# 

**MS806 – Business Application Programming**

# 

**Student Name: Akhil Srinivas Anantha Rajan**

**NUI Galway ID Number: 20230232**

**Assignment:**  4

**Date of Submission:** 17/01/2021

**TABLE OF CONTENTS**

**S.NO TITLE PAGE NO.**

**1** **Client Profile 5**

1.1 General System Specification 6

**2 Developers Manual**  **7**

2.1 Tools & Skills requirement for development of

EPOS Application 7

2.2 Basic Design 7

2.2.1 Forms 8

2.2.2 Panels 12

2.2.3 Listboxes 15

2.2.4 Radio Buttons 15

2.2.5 Grid View 15

2.2.6 Buttons 15

**3 Appendix 17**

3.1 Primary Form 17

3.2 SearchOrderDetails Form 24

3.3 Stock Availability Form 25

**LIST OF FIGURES**

**FIGURE NO. TITLE PAGE NO.**

2.2.1A Primary Form 8

2.2.1B Search Order Details 11

2.2.1C Stock Availability 11

2.2.2A Flowchart – Ordering Panel 12

2.2.2B Flowchart – Basket Panel 13

2.2.2C Confirmation on the Order 14

2.2.4A Warranty Options 15

**LIST OF TABLES**

**TABLE NO. TITLE PAGE NO.**

1A Watch Brands and their Price 5

1B Category and their Cost 5

1C Opening Stock Values 5

2.2A Buttons Available 8

1. **CLIENT PROFILE**

Weir & Sons, official retailers of some of the world’s most exquisite and renowned watch brands since 1869 has an incredible selection of designer and fashion watches from various brands. Our client Weir & Sons is proud to open a new branch in Galway with the view of selling its beautiful collections in Galway. Weir & Sons are best known for selling products with peculiarity and perfection. Currently it sells 13 different brands of watches with 5 different categories. The brands and their prices are listed below:



***Table1A: Watch Brands and their Price***

The category of the watch and their prices are listed below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Digital | Analog | Sporty | Leather | Hybrid |
| € 15 | € 8 | € 10 | € 13 | € 15 |

***Table1B: Category and their Cost***

Each of the brands are sold in different categories. Each brand has their own price and the categories too. So if a Cartier of category Sporty is sold, the price would be €100 + €10 = €110. In this way, all the watches sold are based on the category and hence there are 65 different watches in total.

The number of units available for the brands of the watches based on its categories are populated in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Digital | Analog | Sporty | Leather | Hybrid |
| Cartier | 30 | 20 | 64 | 34 | 89 |
| Benson | 40 | 34 | 55 | 39 | 45 |
| Tudor | 34 | 32 | 14 | 20 | 14 |
| Omega | 67 | 14 | 89 | 45 | 45 |
| Rolex | 44 | 22 | 43 | 67 | 12 |
| Titan | 56 | 23 | 45 | 23 | 34 |
| Gusto | 34 | 22 | 23 | 56 | 34 |
| Timex | 45 | 20 | 23 | 45 | 24 |
| Casio | 45 | 33 | 32 | 31 | 30 |
| Fossil | 28 | 23 | 46 | 20 | 29 |
| Sonata | 34 | 27 | 23 | 24 | 16 |
| Tommy | 20 | 30 | 40 | 39 | 23 |
| Diesel | 23 | 23 | 43 | 20 | 23 |

***Table1C: Opening Stock Values***

* 1. **General System Specification:**

The client mentioned above wants to design an Electronic Point of Sale Application for the sales team for their new showroom which is to be opened. The features required by the application are listed below:

* The Watches available along with its category should be displayed and the rate of each of these item should be displayed when selected.
* There should also be an option to alter the number of units for the selected watch in case the customer wants to buy multiple watches of the same category.
* There should be Warranty option for 2 Years and 1 Year for the purchasing items which should be highlighted for the customers for increasing the sale revenue. (Note: Warranty is optional, i.e., may or may not be added as per the customer)
* The developing application should be able to handle sale of the items for their customers and generate unique transaction ID for saving the order details.
* A text file containing the details of the Order(s) should be generated and appended accordingly to the transactions.
* A text file containing the Closing Stock value report should also be generated at EOD. The initial stock values are listed in ***Table 1C*** which would be the opening stock value of the watches.
* A text file containing the details of the sold stock items along with the revenue generated (both Total Revenue generated in the day as well as the revenue generated by each Watch Brands) should also be created at every EOD (during exiting of the Application) and should be saved as “DATE--SaleReport”. This is for the management to decide which stock value to increase/decrease in the near future.
* Display the existing stock available in a new form.
* Displaying details of a transaction ID if available.

All exceptions should be handled accordingly for the client to run the application smoothly.

1. **DEVELOPERS MANUAL**
   1. **Tools & Skills requirement for development of EPOS Application:**

Visual studio was the IDE used for development of our Electronic Point of Sales application. This Integrated development environment offers a platform to build applications and storage of various data types. We have built an application that is suitable for our client consisting of the below characteristics:

* Fetching quantity of stocks available during opening of the application.
* Sale of watches based on category.
* Updating quantity of stocks accordingly for the above after sale.
* Saving details of the orders in text file & searching those details through the Order ID when needed.
* Generating currently available stock report when required.
* Creating sales reports for the management during EOD in text file format which is saved by their date.

Note: The initial stock values are loaded into the application through which the text files are created. The Sales report generated is during the closing of application (Exit) which is considered as EOD.

