Criterion C: Development

Techniques used to create the web application:

1. A login page for security
2. If…else condition (selection)
3. Methods and functions
4. Recursion
5. For…next and While loops
6. Parallel arrays
7. Database relationships (one-many)
8. DML (select, update, insert into, delete)
9. Complex queries
10. Selection sort
11. Nested loops
12. External tools
13. Filtering
14. 2D arrays
15. Validation
16. Verification
17. Efficient sequential search

1. A login page for security

I made a login page which requires a specific set of username and password in order to access the rest of the application. This authenticates the user, thus minimizes security risks.

Figure 1.1 Display of the login form

```
Partial Class login
    Inherits System.Web.UI.Page
    Protected Sub loginbutton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles loginbutton.Click
        Dim reader As Data.IDataReader = CType(LoginDataSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
        reader.Read()
        If uname.Text = reader("Username") Then
            If pass.Text = reader("Password") Then
                MsgBox("Access Granted")
                Response.Redirect("home.aspx")
            Else
                MsgBox("Wrong password")
            End If
        Else
            MsgBox("Wrong username")
        End If
    End Sub
End Class
```

Figure 1.2 Code of the login form

2. If…else condition (selection)

I used the if…else logical condition in selecting the product name based off its respective product ID.

```
Protected Sub SearchProduct_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles SearchProduct.Click
    Dim ans As String
    Dim flag As Boolean = False
    ans = InputBox("Enter ID")
    If ans <> "" Then
        Dim reader As Data.IDataReader = CType(ProductListSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
        While reader.Read()
            If reader("Product_ID") = ans Then
                ProductDropDownList.SelectedValue = reader("Product_ID")
                flag = True
                Exit While
            End If
        End While
        reader.Close()
        If flag = False Then
            MsgBox("ID not found.", MsgBoxStyle.Exclamation)
        End If
        refreshAmount()
    End If
End Sub
```

Figure 2.1 Code to search for product by ID containing if…else condition

3. Methods and functions

I used multiple methods and functions which facilitate code reuse.

One of which is the refreshAmount() method which has the purpose of refreshing the amount textbox depending on the product being selected and the order it belongs to.

```
Public Sub refreshAmount()
    GridView1.DataBind()
    If GridView1.Rows.Count > 0 Then
        If ProductDropDownList.SelectedItem IsNot Nothing Then
            Dim reader1 As Data.IDataReader = CType(GridViewDataSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
            Dim flag As Integer = 0
            AmountInput.Text = 0
            While reader1.Read()
                If ProductDropDownList.SelectedItem.Text = reader1("Product_Name") Then
                    AmountInput.Text = reader1("Amount")
                    Exit While
                Else
                    AmountInput.Text = 0
                End If
            End While
            reader1.Close()
        Else
            selectProductGridView()
        End If
    Else
        AmountInput.Text = 0
    End If
End Sub
```

Figure 3.1 Code for refreshAmount() method

The getRecordCount() function serves to return the number of orders that are in the Orders table.

```
Public Function getRecordCount() As Integer
    Dim count As Integer
    Dim countreader As Data.IDataReader = CType(OrdersSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
    While countreader.Read()
        count = count + 1
    End While
    countreader.Close()
    Return count
End Function
```

Figure 3.2 Code for getRecordCount() function

4. Recursion

I used recursion in the changeUnamePass() method, which has the purpose of updating the username and password. The method calls itself whenever the user makes a mistake in typing to request the reentering of information. Recursion is used over iteration as it adds more clarity to the workings of the block of code.

```
    Public Sub changeUnamePass()
        Dim ans, ans1, ans2 As String
        Dim reader As Data.IDataReader = CType(LoginDataSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
        reader.Read()
        ans1 = InputBox("Username", DefaultResponse:=reader("Username"))
        ans2 = InputBox("Password", DefaultResponse:=reader("Password"))
        reader.Close()
        If ans1 <> "" And ans2 <> "" Then
            ans = MsgBox("Is the information provided below correct?" & vbCrLf & vbCrLf & "Username: " & ans1 & vbCrLf & "Password: " & ans2, MsgBoxStyle.YesNo)
            If ans = MsgBoxResult.Yes Then
                LoginDataSource.UpdateParameters("uname").DefaultValue = ans1
                LoginDataSource.UpdateParameters("pass").DefaultValue = ans2
                LoginDataSource.Update()
                MsgBox("Username and Password successfully updated.")
            Else
                MsgBox("Please reenter the Username and Password.")
                changeUnamePass()
            End If
        End If
    End Sub
End Class
```

