Online shopping and stock management system

NEA Project

Matthew Tomlinson

Contents

[**Analysis** 4](#_Toc130503660)

[Problem Identification 4](#_Toc130503661)

[Background to project 4](#_Toc130503662)

[Interview 4](#_Toc130503663)

[Summary 5](#_Toc130503664)

[End user 5](#_Toc130503665)

[Modelling 5](#_Toc130503666)

[Data volumes 5](#_Toc130503667)

[Entity relationship diagram 6](#_Toc130503668)

[Entity attribute model 6](#_Toc130503669)

[Data flow diagram 7](#_Toc130503670)

[Research 8](#_Toc130503671)

[Research Summary 9](#_Toc130503672)

[Limitations 10](#_Toc130503673)

[Objectives 10](#_Toc130503674)

[Optional objectives 11](#_Toc130503675)

[**Design** 12](#_Toc130503676)

[Diagrams 12](#_Toc130503677)

[Entity relationship diagram 12](#_Toc130503678)

[Entity attribute model 12](#_Toc130503679)

[Normalisation 13](#_Toc130503680)

[SQL statements 15](#_Toc130503681)

[Top-down diagram 16](#_Toc130503682)

[IPSO chart 17](#_Toc130503683)

[OOP design 21](#_Toc130503684)

[Data Dictionary 24](#_Toc130503685)

[Interface Design 26](#_Toc130503686)

[Algorithms 36](#_Toc130503687)

[Product Search (customer) 36](#_Toc130503688)

[Get any open order from a customer 37](#_Toc130503689)

[Get products that are low on stock 38](#_Toc130503690)

[Edit a product 39](#_Toc130503691)

[Delete a product 40](#_Toc130503692)

[Add a new product 40](#_Toc130503693)

[**Technical Solution** 42](#_Toc130503694)

[Introduction 42](#_Toc130503695)

[Pointers 42](#_Toc130503696)

[Startup 42](#_Toc130503697)

[Models 43](#_Toc130503698)

[Views 44](#_Toc130503699)

[Login 44](#_Toc130503700)

[Customer Side 46](#_Toc130503701)

[Staff side 53](#_Toc130503702)

[Controllers 60](#_Toc130503703)

[Account controller 61](#_Toc130503704)

[Order controller 63](#_Toc130503705)

[Stock controller 66](#_Toc130503706)

[Database 69](#_Toc130503707)

[Customer repository 69](#_Toc130503708)

[Order repository 70](#_Toc130503709)

[Stock repository 74](#_Toc130503710)

[User repository 79](#_Toc130503711)

[SQL server connection 80](#_Toc130503712)

[SQL stored procedures 80](#_Toc130503713)

[**Testing** 91](#_Toc130503714)

[Test 1 91](#_Toc130503715)

[Test 2 91](#_Toc130503716)

[Test 3 92](#_Toc130503717)

[Test 4 92](#_Toc130503718)

[Test 5 93](#_Toc130503719)

[Test 6 93](#_Toc130503720)

[Test 7 94](#_Toc130503721)

[Test 8 94](#_Toc130503722)

[Test 9 95](#_Toc130503723)

[Test 10 95](#_Toc130503724)

[Database tables 96](#_Toc130503725)

[ApplicationUser 96](#_Toc130503726)

[Customers 96](#_Toc130503727)

[OrderHeader 96](#_Toc130503728)

[OrderLines 96](#_Toc130503729)

[ProductMaster 96](#_Toc130503730)

[**Evaluation** 97](#_Toc130503731)

[Objectives achieved 97](#_Toc130503732)

[Feedback from end user 99](#_Toc130503733)

[Summary 99](#_Toc130503734)

[Improvements 99](#_Toc130503735)

# **Analysis**

## **Problem Identification**

Currently, there is no consistency with online shopping websites, if you compare online shopping websites side by side, they all have varying user interfaces, with loads of moving pictures, bright colours and big images. Everything just seems a bit busy. For some websites you would struggle to find the product search box to search for your desired items. A lot of online shopping websites are filled with advertisements, some customers may not be interested in any of these products being advertised.

When I want to shop online for groceries, I want to be able to enter a website, search for desired products, add them to my basket and finalise my order with no struggles. This is where my project comes in.

## **Background to project**

The aim of my project is to create an online shopping system that makes use of a database.

Complaints about current online shopping websites consist of bad search facilities, lack of personalisation, missing important product information, fake product reviews and the list goes on. All of these points result in an overall bad user experience. This lowers the chance of a customer buying from that website.

Customers want to expect a seamless experience with easy navigation with good search facilities, products with correct information and enough of it.

## **Interview**

Date of interview – 28/07/22

Location of interview – End user’s home

**Q1 : Can you give me information about your experience with online shopping?**

“I do weekly shops for food between 2 to 3 supermarkets, because there isn’t one supermarket that sells every item on my list and different supermarkets sell a different quality of the products I want. My list is roughly 70% of the same things each week. I normally have multiple tabs open at once when I’m shopping, so I can compare prices between supermarkets”

**Q2 : Have you had any issues with online shopping?**

“Some websites offer a better user experience than others. Some prepopulate my shopping basket with items that I never ordered, forcing me to delete them. Search facilities are inadequate and inconsistent”

**Q3 : What ideal experience would you like from the product I create?**

“I would like to be able to quickly place an order, so a simple search facility would be useful, I would like a clean user interface with simple and easy to understand navigation. I would like to easily be able to edit and remove items from my basket and not have any prepopulated items already placed in my basket”

## **Summary**

My end user made it clear that some supermarket websites have better user interfaces/experiences than others, some websites are easier to navigate than others.

Some supermarkets try and populate their basket with items that they didn’t add, causing a decline in the level of trust between customers and the supermarket. This is also another factor of the lacking user experience that has been previously mentioned.

Search facilities are also inadequate and inconsistent.

By the time my project is finished, it would be perfectly suited towards my end user, because there will be a clean user interface with simple, easy to understand navigation, with easy control over their shopping basket.

## **End user**

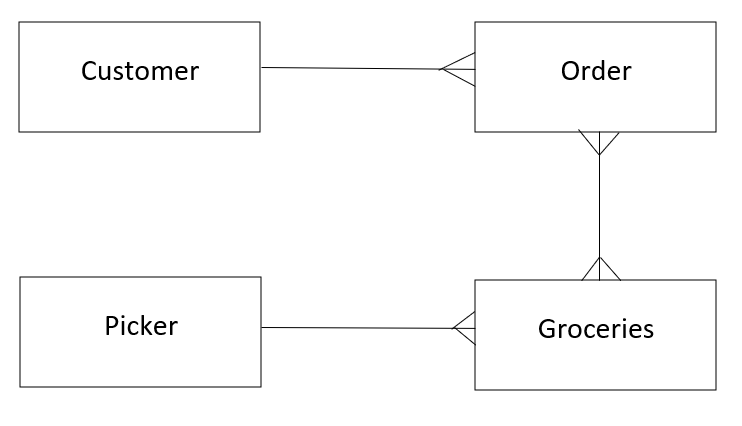
My end user is a regular online shopper. They shop between 2/3 supermarkets. Certain products are put on every order, and some new products. They like to compare prices of the same products from alternate supermarkets by having multiple tabs open displaying the products to see which supermarkets give a better value for money.

## **Modelling**

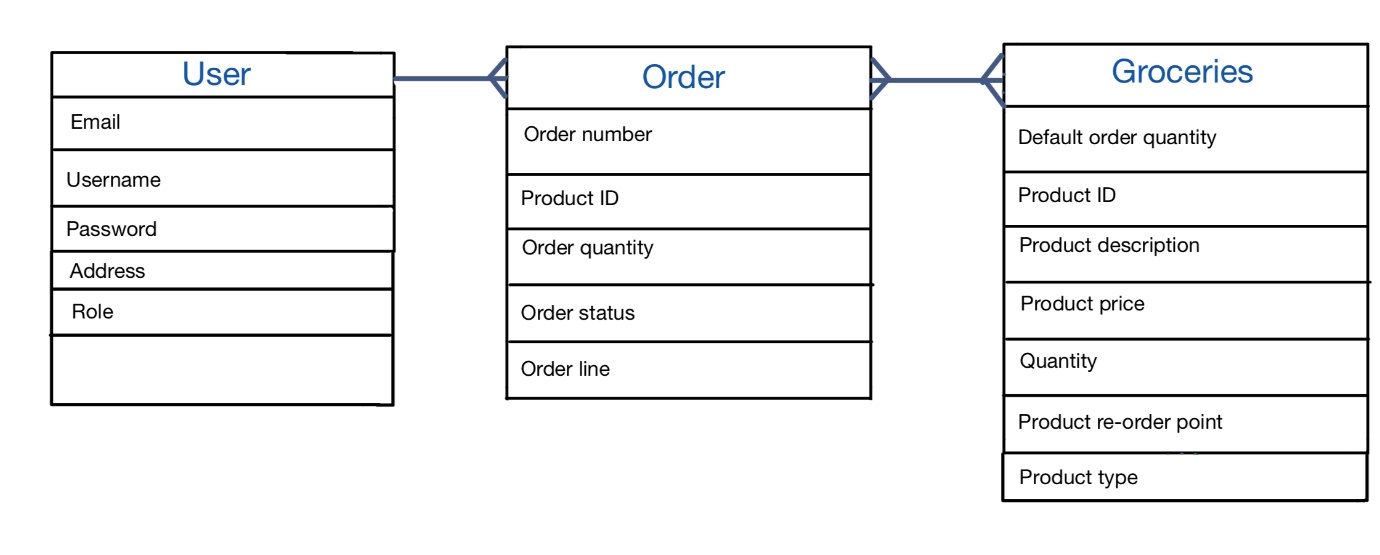
### Data volumes

|  |  |
| --- | --- |
| Data object | Volume of data |
| New customers | Average of 20 per year |
| Number of items in an order | Average of 4 items per order |
| Number of restock orders | Average of 100 restock orders per year |
| Number of customer order | Average of 120 customer orders per year |