Visual Studio also offers various platforms for using different programming languages. The EPOS Application created is done by through C#. Visual Studio enables the developer to chose among its vast options to create an application such as form, buttons, textbox, etc. The Skills required for developing the application are stated below:

* Adept level Programming knowledge of C#
* Rationalization
* Understanding the customer requirements
  1. **Basic Design:**

The Tools used for the making of the applications are as follows:

**Forms** – The design consists of three forms: Primary, SearchOrderDetails and StockAvailability forms

**Panels** – for separation and easy understanding of the flow. The major panels in our application are OrderingPanel and BasketPanel of the Primary Form. The other two forms consists of only one functionality and therefore are not divided through Panels.

**Listboxes** – The primary form consists of the BrandsListBox and CategoryListBox. The SearchOrderDetails form consists of the OrderDetailsListBox. The StockAvailability form consists of the StockAvailabilityListBox.

**Labels** – These are used for reference for prompting the user through providing information regarding the other tools and displaying information.

Ex: Price of the order via PriceDisplayLabel and display of unique order ID via OrderIDLabel.

**Radio Buttons** – For the EPOS radiobuttons are present as an optional selection for obtaining warranty for the customer’s items. The radio buttons available are TwoYearsWarrantyRadioButton, OneYearWarrantyRadioButton and NoThanksRadioButton.

**NumericUpDown** – This tool is placed in the application for the user to select the number of units of a particular item which the customer wants to buy and is given UnitsNumericUpDown as the name for convention.

**Grid View** – The OrderDataGridView is used for displaying the selected items by the user for the customer which will be the final billing items.

**Buttons** – The buttons play the key role for various actions which can be performed by the user. The details of each of these buttons is explained in depth in section 2.2.6. The Buttons in the application are shown below for each form:

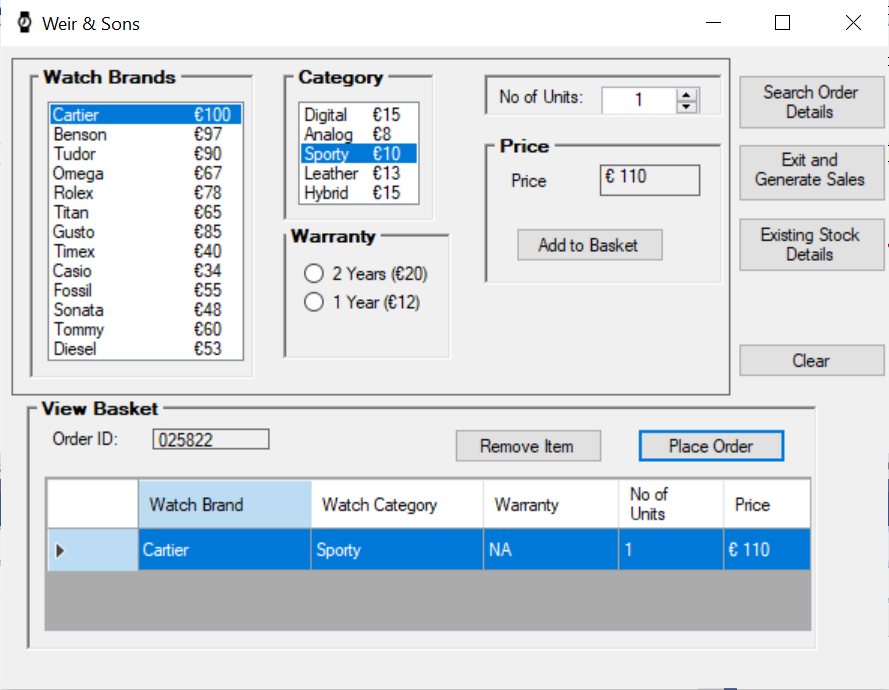
|  |  |  |
| --- | --- | --- |
| **Primary Form** | **SearchOrderDetails Form** | **StockAvailability Form** |
| Add to Basket | Search Order Details | Back |
| Place Order | Back |  |
| Remove Item |  |  |
| Clear |  |  |
| Exit and Generate Sales Report |  |  |
| Search Order Details |  |  |

***Table2.2A: Buttons Available***

**Tooltips –** Appropriate tooltips have been added for the buttons.

**2.2.1 Forms:**

**Primary Form:** The Primary form consists of the two main panels and six buttons.



***Figure 2.2.1A: Primary Form***

In this form the user can select the brand and category of watch which the customer wants and how much units of that watch he/she needs. The user also has an additional option to select Warranty for the customer, if needed. Once the details are selected the price of the selected item will be displayed in the PriceLabelBox and the user can add the item to the cart by selecting the Add to Basket button after which the details of the selected item would be displayed in the Grid View (OrderDataGridView) of the next panel. The OrderDataGridView displays the details of the selected item.

During the form load of the primary form the existing stock values would be fetched from the ClosingStockReport text file (For the first time alone the values are preloaded in the application and the report would be generated and updated accordingly). The values are loaded into a single dimension array **ExistingStockData[]**. The values of this array are then inputted into an two dimension array **LiveStockData[]**. A copy of this array is created as well **CopyofExistingStockData[].**

On clicking of the **Add to Basket** button the stock values in the **CopyofExistingStockData[]** array are manipulated according to the index value selected in the **BrandsListbox** and **CategoryListBox**. The code snippet is given below:

***CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] = CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] – units***

Where **units** are the number selected through **UnitsNumericUpDown** box indicating the number of watches the customer wants to buy of a particular category and brand. This is to make sure the user does not select units higher than the existing stock. If the user selects units higher than the existing stock then a messagebox would be displayed indicating the current available stocks

During this time the **TotalBill** and **Totalitems** are also accumulated through the following code:

totalbill += decimal.Parse(PriceDisplayLabel.Text.Substring(1));

totalitems += UnitsNumericUpDown.Value;

where PriceDisplayLabel is the price indicated for the selected item and UnitsNumericUpDown is the number of units selected.

These are global variables and values are initially set to zero. On the event of adding items to basket the price and units are added to the **TotalBill** and **Totalitems** global variables respectively.

