

**MS806 – Business Application Programming**

**Student Name: Caitlin Hogan**

**NUI Galway ID Number: 20231704**

**Assignment:** 4

**Date of Submission:** 25/01/2021

**Assignment 4 : EPOS Application**

**Developer’s Manual**

**Introduction**

**Purpose / Client Profile**

This simple EPOS system, called YogiIreland, was developed for a Yoga Retreat Company (‘YRC’) that offers various yoga workshops and lodging to its customers at different locations throughout Ireland (assume each workshop is held only once annually for the sake of scope of this project). YRC aims to attract people of all ages and demographics who enjoy the practice of yoga to attend their workshops. Customers may also select additional purchase options – meal plans and yoga teaching certificates.

**General System Specification**

YRC will use YogiIreland to process and record the sales data of these workshop/lodging combinations and optional items purchased by their customers, provide inventory management to keep track and ensure the number of spaces available for each workshop/location, and provide real-time, comprehensive reports that allow for more informed decision-making processes by the appropriate business stakeholders. The system should not allow over-selling of any product to occur.

**Reasons**

The software needs to be built so the client can process and record customer bookings, keep track of inventory, and get feedback from accurate reports that allow for better decision-making.

**Parameters**

This software gives details of current bookings, including line-item costs and total costs. It keeps track of past bookings made on different user sessions and monitors inventory levels and item sales data daily. It supports the purchasing of several different items in the same transaction. It allows updates to be made to orders before a booking is confirmed. It allows for search functionality by Transaction ID and/or date.

This software does not allow sales of items that are out of stock to be processed. It does not accept payments. It is not to be confused with e-commerce software. It does not manage other types of transactions such as returns, rescheduling, sales cancellation, or exchanges. It does not come with built-in tools for staffing or appointment-scheduling. It assumes each workshop/location combo is only to be held once a year (i.e., text files must be renewed at the beginning of each new year) and does not show workshop dates.

**References**

*Best Retail POS System* (no date) Available at: <https://www.g2.com/categories/retail-pos> (Accessed 23 Jan 2021).

# Verasseti (2018) *How to Write a System Specification*. Available at: https://www.verasseti.com/insights/how-to-write-a-system-specification (Accessed: 18 Jan 2021).

# *Visual Studio 2017 Product Family System Requirements* (2018) Available at: https://docs.microsoft.com/en-us/visualstudio/productinfo/vs2017-system-requirements-vs (Accessed: 23 Jan 2021).

# *Visual Studio performance tips and tricks* (2020) Available at: <https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-performance-tips-and-tricks?view=vs-2019> (Accessed: 23 Jan 2021).

**Description**

**A General Description** This is new software built for this specific client’s needs of an EPOS system.

**Who will use it?**

The users of this product are any employees of the YRC permitted to enter bookings for their customers. Customers will get in touch with YRC to request their booking info via phone, email, or through a form filled out on the company website.

**Environment**

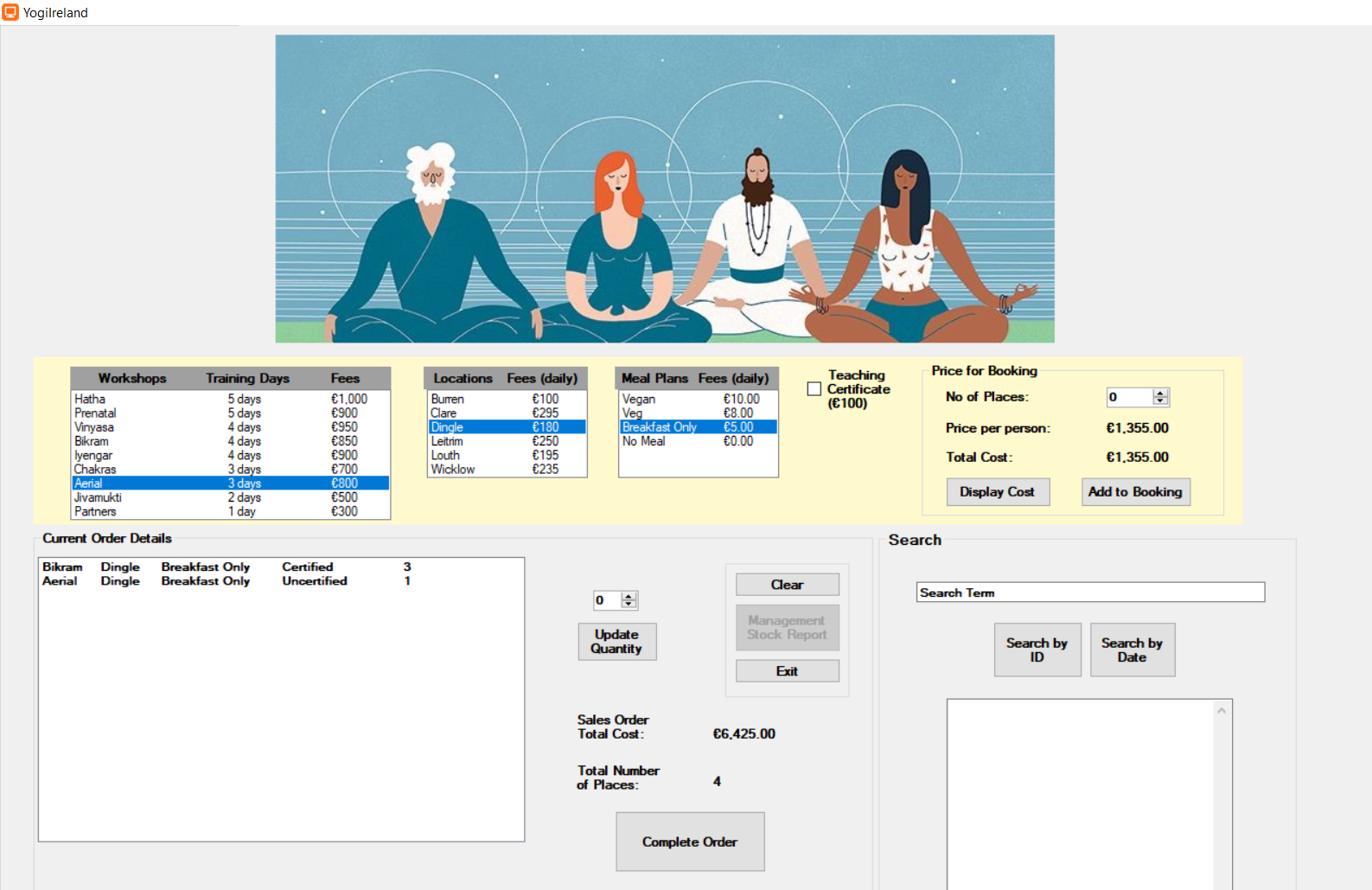
This application was developed in C# using Visual Studio Express 2017 for Windows Desktop (64-bit). System requirements may be found [here](https://docs.microsoft.com/en-us/visualstudio/productinfo/vs2017-system-requirements-vs) (*Visual Studio 2017 Product Family System Requirements*, 2018).

**Constraints**

Virtual memory available to Visual Studio is 4 GB for a 64-bit version of Windows, and 2 GB for a 32-bit version of Windows (*Visual Studio performance tips and tricks*, 2020).

**Interfaces**

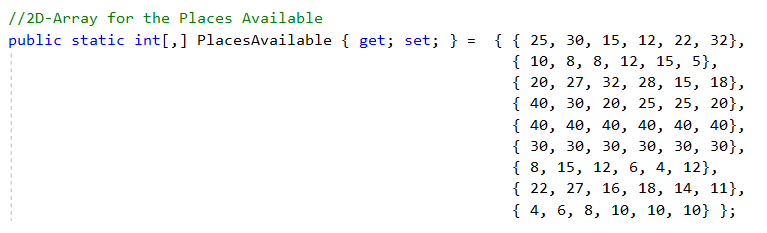
The following will show how the software will present to the end user on the screen.



**Fig. 1(a)**

The client has 1,127 total stock items (i.e., workshop *places)* available to book, where each *place* booked is represented by one and only one person, except in the case of the “Partners” Workshop, in which each *place* represents a booking for two people. In this case, the workshop and lodging location fees remain as those listed in the application graphical user interface (GUI), i.e., they are *not* multiplied by two.

The starting number of places available for each workshop/location combination are hard-coded in Form1.cs as the 2D-array named PlacesAvailable (see ***Fig. 1(b)***). For example, in the first row - which corresponds to the 5-day Hatha Yoga Workshop - there are 25 places available at the Burren, 30 in Clare, 15 in Dingle, 12 in Leitrim, 22 at Louth, and 32 at Wicklow. The workshop and lodging fees are displayed on Form1.cs [Design] in the WorkshopListBox and LocationListBox. The workshop fees are set as what is shown in the WorkshopListBox under column ‘Fees’ (i.e., do *not* multiply by number of days). The length of the workshop is shown under column header ‘Training Days’ in the same list box. LocationListBox shows the lodging locations and their respective *daily* fees. These values *are* multiplied by the number of days associated with the customer’s selected workshop. For instance, if a customer chooses Hatha at the Burren, the base fees would be €1,000 + 5 \*(€100) = €1,500.

