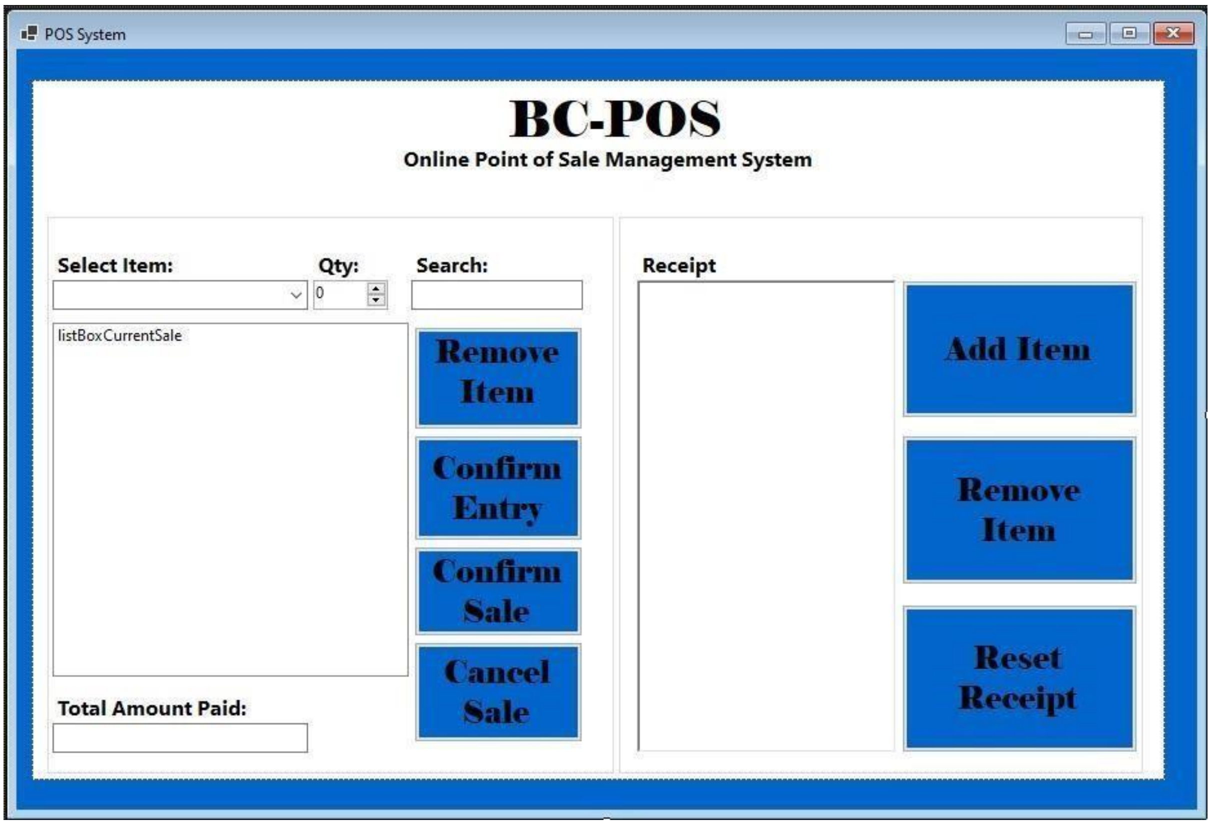


Register Form is responsible for creating a directory in the program folder that stores the login details of the user. If the user enters his/her details and clicks register, the program tries to create a directory named after the user and creates a text file to store the user credentials. If the program successfully creates the directory the user's username and password gets entered, else if the program can't find the directory it creates a new directory in the same fashion.