After an item is added to basket the Basket Panel will become visible. In the Basket panel the **OrderDataGridView** displays the details of the item selected and unique six digit order ID would be generated and displayed in the OrderIDLabel. The gridview consists of five columns (explained in section 2.2.5). Now any number of items of the different brands and category can be added to the basket (OrderDataGridView) which the customer wants to purchase. In case the user has added a wrong item to the OrderDataGridView or if the customer wants to deselect an item, the user simply must select whichever item he/she wants to remove from the basket and click on the Remove Item Button which will show an message regarding which item the user has removed.This will also reset the value for totalbill and totalitems by subtracting the price and units value of the selected item using the following :

totalbill += (decimal.Parse(selectedRowValues[4].Substring(1)));

totalitems += decimal.Parse(selectedRowValues[3]);

Where selectedRowValues[4] is the price and selectedRowValues[3] is the number of units of the selected row in the OrderDataGridView respectively.

The stock values in the CopyExistingStockData array would also be updated as follows:

***CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] = CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] + units***

The units are the units of the item selected in OrderDataGridView Column 4.

Once the OrderDataGridView is satisfied for the customer the user would select the Place Order button after which a message box containing the details of the items in the OrderDataGridView would be displayed along with the total bill value and total units (which is the result generated by TotalBill and Totalitems global variables).

If the user selects OK button in the messagebox then the stock values would be updated and the order details would also be saved in orderdetails text file. The sold stock values are also updated by subtracting the values of LiveStockData array with the CopyofExistingStockData array. The code used is as follows:

WatchesSoldData[i, j] += (LiveStockData[i, j] - CopyofExistingStockData[i, j]);

dummyWatchesSoldData[i, j] = (LiveStockData[i, j] - CopyofExistingStockData[i, j]);

Where WatchesSoldData array is the overall stock sold and dummyWatchesSoldData array is the sold stock in that particular Order. This is achieved by using For Loop statements.

The arrays containing the stock values are also updated by,

LiveStockData[i, j] = CopyofExistingStockData[i, j];

ExistingStockData[singlearrayloop] = LiveStockData[i, j];

Where i and j represents the index of the array and singlearrayloop represents the index of the one dimensional array.

These stock values are then used to overwrite the existing stock values in the **ClosingStockReport** text file. This is for updating the stock values.

Finally the clear Button click method is called (explained more in section 2.2.6) to reset the form.

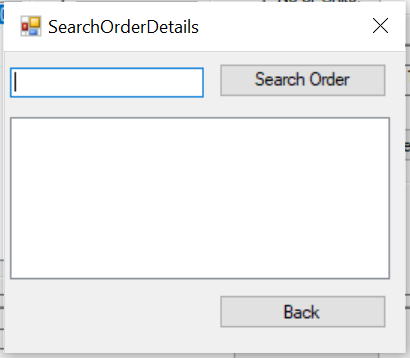
The Primary form also has other buttons which would be explained in the forthcoming sections.

**SearchOrderDetails Form:** This form is displayed when the user clicks on the Search Order Details button through the code:

SearchOrderDetails SearchFORM= new SearchOrderDetails();

SearchFORM.ShowDialog();

The ShowDialog() would display the form and disable the Primary Form which can only be accessed if the user closes the current form.

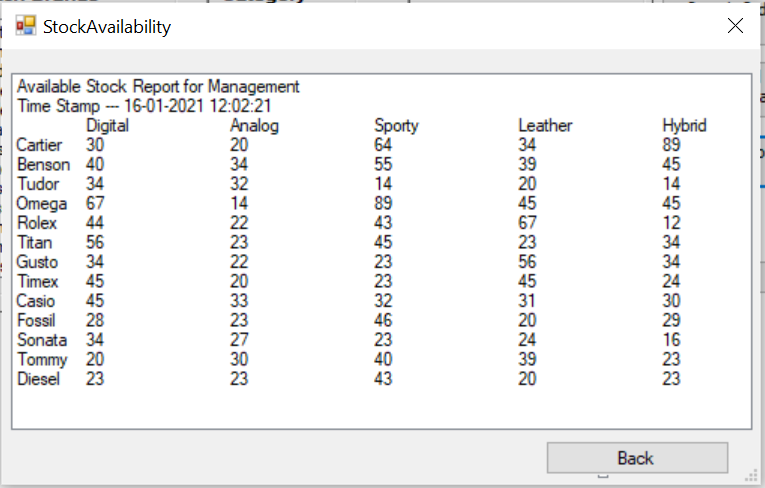


***Figure2.2.1B: Search Order Details***

This form is used to search a particular order ID details by the user. The order ID is entered in the text box after which Search Order Details button is clicked. A function from the mainform (declared as public) is called during this button click to fetch all the lines in the orderdetails text file to the **OrderDetails** list of this form (also declared in public). Then using looping statement the list is searched for the entered Order ID (if present) and the details of the order ID would be displayed in the below **OrderDetailsListBox**.

The Back button in the form would redirect the user to move back to the Primary form and thereby exiting the current form.

**StockAvailibility Form:** This form is loaded when the user selects on the Existing Stock Details button. The details of the current stock values/existing stocks are displayed in this form through the StockAvailabilityListBox.



***Figure2.2.1C: Stock Availability***

Like the SearchOrderDetails form, the Back button in this form would also redirect the user to move back to the Primary form and thereby exiting the current form.