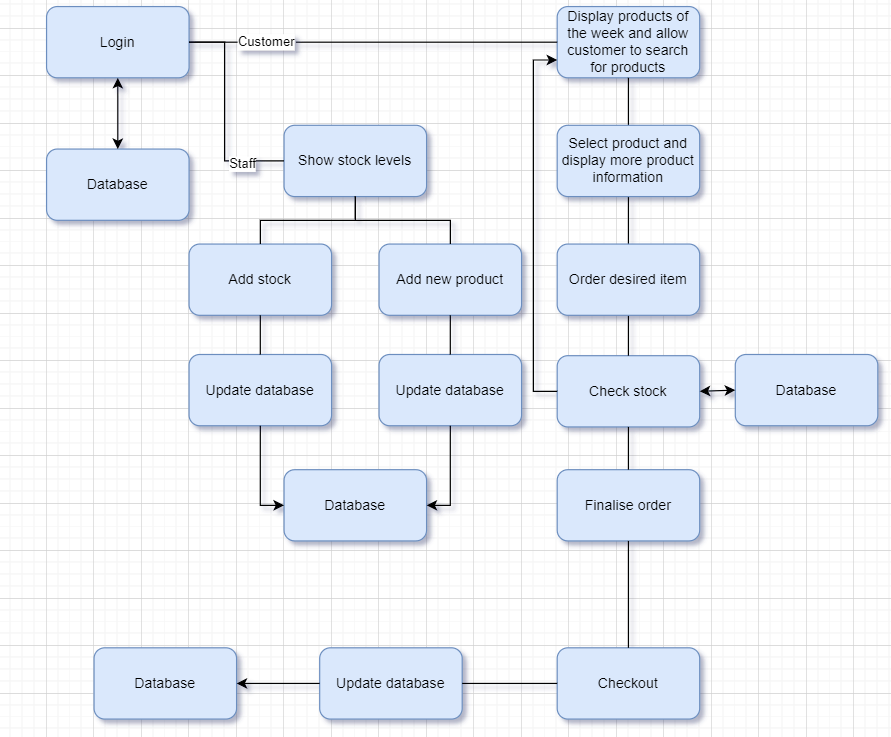
### Entity relationship diagram



### Entity attribute model

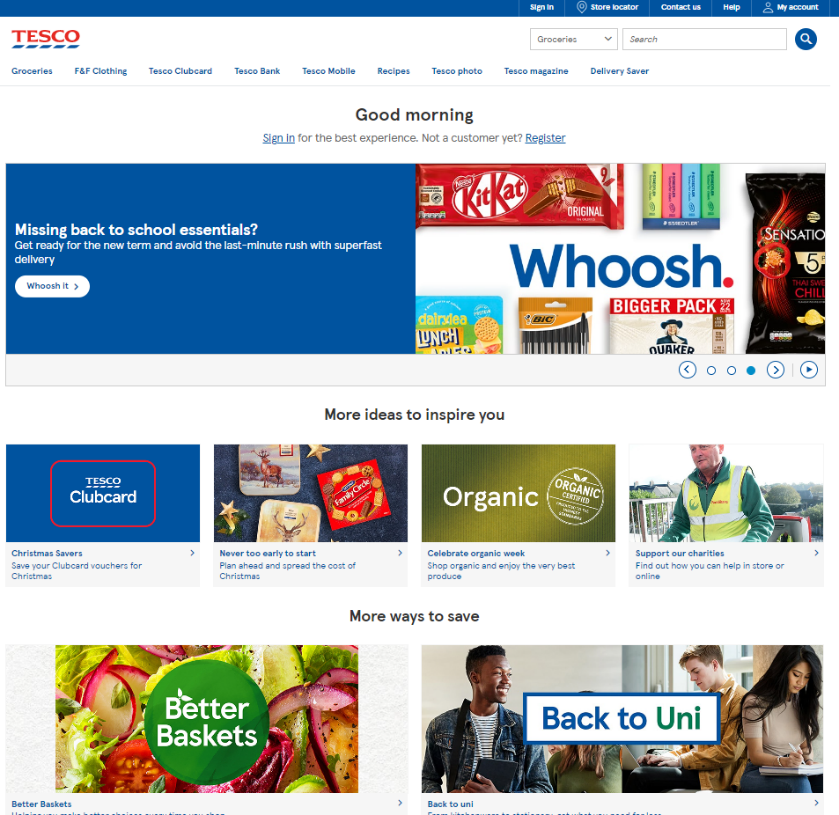


### Data flow diagram

****

## 

## **Research**



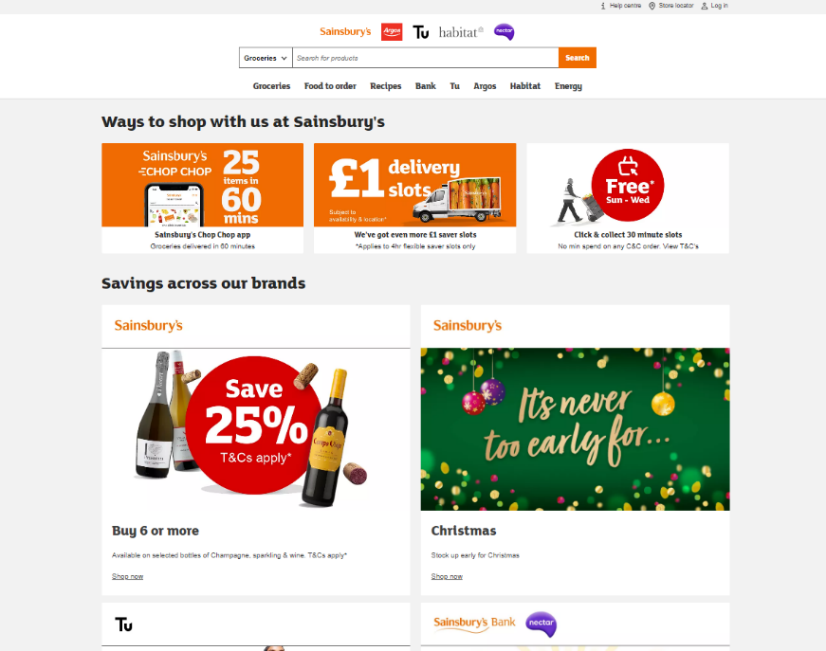
This website has a simple design with a primary colour used throughout.

The search box has a drop-down box next to it to filter results, making it easier for customers to find what they are looking for.

The website is kept up to date with current affairs like going back to school or university and offering product related to this.

There is lots of information displayed on this first page which could be seen as busy and hard to navigate.

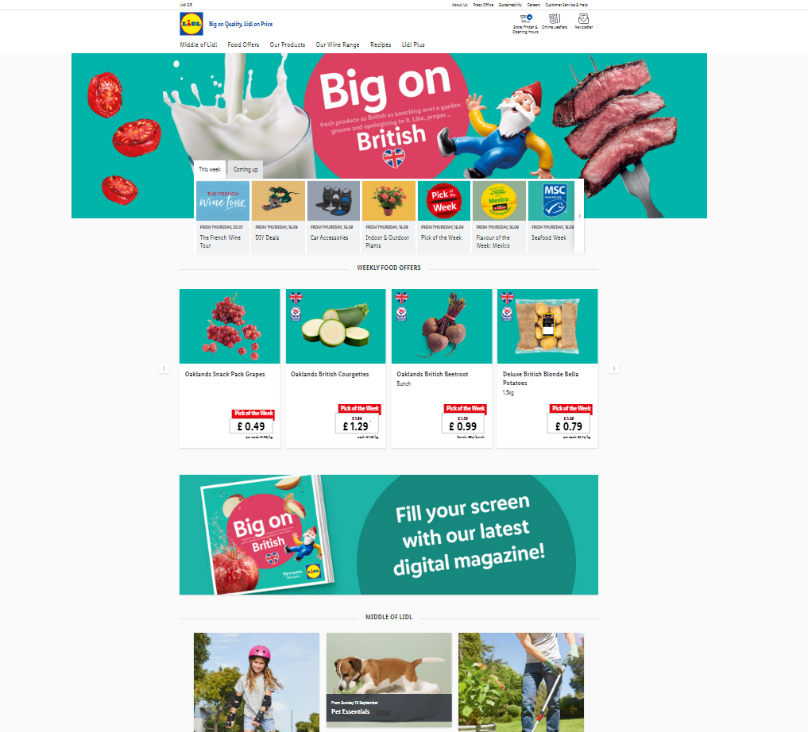
There is also a welcome header that I would like to implement onto my website, making the customer feel welcome.

****

This website is similar to the previous in which there is a common theme, there is a filter next to the search bar to make it easier for customers to find what they are looking for.

Lots of useful information is displayed on the first page like deals and upcoming events like Christmas.

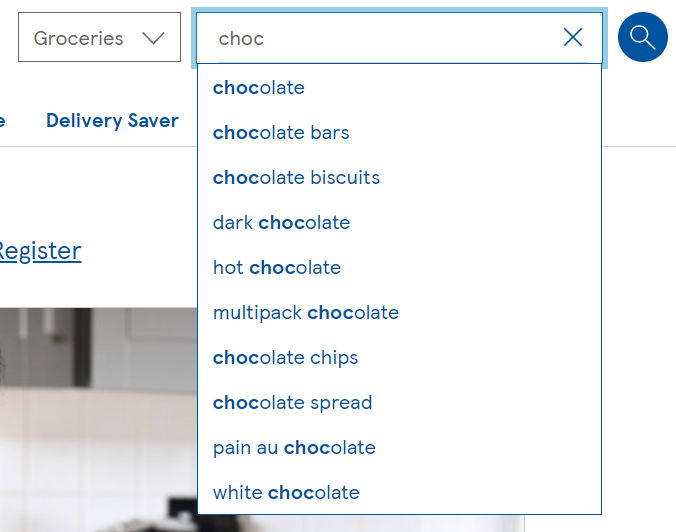
I would say that this website is slightly easier to look at than the previous

****

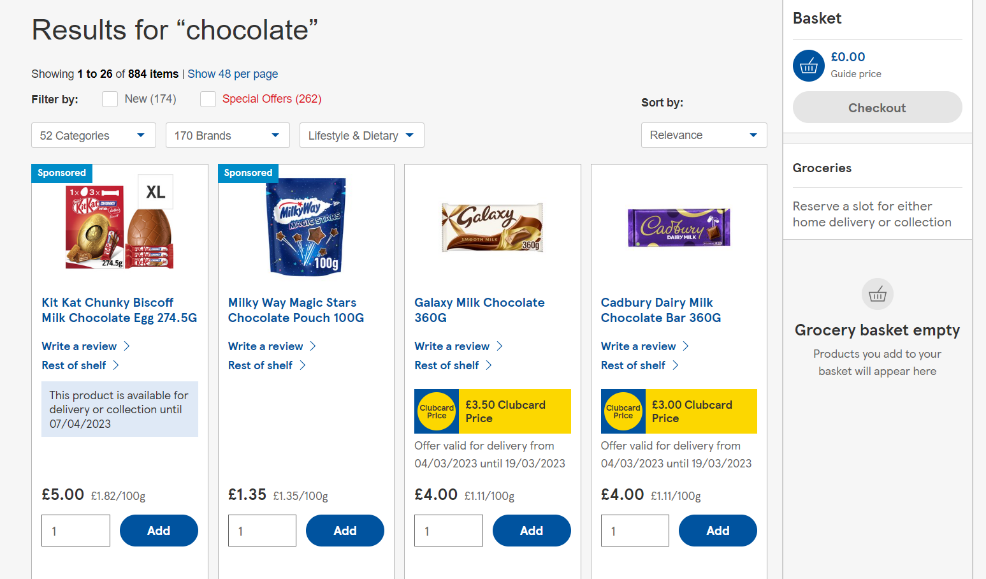
This website has a feature that I wish to implement onto my website, this feature consists of displaying products of the week.

Like both of the previous websites, there is a consistent theme and primary colour. However, there isn’t any obvious search box this makes it harder for customers to navigate.

This website has a simpler design than the both the previous with colourful pictures and a clean interface, making it easier for customers to navigate.

****

Search bar has autocomplete features that helps the user navigate to their desired products. It takes the input in the search box and relates it to products that have a name similar to this input. This is really useful because it makes navigating to products a lot simpler and quicker.

****

The search results page for this website is very busy, there is lots of buttons, lots of text. This makes the page harder to navigate. However, all the products have relevant and useful information displayed. Also, the shopping basket is always displayed to the side of the page, making it easier and quicker for customers to edit their orders.

## **Research Summary**

All of the existing systems have similar pros and cons. They all have a friendly, welcoming interface, however it can sometimes get too busy due to lots of different buttons and shapes displayed, this makes the website’s interfaces misleading and confusing towards the customer trying to navigate. All of the websites share great features like advanced autocomplete search boxes, making it super easy for customer to find their desired products. However, on some of these websites, the search box can be quite challenging to find. I wish to create a website that shares the same vibrant, welcoming user interface, whilst still managing to maintain simple navigation.

## **Limitations**

There is going to be some limitations that I need to be aware of.

* I will be creating my website using MVC ASP.Net Core in Visual Studio which I am completely new to so I have allowed time to learn this way of coding
* There will be no method to pay on my website, just the ability to select order and possibly book a delivery.