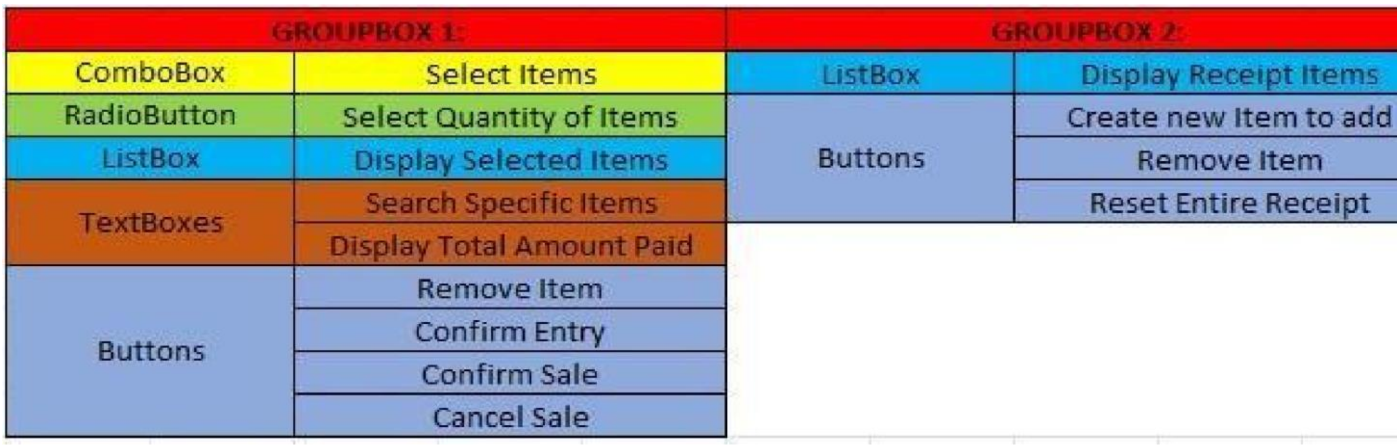
2.4 Pseudocode of the Register form

```
Private Void When Register Button is Clicked()
{
    TRY
        {
            Sw = new StreamWriter that creates a folder called (Username Textbox + "\\login.ID"+);
            Sw Writes inside login.ID (Text in Username TextBox + "\n" + Text in Password TextBox);
            Close Sw;
            Show Message ("You are now successfully registered");
            Close Register-Form;
        }
    CATCH (System.IO Could Not Find Directory Exception)
        {
            System.IO Creates New Directory called (Text in Username TextBox);
            Sw = new StreamWriter that creates a folder called (Username Textbox + "\\login.ID"+);
            Sw Writes inside login.ID (Text in Username TextBox + "\n" + Text in Password TextBox);
            Close Sw;
         }
}
Private Void When Back Button is Clicked()
{
    Close Register-Form;
}
```

2.5Layout for the POS form.



POS Form is split up into 2 main Group Boxes which contain:



GROUPBOX 1:		GROUPBOX 2:	
ComboBox	Select Items	ListBox	Display Receipt Items
RadioButton	Select Quantity of Items	Buttons	Create new Item to add
ListBox	Display Selected Items		Remove Item
TextBoxes	Search Specific Items		Reset Entire Receipt
	Display Total Amount Paid		
Buttons	Remove Item		
	Confirm Entry		
	Confirm Sale		
	Cancel Sale		

Within the project there will be two main text files from which data will be retrieved and written to, these text files will be named SalesList.txt and Items.txt. The POSForm.Load event is responsible for reading all the data that exists within our text files and populating them into the storage objects that have been created in the code like the various lists and other forms of containers.