 **Fig. 1(b)**

Optional costs include a Meal Plan (user can select Vegan, Vegetarian, Breakfast Only, or No Meal) and/or Teaching Certificate. Meal plan options are shown in the MealListBox. The Vegan meal plan costs €10 daily, Vegetarian is €8 daily, Breakfast only is €5 daily, and the No Meal option costs nothing. Since these are *daily* fees, they are multiplied by the number of Training Days from the line item selected by the user in WorkshopListBox. The teaching certificate option costs an additional €100 flat fee. The user can select or unselect this by ticking the CertificateCheckBox (unchecked means not selected, checked means selected).

The user (the Sales Agent/Employee of YRC) can select how many places will be booked with the ‘No. of Places’ NumericUpDown control. If all spots have been filled for that selection, a message box will display reading “Not enough places available” followed by the number of places currently available. The price per person will also be displayed and the total cost for the selected combination, upon pressing the button “Display Cost”. When the user selects “Add to Booking”, the selection details will be added to the ItemBasket List Box shown in the “Current Order Details” Group Box, and No. of Places will be cleared to 0. See ***Fig.1(a)*** for an example, showing 3 bookings for Bikram with Certifications included and 1 booking for Aerial Yoga with no Certifications, both in Dingle, with Breakfast Only. Sales Order Total Cost shows the total cost of the orders, €6,425.00. Total Number of Places in the order basket shows exactly that, which in this case is 4.

**Functional requirements (Development Application Process)**

This program consists of two Forms - Form1 and Form2. Form1 will be discussed first. Arrays are initialized for Workshops, Days, Fees (workshop fees), Locations, LodgingFees, MealOptions, MealCosts, PlacesAvailable, and OpeningStock. A constant is declared for TEACH\_CERT\_FEE, set to 100. Lists of arrays are also declared – List<int[]> Basket to store basket details, List<int[]> AllTransactions to store items from the ItemBasket list box, and List<string[]> SalesData to store sales / transaction data. A file reader object, InputFile, is created. File names are also stored as strings for the three needed reports - DAILY\_ITEM\_SALES\_FILE\_NAME, CLOSING\_STOCK\_FILE\_NAME, and MGMT\_REPORT\_FILE\_NAME.

Upon Form1 load, the CLOSING\_STOCK\_FILE\_NAME file is checked for existence. If the file does exist, it is opened and saved as InputFile, then read, with each line checking after the “:” character for the available places remaining of each product, parsed as an int. These values update both OpeningStock and PlacesAvailable, effectively setting the closing stock of the day prior to the opening stock of the current day. If there is no ClosingStock.txt found, the values from PlacesAvailable are passed to OpeningStock.

To first obtain a quote for the customer based on their product selection, the user will opt to use the “Display Cost” button, i.e., the DisplayButton\_Click event. The ints WorkshopIndex, LocationIndex, and MealIndex are initialized, getting the SelectedIndex from the respective list boxes (WorkshopListBox, LocationListBox, and MealListBox). Error messages are thrown if the user does not make a complete selection, or if the number of places booked is 0. CertCost is set to 0 if the CertificateCheckBox is unchecked and set to 100 if CertificateCheckBox is checked. TotalCost is calculated by getting the appropriate index of each selection and plugging those into the respective arrays. The formula is:

decimal TotalCost = (Fees[WorkshopIndex]

+ (Days[WorkshopIndex] \* LodgingFees[LocationIndex])

+ (Days[WorkshopIndex] \* MealCosts[MealIndex])

+ CertCost) \* NumPlaces.Value;

The price per person is then calculated as the total cost divided by the number of places booked (variable names are changed in the source code to account for currency formatting and parsing between data types):

CurrentCostValue.Text = TotalCost.ToString("C", Locality);

int NumPlacesValue = Int32.Parse(NumPlaces.Value.ToString());

decimal CurrentCost = Decimal.Parse(CurrentCostValue.Text.Remove(0, 1));

if (NumPlaces.Value > 0)