## **Objectives**

1. **Create an online shopping website**
   1. User-friendly interface
      1. single theme
      2. easily identifiable buttons to give more information on products
      3. simple/clean design
   2. Log in capabilities
      1. login if customer already has an account
      2. ability to create a new customer account
      3. ability to log in as a customer or a staff member, giving access to different features
   3. Customer accounts
      1. a special feature such as: products of the week
      2. pictures of products
      3. the ability to search for products using a search box
      4. ability to add products to a list and click on a finalise button to complete order
      5. ability to edit/delete any outstanding orders
   4. Staff accounts
      1. can view products that are currently low on stock
      2. ability to search for products using a search box
      3. ability to increase stock levels of a certain product
      4. ability to add a new product
      5. ability to edit current products
      6. ability to delete products
2. **Create database**
   1. add password protection
   2. create tables containing all data such as accounts and orders
   3. create procedures to maintain and list data

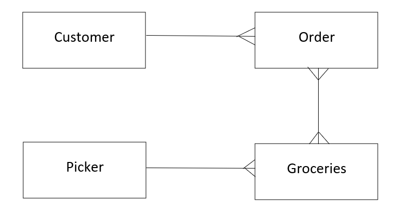
## **Optional objectives**

1. **Create an android app to display orders to staff**
   1. add login capabilities with device ID validation
   2. ability to enter an order number
   3. a list of products corresponding to the order number entered
   4. ability to add picked product quantity from order
   5. update database accordingly
   6. update list of products corresponding to order number but excluding products already entered as picked
2. **add personalisation feature to website which allows customer to pick the primary colour of the website**
3. **add delivery selection capability to website**
4. **add delivery transaction to the android app**

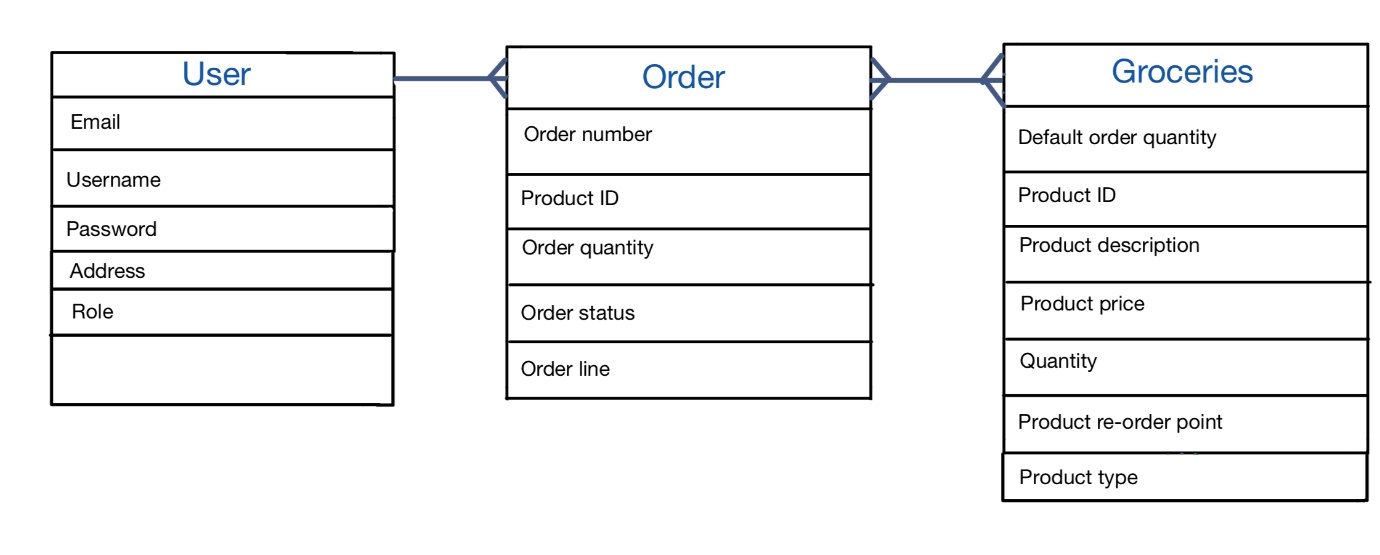
# **Design**

## **Diagrams**

### Entity relationship diagram

****

### Entity attribute model



## **Normalisation**

ApplicationUser (Email, Username, Password, Role)

OrderLines (OrderNumber, OrderLine, OrderQuantity, OrderStatus, ProductID, ProductCode, ProductDescription, ProductPrice)

ProductMaster (ProductID, StockQuantity, ProductDescription, ReOrderQuantity, ProductPrice, DefaultOrderQuantity, ProductType, ImageName)

Customers (CustomerID, Username, Password,Email, FirstName, Surname, Address1, Address2, Address3, Address4, Postcode, Phone)

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | ApplicationUser | | |
| Primary Key | Email | | |
| Foreign Keys | N/A | | |
| Data Item | **Data Type** | **Validation** | **Sample Data** |
| Email | String | Not Null, contain ‘@’ symbol | sam@bob.com |
| Username | String | Not Null, 6 – 100 characters long | Shopper123 |
| Password | String | Not Null, 6 – 100 characters long | Password1 |
| Role | String | Not Null | User |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | OrderLines | | |
| Primary Key | OrderNumber | | |
| Foreign Keys | N/A | | |
| Data Item | **Data Type** | **Validation** | **Sample Data** |
| OrderNumber | Integer | Not Null | 123 |
| OrderLine | Integer | Not Null | 1 |
| OrderQuantity | Integer | Not Null | 3 |
| OrderStatus | Integer | Not Null | 1 |
| ProductID | String | N/A | 1 |
| ProductCode | String | Not Null | HZB200g |
| ProductDescription | String | Not Null | Heinz Beans 200g |
| ProductPrice | Float | Not Null | 1.45 |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | ProductMaster | | |
| Primary Key | ID | | |
| Foreign Keys | N/A | | |
| Data Item | **Data Type** | **Validation** | **Sample Data** |
| ProductID | Integer | N/A | 1 |
| ProductCode | String | Not Null | HZB200g |
| StockQuantity | Integer | Not Null | 1000 |
| ProductDescription | String | Not Null | Heinz Baked Beans 200g |
| ReOrderQuantity | Integer | Not Null | 1500 |
| ProductPrice | Real | Not Null | £3.65 |
| DefaultOrderQuantity | Integer | Not Null | 5 |
| ProductType | String | Not Null | Chocolate |
| ImageName | String | Not Null | cheese.png |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Customers | | |
| Primary Key | Username | | |
| Foreign Keys | N/A | | |
| Data Item | **Data Type** | **Validation** | **Sample Data** |
| CustomerID | Integer | N/A | 1 |
| Username | String | Not Null, 6 – 100 characters long | LoveShopping123 |
| Password | String | Not Null, 6 – 100 characters long | Password1234 |
| Email | String | Not Null, contain ‘@’ symbol | sam@bob.com |
| FirstName | String | Not Null | Matthew |
| Surname | String | Not Null | Tomlinson |
| Address1 | String | N/A | 9 Bridge Court |
| Address2 | String | N/A | Clitheroe |
| Address3 | String | N/A | Lancashire |
| PostCode | String | N/A | BB3 2TG |
| Phone | Integer | N/A | 07355 23789 |

## **SQL statements**

**A query to extract all items part of an unfinalized order.**

GetOpenOrderByCustomer

Called to display screen 4

**A query to extract all products that have a lower stock level than their re-order pint**

GetLowStock

Called to display screen 6

**A query to extract products that are related to what to customer searched for on both customer homepage and staff homepage**

ProductTypeSearch

Called on screen 4 when the search box is used

ProductDescriptionSearch

Called on screen 7 when the search box is used

**A query to delete a product from the product table or delete an order from the order table**

DeleteProductById

DeleteOrder

Both called when a delete button is clicked

**A query to add order lines to unfinalized orders**

InsertOrderLines

Called when order button is clicked on screen 5

**A query to increase low stock items by default order quantity**

IncreaseLowStock

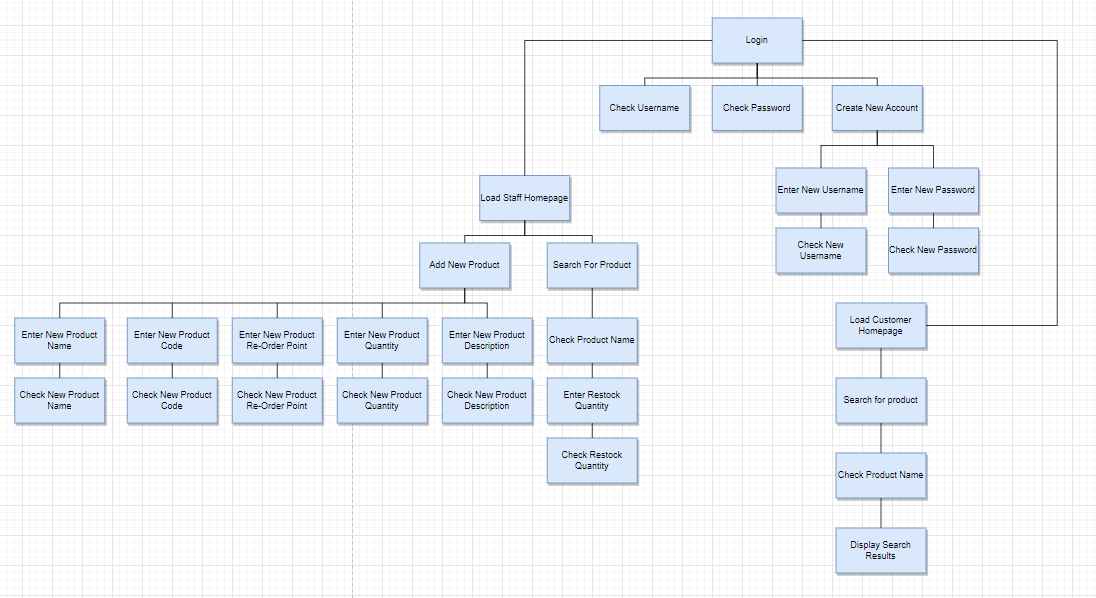
Called when order buttons on screen 6 are clicked

**A query to mark a finalized order as ‘ordered’**

MarkOrderAsOrdered

Called when the finalise button is clicked on screen 4

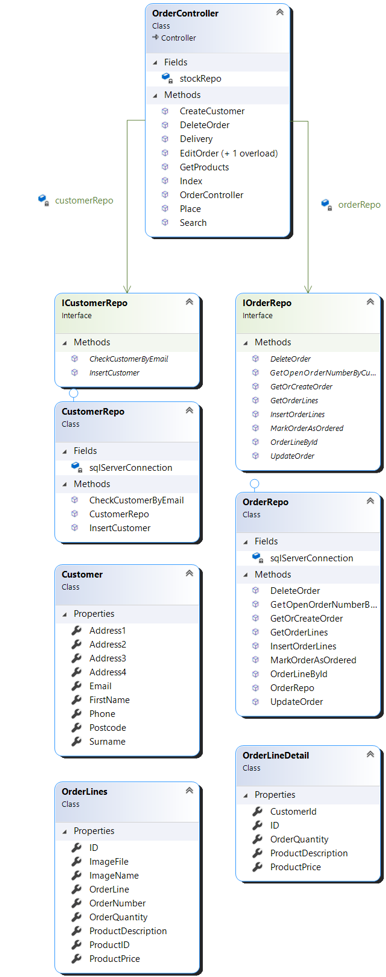
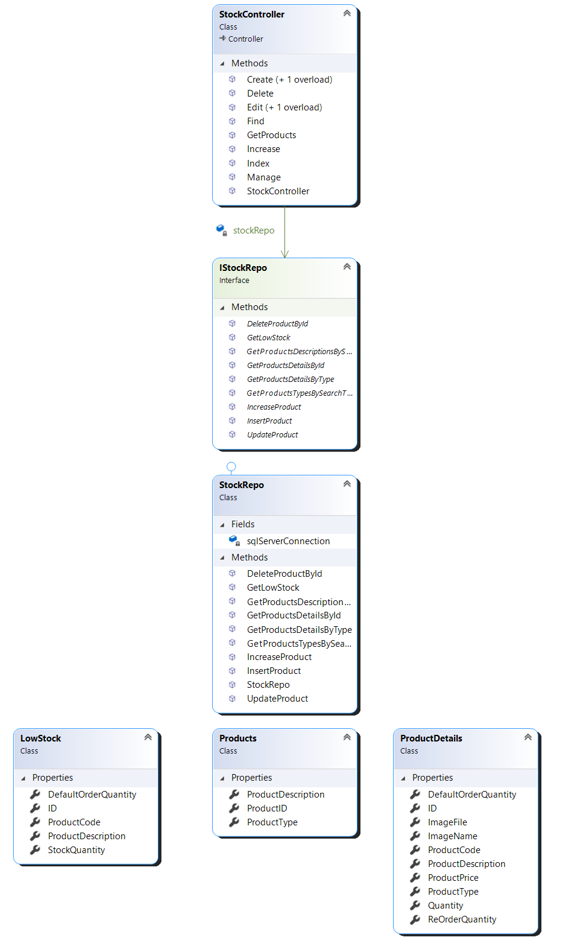
## **Top-down diagram**