Figure 4.1 Code of recursive method changeUnamePass()

5. For…next loop and While loop

I used For…Next and While loop in order to improve the efficiency of my code.

I used For loop in the creation of my selection sort code, where there is a set number of values being compared, and so it is possible to predict the number of iterations that will be required.

```
For j = i + 1 To getRecordCount() - 1
    If shipdate(j) < shipdate(min) Then
        min = j
    End If
Next
```

Figure 5.1 Code containing For…Next loop within selection sort

I used While loop to fill my parallel arrays with data read from the database in which, the number of records, and therefore the number of repetitions needed to be done, cannot be initially known.

```
While reader.Read()
    id(count) = reader("Order_ID")
    shipdate(count) = reader("Date_to_be_Shipped")
    count = count + 1
End While
reader.Close()
```

Figure 5.2 Code containing While loop within filling parallel arrays with values read from database

6. Parallel arrays

I used parallel arrays in storing the to-be-sorted order id and date_to_be_shipped values, in which the index represents the position of the order of dates in ascending order, as they are related values, denoting specific orders.

```
While reader.Read()
    id(count) = reader("Order_ID")
    shipdate(count) = reader("Date_to_be_Shipped")
    count = count + 1
End While
reader.Close()
```

Figure 6.1 Code containing the filling of parallel arrays

7. Database relationships (one-many)

I made use of database relationships (specifically one-many) in linking my tables. Each primary key acts as a foreign key in the linked tables namely: Products_of_Orders and Ingredients_Needed. The relationships allow complex queries inner joining one table with another to be created.