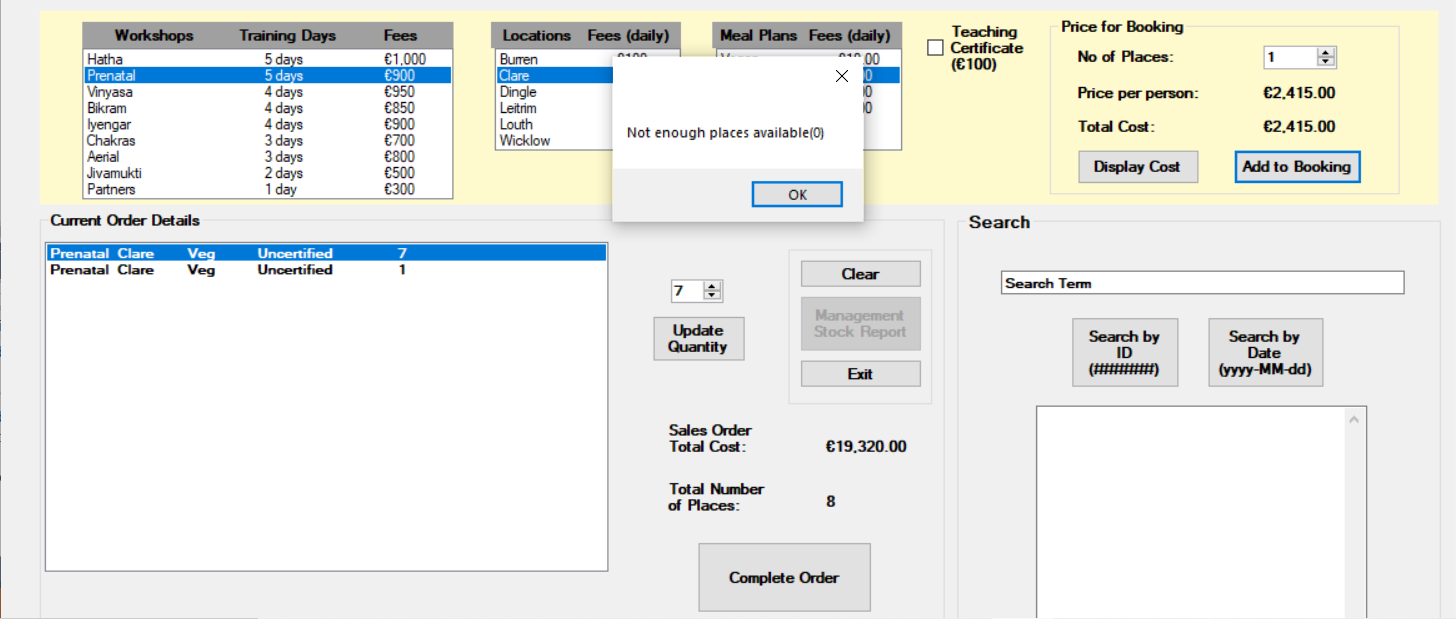
{

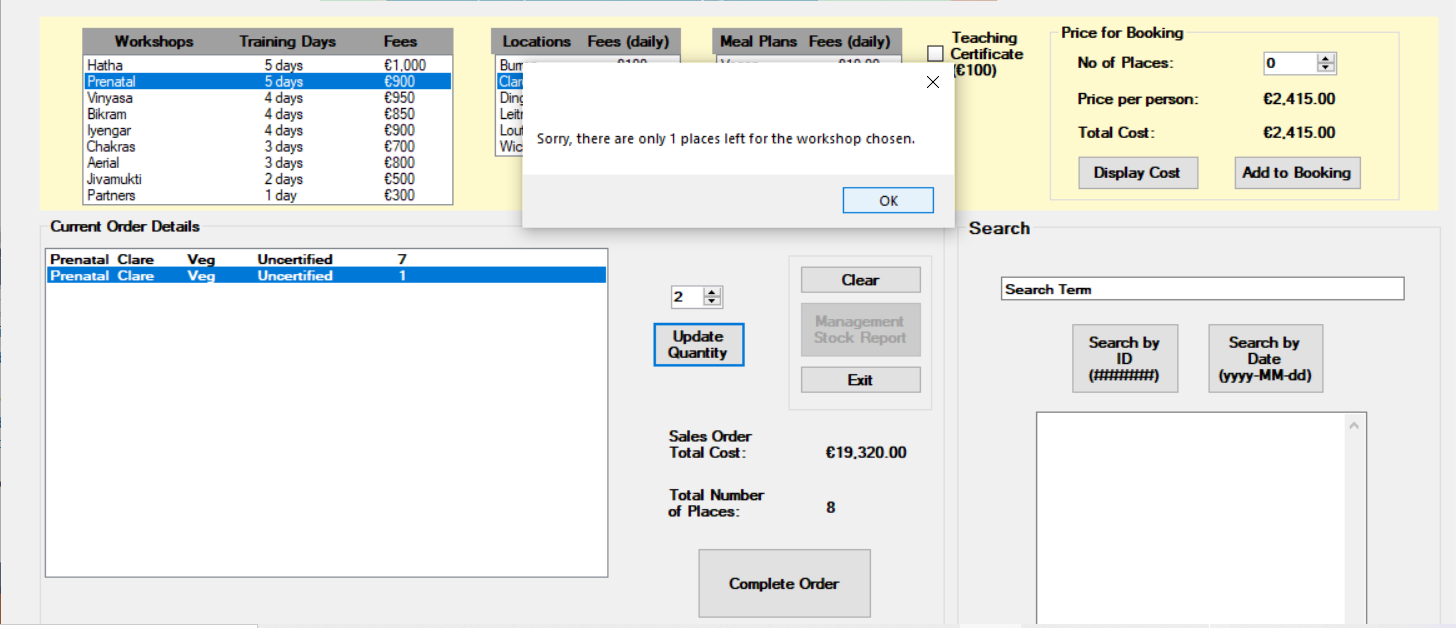
PricePerPersonLabel.Text = (CurrentCost / NumPlacesValue).ToString("C", Locality);

}

For the “Add to Booking” button, the same exception handling / error messaging is performed. If all selections are correct and complete, the next step is to check that enough places are available to book, before adding the most recently selected items to the ItemBasket. The quantity needs to be subtracted from the number of places available if the same item is already in the basket. If the chosen quantity is greater than this updated number of places that are truly available, an error message must be thrown and the quantity reduced. If it is a valid quantity, the item details are added to the basket. These details are also stored as an integer array ItemDetails, which are added to the list of arrays, Basket. This list of arrays contains all details and complete information so that parsing can be done and each line item’s cost or total cost can always be worked out, even if the user clicks another button in the interim. Totals are then updated, which will be discussed soon.

The “Update Quantity” button allows for a user to change the quantity of bookings made after adding the order to the item basket. A basket item must first be selected, else a message box will appear to prompt the user to do so. The quantity must be at least 0 (with 0 effectively removing an item from the basket, using the RemoveAt(ItemBasket.SelectedIndex) method). A check is performed to see if there are enough places available to book with this updated quantity amount. If there are, this NewQuantity is reflected in the list box with the other details. Totals are updated. If the item is found to already be in the basket, the spaces available are reduced. See the example below. A booking was successfully added to the basket for 7 places of Prenatal, Clare, and then another separate booking for 1 place for the same product. When the user tries to make another booking, they are prompted with “Not enough places available(0)”. If the user tries to update the quantity of the second item in the basket from 1 to 2 places, another error is thrown: “Sorry, there is/are only 1 place(s) left for the workshop chosen.” The user must then change the quantity back to at most 1 place if desired and proceed to complete the order. Totals are updated.





UpdateTotals() calculates and displays the total sales order and total number of places in the basket. It uses a similar formula as TotalCost from the “Display Cost” button, but since Basket is a list of arrays, it produces the running total of *all* items in the basket, not just one line item at a time. The formulae are:

TotalCost += (Fees[Basket[i][0]]

+ (Days[Basket[i][0]] \* LodgingFees[Basket[i][1]])

+ (Days[Basket[i][0]] \* MealCosts[Basket[i][2]])

+ (TEACH\_CERT\_FEE \* Basket[i][3])) \* Basket[i][4];

TotalPlaces += Basket[i][4];

For CompleteOrderButton\_Click, TotalCost is calculated in the same manner as in DisplayButton\_Click. Details are then updated and printed for all items in Basket, showing the price of each item as well as the total cost of sales order in a Confirmation (Y/N) Message Box. If the user selects ‘Yes’, a random 6 char int is generated between 100000 and 999999 as the transaction ID, using Random(DateTime.Now.Millisecond). An array T is created to store the TransactionID as a string, OrderDate and Details. T is added to SalesData to save transaction info to memory for search functionality. Item basket places available are adjusted and Basket[i] is added to AllTransactions. TransactionID and OrderDate are displayed and basket is cleared. ManagementStockButton is enabled for a summary report on Form2 since an order has now been completed.

ManagementStockButton\_Click has the main form, Form1, access the second form, Form2, which pops up with date and time, and loops through both Locations and Workshops to display the stock levels for each combination in the format OpeningStock / PlacesAvailable. This is a running summary report, so it holds the total places left based on all orders, even those submitted on previous days. This allows management to know when workshops are almost or already completely booked for the year.

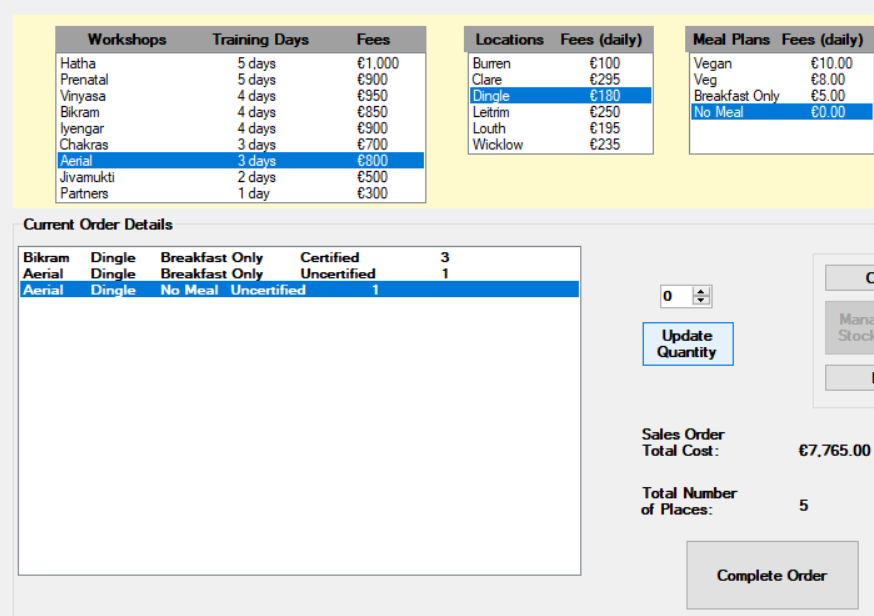
SearchByIDButton\_Click loops through SalesData to check all transaction IDs, and if there is a match, it will print the transaction ID, order date, and details of the transaction in the search results text box.

SearchByDateButton\_Click has the same as SearchByIDButton\_Click but checks SalesData at t[1], order date, instead of t[0], transaction ID. If there is a match, text is appended to the search results text box, displaying transaction ID, order date, and details.

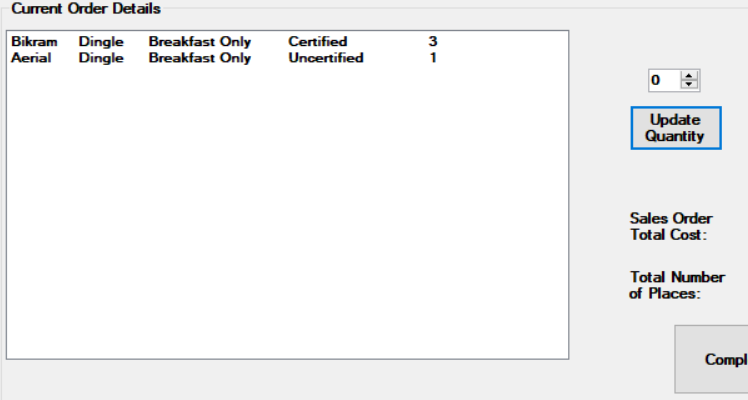
Form1\_FormClosing generates the End-Of-Day reports – for closing stock and daily item sales. Each file is check for existence, and if the files do not exist, they are created and opened as text files, with client details appended to the end of the file. For closing stock these details are the places available for each workshop/location. For the daily sales report there is a header showing date and time to keep track of running sessions of the app (a new report is created daily but each daily report can contain multiple session – opening and closing of the app – data). Here we are only concerned with the primary purchase data (workshop and lodging fees, not meal or certification costs). The fee amount for each item is multiplied by the difference of opening stock and places available, which is the amount of that item sold in one day. Each printed line shows the opening stock, places available, and cost as calculated above. Then, each meal option with its cost, quantity sold, and total cost is printed. The same is done for certificate costs.