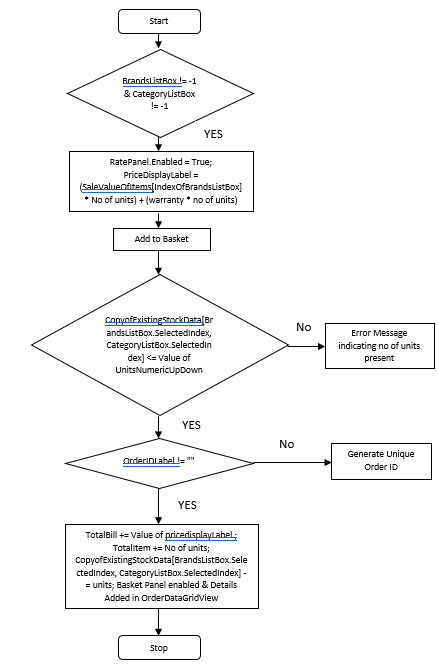
**2.2.2. Panels:**

**OrderingPanel:** The ordering panel comes under the Primary form (Forms Section 2.2.1).

The OrderingPanel consists of the below elements:

* BrandsListBox and CategoryListBox
* Warranty Radio Buttons (2 Years, One Year and No Thanks)
* UnitsNumericUpDown
* RatePanel
* Add to Basket

The basic flow occurring in the OrderingPanel are shown via flowchart:



***Figure2.2.2A: Flowchart – Ordering Panel***

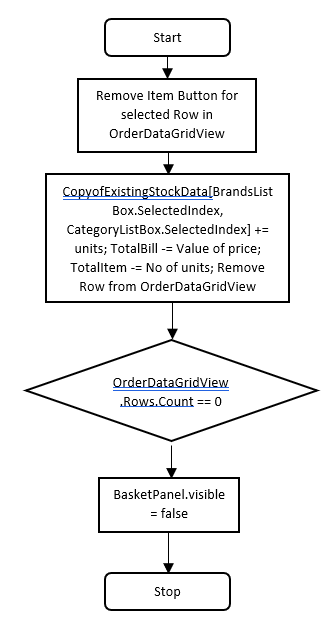
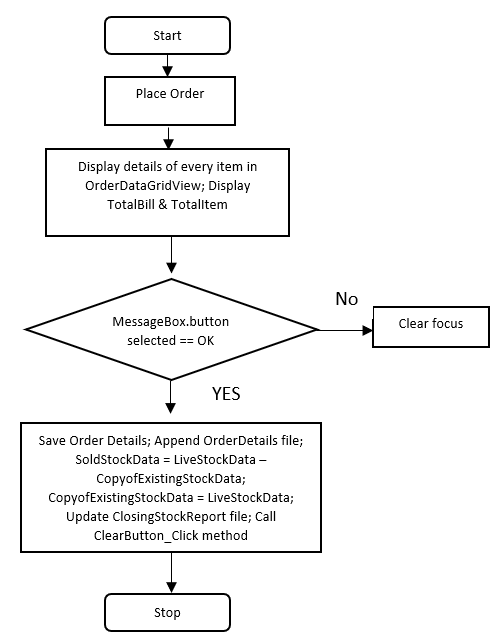
The user selects a watch and category based on the customer after which the RatePanel is enabled. Now using the PriceDisplayLabel the price of the item selected is shown. The UnitsNumericUpDown box enables the user to select the number of units of the selected item which the customer wants to buy. Also the user has the option to select Warranty for the selected item(s) if wanted by the customer. Note that the warranty is optional and may or may not be added. The price is dynamically changed during the selection of the list values in the BrandsListBox and CategoryListBox, during changing of the warranty, altering the number of units to be purchased and displayed in the PriceDisplayLabel.

If the user selects the Add to Basket button the items will be added to the OrderDataGridView in the BasketPanel as well as displaying the BasketPanel and its indicating BasketLabel. The method is explained more in depth in Primary form in Forms section 2.2.1.

**BasketPanel:** As mentioned above the panel is enabled once an item has been added to basket (OrderDataGridView) using the Add to Basket button. The panel consists of the following elements:

* OrderIDLabel
* Remove Item and Place Order Buttons
* OrderDataGridView

The basic flow diagram in the panel is shown:

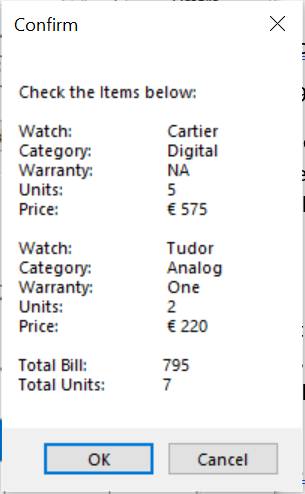
***Figure2.2.2B: Flowchart – Basket Panel***

hilst clicking on the Add to Basket button an Unique Order ID is generated and displayed in the OrderIDLabel. This Order ID acts as a reference for the particular transaction done. The order ID would be generated only once for the entire transaction until Clear button is used (explained in section 2.2.6). The Order ID is created using the below method:

Random UniqueNumber = new Random();

int receiptNO = UniqueNumber.Next(0, 999999);

This Order ID is used as reference for the Order in progress. The user also has the option to remove any of the items in the Cart (OrderDataGridView) by selecting the remove button. If the user removes the final item from the OrderDataGridView then the BasketPanel and its subsequent label will be hidden.



***Figure2.2.2C: Confirmation on the Order***

The prompt message displayed in the message box after selecting the Place Order button is generated using the **StringBuilder** method and appending lines to the object created through the StringBuilder. The lines appended in the string builder are the cells of the rows present in the OrderDataGridView using For Loop statement.

StringBuilder OrderDetails = new StringBuilder();

OrderDetails.AppendLine(string.Format(“Text”));

If the user selects the Ok button in the message box then the order is completed and the details of the order are saved in a text file(explained in Section 2.2.1). The details of the order are saved in the below format:

OrderID

Date Of Order

Brand|Category|UnitsSold\*

Sale Value

The line indicated by \* represents that the value can be multiple lines (for different types of Brand and category) for a single Order.