****

## **IPSO chart**

|  |  |  |
| --- | --- | --- |
| **IPSO** | **Program Section** | **Item** |
| **Input** | Search For Product (Customer) | ProductType |
| Search For Product (Staff) | ProductDescription |
| Adding New Product | ProductID  ProductDescription  ReOrderQuantity  Quantity  ProductPrice  DefaultOrderQuantity  ProductType  ImageName |
| Editing a Product | ProductDescription  ReOrderQuantity  ProductPrice  DefaultOrderQuantity  Quantity  ImageName |
| Editing an order | OrderQuantity |
| Login | Email  Password |
| Create New Account | Username  Email  Password |
| Customer Details | FirstName  Surname  Address1,2,3,4  Postcode  Phone |
| **Processing** | Check Username | Does it match any login details in the database? Identify if it’s a customer or staff login |
| Check Password | Does it match any login details in the database? Identify if it’s a customer or staff login |
| Check New Username | Does it follow validation rules? Do these details already exist in the database? |
| Check New Password | Does it follow validation rules? Do these details already exist in the database? |
| Check Product Type | Has the field been populated? |
| Check Product Code | Has the field been populated? |
| Check Product Description | Has the field been populated? |
| Check Product Re-Order Point | Has the field been populated? |
| Check Product Quantity | Has the field been populated? |
| Check Product Price | Has the field been populated? |
| Check Product Default Order Quantity | Has the field been populated? |
| Check Image Name | Has the field been populated? |
| **Storage** | Save New Login Details | Saves details entered in screen 2 |
| Save New Product Details | Saves details entered in screen 9 |
| Save Customer Details | Save details entered in screen 3 |
| Save changes made to product | Save details entered on screen 9 |
| Save changes made to order | Save details entered on edit order screen |
| **Output** | Check Username | Message “invalid username” |
| Check Password | Message “invalid password” |
| Check New Username | Message “invalid username, username must be between 6 and 100 characters long” |
| Check New Password | Message “invalid password, password must be between 6 and 100 characters long” |
| Display Customer Homepage | Display screen 4 |
| Display Staff Homepage | Display screen 6 |
| Check Product Type | Message “this field is required” |
| Check Product Name | Message “this field is required” |
| Check Product Description | Message “this field is required” |
| Check Product Re-Order Point | Message “this field is required” |
| Check Product Quantity | Message “this field is required” |
| Check Product Price | Message “this field is required” |
| Check Product Default Order Quantity | Message “this field is required” |
| Check Image Name | Message “this field is required” |

## **OOP design**

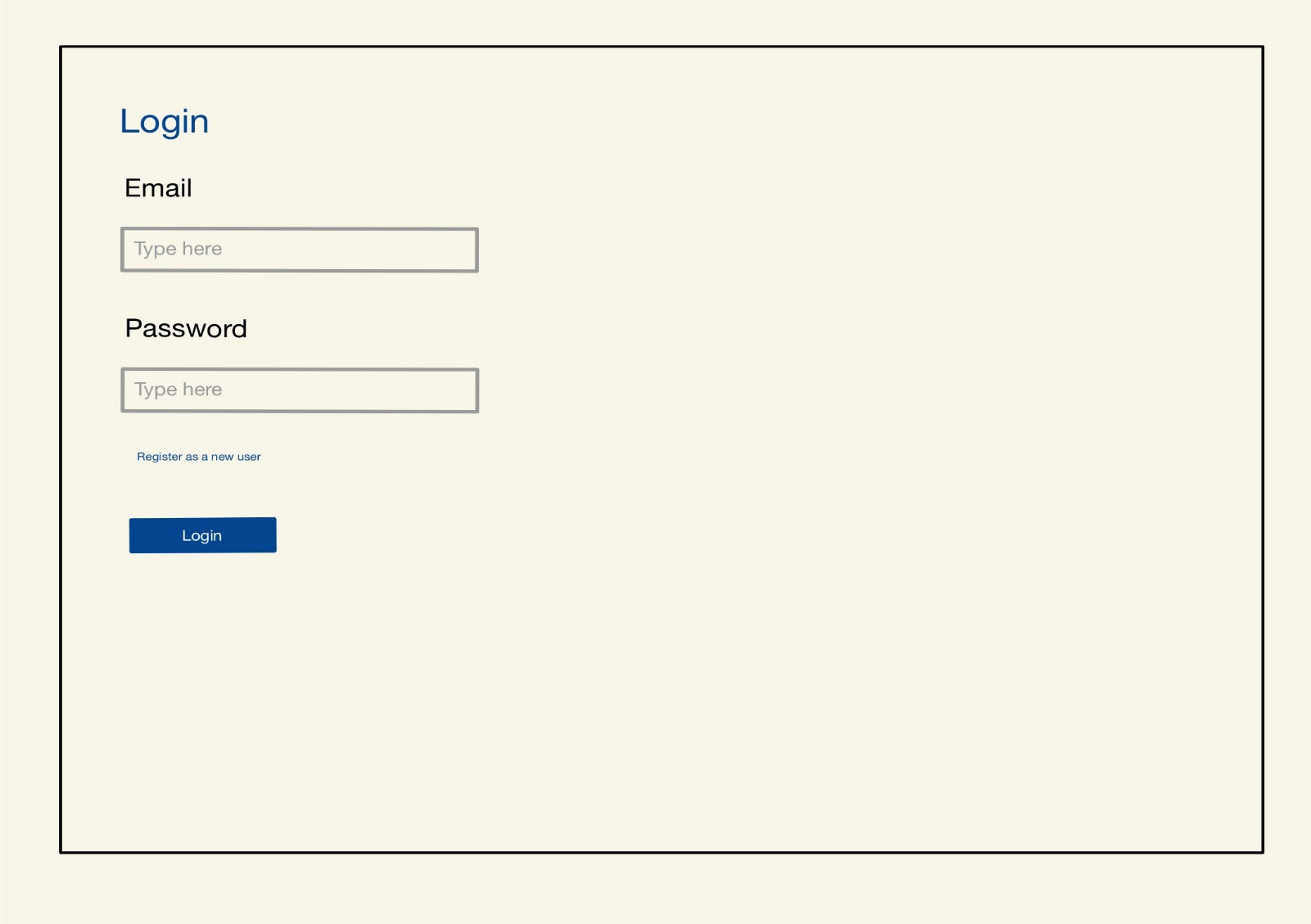


## **Data Dictionary**

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data Type | Validation | Sample Data |
| Username | String | Not Null, 6 – 100 characters | Shopping123 |
| Password | String | Not Null, 6 – 100 characters | Password1 |
| Email | String | Not Null, contain ‘@’ symbol | Shop@yay.com |
| Role | String | N/A | User |
| FirstName | String | N/A | Matthew |
| Surname | String | N/A | Tomlinson |
| CustomerID | String | N/A | 2 |
| ProductID | String | N/A | 3 |
| ProductCode | String | Not Null | HZB200g |
| DefaultOrderQuantity | Integer | Not Null | 1000 |
| StockQuantity | Integer | Not Null | 1500 |
| ReOrderQuantity | Integer | Not Null | 500 |
| ProductDescription | String | Not Null | Heinz Baked Beans 200g |
| ProductPrice | Float | Not Null | 1.40 |
| ImageName | String | Not Null | cheese.png |
| OrderNumber | String | Not Null | 456 |
| OrderLine | Integer | Not Null | 2 |
| OrderQuantity | Integer | Not Null | 3 |
| Address1 | String | N/A | 11 BrightField Drive |
| Address2 | String | N/A | Clitheroe |
| Address3 | String | N/A | Lancashire |
| Address4 | String | N/A | England |
| Postcode | String | N/A | BB9 6TF |
| Phone | Integer | N/A | 07890 45576 |

## **Interface Design**

**Screen 1**



**1**

**2**

**3**

**4**

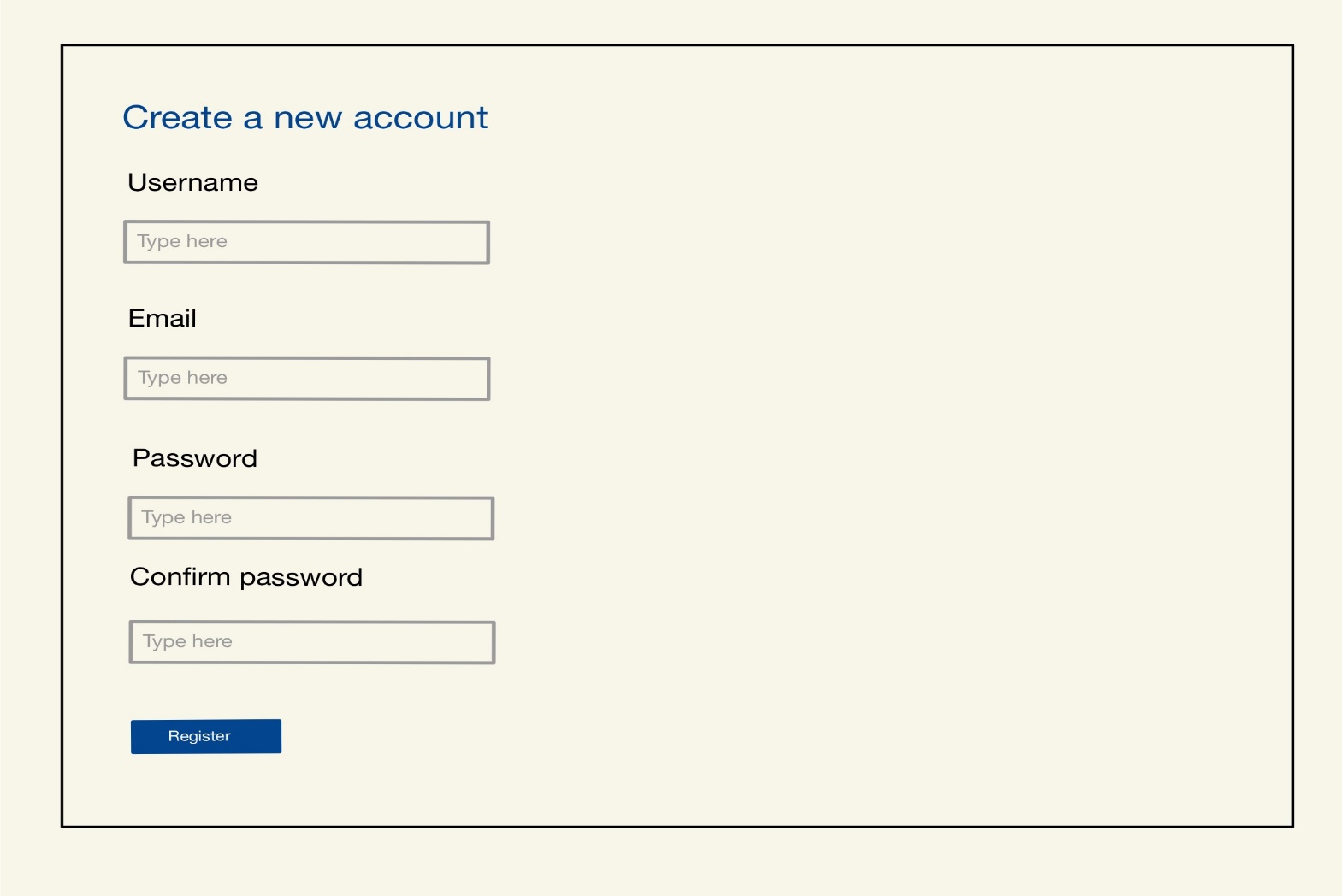
1 – the user will enter the email their account is under in this text box, it is validated and will display an error message if the details are incorrect