### User documentation

The user can highlight a selection in the Item Basket and update the quantity of bookings for that selection. Say the user adds ‘Aerial, Dingle, No Meal, Uncertified, 1’ as shown in ***Fig. 2***, but then changes their mind and wants to remove this. They can click on this row, then select 0 in the drop-down list above the ‘Update Quantity’ button, and then press this button. This will remove the row from the basket (see ***Fig. 3***).

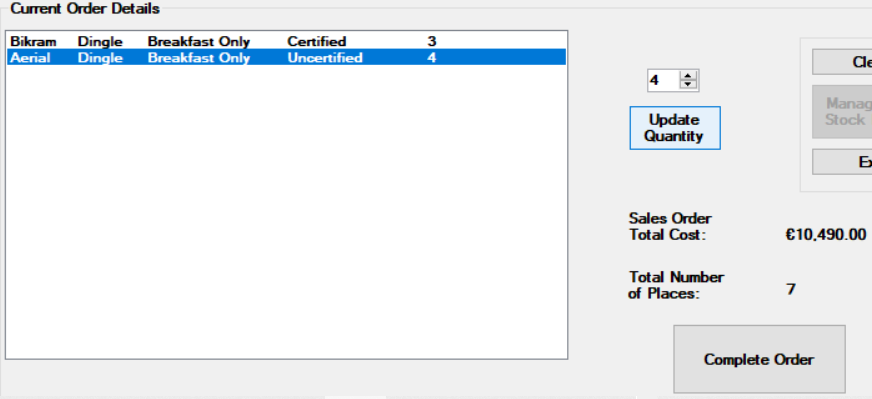


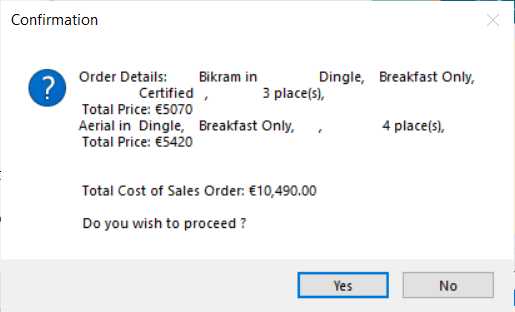
**Fig 2**



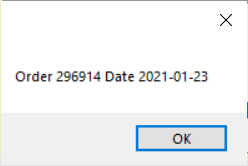
**Fig 3**

Likewise, the user can add/subtract the quantity of places booked for a selection in the basket. In ***Fig. 4*** we see how the user can change the ‘Aerial, Dingle, Breakfast Only, Uncertified, 1’ booking to 4 places instead of just 1. The sales order total cost and total number of places also updates to account for this adjustment.

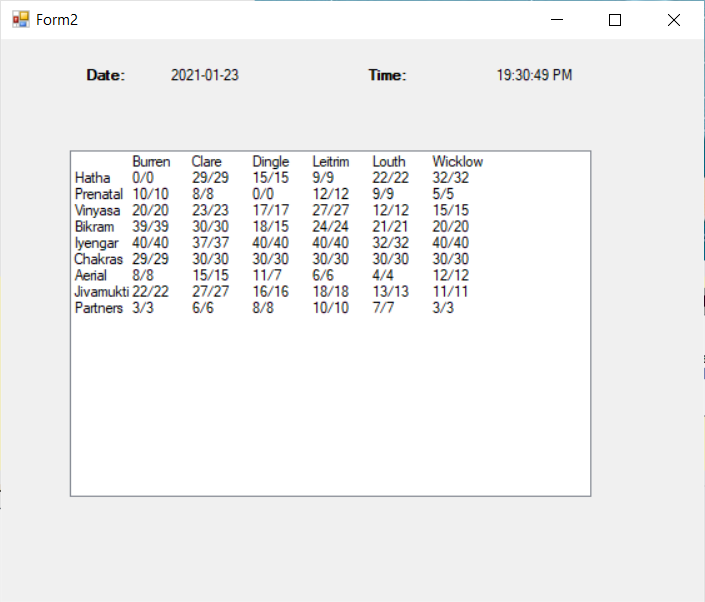
 **Fig 4**

 **Fig 5**

A message box is then displayed showing each line item’s order details, including the price for each and the Total Cost of Sales Order. When asked “Do you wish to proceed?”, the user can select the Yes or No button.

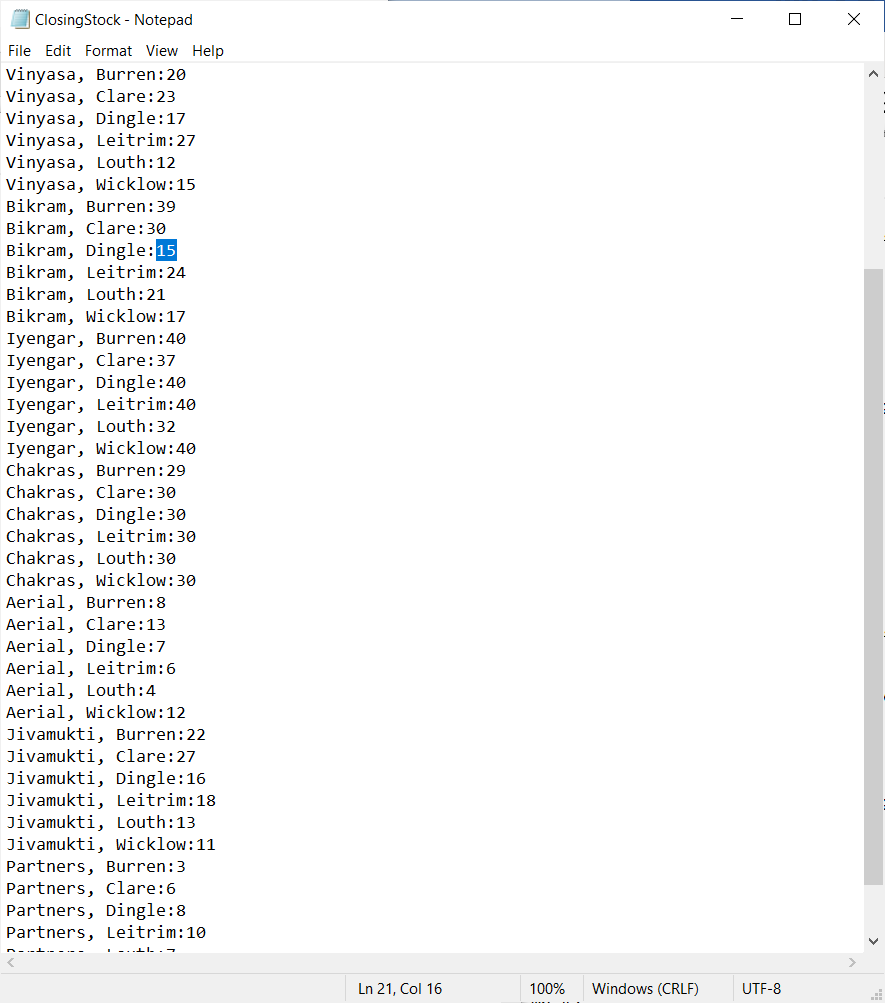
 **Fig 6**

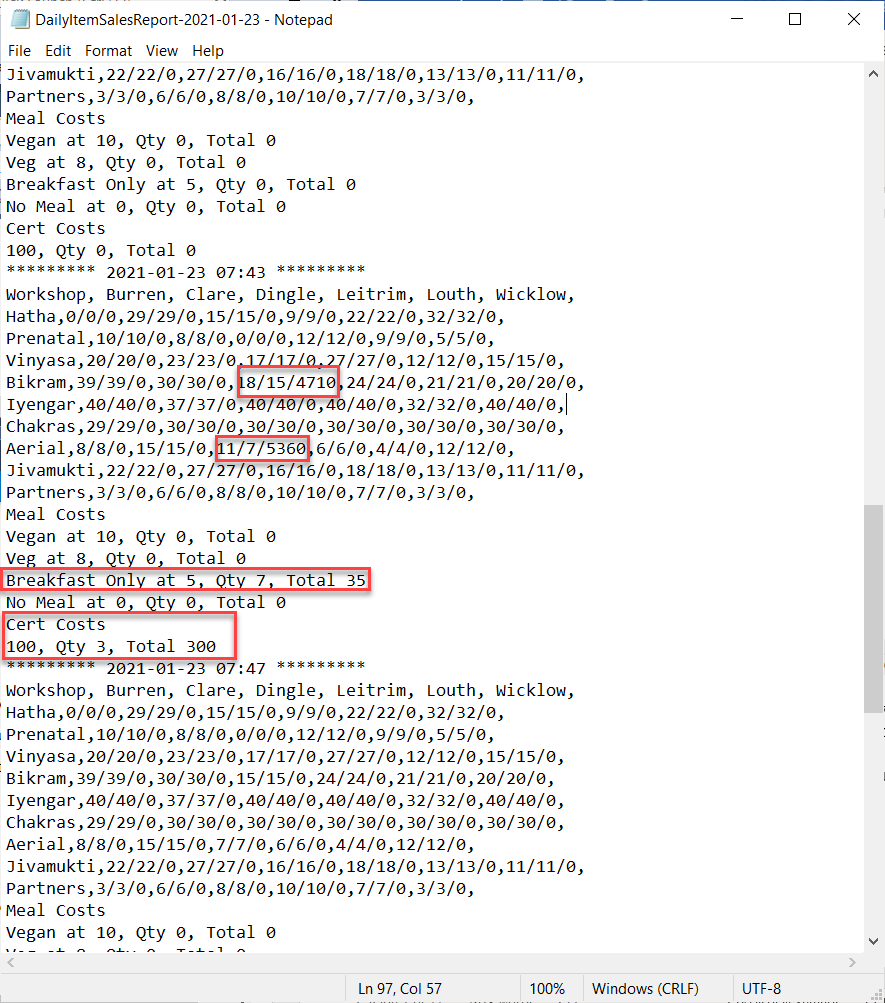
If the user selects ‘No’, they will be brought back to the previous screen and can either make modifications and then Complete the Order again, or can simply Clear the application and start over, or Exit the application completely. If the user chooses ‘Yes’, a new message box will display the Transaction ID and the Date. These values can be searched in the Search group box (either by ID or by Date). This order has now been successful and has been added to the Management Stock Report (***Fig. 7***).