2.6Pseudocode for POS-FORM.LOAD

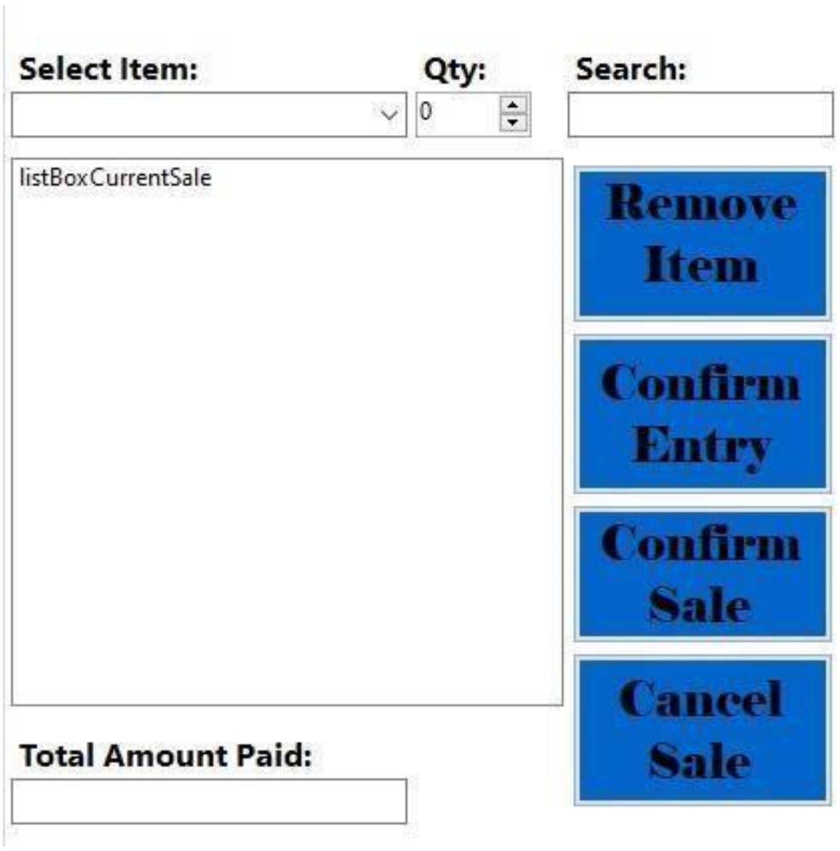
```
Private Void POS-Form.Load
{
   Declare decimal ItemQuantToObject, SaleTotal = 0;
```

```
    Create List<int> StockItemsTotal;
    Create List<StockItem> StockInSale;
    Create instance of class SaleInstance called CurrentSaleInstance;
    Create instance of class StockItem called CurrentItem;
    Declare string fileselected = " ";
    Declare string[] TextFileContent;
    Declare string ItemNameToObject;
    Declare decimal ItemPriceToObject;

    fileSelected = Path of Directory + "\\Items\\ItemList.txt";      TRY
        {
            TextFileContent = ReadAllLines from (fileSelected);
        }
    CATCH
        {
            Create FileStream Fs = fileSelected;
            Close Fs
        }
     FOR (0 to length of TextFileContent,i++)
        {
            FOR(0 to length of TextFileContent -1 ,j++)
                {
                    IF(TextFileContent[i].Substring(j,1) = "=";
                     {
                        IF(TextFileContent[i].Substring(j+1,1) = "=";
                          {
                             ItemNameToObject = first section of string before "=";
                             ItemPriceToObject= second section of string after "=";
                             CurrentItem = new iteration of class StockItem(returns ItemNameToObject,
ItemPriceToObject);
                              Add CurrentItem to Variables;
                          }
                     }
                }
        }


Private Void POS-Form.Load
{
        Fileselected = Path of Directory + \\Sales\\SalesList.txt;    TRY
      {
          TextFileContent = ReadAllLines from FileSelected;
      }
    CATCH
      {
          Create FileStream Fs = Create FileSelected;
          Close Fs;
      }
            TextFileContent = ReadAllLines from FileSelected;
    FOR(i=0 to length of TextFileContent -1,i++)
      {
        IF(TextFileContent[i].Substring(0,3) = “+++”
          {
            SaleTotal = TextFileContent[i].Substring(0 to length of TextFileContent[i]-3);
          }
        ELSE IF(TextFileContent[i].Substring(0,3) = “---”
          {
            CurrentSaleInstance = new instance of class SaleInstance(returns
StockInSale,StockItemTotal,SaleTotal);
            Add CurrentSaleInstance to Variables;
            Clear StockInSale;
            Clear StockItemsTotal;
          }
         ELSE
          {
             FOR(j=0 to length of TextFileContent[i] -1,j++)
               {
                   IF(TextFileContent[i].Substring(j,1) = “=”)
                     {
                        IF(TextFileContent[i].Substring(j+1,1) = “=”)
                          {
                             String sub = TextFileContent[i].Substring(0,j);
                             ItemNameToObject = TextFileContent[i].Substring(0,j);
      Sub      =      TextFileContent[i].Substring(j+2,length      of      TextFileContent-j-2;
ItemQuantToObject= TextFileContent[i].Substring(j+2,length of textfilecontent-j-2);
CurrentItem = new instance of class StockItem(returns itemNameToObject,ItemQuantToObject);
                             Add CurrentItem to StockInSale;
                             Add ItemQuantToObject to StockItemsTotal;
                          }
                     }
               }
          }
      }
```

2.7 Layout for GroupBox1



In the first group box of the POS-FORM there are 3 types of buttons namely:

- **Remove Item Button:**

This button is used to remove an item from the current sale **Confirm Entry Button:**

Each time an item is logged to the sale the Confirm Entry Button is used to append the item to the current sale

- **Confirm Sale Button:**

Once all the items have been appended to the current sale the Confirm Sale button is clicked. This will write the entire sale to the SalesList text file in the program folder. After this is done a grand total is also displayed to the user.

2.8 Pseudo for remove item from sale button

```
Private Void When RemoveItemFromSale Button is clicked
{
   Declare int CurrentListPos, Index;
   Declare decimal Price;

   Index = selected item in the ListBox;
   Remove value of Index from list ItemLineTotals;
   Remove index from list CurrentSales;
   Remove the quantity of index from list QtyItems;
   Clear entire ListBox for current sales;
   FOREACH(item in list CurrentSales)
```

```
          {
              Add item to ListBox CurrentItems;
             Iterate through list position;
          }

}
```