2 – the user will enter the password to their account, this will be validated and display an error message if the details are incorrect

3 – this ‘register as a new user’ button will send the user to screen 2

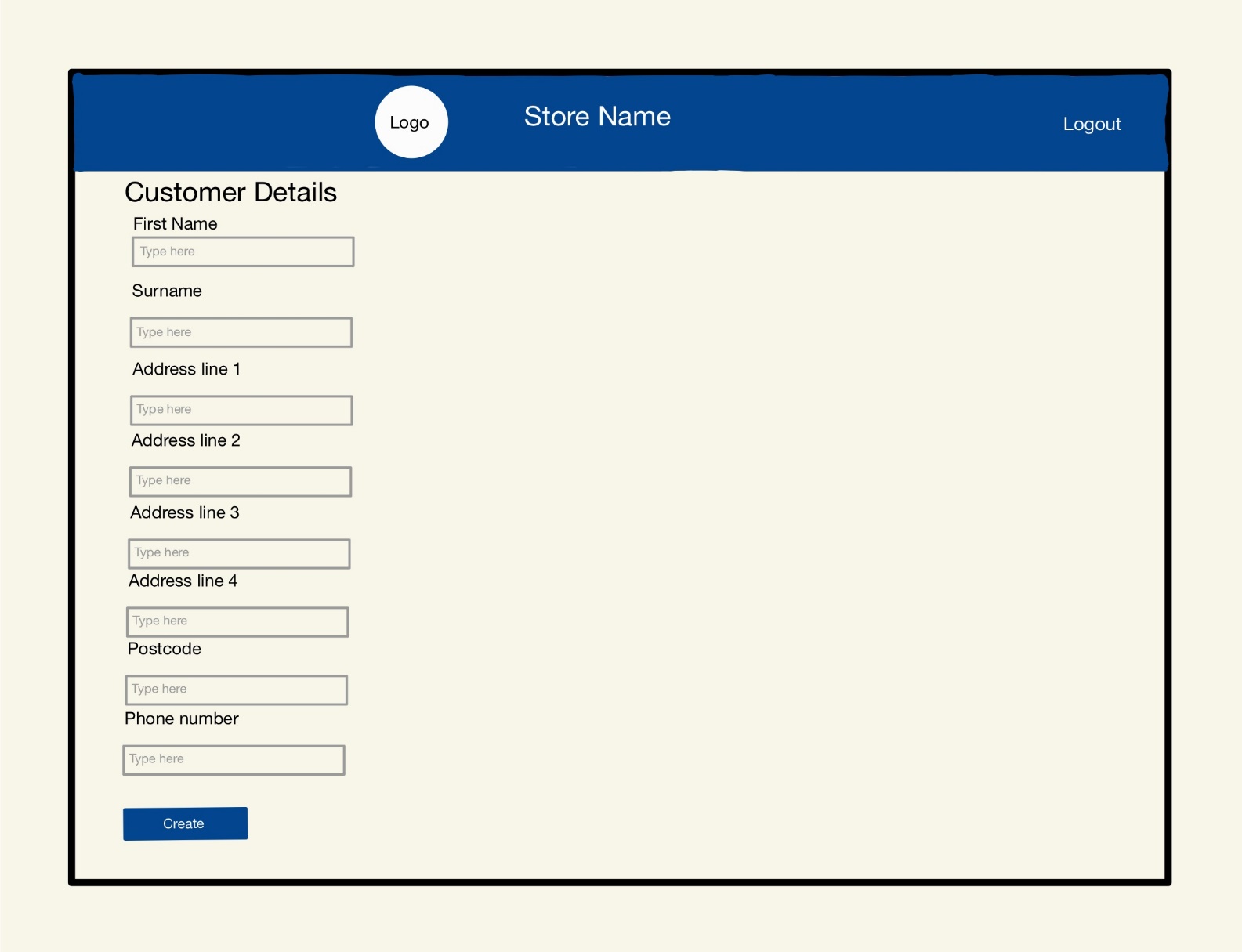
4 – this button will be clicked once the user has filled in the boxes, there will be an error message displayed if the boxes are left empty, or the details are incorrect

**Screen 2**



The user will fill in all the input boxes with the details of their new account, all of these input boxes are validated to ensure they are all filled and follow requirements.

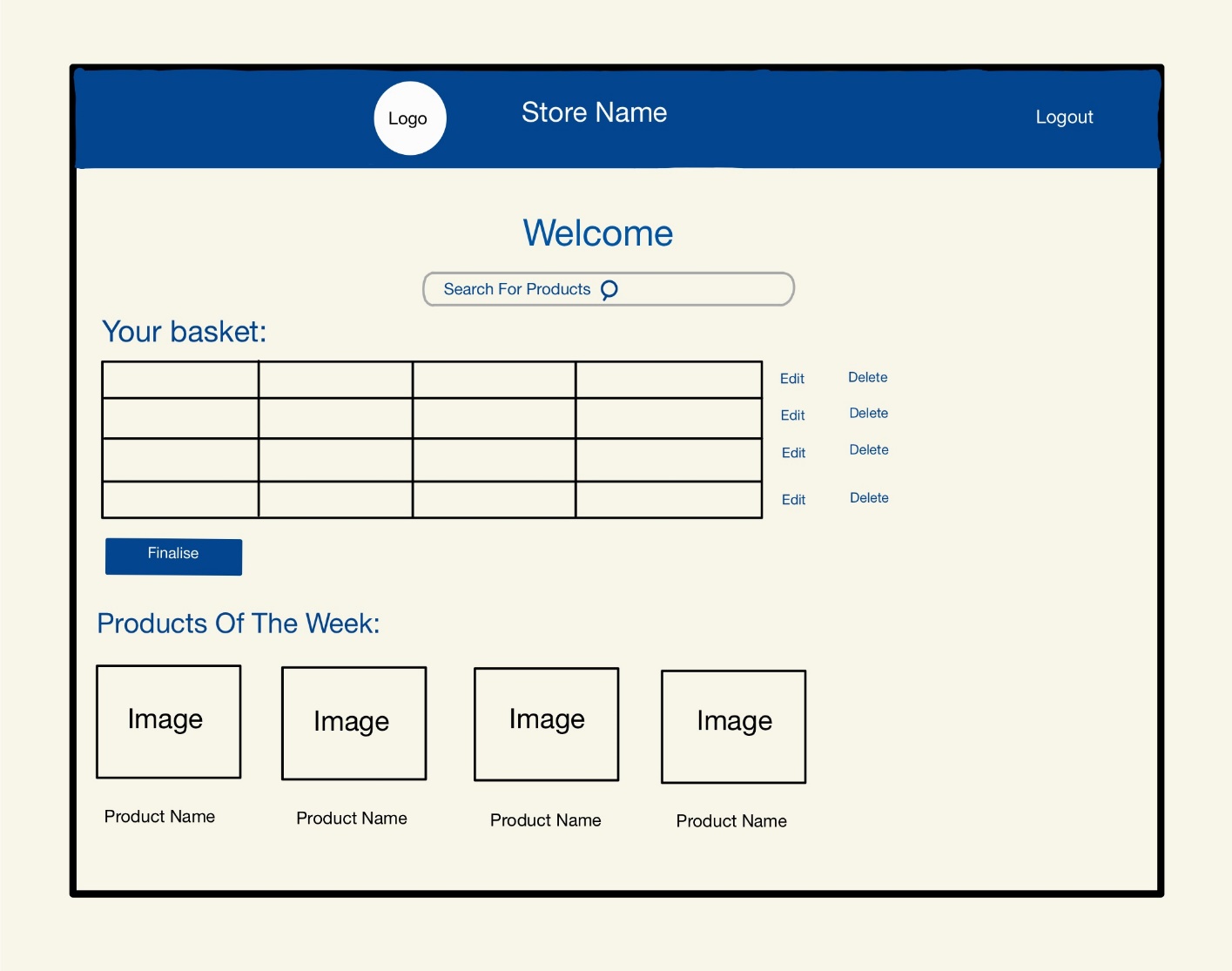
Once all boxes are filled, and the details are validated, the details will be stored in the ApplicationUser table in my database.

**Screen 3**

This page is displayed after the first login from the creation of a new account, the customer will provide their name, address and phone number.

Once the boxes have been filled and validated, the details will be saved to the ApplicationUser table in my database.

**Screen 4**



**4** **1**

**2**

**6**

**5**

**3**

1 – the customer will enter the product they want to search for in the search bar, this will have autocomplete and auto search features that will appear underneath the search bar as the customer types more. Once the user selects a product from the drop down box, screen 5 will be displayed

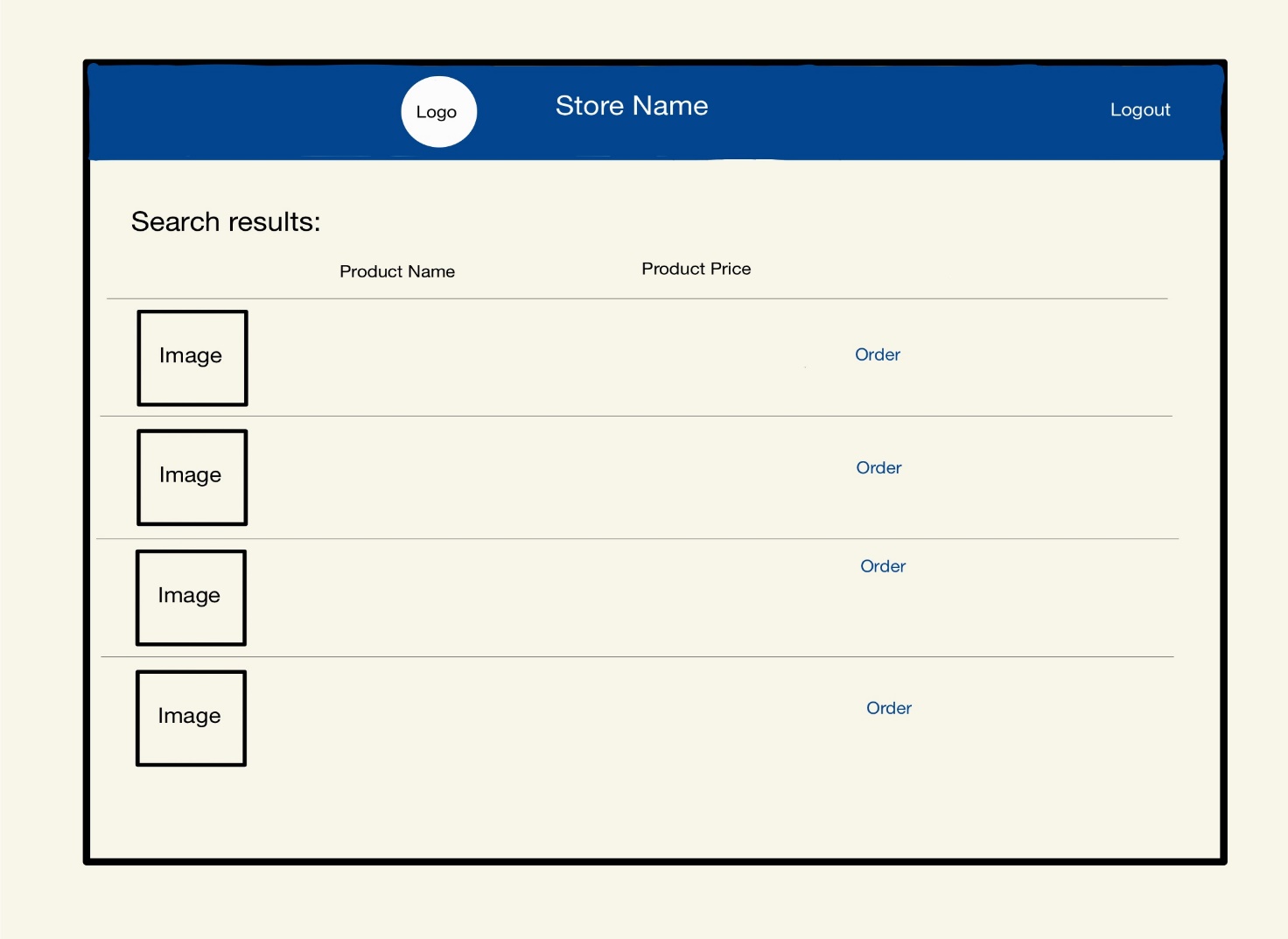
2 – this table will display the items of any order that has not yet been finalised. If the edit button is clicked, a screen similar to screen 8 will be displayed. If the delete button is clicked, the item will be deleted from that order

3 – this gives the ability to advertise products to customers, these products will be random and will show the product image and the product name.