 **Fig 7**

The Management Stock Report is essentially a summary of workshop/location places showing the starting number of places available for each combination at time of opening the application, followed by the current number of places available for each product combination. For example, we can see in Fig. 7 that Bikram at Dingle started with 18 places available to book at running time of the application, but after the last order of 3 bookings was made for this workshop/location, there are now only 15 places available. We see that the number of places available for Aerial Yoga in Dingle has also decreased from 11 to 7 (i.e., by 4 places).

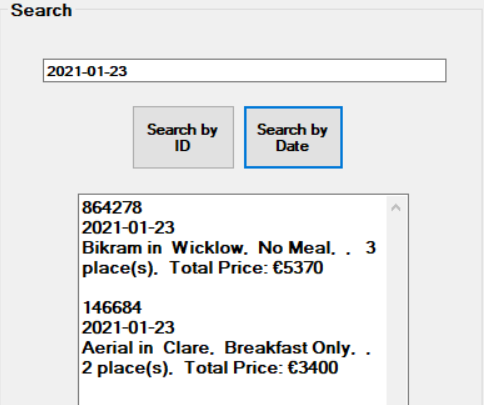
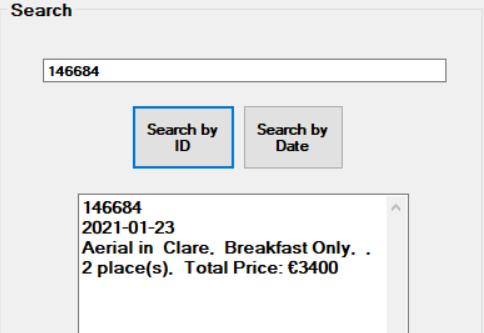
The ClosingStock text file (available in Debug folder) also shows the final number of places available of each workshop/location (see ***Fig. 8***). These stock values are read into a 2D array in the application from file at the start of each day (OpeningStock) and read out at the end of each day (ClosingStock).

 **Fig 8**



**Fig 9**

A Daily Item Sales Report (See ***Fig. 9***) text file (also saved to Debug folder) is created every day the application is run. The date and time are displayed as headers. We see 3 numbers for each workshop/location shown in the format xx/xx/xx. The first is the number of places available as opening stock, the second is how many places are currently available, and the final is the total sales of each item combo made in one session. A session is defined as the time between when the app is opened and exited. There can be many user sessions with the app within one day. The meal costs and quantities are also shown, as well as the cert costs and quantities.

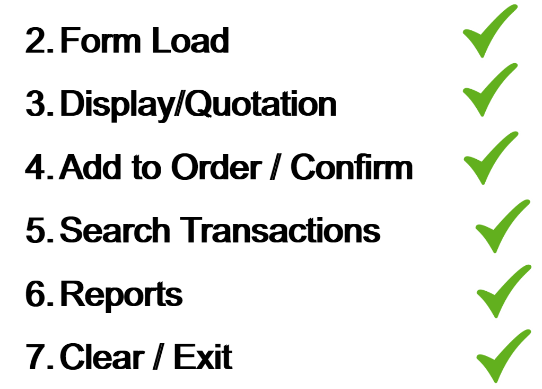
**Fig 10 Fig 11**

If the user searches by date (yyyy-MM-dd) format, each transaction made on that date (1:M) will be returned in the Search Results text box, showing the Transaction ID, date, and booking details (see ***Fig. 10***). If the user keeps track of the transaction IDs, they may also search by these 6-digit randomly generated numbers (1:1) to get the same details returned (see ***Fig. 11****)*.

 **Fig 12**

Lastly, the group box in ***Fig. 12*** shows the buttons for ‘Clear’, ‘Management Stock Report’, and ‘Exit’.

**Appendix**



**Fig: Application Event Flow**

**Source Code:**

/\* Student Name: Caitlin Hogan

\* Student ID: 20231704

\* Date: 25/01/2021

\* Assignment: 4

\* Assignment: Create an EPOS application for a business of your choice.

\*/

**using** System**;**

**using** System**.**Windows**.**Forms**;**

**using** System**.**Globalization**;**

**using** System**.**IO**;**

**using** System**.**Collections**.**Generic**;**

**namespace** yogi\_ireland

**{**

**public** **partial** class Form1 **:** Form

**{**

//Array for the workshops

**public** static string**[]** Workshops **{** **get;** **set;** **}** **=** **{** "Hatha"**,** "Prenatal"**,** "Vinyasa"**,** "Bikram"**,** "Iyengar"**,** "Chakras"**,**

"Aerial"**,** "Jivamukti"**,** "Partners" **};**

//Array for the days

**public** static int**[]** Days **{** **get;** **set;** **}** **=** **{** 5**,** 5**,** 4**,** 4**,** 4**,** 3**,** 3**,** 2**,** 1 **};**

//Array for the workshop fees

**public** static decimal**[]** Fees **{** **get;** **set;** **}** **=** **{** 1000m**,** 900m**,** 950m**,** 850m**,** 900m**,** 700m**,** 800m**,** 500m**,** 300m **};**

// Array for the locations

**public** static string**[]** Locations **{** **get;** **set;** **}** **=** **{** "Burren"**,** "Clare"**,** "Dingle"**,** "Leitrim"**,** "Louth"**,** "Wicklow" **};**

//Array for the lodging fees

**public** static decimal**[]** LodgingFees **{** **get;** **set;** **}** **=** **{** 100m**,** 295m**,** 180m**,** 250m**,** 195m**,** 235m **};**

const decimal TEACH\_CERT\_FEE **=** 100m**;**

decimal CertCost **=** 0**;**

// Creates a CultureInfo for Euro in Ireland

**private** CultureInfo Locality **=** **new** CultureInfo**(**"en-IE"**);**

// order details file path

const string ORDER\_DETAILS **=** "C:\\Users\\caitl\\OneDrive\\Desktop\\Assignment 4\\Assignment 4\\Hogan\_Caitlin\_Assignment 4\_MS806\\bin\\Debug\\OrderDetails.txt"**;**

//2D-Array for the Meal Prices based on customer meal and workshop selection

**public** static string**[]** MealOptions **{** **get;** **set;** **}** **=** **{** "Vegan"**,** "Veg"**,** "Breakfast Only"**,** "No Meal" **};**

**public** static decimal**[]** MealCosts **{** **get;** **set;** **}** **=** **{** 10m**,** 8m**,** 5m**,** 0m **};**

//use a list of arrays to store basket details

List**<**int**[]>** Basket **=** **new** List**<**int**[]>** **();**

//use a list of arrays to store sales (transaction) data

List**<**string**[]>** SalesData **=** **new** List**<**string**[]>();**

//create int array store items from ItemBasket

List**<**int**[]>** AllTransactions **{** **get;** **set;** **}** **=** **new** List**<**int**[]>();**

//2D-Array for the Places Available

**public** static int**[,]** PlacesAvailable **{** **get;** **set;** **}** **=** **{** **{** 25**,** 30**,** 15**,** 12**,** 22**,** 32**},**