2.9 Pseudo for confirm entry button.

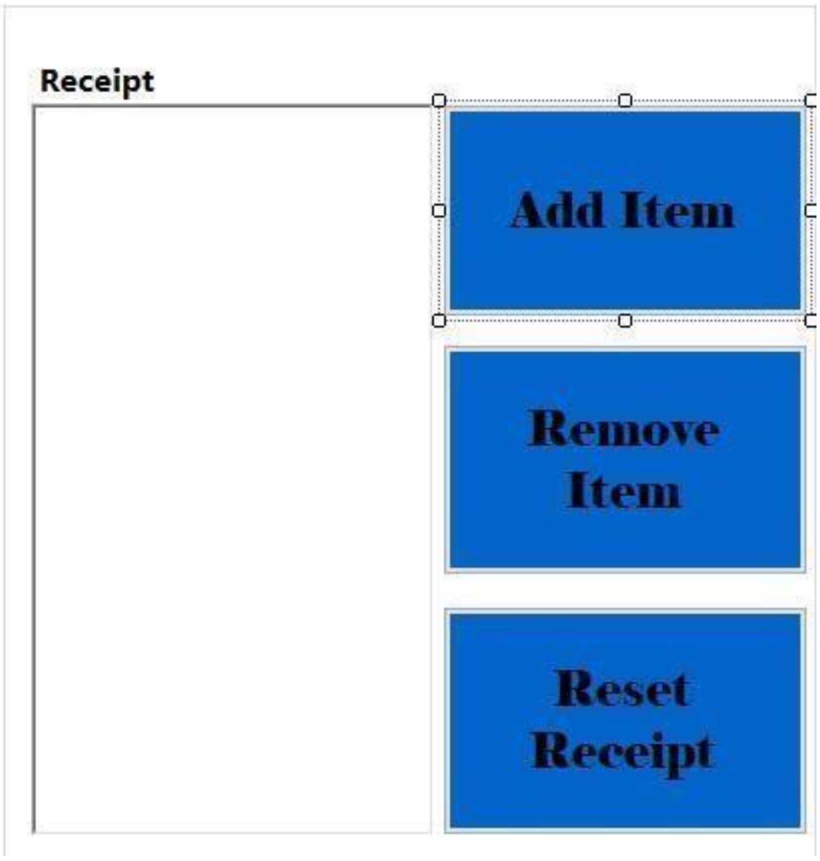
```
Private Void When ConfirmEntry Button is clicked
{
   Declare string tempstring;
   Declare int tempint;

   Tempstring = selected item in the combobox;     Tempint
= quantity from quantity combobox;
Add tempint to class Variables;
   FOREACH(item in Variables.items)
       {
           IF(itemdescription = tempstring)
             {
                 Variables.tempdecimal = itemprice;
             }
       }
   Add tempint to itemtotals;
   Add tempdecimal to salestotals;
   Add tempstring to Stockitem;   }
```

2.10 Pseudo for confirm sale button

```
Private Void When ConfirmSale Button is clicked
{
    Create new instance of class SaleInstance;
    Add currentsale to  SaleHistory list;
}
```

2.11 Layout for GroupBox2



 Add stock Item Button:

Redirects user to the Add Item Form
Remove stock Item Button: Redirects user to the Remove Item Form **Reset Receipt Button:**
Clears receipt display

2.12 Pseudo for add item button

```
Private Void When AddItem Button is clicked
{
Create new instance of FormAddItem();
                    Show
FormAdd;   }
```

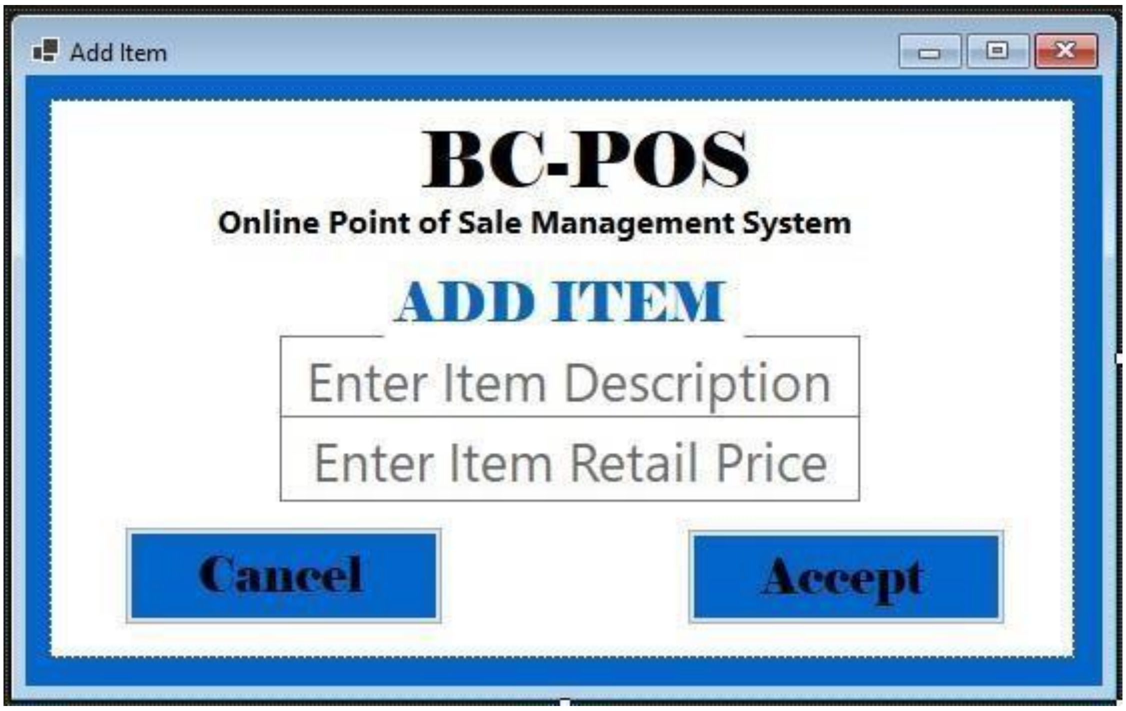
2.13 Pseudo for remove item button

```
Private Void When RemoveItem Button is clicked
{
Create new instance of FormRemoveItem();
          Show FormRemove;
}
```