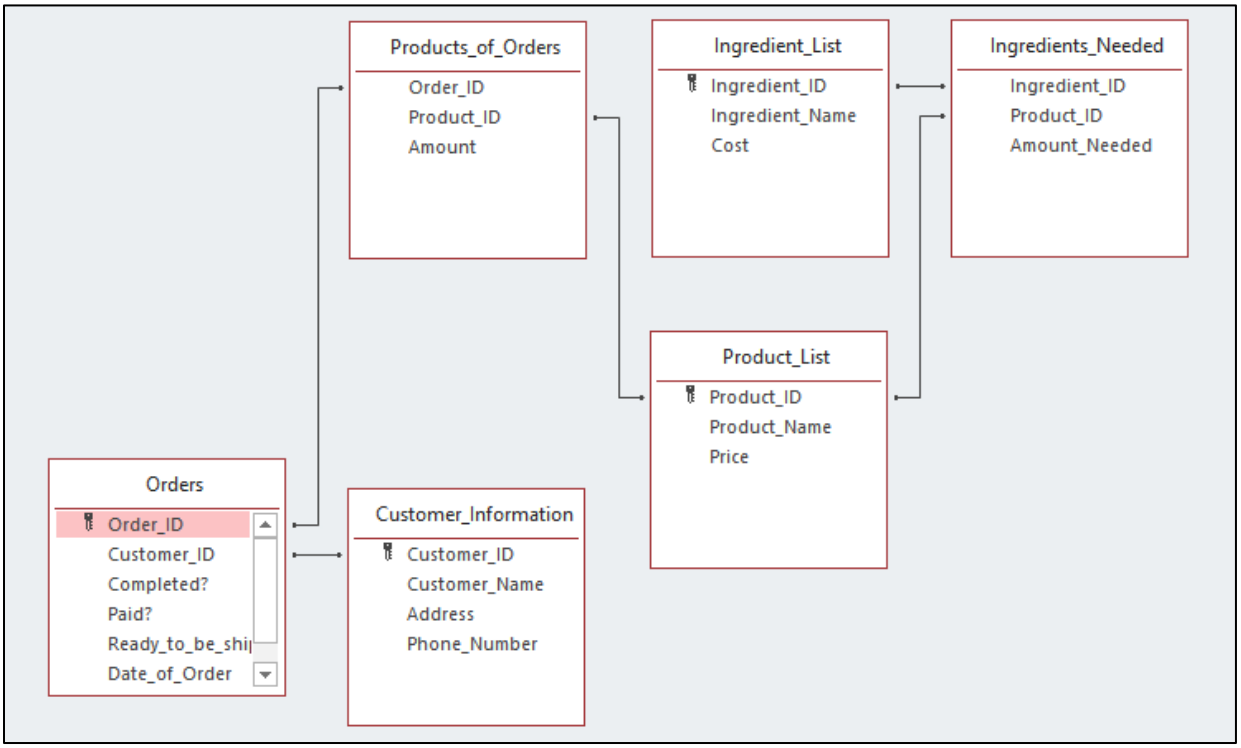

Figure 7.1 Diagram of database relationships

8. DML (select, update, insert, delete)

I made use of DML to modify the database from an external application. I used SELECT to read values, DELETE to delete records, INSERT to add new records and UPDATE to edit existing records of my database.

```
SelectCommand="SELECT * FROM [Products_of_Orders]"
DeleteCommand="DELETE FROM Products_of_Orders WHERE (Products_of_Orders.Order_ID = @orderid)"
InsertCommand="INSERT INTO Products_of_Orders (Product_ID, Amount, Order_ID) VALUES (@productid, @amount, @orderid)"
UpdateCommand="UPDATE Products_of_Orders SET Products_of_Orders.Amount = @amount
               WHERE (Products_of_Orders.Product_ID = @productid AND Products_of_Orders.Order_ID = @orderid)">
```

Figure 8.1 Code containing all types of DML queries

9. Complex queries

I used complex queries to select or read values from two different tables, one a linked table, joined by a single one-many relationship that is the Customer_ID.

```
SELECT Orders.Order_ID, Customer_Information.Customer_Name, Orders.[Completed?], Orders.[Paid?], Orders.
[Ready_to_be_shipped?], Customer_Information.Customer_ID, Orders.Date_of_Order, Orders.Date_to_be_Shipped
FROM (Orders INNER JOIN Customer_Information ON Orders.Customer_ID = Customer_Information.Customer_ID)
```

Figure 9.1 Code containing a complex query using INNER JOIN

10. Selection sort

Selection sort is used to sort the orders by their date to be shipped in ascending order. I chose selection sort over bubble sort because of its efficiency.

```
For i = 0 To getRecordCount() - 2
    min = i
    For j = i + 1 To getRecordCount() - 1
        If shipdate(j) < shipdate(min) Then
            min = j
        End If
    Next
    Dim tempid As Integer = id(min)
    Dim tempdate As Date = shipdate(min)
    id(min) = id(i)
    id(i) = tempid
    shipdate(min) = shipdate(i)
    shipdate(i) = tempdate
Next
```

Figure 10.1 Code containing selection sort

11. Nested loops

I made use of nested while loops to check for the existence of a previous record to read from the database.

```
While reader1.Read() And nextflag = False
    If Val(OrderIDData.Text - 1) = reader1("Order_ID") Then
        nextflag = True
    End If
    If nextflag = True Then
        OrderIDData.Text = Val(OrderIDData.Text) - 1
        Dim reader2 As Data.IDataReader = CType(CustInfoSource.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
        While reader2.Read()
            If reader2("Customer_ID") = reader1("Customer_ID") Then
                NameDropDownList.Text = reader2("Customer_Name")
                DateofOrderData.Text = reader1("Date_of_Order")
                shipdate = reader1("Date_to_be_Shipped")
                DayInput.Text = Day(shipdate)
                MonthInput.Text = Month(shipdate)
                YearInput.Text = Year(shipdate)
            End If
        End While
        reader2.Close()
    End If
End While
reader1.Close()
```

Figure 11.1 Code containing nested loops in the displayInformation() method

12. External tools

I made use of external tools from the AJAX Toolkit in my program as it offered features that are able to optimize my program (ASP.NET Ajax community & ASP.NET team, 2019). Specifically, I used the ComboBox and the NumericUpDownExtender. The combobox allows for autocompletion as well as the features of a dropdownlist to be combined into one tool.