**{** 10**,** 8**,** 8**,** 12**,** 15**,** 5**},**

**{** 20**,** 27**,** 32**,** 28**,** 15**,** 18**},**

**{** 40**,** 30**,** 20**,** 25**,** 25**,** 20**},**

**{** 40**,** 40**,** 40**,** 40**,** 40**,** 40**},**

**{** 30**,** 30**,** 30**,** 30**,** 30**,** 30**},**

**{** 8**,** 15**,** 12**,** 6**,** 4**,** 12**},**

**{** 22**,** 27**,** 16**,** 18**,** 14**,** 11**},**

**{** 4**,** 6**,** 8**,** 10**,** 10**,** 10**}** **};**

//Array for live count of stock

const int ROW\_SIZE **=** 9**,** COLUMN\_SIZE **=** 6**;**

**public** static int**[,]** OpeningStock **=** **new** int**[**ROW\_SIZE**,** COLUMN\_SIZE**];**

//file reader and writer objects

//StreamWriter OutputFile;

StreamReader InputFile**;**

//file names

string DAILY\_ITEM\_SALES\_FILE\_NAME **=** "DailyItemSalesReport-" **+** System**.**DateTime**.**Today**.**ToString**(**"yyyy-MM-dd"**)** **+** ".txt"**;**

string CLOSING\_STOCK\_FILE\_NAME **=** "ClosingStock.txt"**;**

string MGMT\_REPORT\_FILE\_NAME **=** "MgmtReport-" **+** System**.**DateTime**.**Today**.**ToString**(**"yyyy-MM-dd"**)** **+** ".txt"**;**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

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

**{**

ClearButton\_Click**(**sender**,** e**);**

ManagementStockButton**.**Enabled **=** **false;**

//Read Closing Stock of Yesterday; set as Opening Stock of Today

**if** **(**File**.**Exists**(**CLOSING\_STOCK\_FILE\_NAME**))**

**{**

**try**

**{**

InputFile **=** File**.**OpenText**(**CLOSING\_STOCK\_FILE\_NAME**);**

//Read File and Save to PlacesAvailable

**for** **(**int i **=** 0**;** i **<** Workshops**.**Length**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

string Line **=** InputFile**.**ReadLine**();**

**if** **(**Line**.**IndexOf**(**":"**)** **>=** 0**)**

**{**

Line **=** Line**.**Substring**(**Line**.**IndexOf**(**":"**)** **+** 1**);**

OpeningStock**[**i**,** j**]** **=** int**.**Parse**(**Line**);**

PlacesAvailable**[**i**,** j**]** **=** int**.**Parse**(**Line**);**

**}**

**}**

**}**

InputFile**.**Close**();**

**}**

**catch** **(**Exception ex**)**

**{**

MessageBox**.**Show**(**"Can't read last Transaction day's file:" **+** ex**.**Message**);**

**}**

**}**

//in case where there is no Closing Stock, the array PlacesAvailable is taken as Opening Stock

**else**

**{**

**for** **(**int i **=** 0**;** i **<** Workshops**.**Length**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

OpeningStock**[**i**,** j**]** **=** PlacesAvailable**[**i**,** j**];**

**}**

**}**

**}**

**}**

**private** void DisplayButton\_Click**(object** sender**,** EventArgs e**)**

**{**

int WorkshopIndex **=** WorkshopListBox**.**SelectedIndex**;**

int LocationIndex **=** LocationListBox**.**SelectedIndex**;**

int MealIndex **=** MealListBox**.**SelectedIndex**;**

CertCost **=** 0**;**

**if** **(**WorkshopListBox**.**SelectedIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a workshop."**);**

**}**

**else** **if** **(**LocationListBox**.**SelectedIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a location."**);**

**}**

**else** **if** **(**MealListBox**.**SelectedIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a meal plan."**);**

**}**

**else** **if** **(**NumPlaces**.**Value **==** 0**)**

**{**

MessageBox**.**Show**(**"No. of Places should be more than 0."**);**

**}**

**if** **(**CertificateCheckBox**.**Checked**)**

**{**

CertCost **=** TEACH\_CERT\_FEE**;**

**}**

**try**

**{**

decimal TotalCost **=** **(**Fees**[**WorkshopIndex**]**

**+** **(**Days**[**WorkshopIndex**]** **\*** LodgingFees**[**LocationIndex**])**

**+** **(**Days**[**WorkshopIndex**]** **\*** MealCosts**[**MealIndex**])**

**+** CertCost**)**

**\*** NumPlaces**.**Value**;**

CurrentCostValue**.**Text **=** TotalCost**.**ToString**(**"C"**,** Locality**);**

int NumPlacesValue **=** Int32**.**Parse**(**NumPlaces**.**Value**.**ToString**());**

decimal CurrentCost **=** Decimal**.**Parse**(**CurrentCostValue**.**Text**.**Remove**(**0**,** 1**));**

**if** **(**NumPlaces**.**Value **>** 0**)**

**{**

PricePerPersonLabel**.**Text **=** **(**CurrentCost **/** NumPlacesValue**).**ToString**(**"C"**,** Locality**);**

**}**

**}**

**catch** **(**Exception e1**)**

**{**

PricePerPersonLabel**.**Text **=** "N/A"**;**

MessageBox**.**Show**(**"Please ensure all selections are complete."**);**

**}**

**}**

**private** void AddToBookingButton\_Click**(object** sender**,** EventArgs e**)**

**{**

int WorkshopIndex **=** WorkshopListBox**.**SelectedIndex**;**

int LocationIndex **=** LocationListBox**.**SelectedIndex**;**

int MealIndex **=** MealListBox**.**SelectedIndex**;**

**if** **(**WorkshopIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a workshop."**);**

**}**

**else** **if** **(**LocationIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a location."**);**

**}**

**else** **if** **(**MealIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a meal plan."**);**

**}**

**else**

**{**

//check that quantity is a number greater than 0

int Quantity **=** 0**;**

**if** **(**Int32**.**TryParse**(**NumPlaces**.**Text**,** **out** Quantity**))**

**{**

**if** **(**Quantity **>** 0**)**

**{**

//check that you have enough places available to book

int available **=** PlacesAvailable**[**WorkshopIndex**,** LocationIndex**];**

//need to subtract quantity if item is already in the basket

**for** **(**int i **=** 0**;** i **<** Basket**.**Count**;** i**++)**

**{**

**if** **(**Basket**[**i**][**0**]** **==** WorkshopIndex **&&** Basket**[**i**][**1**]** **==** LocationIndex**)**

**{**

available **-=** Basket**[**i**][**4**];**

**}**

**}**

**if** **(**Quantity **>** available**)**

**{**

MessageBox**.**Show**(**"Not enough places available (" **+** available **+** ")"**);**

**}**

**else**

**{**

//valid quant

string Certification **=** "Uncertified"**;**

int CertificationValue **=** 0**;**

**if** **(**CertificateCheckBox**.**Checked**)**

**{**

Certification **=** "Certified "**;**

CertificationValue **=** 1**;**

**}**

string Details **=** Workshops**[**WorkshopIndex**]** **+** "\t" **+** Locations**[**LocationIndex**]** **+** "\t" **+**

MealOptions**[**MealIndex**]** **+** "\t" **+** Certification **+** "\t" **+** Quantity**;**

ItemBasket**.**Items**.**Add**(**Details**);**

//MessageBox.Show(ItemBasket.Items.Count.ToString());

int**[]** ItemDetails **=** **{** WorkshopIndex**,** LocationIndex**,** MealIndex**,** CertificationValue**,** Quantity **};**

Basket**.**Add**(**ItemDetails**);**

CompleteOrderButton**.**Enabled **=** **true;**

CompleteOrderButton**.**Focus**();**

UpdateTotals**();**

**}**

**}**

//invalid quant for number of places

**else**

**{**

MessageBox**.**Show**(**"Number of places needs to be greater than 0"**);**

**}**

**}**

**else**

**{**

MessageBox**.**Show**(**"Invalid number of places"**);**

**}**

**}**

NumPlaces**.**Value **=** 0**;**

WorkshopListBox**.**Focus**();**

**}**

**public** void UpdateQuantity\_Click**(object** sender**,** EventArgs e**)**

**{**

//allow user to update quantity of the selected basket item

**if** **(**ItemBasket**.**SelectedIndex **==** **-**1**)**

**{**

MessageBox**.**Show**(**"You must select a basket item"**);**

**}**

**else**

**{**

int NewQuantity **=** Int32**.**Parse**(**UpdateQuant**.**Value**.**ToString**());**

**if** **(**NewQuantity **<** 0**)**

**{**

MessageBox**.**Show**(**"Quantity must be at least 0."**);**

**}**

**else**

**{**

//if quantity is 0, remove the item

**if** **(**NewQuantity **==** 0**)**

**{**

//remove the item

Basket**.**RemoveAt**(**ItemBasket**.**SelectedIndex**);**

ItemBasket**.**Items**.**RemoveAt**(**ItemBasket**.**SelectedIndex**);**

//update total

UpdateTotals**();**

**}**

**else**

**{**

//check that you have enough places available to book

int available **=** PlacesAvailable**[**WorkshopListBox**.**SelectedIndex**,** LocationListBox**.**SelectedIndex**];**