2.14 Pseudo for reset receipt button

```
Private Void When ResetReceipt Button is clicked
{
Clear RichTextBoxReceipt
    }
```

2.15 Layout for Add item form



This form will be used to create new items and log them into the items textbox to be read from when a sale is made.

2.16 Pseudo for add item form

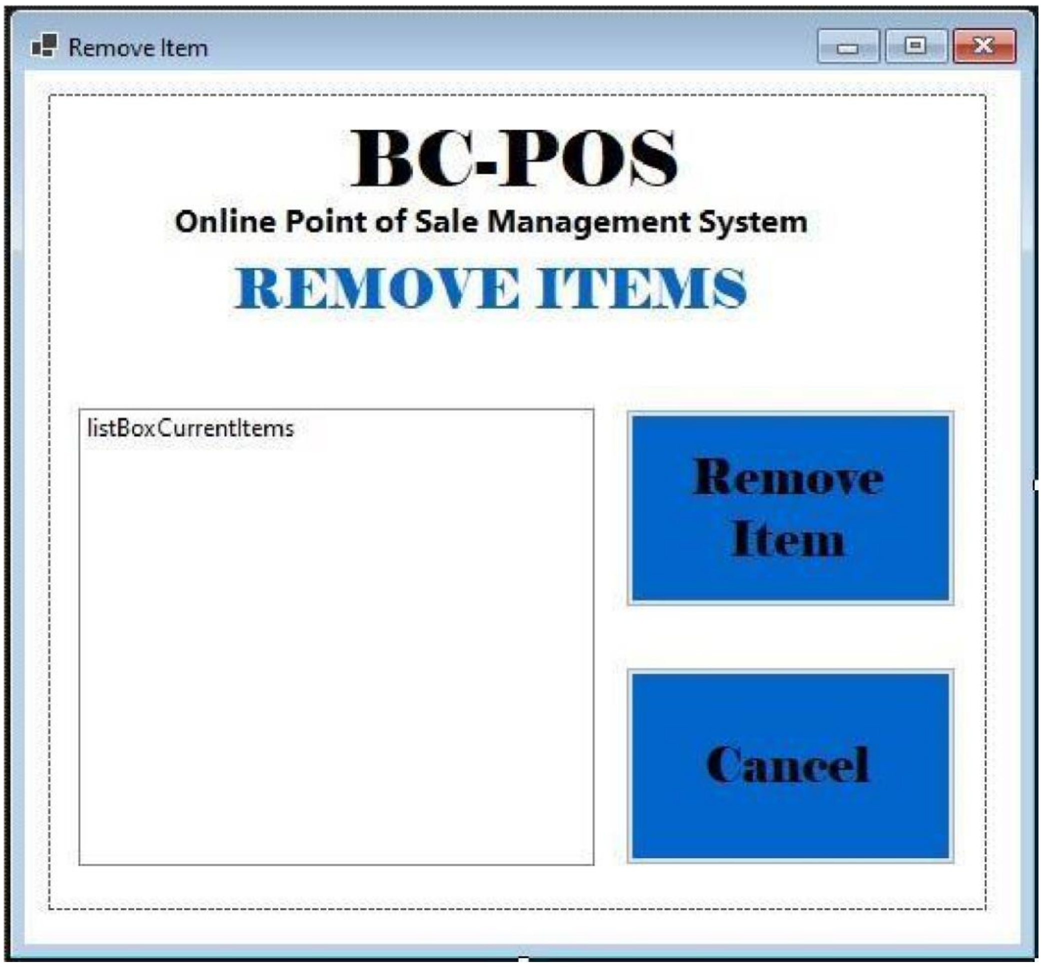
```
START
{
        print 'BC-POS',
        print 'Online Point of Sale Management System',
        print 'ADD ITEM'

        string name;
        decimal price;

        textbox input 'Enter Item Description';
        textbox input 'Enter Item Retail Price';

        if (buttonAcceptClick == true)
                {
                        Add TextboxDesc and TextboxPrice to StockItem list;
                        Write TextboxDesc and TextboxPrice to Item textfile;
}
        If (buttonCancelClick == true)
                {
                        Close AddItemForm
                }
}
END
```

2.17 Layout for remove item form



This form will be used to permanently remove items from the items textbox so it cannot be read when creating a new sale.