4 – if clicked, screen 1 will be displayed

5 – if clicked, a screen similar to screen 9 will be displayed

6 – if clicked, the item next to the delete button will be removed from the table on the screen and the database



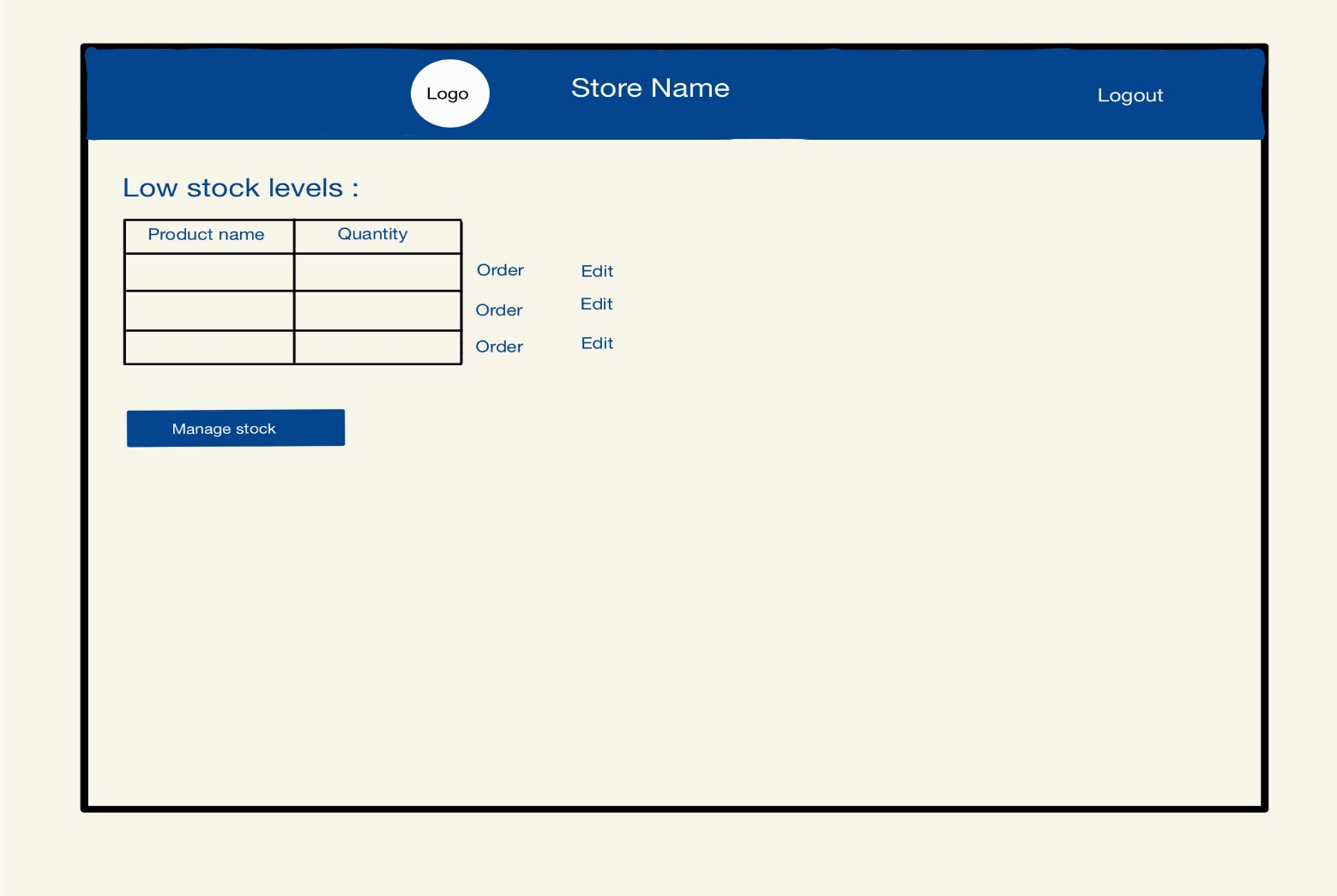
**Screen 5**

This page appears after the customer types a product into the search bar on screen 4.

It displays images of the products, their names and their prices.

1 – if this button is clicked, the customer will be prompted to enter the quantity they wish to order of this product and it will be added to the table on screen 4.

**1**



**Screen 6**

**3**

**1 2**

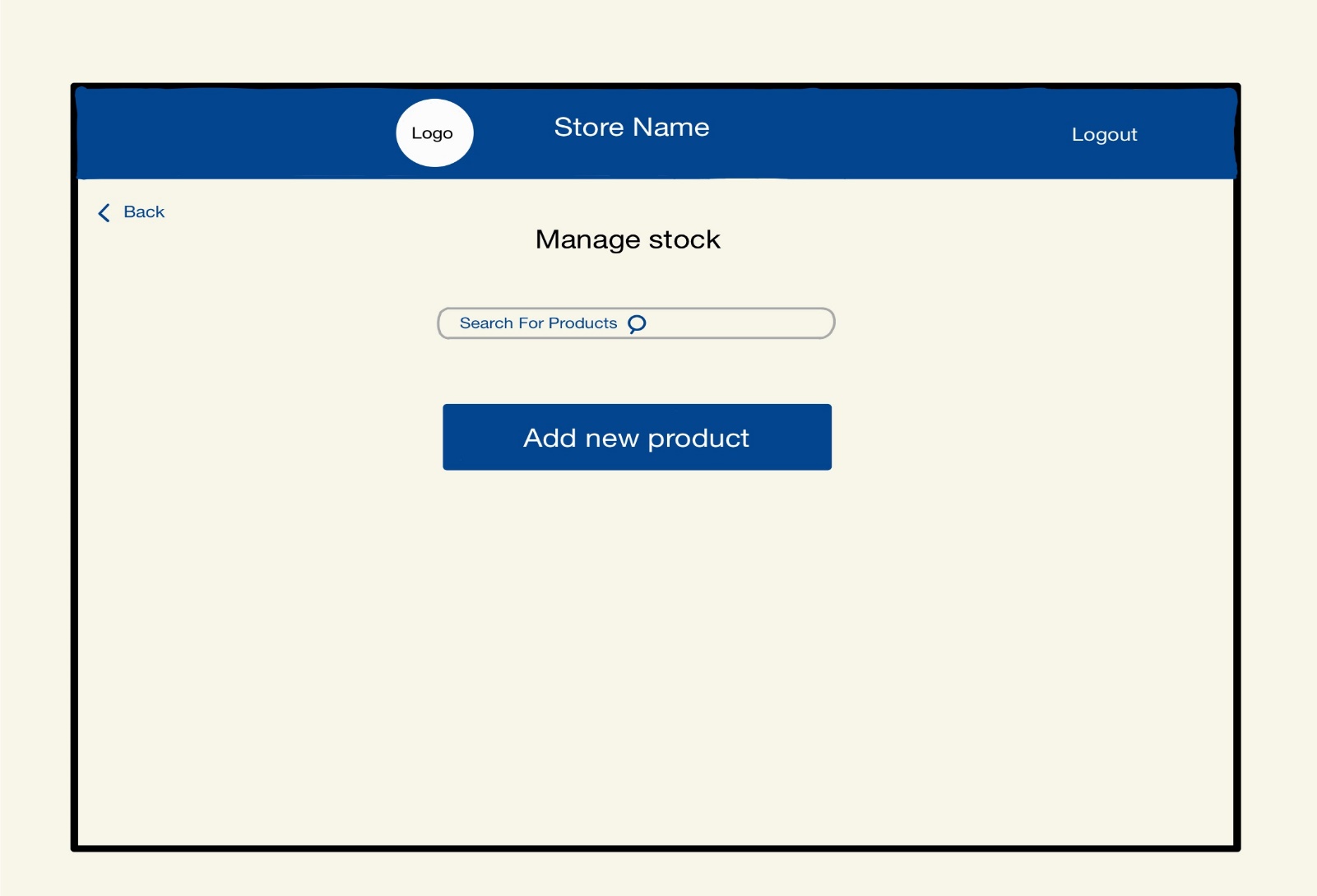
**4**

1 - this table will display products that have a lower stock quantity than their re-order point, the table will have two columns which are titled ‘product name’ and ‘quantity’

2 – when the staff clicks this button the staff will be taken to screen 9

3 – if the order button is clicked, the quantity of the item will be increased by its default order quantity

4 – when the staff clicks this button, they will be taken to screen 7



**Screen 7**

**3**

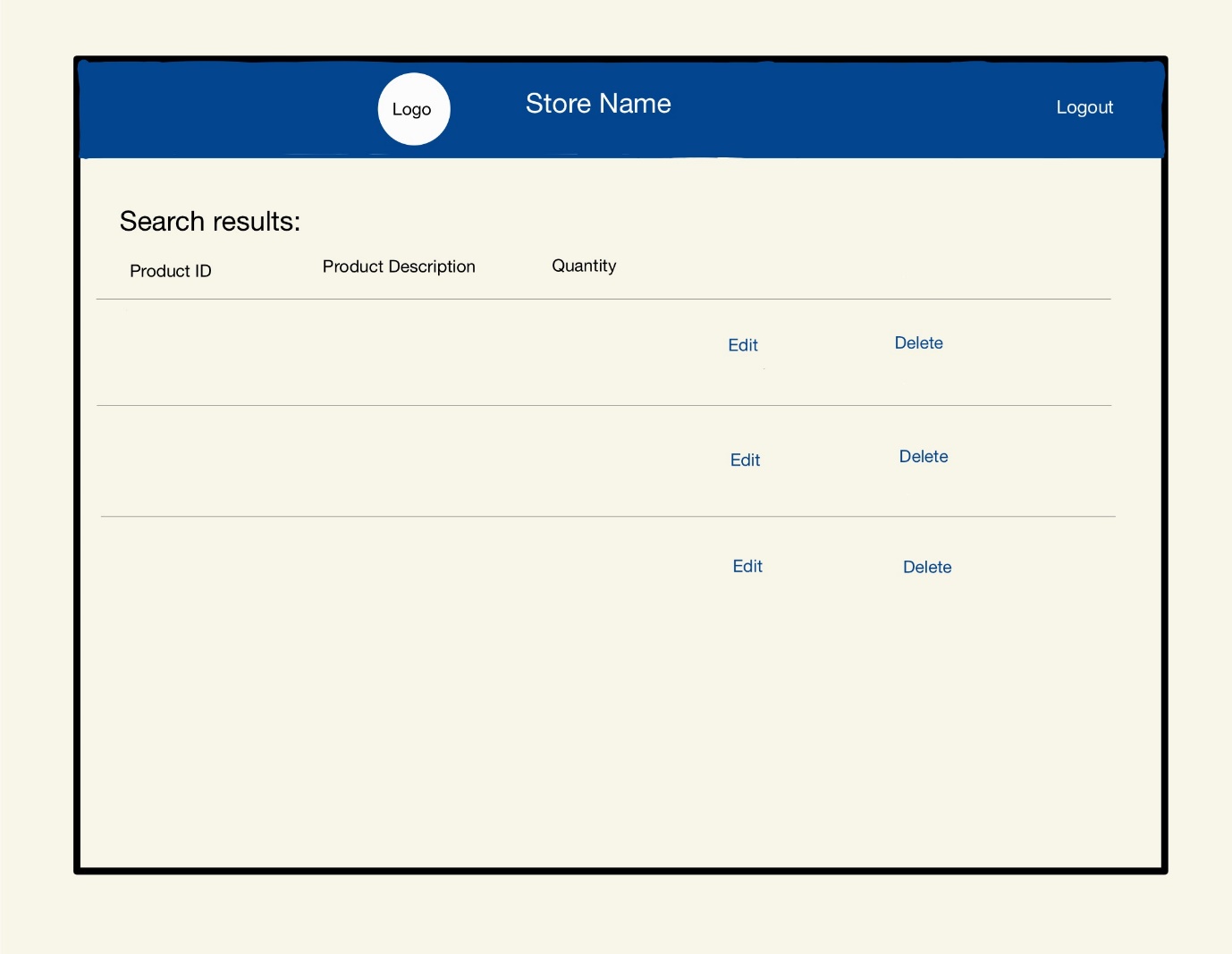
**1**

**2**

1 – the staff will enter the product they want to search for in the search bar, this will have autocomplete and auto search features that will appear underneath the search bar as the customer types more. Once the user selects a product from the drop-down box, screen 8 will be displayed