//need to subtract quantity if item is already in the basket

**for** **(**int i **=** 0**;** i **<** Basket**.**Count**;** i**++)**

**{**

**if** **(** i **!=** ItemBasket**.**SelectedIndex **&&** Basket**[**i**][**0**]** **==** WorkshopListBox**.**SelectedIndex **&&** Basket**[**i**][**1**]** **==** LocationListBox**.**SelectedIndex**)**

**{**

available **-=** Basket**[**i**][**4**];**

**}**

**}**

**if** **(**NewQuantity **>** available**)**

**{**

MessageBox**.**Show**(**"Sorry, there is/are only " **+** available **+** " place(s) left for the workshop chosen."**);**

**}**

**else**

**{**

string Certification **=** "Uncertified"**;**

**if** **(**Basket**[**ItemBasket**.**SelectedIndex**][**3**]** **==** 1**)**

**{**

Certification **=** "Certified "**;**

**}**

Basket**[**ItemBasket**.**SelectedIndex**][**4**]** **=** NewQuantity**;**

string Details **=** Workshops**[**Basket**[**ItemBasket**.**SelectedIndex**][**0**]]** **+** "\t" **+** Locations**[**Basket**[**ItemBasket**.**SelectedIndex**][**1**]]** **+**

"\t" **+** MealOptions**[**Basket**[**ItemBasket**.**SelectedIndex**][**2**]]** **+**

"\t" **+** Certification **+** "\t" **+** NewQuantity**;**

ItemBasket**.**Items**[**ItemBasket**.**SelectedIndex**]** **=** Details**;**

UpdateTotals**();**

**}**

**}**

**}**

**}**

**}**

**private** void Form1\_FormClosing**(object** sender**,** FormClosingEventArgs e**)**

**{**

//generate end-of-day reports

//CLOSING\_STOCK\_FILE\_NAME

//if file does not exist, create it.

//Free unmanaged resources. Use method dispose() whenever there is a close() method -- see above

**if** **(!**File**.**Exists**(**CLOSING\_STOCK\_FILE\_NAME**))**

**{**

File**.**CreateText**(**CLOSING\_STOCK\_FILE\_NAME**).**Dispose**();**

**}**

**try**

**{**

//open file as text file and append client details to end of text file

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**CLOSING\_STOCK\_FILE\_NAME**))**

**{**

**for(**int i **=** 0**;** i **<** Workshops**.**Length**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

stream**.**WriteLine**(**Workshops**[**i**]** **+** ", " **+** Locations**[**j**]** **+** ":" **+** PlacesAvailable**[**i**,**j**]);**

**}**

**}**

**}**

**}**

**catch** **(**Exception e1**)**

**{**

MessageBox**.**Show**(**"File does not exist. Please try again." **+** e1**.**Message**);**

**}**

//do similar process for DAILY\_ITEM\_SALES\_FILE\_NAME

//if file does not exist, create it

**if** **(!**File**.**Exists**(**DAILY\_ITEM\_SALES\_FILE\_NAME**))**

**{**

File**.**CreateText**(**DAILY\_ITEM\_SALES\_FILE\_NAME**).**Dispose**();**

**}**

**try**

**{**

//open file as text file and append client details to end of text file

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**DAILY\_ITEM\_SALES\_FILE\_NAME**,** **true))**

**{**

string DateInfo **=** "\*\*\*\*\*\*\*\*\* " **+** DateTime**.**Now**.**ToString**(**"yyyy-MM-dd hh:mm"**)** **+** " \*\*\*\*\*\*\*\*\*"**;**

stream**.**WriteLine**(**DateInfo**);**

string Line **=** "Workshop, "**;**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

Line **+=** Locations**[**j**]** **+** ", "**;**

**}**

stream**.**WriteLine**(**Line**);**

**for** **(**int i **=** 0**;** i **<** Workshops**.**Length**;** i**++)**

**{**

Line **=** Workshops**[**i**]** **+** ':'**;**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

// (Fees[i] + (Days[i] \* LodgingFees[j])) is the fee amount for the given selection

// OpeningStock[i,j] - PlacesAvailable[i,j] is the amount of selected item combo sold in one day

// Hence, Cost gives the total cost of each item combo booked in current day

decimal Cost **=** **(**Fees**[**i**]** **+** **(**Days**[**i**]** **\*** LodgingFees**[**j**]))** **\*** **(**OpeningStock**[**i**,**j**]** **-** PlacesAvailable**[**i**,**j**]);**

//display how many places were available as opening stock, how many are currently available, and total sales of each item combo in one day/'session'

Line **+=** OpeningStock**[**i**,** j**].**ToString**()** **+** "/" **+** PlacesAvailable**[**i**,** j**]** **+** "/" **+** Cost**.**ToString**()** **+** ','**;**

**}**

stream**.**WriteLine**(**Line**);**

**}**

//work out costs of meals and certificates

stream**.**WriteLine**(**"Meal Costs"**);**

int CertsCount **=** 0**;**

**for** **(**int i **=** 0**;** i **<** MealOptions**.**Length**;** i**++)**

**{**

int MealsCount **=** 0**;**

**for** **(**int j **=** 0**;** j **<** AllTransactions**.**Count**;** j**++)**

**{**

**if** **(**AllTransactions**[**j**][**2**]** **==** i**)**

**{**

MealsCount **+=** AllTransactions**[**j**][**4**];**

**}**

**if** **(**i **==** 0 **&&** AllTransactions**[**j**][**3**]** **==** 1**)**

**{**

//make sure cert counts are only counted once

CertsCount **+=** AllTransactions**[**j**][**4**];**

**}**

**}**

Line **=** MealOptions**[**i**]** **+** " at " **+** MealCosts**[**i**]** **+** ", Qty " **+** MealsCount**.**ToString**()** **+** ", Total " **+** **(**MealsCount **\*** MealCosts**[**i**]).**ToString**();**

stream**.**WriteLine**(**Line**);**

**}**

stream**.**WriteLine**(**"Cert Costs"**);**

Line **=** TEACH\_CERT\_FEE**.**ToString**()** **+** ", Qty " **+** CertsCount**.**ToString**()** **+** ", Total " **+** **(**CertsCount **\*** TEACH\_CERT\_FEE**).**ToString**();**

stream**.**WriteLine**(**Line**);**

**}**

**}**

**catch** **(**Exception e1**)**

**{**

MessageBox**.**Show**(**"File does not exist. Please try again." **+** e1**.**Message**);**

**}**

**}**

**private** void SearchByDateButton\_Click**(object** sender**,** EventArgs e**)**

**{**

SearchResultsTB**.**Text **=** ""**;**

string SearchTerm **=** SearchTermTextBox**.**Text**;**

bool Result **=** **false;**

**foreach** **(**string**[]** t **in** SalesData**)**

**{**

**if** **(**t**[**1**]** **==** SearchTerm**)**

**{**

SearchResultsTB**.**AppendText**(**t**[**0**]** **+** Environment**.**NewLine**);**

SearchResultsTB**.**AppendText**(**t**[**1**]** **+** Environment**.**NewLine**);**

SearchResultsTB**.**AppendText**(**t**[**2**].**Replace**(**"\t"**,** " "**)** **+** Environment**.**NewLine**);**

SearchResultsTB**.**AppendText**(**Environment**.**NewLine**);**

Result **=** **true;**

**}**

**}**

**if(**Result **==** **false)**

**{**

MessageBox**.**Show**(**"Result not found. Please try again."**);**

**}**

**}**

**private** void SearchByIDButton\_Click**(object** sender**,** EventArgs e**)**

**{**

SearchResultsTB**.**Text **=** ""**;**

string SearchTerm **=** SearchTermTextBox**.**Text**;**

bool Result **=** **false;**

**foreach** **(**string**[]** t **in** SalesData**)**

**{**

**if** **(**t**[**0**]** **==** SearchTerm**)**

**{**

SearchResultsTB**.**AppendText**(**t**[**0**]** **+** Environment**.**NewLine**);**

SearchResultsTB**.**AppendText**(**t**[**1**]** **+** Environment**.**NewLine**);**

SearchResultsTB**.**AppendText**(**t**[**2**].**Replace**(**"\t"**,** " "**)** **+** Environment**.**NewLine**);**

Result **=** **true;**

**}**

**}**

**if** **(**Result **==** **false)**

**{**

MessageBox**.**Show**(**"Result not found. Please try again."**);**

**}**

**}**

Form2 SecondForm **=** **new** Form2**();**

**private** void ManagementStockButton\_Click**(object** sender**,** EventArgs e**)**

**{**

SecondForm**.**Show**();**

SecondForm**.**MgmtListBox**.**Items**.**Clear**();**

//loop through locations and workshops to display stock for each combination

string Line **=** " \t"**;**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