**2.2.3 Listboxes:**

The listboxes present in the various forms are listed below:

Primary Form:

BrandsListBox – Consists of the different Brands of Watches available along with its price rate.

CategoryListBox – Consists of the different Categories of Watches available along with its price rate.

Note : The Price is calculated by adding both the value of the selected item in the BrandsListBox and CategoryListBox (Explained more in detail in Section 2.2.1)

SearchOrderDetails form:

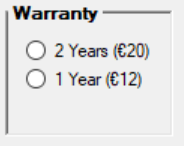
OrderDetailsListBox – Displays the fetched Particular Order details from the OrderDetails List.

StockAvailability form:

OrderDetailsListBox – Displays the available stock data for the items.

**2.2.4 Radio Buttons:**

In the Primary form the radio buttons display the warranty options available for the items selected by the customer.



***Figure2.2.4A: Warranty Options***

Initially there are only two Radio buttons displayed: 2 Years (€20) and 1 Year (€12) prompting the customer to select any of the warranty thereby increasing the sale value. However after selecting the radio button (If wanted) the No Thanks Radio button would be visible in the panel indicating that the warranty can also be cancelled.   
Note: Even if no warranty is selected it is assumed that the customer has not selected warranty for the item and will be neglected (No Thanks).

**2.2.5** **Grid View:**

The OrderDataGridView contains the following columns: **Watch Brand, Watch Category, Warranty, No of Units** and **Price**. Whilst an item is added through the Add to Cart button the details are added to the columns accordingly.

**2.2.6** **Buttons:**

**Add to Basket** – Used to Add the selected item in the panel along with its price and no of units (Explained in section2.2.1)

**Place Order** – Displays a message box indicating the Full Order details and upon confirmation (Pressing OK) the stock values are updated, and Order details are saved (Explained in section 2.2.1).

**Remove Item** – Removes an item from the OrderDataGridView as well as updates the stock values accordingly.

**Clear** – Clears the screen and displays the initial load screen for the user. Also resets any pending order by removing rows from the grid and thereby its stock values (Based on the last count of the stock values) and clears the order ID generated. This is achieved by using:

CopyofExistingStockData[x, y] = LiveStockData[x, y];

Where x and y are indexes of the two-dimensional array. This statement is executed if there are any items in the OrderDataGridView using Decision statement and looping to reset the value based on the last available stocks.

**Exit and Generate Sales Report** – Generates and displays a text file containing the Number of items sold along with its sale value and the total sale value generated. The generated text file will be saved based on the date and can be used by the Management for analysis purposes. On closing of this text file a message box for confirmation is displayed whether the user wants to close the application or not.

**Search Order Details** (Primary Form)– Opens the SearchOrderDetails Form and disabling the Primary form (explained in section 2.2.2).

**Search Order Details** (SearchOrderDetails Form) – This Button is used for finding the details of the order ID entered in the InputTextBox and displays the details in the OrderDetailsListBox. Initially the values in the OrderDetails List and OrderDetailsListBox are cleared. Also Exceptions are handled in the button in case the user has entered non-numeric values and if the length of the entered ID is not equal to 6. Also if the entered 6 digit ID is not present in the OrderDetails text file (where the details of the Order ID are saved – Explained in section 2.2.1).

**Back** (SearchOrderDetails Form and StockAvailability Form) – Brings the user back to the Primary form by closing the existing form.

**Note:** *It is important to note that the stock value should be updated via the ClosingStockReport text file (by adding the stock values one below the other in sequential format) if new stock items are added since there was no indication of updating the stock value through the application in the system requirement. It is easy to update the value by simply entering the stock of each item via brand first followed by its category. For example the Cartier Watch stocks are added one below the other sequentially for all the five categories followed by the next Benson Watch and so on. Initially the number of available stocks are pre-loaded into the application.*

1. **APPENDIX**
   1. **Primary Form:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Srinivas\_Akhil\_Assignment4\_MS806

{

public partial class Primaryform : Form

{

public Primaryform()

{

InitializeComponent();

}

//Global Variable Declaration

static List<string> Watches = new List<string>()

{"Cartier","Benson","Tudor","Omega","Rolex","Titan","Gusto","Timex","Casio","Fossil","Sonata","Tommy","Diesel"};

static List<string> CategoryofWatch = new List<string>()

{"Digital","Analog","Sporty","Leather","Hybrid"};

static decimal[,] SaleValueofItems = { { 115, 108, 110, 113, 115 },

{ 112, 105, 107, 110, 112 },

{ 105, 98, 100, 103, 105 },

{ 82, 75, 77, 80, 82 },

{ 93, 86, 88, 91, 93 },

{ 80, 73, 75, 78, 80 },

{ 100, 93, 95, 98, 100 },

{ 55, 48, 50, 53, 55 },

{ 49, 42, 44, 47, 49 },

{ 70, 63, 65, 68, 70 },

{ 63, 56, 58, 61, 63 },

{ 75, 68, 70, 73, 75 },

{ 68, 61, 63, 66, 68 } };

int[,] InitialStockData = { {30, 20, 64, 34, 89 },

{ 40, 34, 55, 39, 45 },

{ 34, 32, 14, 20, 14 },

{ 67, 14, 89, 45, 45 },

{ 44, 22, 43, 67, 12 },

{ 56, 23, 45, 23, 34 },

{ 34, 22, 23, 56, 34 },

{ 45, 20, 23, 45, 24 },

{ 45, 33, 32, 31, 30 },

{ 28, 23, 46, 20, 29 },

{ 34, 27, 23, 24, 16 },

{ 20, 30, 40, 39, 23 },

{ 23, 23, 43, 20, 23 } };

int[,] WatchesSoldData = new int[13, 5]; int[] ExistingStockData = new int[65];

int[,] CopyofExistingStockData = new int[12, 4];

int[,] LiveStockData = new int[13, 5];

private void Form1\_Load(object sender, EventArgs e)