2 – if clicked, screen 9 will be displayed

3 – if clicked, the staff will be taken back to screen 6



**Screen 8**

**2**

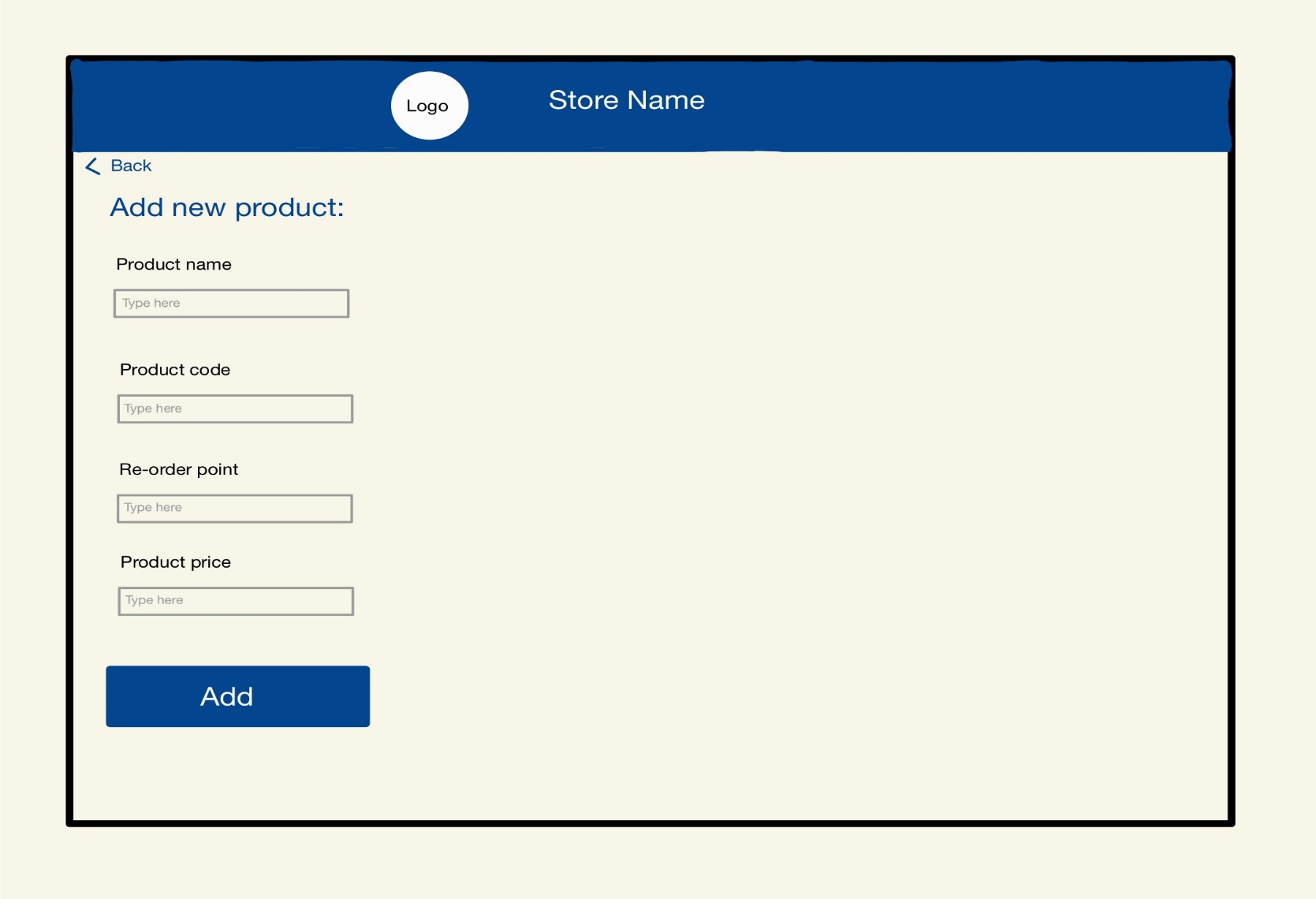
**1**

This page appears after staff types a product into the search bar on screen 7.

It displays images of the products, their names, and their stock levels

1 – if this button is clicked, screen 10 will be displayed

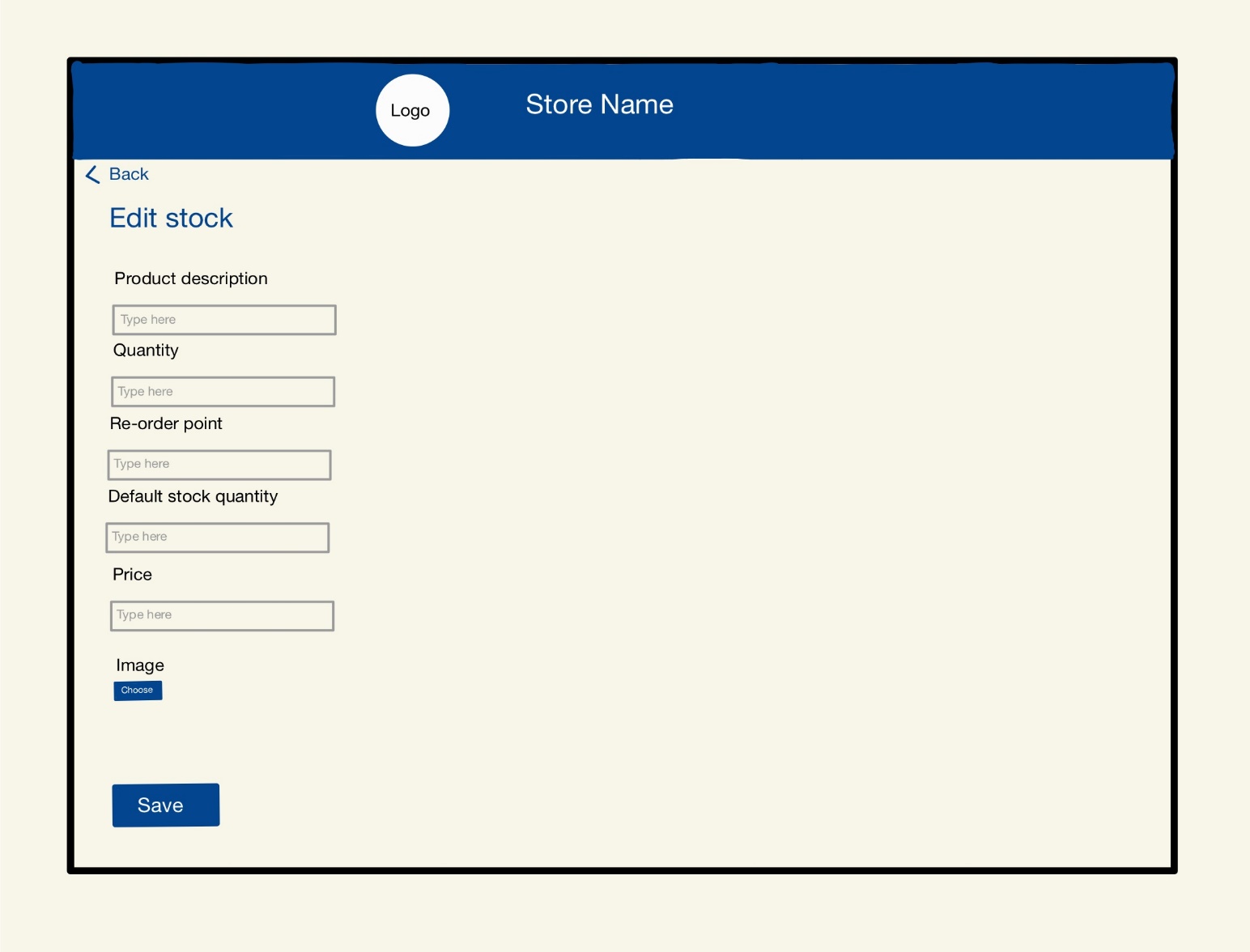
2 – if this button is clicked, the product will be removed from the product table in the database



**Screen 9**

Staff will populate all these boxes with details of a new product they wish to add to the system, all boxes are validated to ensure none are left blank and that they meet validation requirements for the specific box.

Once the add button has been clicked, the new item, along with its details will be added to the ProductMaster table in my database, providing all boxes have been validated.



**Screen 10**

Staff will fill in any of the boxes on this page they wish to edit, all of these boxes are pre populated with the product’s current details and will be validated to ensure no boxes are left blank and any edited boxes meet validation requirements.

Once the save button is clicked, any changed details will be saved to the ProductMaster table in my database, providing edited boxes have been validated.