Line **+=** Locations**[**j**]** **+** '\t'**;**

**}**

SecondForm**.**MgmtListBox**.**Items**.**Add**(**Line**);**

**for** **(**int i **=** 0**;** i **<** Workshops**.**Length **;** i**++)**

**{**

Line **=** Workshops**[**i**]+** '\t'**;**

**for** **(**int j **=** 0**;** j **<** Locations**.**Length**;** j**++)**

**{**

Line **+=** OpeningStock**[**i**,** j**].**ToString**()** **+** "/" **+** PlacesAvailable**[**i**,**j**]** **+** '\t'**;**

**}**

SecondForm**.**MgmtListBox**.**Items**.**Add**(**Line**);**

**}**

ClearButton**.**Focus**();**

**}**

**private** void UpdateTotals**()**

**{**

//calculate and display basket totals

decimal TotalCost **=** 0**;**

int TotalPlaces **=** 0**;**

**for** **(**int i **=** 0**;** i **<** Basket**.**Count**;** i**++)**

**{**

TotalCost **+=** **(**Fees**[**Basket**[**i**][**0**]]**

**+** **(**Days**[**Basket**[**i**][**0**]]** **\*** LodgingFees**[**Basket**[**i**][**1**]])**

**+** **(**Days**[**Basket**[**i**][**0**]]** **\*** MealCosts**[**Basket**[**i**][**2**]])**

**+** **(**TEACH\_CERT\_FEE **\*** Basket**[**i**][**3**]))**

**\*** Basket**[**i**][**4**];**

TotalPlaces **+=** Basket**[**i**][**4**];**

**}**

LabelTotalSalesOrder**.**Text **=** TotalCost**.**ToString**(**"C"**,** Locality**);**

LabelTotalPlaces**.**Text **=** TotalPlaces**.**ToString**();**

**}**

//public void UpdateListBox(string lstValue)

//{

// ItemBasket.Items.Add(lstValue);

//}

**private** void Form2\_Load**(object** sender**,** EventArgs e**)**

**{**

ClearButton\_Click**(**sender**,** e**);**

ManagementStockButton**.**Enabled **=** **false;**

**}**

**private** void CompleteOrderButton\_Click**(object** sender**,** EventArgs e**)**

**{**

//ask user if details can be confirmed

string Details **=** ""**;**

**for** **(**int i **=** 0**;** i **<** Basket**.**Count**;** i**++)**

**{**

string Certification **=** ""**;**

**if** **(**Basket**[**i**][**3**]** **==** 1**)**

**{**

Certification **=** "Certified "**;**

**}**

decimal TotalCost **=** **(**Fees**[**WorkshopListBox**.**SelectedIndex**]** **+**

**(**Days**[**WorkshopListBox**.**SelectedIndex**]** **\*** LodgingFees**[**LocationListBox**.**SelectedIndex**])** **+**

**(**Days**[**WorkshopListBox**.**SelectedIndex**]** **\*** MealCosts**[**MealListBox**.**SelectedIndex**])** **+**

CertCost**)** **\*** NumPlaces**.**Value**;**

//update details of each booking

Details **+=** Workshops**[**Basket**[**i**][**0**]]** **+** " in \t" **+** Locations**[**Basket**[**i**][**1**]]** **+** ", \t" **+** MealOptions**[**Basket**[**i**][**2**]]** **+**

", \t" **+** Certification **+** ", \t " **+** Basket**[**i**][**4**]** **+** " place(s), \n Total Price: €" **+**

**(**Fees**[**Basket**[**i**][**0**]]** **+** **(**Days**[**Basket**[**i**][**0**]]** **\*** LodgingFees**[**Basket**[**i**][**1**]])** **+** **(**TEACH\_CERT\_FEE **\*** Basket**[**i**][**3**])**

**+** **(**Days**[**Basket**[**i**][**0**]]** **\*** MealCosts**[**Basket**[**i**][**2**]]))** **\*** Basket**[**i**][**4**]** **+** "\n"**;**

**}**

//print out details of each booking to a message box before confirming order

var result **=** MessageBox**.**Show**(**"Order Details: " **+** "\t" **+** Details **+** "\n\n Total Cost of Sales Order: " **+** LabelTotalSalesOrder**.**Text **+**

" \n\n Do you wish to proceed ?"**,** "Confirmation"**,** MessageBoxButtons**.**YesNo**,** MessageBoxIcon**.**Question**);**

**if** **(**result **==** DialogResult**.**Yes**)**

**{**

//confirm and save details

int TransactionID **=** 123456**;**

//generate random transaction ID value that is 6 characters long

//integer generated is between 100000 and 999999

Random R **=** **new** Random**(**DateTime**.**Now**.**Millisecond**);**

TransactionID **=** R**.**Next**(**100000**,** 999999**);**

// Check if NewID is duplicate

**while** **(**CheckDuplicates**(**TransactionID**)** **==** **true)**

**{**

R **=** **new** Random**(**DateTime**.**Now**.**Millisecond**);**

TransactionID **=** R**.**Next**(**100000**,** 999999**);**

**}**

//get today's date

string OrderDate **=** DateTime**.**Now**.**ToString**(**"yyyy-MM-dd"**);**

string**[]** T **=** **{** TransactionID**.**ToString**(),** OrderDate**,** Details **};**

//add data to the AllTransactions list

**for** **(**int i **=** 0**;** i **<** Basket**.**Count**;** i**++)**

**{**

string Certification **=** ""**;**

**if** **(**Basket**[**i**][**3**]** **==** 1**)**

**{**

Certification **=** "Certified "**;**

**}**

//TransDetails += Workshops[Basket[i][0]] + "\t" + Locations[Basket[i][1]] + "\t" + MealOptions[Basket[i][2]] +

//"\t" + Certification + "\t" + Basket[i][4] + '\n';

//adjust quantity available

PlacesAvailable**[**Basket**[**i**][**0**],** Basket**[**i**][**1**]]** **-=** Basket**[**i**][**4**];**

AllTransactions**.**Add**(**Basket**[**i**]);**

**}**

// we use the SalesData list to save sales info in memory so we may search

SalesData**.**Add**(**T**);**

MessageBox**.**Show**(**"Order " **+** TransactionID **+** " Date " **+** OrderDate**);**

//clear basket

ClearButton\_Click**(**sender**,** e**);**

ManagementStockButton**.**Enabled **=** **true;**

**}**

**}**

**private** bool CheckDuplicates**(**int NewID**)**

**{**

**try**

**{**

//open file as text file and append client details to end of text file

**using** **(**StreamReader stream **=** **new** StreamReader**(**ORDER\_DETAILS**))**

**{**

string Line **=** ""**;**

//int RecordLength = 0;

// Read and write lines from the file until end of file is reached

**while** **((**Line **=** stream**.**ReadLine**())** **!=** **null)**

**{**

**if** **(**Int32**.**Parse**(**Line**)** **==** NewID**)**

**{**

MessageBox**.**Show**(**"ID is already taken."**);**

**return** **true;**

**}**

**for** **(**int i **=** 0**;** i **<** 6**;** i**++)**

**{**

Line **=** stream**.**ReadLine**();**

**}**

**}**

**return** **false;**

**}**

**}**

**catch** **(**Exception e1**)**

**{**

MessageBox**.**Show**(**"File does not exist. Please try again." **+** e1**.**Message**);**

**}**

**return** **false;**

**}**

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

**{**

WorkshopListBox**.**SelectedIndex **=** **-**1**;**

LocationListBox**.**SelectedIndex **=** **-**1**;**

MealListBox**.**SelectedIndex **=** **-**1**;**

CertificateCheckBox**.**Checked **=** **false;**

CompleteOrderButton**.**Enabled **=** **false;**

//DisplayButton.Enabled = true;

Basket**.**Clear**();**

ItemBasket**.**Items**.**Clear**();**

UpdateQuant**.**Value **=** 0**;**

LabelTotalPlaces**.**Text **=** ""**;**

LabelTotalSalesOrder**.**Text **=** ""**;**

**}**

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

**{**

**this.**Close**();**

**}**

**}**

**}**

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;

namespace yogi\_ireland

{

public partial class Form2 : Form

{

//public ListBox FormList { get { return MgmtListBox; } }

public Form2()

{

InitializeComponent();

}

public void Form2\_Load(object sender, EventArgs e)

{

//get today's date

string Date = DateTime.Now.ToString("yyyy-MM-dd");

string Time = DateTime.Now.ToString("HH:mm:ss tt");

DateLabel.Text = Date;

TimeLabel.Text = Time;

}

}