{

toolTip1.SetToolTip(AddToBasketButton, "Press to add the item to basket");

toolTip1.SetToolTip(RemoveButton, "Press to remove the item from basket");

toolTip1.SetToolTip(PlaceOrderButton, "Press to place the order");

TwoYearsWarrantyRadioButton.Checked = false;

OneYearWarrantyRadioButton.Checked = false;

NoThanksRadioButton.Visible = false;

if (File.Exists("ClosingStockReport.txt"))

{

try

{

StreamReader Outputfile = File.OpenText("ClosingStockReport.txt");

for (int k = 0; k <= 64; k++)

{ ExistingStockData[k] = int.Parse(Outputfile.ReadLine()); Console.WriteLine(ExistingStockData[k]); }

int singlearrayvalue = 0;

for (int i = 0; i <= 12; i++)

{

for (int j = 0; j <= 4; j++)

{

LiveStockData[i, j] = ExistingStockData[singlearrayvalue]; singlearrayvalue++;

CopyofExistingStockData[i, j] = LiveStockData[i, j];

WatchesSoldData[i, j] = 0;

Console.WriteLine(WatchesSoldData[i, j] = 0);

}

}

Outputfile.Close();

}

catch (Exception ex) { Console.WriteLine("Closing Stock Report does not exist\n" + ex.Message); }

}

else

{

LiveStockData = (int[,])this.InitialStockData.Clone();

CopyofExistingStockData = (int[,])this.LiveStockData.Clone();

}

}

//Global variable Declaration

decimal unitsofwatches;

//Enabling PriceBox only when values inthe list boxes are selected

private void EnablingPriceBox()

{

decimal warrantyrate;

if (BrandsListBox.SelectedIndex != -1)

{

if (CategoryListBox.SelectedIndex != -1)

{

warrantyrate = warrantycalculation();

RatePanel.Enabled = true;

unitsofwatches = UnitsNumericUpDown.Value;

PriceDisplayLabel.Text = "€ " + (((unitsofwatches \* SaleValueofItems[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex])) + (unitsofwatches \* warrantyrate)).ToString();

AddToBasketButton.Focus();

}

}

}

//Calculating price for optional warranty

decimal[] warrantyrates = new decimal[] { 20, 12, 0 }; string warranty;

private decimal warrantycalculation()

{

decimal warrantyvalue;

if (TwoYearsWarrantyRadioButton.Checked) { warrantyvalue = warrantyrates[0]; NoThanksRadioButton.Visible = true; warranty = "Two"; }

else if (OneYearWarrantyRadioButton.Checked) { warrantyvalue = warrantyrates[1]; NoThanksRadioButton.Visible = true; warranty = "One"; }

else { warrantyvalue = warrantyrates[2]; warranty = "NA"; }

return warrantyvalue;

}

private void TwoYearsWarrantyRadioButton\_CheckedChanged(object sender, EventArgs e)

{

EnablingPriceBox();

NoThanksRadioButton.Visible = true;

}

private void OneYearWarrantyRadioButton\_CheckedChanged(object sender, EventArgs e)

{

EnablingPriceBox();

NoThanksRadioButton.Visible = true;

}

private void BrandsListBox\_SelectedIndexChanged(object sender, EventArgs e)

{

EnablingPriceBox();

}

private void CategoryListBox\_SelectedIndexChanged(object sender, EventArgs e)

{

EnablingPriceBox();

}

private void QuantityNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

EnablingPriceBox();

}

//Global Variable Declaration

decimal totalbill = 0, totalitems = 0;

//Action performed when add to basket button is clicked

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

{

int units = int.Parse(UnitsNumericUpDown.Value.ToString());

if (CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] >= units)

{

BasketPanel.Visible = true; ViewBasketLabel.Visible = true;

int uniqueReceiptNO = generateUniquereceiptnumber();

decimal warrantyvalue = warrantycalculation();

totalbill += decimal.Parse(PriceDisplayLabel.Text.Substring(1));

totalitems += UnitsNumericUpDown.Value;

if (OrderIDLabel.Text == "") { OrderIDLabel.Text = uniqueReceiptNO.ToString("D6"); }

else { Console.WriteLine("Unique ReceiptNO already created"); }

CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] = CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex] - units;

OrderDataGridView.Rows.Add(Watches[BrandsListBox.SelectedIndex], CategoryofWatch[CategoryListBox.SelectedIndex], warranty, UnitsNumericUpDown.Value.ToString(), PriceDisplayLabel.Text);

PlaceOrderButton.Focus();

}

else {

string unitserror = "Only " + CopyofExistingStockData[BrandsListBox.SelectedIndex, CategoryListBox.SelectedIndex].ToString() + " are available right now for " + Watches[BrandsListBox.SelectedIndex] + " of category " + CategoryofWatch[CategoryListBox.SelectedIndex];

MessageBox.Show(unitserror, "OOPS!!",MessageBoxButtons.OK,MessageBoxIcon.Exclamation);

UnitsNumericUpDown.Focus();

}

}

//Generating Unique Receipt Number

private int generateUniquereceiptnumber()

{

Random UniqueNumber = new Random();

int receiptNO = UniqueNumber.Next(0, 999999);

string UniqueReceiptValue = "Receipt Number:" + receiptNO.ToString("D6");

try

{

StreamReader Outputfile = new StreamReader("orderdetails.txt");

while (!Outputfile.EndOfStream)

{

string temp = Outputfile.ReadLine();

temp = "Order details: " + temp;

if (temp != UniqueReceiptValue) { Console.WriteLine("Searching for repeated TranscationID"); }

else { generateUniquereceiptnumber(); };