2.18 Pseudo for remove item list

```
frmRemove list

List added items

 If (RemoveItemsClick == true)
{
        print "which item would you like to remove?" Get userInput

                If (ItemFound == true)
{                       remove item
}
       Else
{
                        print "item not found".
}
}
  If (CancelClick == true)
{
Rollback changes
}
```

3. Topics that will be used

3.1 Classes and objects

In C#, a class consists solely of the attributes and methods necessary to represent a real-time entity (Thomspon, 2022). Data methods and properties are contained within classes (Thomspon, 2022). The properties describe the data the class holds, and the methods tell you what operations can be performed on the data (Thomspon, 2022).

In C#, an object represents actual entities and is the fundamental building block of object-oriented programming (Kirloskar, 2021). These are things such as car, pen, laptop etc (Kirloskar, 2021). At each given time, an object is an instance of a class (Thomspon, 2022). The fact that an object has values for the properties as opposed to a class distinguishes them (Thomspon, 2022).

To achieve the highest amount of functionality and readability the developers choose to make use of multiple classes containing sale instances and objects that can be sold in the store using the POS system. This will also ensure ease of coding for the team in the development process.

3.2 Encapsulation

Encapsulation is one of the main language features of an object-oriented programming language (Ganesh, 2019). The definition of encapsulation is the grouping of data into a single unit (Saini, 2019). The process of combining data and functions into a single unit (called class) is termed encapsulation (Ganesh, 2019). It is the mechanism that connects the data the code manipulates with the code itself (Saini, 2019).

Data can be safeguarded against unintentional corruption using encapsulation (Ganesh, 2019). Encapsulation, on the other hand, also functions as a barrier that stops code from outside the barrier from accessing the data (Saini, 2019).

Technically speaking, encapsulation means that a class's variables or data are kept private from other classes and are only accessible through member functions of the class in which they were declared (Saini, 2019). This action of hiding the data is also known as data-hiding (Saini, 2019). Encapsulation can be accomplished by declaring all class variables as private and using C# class properties to set and retrieve variable values (Saini, 2019).

To avoid corruption and protect the data that are read from the text files and added to the lists the developers chose to use encapsulation. This allows the developers to read and write the needed data from the list without the fear of data corruption. This will also speed up the coding process as referencing a single object can return multiple data points.

3.3 Best coding practises

Consider coding standards as a set of guidelines, methods, and best practices for writing cleaner, easier-to-read, more effective code with few mistakes (Bose, 2021). They provide a standard structure that software engineers can utilize to create complex, highly functioning code (Bose, 2021). Code readability is essential for development since it is essential for maintainability and teamwork (Guzel, 2021).

Some of these coding practices include:

- Try to keep your writing to a minimum (Bose, 2021). Apply the proper naming conventions (Guzel, 2021).
- Create paragraphs out of code blocks that are within the same section (Guzel, 2021).
- To indicate the start and end of control structures, use indentation. Indicate the connection between them in detail. (Bose, 2021)
- Avoid using long functions. A single function should, ideally, complete a single task (Bose, 2021).
- Don't repeat yourself; use the DRY principle (Bose, 2021). When necessary, automate repetitious processes (Bose, 2021). The script shouldn't contain several instances of the same piece of code (Bose, 2021).
- Don't deep nest (Guzel, 2021). Coding becomes more complex when there are too many levels of nesting (Guzel, 2021).
- To distinguish them from table and column names, capitalize the special characters in SQL function names (Bose, 2021).
- Stay away from long lines (Bose, 2021). Humans find it simpler to read blocks of lines that are short on the horizontal axis, but long on the vertical axis (Bose, 2021).

To ensure readability the developers chose to include these above best practices. A uniform naming convention will be used as well as, using the dry principles where possible.

3.4 Form events

Windows Forms lets you drag UI elements onto a canvas to graphically design desktop applications (Khan, 2022). Widgets such as buttons, panels, or checkboxes are examples of these UI elements (Khan, 2022). Events are sent to each UI element individually (Khan, 2022). For instance, you may have a drag-and-drop event for panels, a modified event for checkboxes, or a click event for buttons (Khan, 2022).