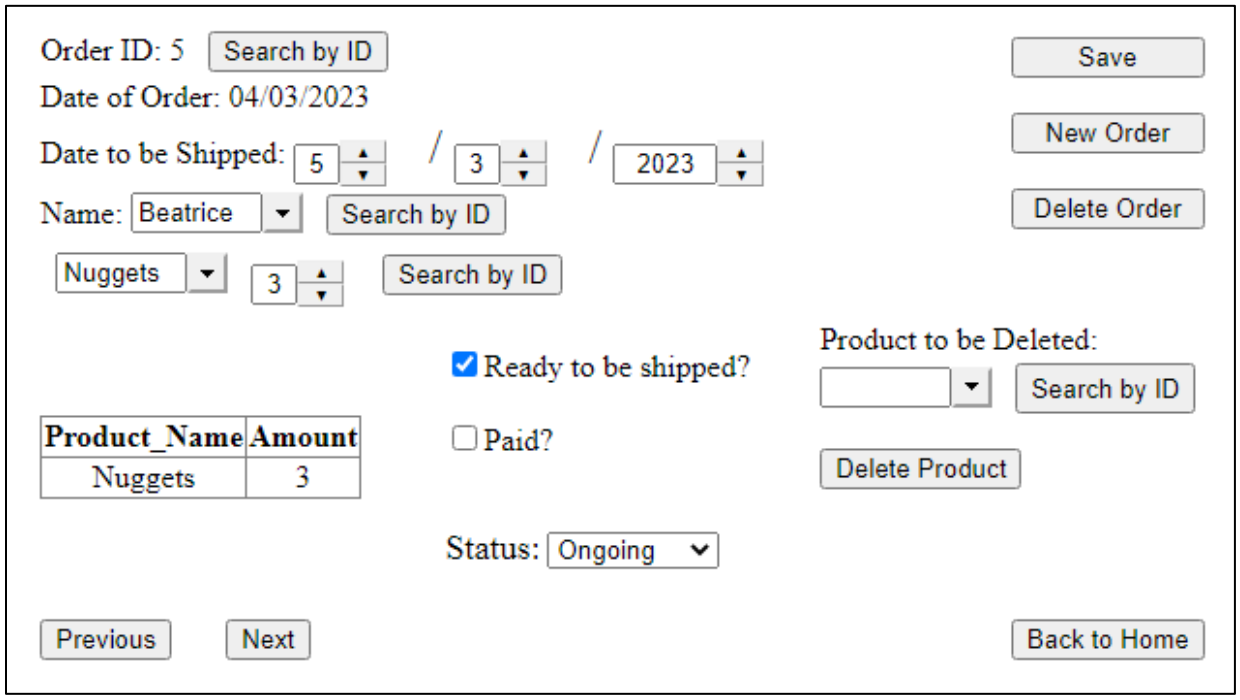

Figure 12.1 Display of the Edit Orders form containing external tools such as ComboBox and NumericUpDown Extender

```
<ajaxToolkit:ComboBox ID="ProductDeleteDropDownList" runat="server" AutoCompleteMode="SuggestAppend"
                      AutoPostBack="True" DataSourceID="GridViewDataSource" DataTextField="Product_Name"
                      DataValueField="Product_ID" DropDownStyle="DropDownList" MaxLength="0" style="display: inline;" Width="60px">
</ajaxToolkit:ComboBox>
```

Figure 12.2 ASP code of Combobox used

13. Filtering

I made use of filtering in the View_Orders form where the users may choose to filter for whether the order is ready to be shipped, paid for, completed as well as by customer. If an order read from the database does not meet the conditions of the filter, it will not be shown and the program checks the next order. If no orders meet the conditions, an error message is shown. This allows the efficient searching of specific records.