}

Outputfile.Close();

}

catch { Console.WriteLine("File does not exist"); }

return receiptNO;

}

//Action Performed when Remove Button is clicked

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

{

List<string> selectedRowValues = new List<string>();

selectedRowValues.Clear();

int rowNumber = OrderDataGridView.CurrentRow.Index;

DataGridViewRow selectedR = OrderDataGridView.Rows[rowNumber];

for (int index = 0; index <= 4; index++)

{

selectedRowValues.Add(selectedR.Cells[index].Value.ToString());

}

int positionoftemptoy = Watches.IndexOf(selectedRowValues[0]);

int positionoftempmake = CategoryofWatch.IndexOf(selectedRowValues[1]);

int quantity = int.Parse(selectedRowValues[3]);

CopyofExistingStockData[positionoftemptoy, positionoftempmake] = CopyofExistingStockData[positionoftemptoy, positionoftempmake] + quantity;

RemoveButton.Focus();

totalbill -= (decimal.Parse(selectedRowValues[4].Substring(1)));

totalitems -= decimal.Parse(selectedRowValues[3]);

OrderDataGridView.Rows.Remove(OrderDataGridView.Rows[rowNumber]);

MessageBox.Show(selectedRowValues[3] + " Units of watch " + selectedRowValues[0] + " of Category " + selectedRowValues[1] + " was removed");

if (OrderDataGridView.Rows.Count == 0) { ViewBasketLabel.Visible = false; MessageBox.Show("No more items on the cart"); BasketPanel.Visible = false; }

else { Console.WriteLine("Items Still Available"); }

}

//Action Performed when Place Order Button is clicked

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

{

int[,] dummyWatchesSoldData = new int[13, 5];

StringBuilder OrderDetails = new StringBuilder();

OrderDetails.AppendLine(string.Format("Check the Items below:"));

for (int i = 0; i < OrderDataGridView.Rows.Count; i++)

{

OrderDetails.AppendLine(string.Format("\nWatch:\t\t {0}", OrderDataGridView.Rows[i].Cells[0].Value.ToString()));

OrderDetails.AppendLine(string.Format("Category:\t {0}", OrderDataGridView.Rows[i].Cells[1].Value.ToString()));

OrderDetails.AppendLine(string.Format("Warranty:\t {0}", OrderDataGridView.Rows[i].Cells[2].Value.ToString()));

OrderDetails.AppendLine(string.Format("Units:\t\t {0}", OrderDataGridView.Rows[i].Cells[3].Value.ToString()));

OrderDetails.AppendLine(string.Format("Price:\t\t {0}", OrderDataGridView.Rows[i].Cells[4].Value.ToString()));

}

OrderDetails.AppendLine(string.Format("\nTotal Bill:\t\t" + totalbill.ToString() + "\nTotal Units:\t" + totalitems.ToString()));

if (MessageBox.Show(OrderDetails.ToString(), "Confirm", MessageBoxButtons.OKCancel) == DialogResult.OK)

{

try

{

DateTime DateofOrder = DateTime.Now;

StreamWriter OutputFile = File.AppendText("orderdetails.txt");

OutputFile.WriteLine("Order details: " + OrderIDLabel.Text);

OutputFile.WriteLine("Date of Order: " + DateofOrder.ToString("dd-MM-yy"));

for (int i = 0; i <= 12; i++)

{

for (int j = 0; j <= 4; j++)

{

WatchesSoldData[i, j] += (LiveStockData[i, j] - CopyofExistingStockData[i, j]);

dummyWatchesSoldData[i, j] = (LiveStockData[i, j] - CopyofExistingStockData[i, j]);

if (dummyWatchesSoldData[i, j] != 0) { OutputFile.WriteLine(Watches[i] + "|" + CategoryofWatch[j] + "|" + dummyWatchesSoldData[i, j]); }

else { Console.WriteLine("Sale has not occured for this instance"); }

}

}

OutputFile.WriteLine("Sale Value: " + totalbill); OutputFile.WriteLine();

OutputFile.Close(); }

catch (Exception ex) { MessageBox.Show("Error unable to save orderID file contact manager\n" + ex); }

int singlearrayloop = 0;

for (int i = 0; i <= 12; i++)

{

for (int j = 0; j <= 4; j++)

{

LiveStockData[i, j] = CopyofExistingStockData[i, j];

ExistingStockData[singlearrayloop] = LiveStockData[i, j]; singlearrayloop++; } }

try

{

StreamWriter StockFile = File.CreateText("ClosingStockReport.txt"); for (int x = 0; x <= 64; x++)

{ StockFile.WriteLine(ExistingStockData[x]); }

StockFile.Close();

}

catch (Exception ex) { MessageBox.Show("Unable to create closing stock report\n" + ex); }

ClearButton\_Click(sender, e);

MessageBox.Show("Thank you for Shopping!!", "Successful");

}

else

{

ClearButton.Focus();

}

}

//Action Performed when Exit Button is clicked

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

{

try

{

decimal TotalProfitGenerated = 0;

DateTime getdate = DateTime.Now;

string today = getdate.ToString("dd-MM-yy");

StreamWriter OutputFile = File.CreateText(today + "--SaleReport.txt");

OutputFile.WriteLine("Management Report|Details of Sold Items\n" + getdate);

for (int k = 0; k <= 4; k++) { OutputFile.Write("\t" + CategoryofWatch[k]); }

OutputFile.Write("\tProfit");

for (int i = 0; i <= 12; i++)

{

decimal profitperwatch = 0;

OutputFile.WriteLine();

OutputFile.Write(Watches[i]);

for (int j = 0; j <= 4; j++)