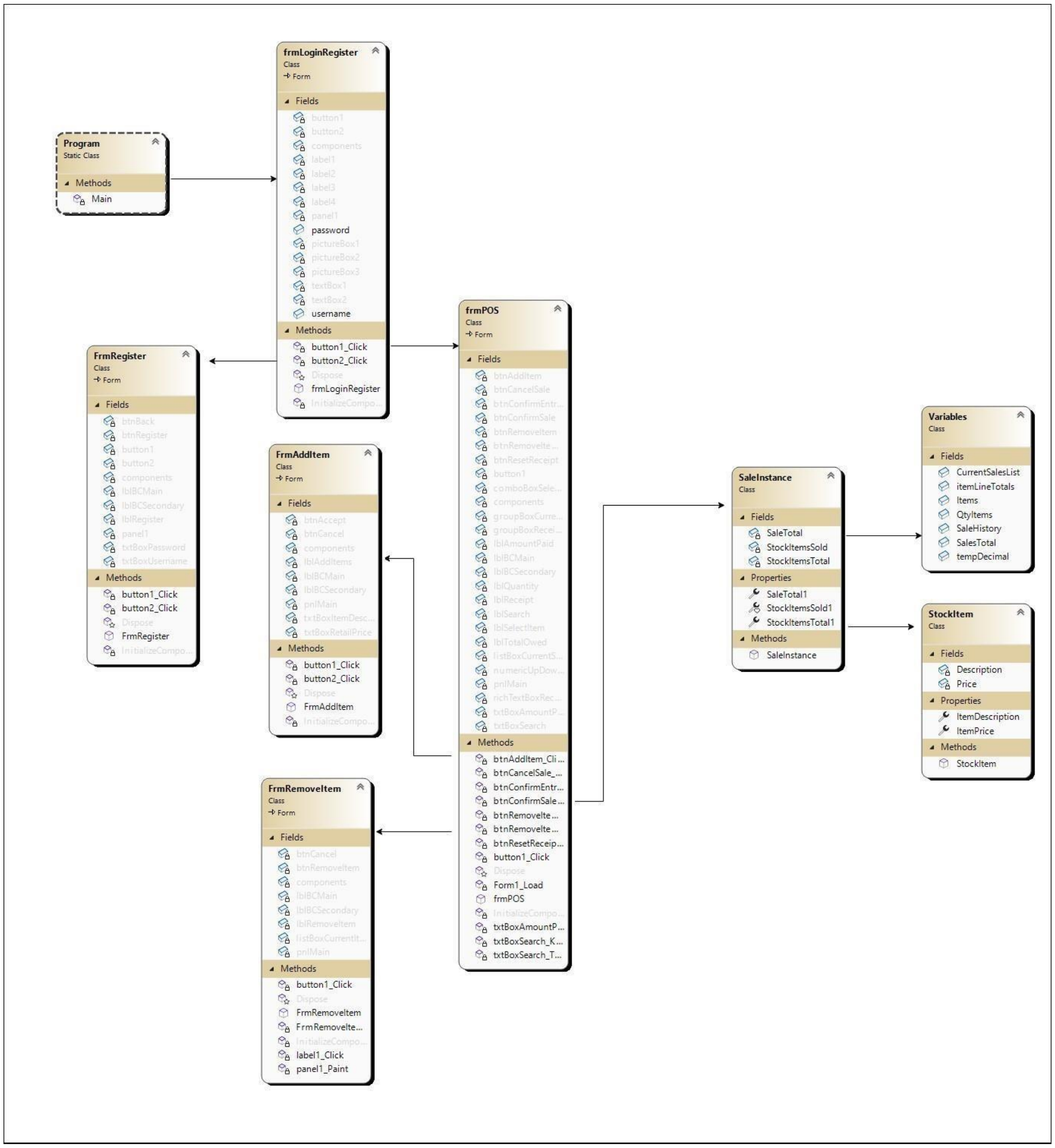
To ensure user friendliness the developers chose to make use of form events. These will include things such as the ability to search the items contained in a drop box, and multiple buttons that will do different functions

3.5 Exception Handling

Unfortunately, one of the most ignored subjects in the instruction of new software engineers is how to manage mistakes and other issues in the code (Watson, 2020). An occurrence that takes place during the execution of a program that the program code does not expect is known as an exception (Kirloskar, 2021). In this situation, we construct an exception object and run the function that handles exceptions (Kirloskar, 2021). Using an exception handler to prevent a program's code from crashing is known as exception handling (Kirloskar, 2021).

To avoid random software crashes due to unexpected actions the developers chose to use exception handling. Exception handling will be used for calculations to ensure that no unexpected values will cause an unwanted result. This will be achieved by using C#’s built in tryParse and try catches.