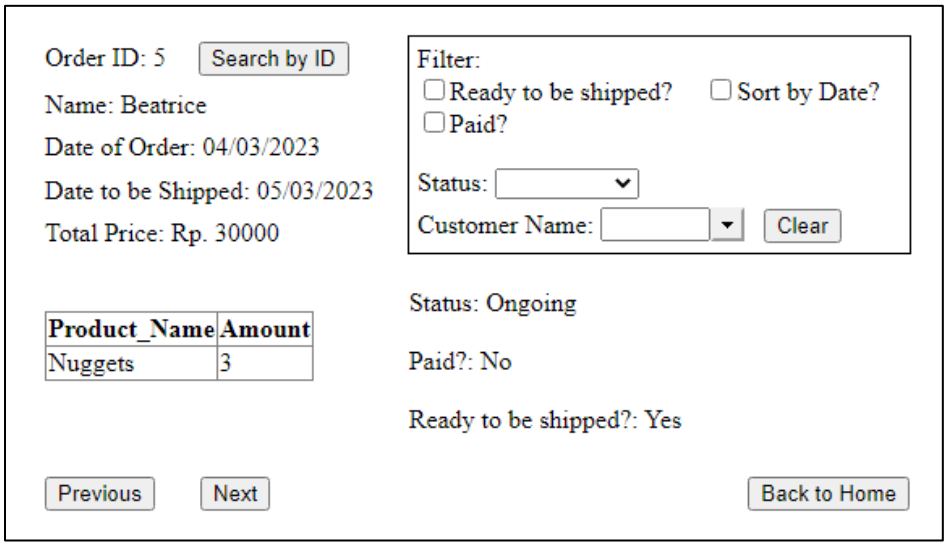

Figure 13.1 Display of the View Orders form containing filtering feature

```
If CompletedDropDownList.SelectedItem.ToString IsNot "" Then
    If CompletedDropDownList.SelectedValue = True And CompletedData.Text = "Ongoing" Then
        filterflag = False
    End If
End If
If paidfilter.Checked = True And paidans.Text = "No" Then
    filterflag = False
End If
If readyfilter.Checked = True And readyans.Text = "No" Then
    filterflag = False
End If
If NameFilterDropDownList.SelectedIndex <> -1 Then
    If NameFilterDropDownList.SelectedItem.ToString IsNot "" Then
        If CStr(NameFilterDropDownList.Text) <> CStr(customernamedata.Text) Then
            filterflag = False
        End If
    End If
End If
If filtercount <= getRecordCount() Then
    If filterflag = False Then
        filtercount = filtercount + 1
        If flag = -1 Then
            displayInformation(0, filtercount)
        Else
            displayInformation(flag, filtercount)
        End If
    End If
Else
    MsgBox("No orders found.")
    readyfilter.Checked = False
    paidfilter.Checked = False
    CompletedDropDownList.Text = ""
    NameFilterDropDownList.Text = Nothing
    displayInformation(-1, 0)
End If
```

Figure 13.2 Code for filtering feature

14. 2D arrays

I used 2D arrays to store the top 5 customers in my selection sort code where the first index determines the rank of the customer and the second to be determining whether the value is the Customer ID or the Customer Name. 2D arrays allows the storing of multiple related values without the need for separate arrays.

```
For i = 0 To 4
    max = 0
    For j = 1 To getMaxCustID()
        If total(j) > total(max) Then
            max = j
        End If
    Next
    topcust(i, 0) = max
    total(max) = 0
Next
```

Figure 14.1 Code containing 2D array

15. Validation

Validation, specifically presence check, is done in my code to search for the product to be deleted. It requires the user to enter an ID, and if the user fails to do so, inputting an empty string, an error message is displayed. This ensures data is logical.

```
ans = InputBox("Enter ID")
If ans <> "" Then
    Dim reader As Data.IDataReader = CType(GridViewDataSource2.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
    While reader.Read()
        If reader("Product_ID") = ans Then
            ProductDeleteDropDownList.SelectedValue = reader("Product_ID")
            flag = True
            Exit While
        End If
    End While
    reader.Close()
    If flag = False Then
        MsgBox("Product not found.", MsgBoxStyle.Exclamation)
    End If
Else
    MsgBox("ID has not been entered, please try again", MsgBoxStyle.Exclamation)
End If
```