{

OutputFile.Write("\t" + WatchesSoldData[i, j].ToString());

if (WatchesSoldData[i, j] != 0) { TotalProfitGenerated += WatchesSoldData[i, j] \* SaleValueofItems[i, j]; }

else { TotalProfitGenerated += 0; }

profitperwatch += WatchesSoldData[i, j] \* SaleValueofItems[i, j];

}

OutputFile.Write("\t" + profitperwatch);

}

OutputFile.WriteLine("\n Total Profit Generated Till Now :" + TotalProfitGenerated);

OutputFile.Close();

System.Diagnostics.Process.Start("notepad.exe", today + "--SaleReport.txt").WaitForExit();

}

catch (Exception ex) { MessageBox.Show("Error file cannot be generated\nContact your manager\n" + ex); }

if (MessageBox.Show("Are you sure you want to close the application?\nNote: Management Report has been generated and can be viewed at any time", "Confirmation", MessageBoxButtons.OKCancel) == DialogResult.OK) { this.Close(); }

}

//Action Performed when Clear Button is clicked

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

{

BasketPanel.Visible = false;

ViewBasketLabel.Visible = false;

CategoryListBox.SelectedIndex = -1;

BrandsListBox.SelectedIndex = -1;

TwoYearsWarrantyRadioButton.Checked = false;

OneYearWarrantyRadioButton.Checked = false;

NoThanksRadioButton.Visible = false;

UnitsNumericUpDown.Value = 1;

PriceDisplayLabel.Text = "";

OrderIDLabel.Text = "";

totalbill = 0; totalitems = 0;

RatePanel.Enabled = false;

if (OrderDataGridView.Rows.Count != 0)

{

for (int x = 0; x <= 12; x++)

{

for (int y = 0; y <= 4; y++)

{

CopyofExistingStockData[x, y] = LiveStockData[x, y];

}

}

OrderDataGridView.Rows.Clear();

}

else { Console.WriteLine("No watches in the grid"); }

}

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

{

SearchOrderDetails SearchFORM= new SearchOrderDetails();

SearchFORM.ShowDialog();

}

//Displaying the available stocks

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

{

try

{

StreamWriter OutputFile = File.CreateText("AvailableStockData.txt");

DateTime getdate = DateTime.Now;

OutputFile.WriteLine("Available Stock Report for Management\nTime Stamp --- " + getdate);

for (int z = 0; z <= 4; z++) { OutputFile.Write("\t" + CategoryofWatch[z]+ "\t"); }

for (int x = 0; x <= 12; x++)

{

OutputFile.WriteLine();

OutputFile.Write(Watches[x]);

for (int y = 0; y <= 4; y++)

{

OutputFile.Write("\t" + LiveStockData[x, y].ToString() + "\t");

}

}

OutputFile.Close();

StockAvailability StockDetails = new StockAvailability();

StockDetails.ShowDialog();

}

catch (Exception ex) { MessageBox.Show("Unable to Generate Existing Stock details file\nContact your Manager\n" + ex); }

}

public void InputAllOrderDetailsintoList()

{

try

{

StreamReader InputFile = File.OpenText("orderdetails.txt");

while (!InputFile.EndOfStream)

{

SearchOrderDetails.OrderDetails.Add(InputFile.ReadLine());

}

InputFile.Close();

}

catch (Exception ex) { MessageBox.Show("Order Details file is missing\ncontact manager\n" + ex); }

}}}

* 1. **SearchOrderDetails Form:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Srinivas\_Akhil\_Assignment4\_MS806

{

public partial class SearchOrderDetails : Form

{

public SearchOrderDetails()

{

InitializeComponent();

}

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

{

this.Close();

}

public static List<string> OrderDetails = new List<String>();

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

{

OrderDetails.Clear(); OrderDetailsListBox.Items.Clear();

Primaryform Mainform = new Primaryform(); Mainform.InputAllOrderDetailsintoList();

try

{

string ordernumber = InputTextBox.Text;

string fullsearchordernumber = "Order details: " + ordernumber;

int Number = int.Parse(ordernumber);

if (Number != 0 && ordernumber.Length == 6)

{

for (int index = 0; index < OrderDetails.Count; index++)

{

if (OrderDetails[index] == fullsearchordernumber)

{

for (int transcationvalues = index; transcationvalues <= OrderDetails.Count; transcationvalues++)

{

index++;

if (OrderDetails[transcationvalues] == "") { break; }

else { OrderDetailsListBox.Items.Add(OrderDetails[transcationvalues]); }

}

}

}

if (OrderDetailsListBox.Items.Count == 0) { MessageBox.Show("This ID does not exist\nPlease Verify"); }

}

else { MessageBox.Show("Enter Valid Order ID"); }

}

catch (Exception ex)

{

MessageBox.Show("Enter Valid Numeric TransactionID in the textBox\t");

Console.WriteLine(ex); InputTextBox.Focus();

}

}

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

{

toolTip1.SetToolTip(ExitButton, "Press to go back");

}

}

}

* 1. **Stock Availability Form:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Srinivas\_Akhil\_Assignment4\_MS806

{

public partial class StockAvailability : Form

{

public StockAvailability()

{

InitializeComponent();

}

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

{

try

{

StreamReader InputFile = File.OpenText("AvailableStockData.txt");

while (!InputFile.EndOfStream)

{

StockAvailabilityListBox.Items.Add(InputFile.ReadLine());

}

InputFile.Close();

}

catch (Exception ex) { MessageBox.Show("Unable to fetch the stock details file\nContact Manager\n\n" + ex); }

}

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

{

this.Close();

}

}}

**Disclaimer:**

“*In submitting this work I confirm that it is entirely my own. I acknowledge that I may be invited to undertake an online interview if there is any concern in relation to the integrity of my submission*”