Class diagram


Program
Static Class
Methods
Main
frmLoginRegister
Class
Form
Fields
button1
button2
components
label1
label2
label3
label4
panel1
password
pictureBox1
pictureBox2
pictureBox3
textBox1
textBox2
username
Methods
button1_Click
button2_Click
Dispose
frmLoginRegister
InitializeCompo...
FrmRegister
Class
Form
Fields
btnBack
btnRegister
button1
button2
components
lblBCMain
lblBCSecondary
lblRegister
panel1
txtBoxPassword
txtBoxUsername
Methods
button1_Click
button2_Click
Dispose
FrmRegister
InitializeCompo...
FrmAddItem
Class
Form
Fields
btnAccept
btnCancel
components
lblAddItems
lblBCMain
lblBCSecondary
pnlMain
txtBoxItemDesc...
txtBoxRetailPrice
Methods
button1_Click
button2_Click
Dispose
FrmAddItem
InitializeCompo...
FrmRemoveItem
Class
Form
Fields
btnCancel
btnRemoveItem
components
lblBCMain
lblBCSecondary
lblRemoveItem
listBoxCurrentIt...
pnlMain
Methods
button1_Click
Dispose
FrmRemoveItem
FrmRemoveIte...
InitializeCompo...
label1_Click
panel1_Paint
frmPOS
Class
Form
Fields
btnAddItem
btnCancelSale
btnConfirmEntr...
btnConfirmSale
btnRemoveItem
btnRemoveIte...
btnResetReceipt
button1
comboBoxSele...
components
groupBoxCurre...
groupBoxRecei...
lblAmountPaid
lblBCMain
lblBCSecondary
lblQuantity
lblReceipt
lblSearch
lblSelectItem
lblTotalOwed
listBoxCurrentS...
numericUpDow...
pnlMain
richTextBoxRec...
txtBoxAmountP...
txtBoxSearch
Methods
btnAddItem_Cli...
btnCancelSale_...
btnConfirmEntr...
btnConfirmSale...
btnRemoveIte...
btnRemoveIte...
btnResetReceip...
button1_Click
Dispose
Form1_Load
frmPOS
InitializeCompo...
txtBoxAmountP...
txtBoxSearch_K...
txtBoxSearch_T...
SaleInstance
Class
Fields
SaleTotal
StockItemsSold
StockItemsTotal
Properties
SaleTotal1
StockItemsSold1
StockItemsTotal1
Methods
SaleInstance
Variables
Class
Fields
CurrentSalesList
itemLineTotals
Items
QtyItems
SaleHistory
SalesTotal
tempDecimal
StockItem
Class
Fields
Description
Price
Properties
ItemDescription
ItemPrice
Methods
StockItem

Division of work

Table that shows the students and each of their responsibilities		
Student ID	**Student Name**	**Responsible for**
576865	Ofentse Mapheto	• Project Manager • Will code functionality of POS form • Class creation for entire system • Will create layout of text files • Creation of add item and remove item forms
576728	Nthabiseng Motshwane	• Will code functionality of POS form • Write pseudo code for milestone 1 document • Describe the use of learning outcomes • Compile milestone 1 document • Write pseudo code for milestone 1 document • Will code sales history functionality
578535	Retema Sedibu Abel	• Created login and register forms • Will code functionality of login and register forms • Supply screenshots of forms • Write pseudo code for login, register and POS form • Help with creation of remove item form • Will code remove item functionality
576745	Sindisiwe Mnukwa	• Help with creation of add item form • Write pseudocode for add item • Will code add item functionality • Wrote pseudocode for remove item • Created class diagram

References

Bose, S., 2021. *Coding Standards and Best Practices To Follow.* [Online]
Available at: https://www.browserstack.com/guide/coding-standards-best-practices
[Accessed 28 7 2022].
Ganesh, G. A., 2019. *C# Encapsulation.* [Online]
Available at: https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/ [Accessed 28 7 2022].
Guzel, B., 2021. *Top 18 Best Practices for Writing Super Readable Code.* [Online] Available at: https://code.tutsplus.com/tutorials/top-15-best-practices-for-writingsuperreadablecode--net-8118 [Accessed 28 7 2022].
Khan, S., 2022. *How Events Work in a Windows Form Application.* [Online]
Available at: https://www.makeuseof.com/windows-form-application-events-how-work/ [Accessed 28 7 2022].
Kirloskar, M., 2021. *Different ways to create an Object in C#.* [Online]
Available at: https://www.geeksforgeeks.org/different-ways-to-create-an-object-in-c-sharp/ [Accessed 28 7 2022].
Kirloskar, M., 2021. *Exception Handling in C#.* [Online]
Available at: https://www.geeksforgeeks.org/exception-handling-in-c-sharp/ [Accessed 28 7 2022].
Saini, A., 2019. *C# | Encapsulation.* [Online]
Available at: https://www.geeksforgeeks.org/c-sharp-encapsulation/ [Accessed 28 7 2022].
Thomspon, B., 2022. *C# Class & Object Tutorial with Examples.* [Online] Available at: https://www.guru99.com/c-sharp-class-object.html [Accessed 28 7 2022].
Watson, M., 2020. *C# Exception Handling Best Practices.* [Online]
Available at: https://stackify.com/csharp-exception-handling-best-practices/ [Accessed 28 7 2022].