Figure 15.1 Code containing presence check

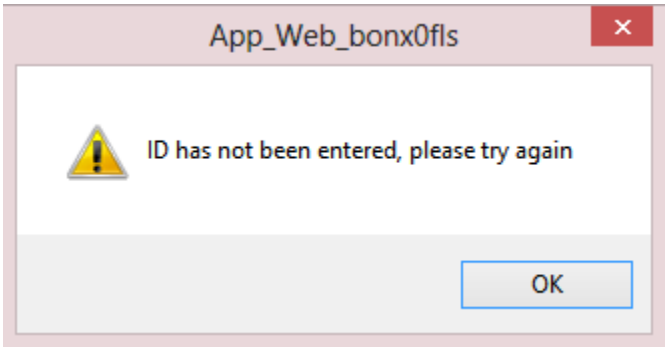

Figure 15.2 Error message displayed

16. Verification

Verification is utilized in the code where the user changes his/her username and/or password. Once this is inputted, the program will display the inputted values and asks the user for confirmation, prompting the user to proofread and therefore verify the data. This ensures data is accurate.

```
ans1 = InputBox("Username", DefaultResponse:=reader("Username"))
ans2 = InputBox("Password", DefaultResponse:=reader("Password"))
reader.Close()
If ans1 <> "" And ans2 <> "" Then
    ans = MsgBox("Is the information provided below correct?" & vbCrLf & vbCrLf & "Username: " & ans1 & vbCrLf & "Password: " & ans2, MsgBoxStyle.YesNo)
    If ans = MsgBoxResult.Yes Then
        LoginDataSource.UpdateParameters("uname").DefaultValue = ans1
        LoginDataSource.UpdateParameters("pass").DefaultValue = ans2
        LoginDataSource.Update()
        MsgBox("Username and Password successfully updated.")
    Else
        MsgBox("Please reenter the Username and Password.")
        changeUnamePass()
    End If
End If
```

Figure 16.1 Code containing verification

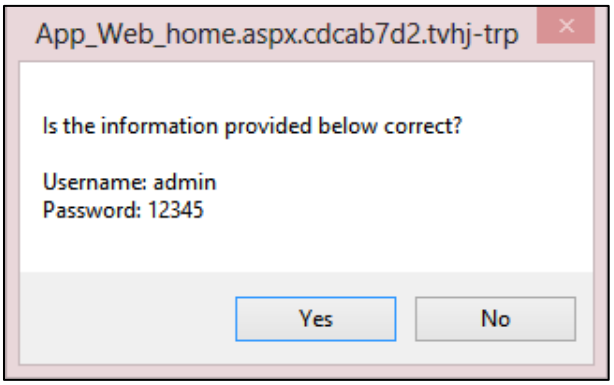

Figure 16.2 Verification window prompting proofreading displayed

17. Efficient sequential search

I used sequential search to search for a product by matching each record to the inputted Product ID. Sequential search was used instead of binary search because the records were not sorted in any order. This is done in an efficient manner as once the record is found, the loop is stopped using Exit While, minimizing the number of iterations to be performed.

```
Dim reader As Data.IDataReader = CType(GridViewDataSource2.Select(DataSourceSelectArguments.Empty), Data.IDataReader)
While reader.Read()
    If reader("Product_ID") = ans Then
        ProductDeleteDropDownList.SelectedValue = reader("Product_ID")
        flag = True
        Exit While
    End If
End While
reader.Close()
```

Figure 17.1 Code containing efficient sequential search

Word Count: 777 words

References

ASP.NET Ajax community & ASP.NET team. (2007). ASP.NET AJAX Control Toolkit (1.0) [Web development tool]. Microsoft. http://www.ajaxtoolkit.net/