## **Algorithms**

### Product Search (customer)

Function GetProductsTypesBySearchTerm(searchTerm)

//this function finds all products related to the ProductType that was entered

products = empty List of Products

conn = sqlServerConnection.GetConnection //establishing a connection to the database

conn.Open()

//create SQL command object

cmd = new SqlCommand("ProductTypeSearch", conn) // calling ProductTypeSearch stored procedure

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@SearchTerm", searchTerm)

reader = cmd.ExecuteReader()

if reader.HasRows Then

while reader.Read()

product = new Products

product.ProductType = reader.GetValueOrDefault<string>("ProductType")

products.Add(product)

endwhile

return products

else

return products

endif

conn.Close() //close the connection to the database

End Function

Function GetProductsDetailsByType(productType)

//this function finds gets all the details of the products returned in the last function

products = empty List of ProductDetails

conn = sqlServerConnection.GetConnection

conn.Open()

cmd = new SqlCommand("ProductDetailsByType", conn) //calls the ProductDetailsByType stored procedure

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@ProductType", productType)

reader = cmd.ExecuteReader()

if reader.HasRows Then

while reader.Read()

product = new ProductDetails

product.ID = reader.GetValueOrDefault<int>("ID")

product.ProductDescription = reader.GetValueOrDefault<string>("ProductDescription")

product.ProductPrice = reader.GetValueOrDefault<double>("ProductPrice")

product.ImageName = reader.GetValueOrDefault<string>("ImageName")

products.Add(product)

endwhile

return products

else

return products

endif

conn.Close()

End Function

### Get any open order from a customer

Function GetOpenOrderNumberByCustomer(customerId)

conn = sqlServerConnection.GetConnection

conn.Open()

//calls the GetOpenOrderByCustomer stored procedure

cmd = new SqlCommand("GetOpenOrderNumberByCustomer", conn)

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@CustomerId", customerId)

reader = cmd.ExecuteReader()

if reader.HasRows Then

reader.Read()

return reader.GetValueOrDefault<string>("OrderNumber")

else

return string.Empty

endif

conn.Close()

End Function

### Get products that are low on stock

Function GetLowStock()

lowStockList = empty List of LowStock

conn = sqlServerConnection.GetConnection

conn.Open()

cmd = new SqlCommand("ProductMaster\_GetLowStock", conn) //calls the GetLowStock stored procedure

cmd.CommandType = CommandType.StoredProcedure

reader = cmd.ExecuteReader()

if reader.HasRows Then

while reader.Read()

line = new LowStock

line.ID = reader.GetValueOrDefault<int>("ID")

line.ProductCode = reader.GetValueOrDefault<string>("ProductCode")

line.ProductDescription = reader.GetValueOrDefault<string>("ProductDescription")

line.StockQuantity = reader.GetValueOrDefault<int>("StockQuantity")

line.DefaultOrderQuantity = reader.GetValueOrDefault<int>("DefaultOrderQuantity")

lowStockList.Add(line)

endwhile

return lowStockList

else

return lowStockList

endif

conn.Close()

End Function

### Edit a product

Procedure UpdateProduct(product)

conn = sqlServerConnection.GetConnection

conn.Open()

cmd = new SqlCommand("UpdateProduct", conn) //calls the UpdateProduct stored procedure

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@Id", product.ID)

cmd.Parameters.AddWithValue("@ProductDescription", product.ProductDescription)

cmd.Parameters.AddWithValue("@ReOrderQuantity", product.ReOrderQuantity)

cmd.Parameters.AddWithValue("@DefaultOrderQuantity", product.DefaultOrderQuantity)

cmd.Parameters.AddWithValue("@ProductPrice", product.ProductPrice)

cmd.Parameters.AddWithValue("@StockQuantity", product.Quantity)

cmd.Parameters.AddWithValue("@ImageName", product.ImageName)

//execute the command and get the number of affected records

numberRecordsAffected = cmd.ExecuteNonQuery()

if numberRecordsAffected <= 0 Then

throw new ApplicationException($"Failed To Update product ID {product.ID}")

endif

conn.Close()

End Procedure

### Delete a product

procedure DeleteProductById(id: integer)

conn = sqlServerConnection.GetConnection

conn.Open()

cmd = SqlCommand("DeleteProductById", conn) //calls DeleteProductById stored procedure

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@Id", id)

numberRecordsAffected = cmd.ExecuteNonQuery()

if numberRecordsAffected <= 0: //check if the product was deleted successfully

throw new ApplicationException("Failed To Delete product ID " + id)

endif

conn.Close()

end procedure

### Add a new product

procedure InsertProduct(product: ProductDetails)

conn = sqlServerConnection.GetConnection

conn.Open()

cmd = new SqlCommand("InsertProduct", conn) //calls InsertProduct stored procedure

cmd.CommandType = CommandType.StoredProcedure

cmd.Parameters.AddWithValue("@ProductCode", product.ProductCode)

cmd.Parameters.AddWithValue("@ProductDescription", product.ProductDescription)

cmd.Parameters.AddWithValue("@StockQuantity", product.Quantity)

cmd.Parameters.AddWithValue("@ReOrderQuantity", product.ReOrderQuantity)

cmd.Parameters.AddWithValue("@DefaultOrderQuantity", product.DefaultOrderQuantity)

cmd.Parameters.AddWithValue("@ProductType", product.ProductType)

cmd.Parameters.AddWithValue("@ProductPrice", product.ProductPrice)

cmd.Parameters.AddWithValue("@ImageName", product.ImageName)

numberRecordsAffected = cmd.ExecuteNonQuery()

if numberRecordsAffected <= 0: //check if insert was successfull

throw new ApplicationException($"Failed To Insert product {product.ProductDescription}")

endif

conn.Close()

endprocedure

# **Technical Solution**

## **Introduction**

My project consists of 4 main areas:

* Models – representation of the data being posted to the controller
* Views – HTML templates with embedded Razor markup (user interface)
* Controllers – responsible for controlling the way that a user interacts with the application
* Database – series of stored procedures in SQL Server Management Studio that all called from functions inside the repositories

## **Pointers**

* Cross table parameterised SQL – Stored Procedures, Get Products In Open order, page 82, and Get order line from ID page 85, and Mark order as ordered, page 85
* Dynamic generation of objects based on complex user-defined use of OOP model – this is reached through use of dependency injection throughout the application
* All controllers and repositories
* Get or create an order, page 83 is a complex SQL stored procedure

## **Startup**

using Database;

using Database.Repos;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc.Authorization;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using ShoppingToYou.Database;

namespace ShoppingToYou

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services) //dependency injection container

{

services.AddControllersWithViews(o => o.Filters.Add(new AuthorizeFilter()));

services.AddScoped<IUserRepo, UserRepo>();

services.AddTransient<ISqlServerConnection, SqlServerConnection>();

services.AddTransient<IStockRepo, StockRepo>();

services.AddTransient<IOrderRepo, OrderRepo>();

services.AddTransient<ICustomerRepo, CustomerRepo>();

services.AddAuthentication(o =>

{

o.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;

})

.AddCookie();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Account}/{action=Login}/{id?}");

});

}

}

}

## **Models**

The models in my application are responsible for defining application data and transporting it around the application, the properties of these models correspond to fields in my database:

The properties in the model are used to populate the Customer table in my database with details entered on the Customer Details page.

namespace ShoppingToYou.Models

{

public class Customer

{

public string Email { get; set; }

public string FirstName { get; set; }

public string Surname { get; set; }

public string Address1 { get; set; }

public string Address2 { get; set; }

public string Address3 { get; set; }

public string Address4 { get; set; }

public string Postcode { get; set; }

public string Phone { get; set; }

}

}

## **Views**

### **Login**

#### Login Page

@\*Login Page\*@

@model ShoppingToYou.Models.LoginModel

@{

Layout = "~/Views/Account/LoginLayout.cshtml";

}

<div class="login-page">

<div class="page-header">

<h1>Login</h1>

</div>

<div class="row">

<div class="col-sm-6">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Please enter your credentials</h3>

</div>

<div class="panel-body">

<form **asp-route**="Login">

<input **type**="hidden" **asp-for**="ReturnUrl" />

<fieldset>

<div class="form-group">

<label **asp-for**="Email"></label>

<input class="form-control" placeholder="Email" **asp-for**="Email" autofocus>

</div>

<div class="form-group">

<label **asp-for**="Password"></label>

<input **type**="password" class="form-control" placeholder="Password" **asp-for**="Password" autocomplete="off">

</div>

<div class="form-group">

<button class="btn btn-primary" style="color: white;">Login</button>

</div>

<div class="form-group">

<p>

<a **asp-action**="Register" **asp-route-returnurl**="@ViewData["ReturnUrl"]">Register as a new user?</a>

</p>

</div>

</fieldset>

</form>

</div>

</div>

</div>

</div>

</div>

#### New account page

@\*Create new account page\*@

@model ShoppingToYou.Models.RegisterViewModel

<h2>@ViewData["Title"]</h2>

@{

Layout = "~/Views/Account/LoginLayout.cshtml";

}

<div>

<a **asp-action**="Login">Back</a>

</div>

<div class="row">

<div class="col-md-4">

<form **asp-route**="Register">

<input **type**="hidden" **asp-for**="ReturnUrl" />

<h1 class="heading">New Account</h1>

<div **asp-validation-summary**="All" class="text-danger"></div>

//each of these blocks of code represents an input box on the page

<div class="form-group">

<label **asp-for**="UserName"></label>

<input **asp-for**="UserName" class="form-control" />

<span **asp-validation-for**="UserName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Email"></label>

<input **asp-for**="Email" class="form-control" />

<span **asp-validation-for**="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Password"></label>

<input **asp-for**="Password" class="form-control" />

<span **asp-validation-for**="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ConfirmPassword"></label>

<input **asp-for**="ConfirmPassword" class="form-control" />

<span **asp-validation-for**="ConfirmPassword" class="text-danger"></span>

</div>

<button class="btn btn-primary" style="color: white;">Register</button>

</form>

</div>

</div>

### **Customer Side**

#### Customer details page

@\*Customer Details Page\*@

@model ShoppingToYou.Models.Customer

<div id="modal\_dialog" role="dialog"></div>

<head>

<meta name="viewport" content="width=device-width" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

<link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

<link rel="stylesheet" href="../css/site.css" />

</head>

<body>

<h4 class="heading">Customer Details</h4>

<div class="form-styling">

<div class="row">

<div class="col-md-4">

<form **asp-action**="CreateCustomer">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **asp-for**="Email" class="form-control" **type**="hidden" />

<div class="form-group">

<label **asp-for**="FirstName" class="control-label"></label>

<input **asp-for**="FirstName" class="form-control" />

<span **asp-validation-for**="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Surname" class="control-label"></label>

<input **asp-for**="Surname" class="form-control" />

<span **asp-validation-for**="Surname" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Address1" class="control-label"></label>

<input **asp-for**="Address1" class="form-control" />

<span **asp-validation-for**="Address1" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Address2" class="control-label"></label>

<input **asp-for**="Address2" class="form-control" />

<span **asp-validation-for**="Address2" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Address3" class="control-label"></label>

<input **asp-for**="Address3" class="form-control" />

<span **asp-validation-for**="Address3" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Address4" class="control-label"></label>

<input **asp-for**="Address4" class="form-control" />

<span **asp-validation-for**="Address4" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Postcode" class="control-label"></label>

<input **asp-for**="Postcode" class="form-control" />

<span **asp-validation-for**="Postcode" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Phone" class="control-label"></label>

<input **asp-for**="Phone" class="form-control" />

<span **asp-validation-for**="Phone" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" style="background-color: #13921e; color: white;" />

</div>

</form>

</div>

</div>

</div>

</body>

#### Customer homepage

@\*Customer Homepage\*@

@model IEnumerable<ShoppingToYou.Models.OrderLines>

@{

ViewData["Title"] = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1 class="greeting">Welcome</h1>

<input type="text" id="productTypes" placeholder="Search Products, Enter Product Type" style="margin-left: 420px; width: 270px; border: solid 1px #1b6aba;" />

<div class="carouselposition">

<div class="carousel slide" data-ride="carousel" data-interval="3000">

<div class="carousel-inner">

<div class="carousel-item active">

<img src="~/Uploads/3redonionsPOTW.png" />

<img src="~/Uploads/babybellPOTW.png" />

<img src="~/Uploads/milkPOTW.png" />

</div>

<div class="carousel-item">

<img src="~/Uploads/bananaPOTW.png" />

<img src="~/Uploads/chickenPOTW.png" />

<img src="~/Uploads/cherrycoke8packjpegPOTW.png" />

</div>

</div>

</div>

</div>

<div id="partialViewProducts">

</div>

<input type="hidden" id="orderNo" value="@ViewBag.OrderNumber" />

<input type="hidden" class="customer" id="custId" value="@ViewBag.CustomerId" />

<div id='editOrderModal' class='modal'>

<div class="modal-dialog">

<div class="modal-content">

<div id='modalContent'></div>

</div>

</div>

</div>

@if (ViewBag.OrderNumber != "")

{

<table class="table">

<thead>

<tr>

<th>

</th>

<th>

@Html.DisplayNameFor(model => model.OrderNumber)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductDescription)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductPrice)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderQuantity)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

<img **src**="@FileLocation.RetriveFileFromFolder@item.ImageName" class="img-thumbnail" height="80" width="80" **asp-append-version**="true" />

</td>

<td>

@Html.DisplayFor(modelItem => item.OrderNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductDescription)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductPrice)

</td>

<td>

@Html.DisplayFor(modelItem => item.OrderQuantity)

</td>

<td>

<a href="#" data-id="@item.ID" class="delete-order btn btn-success">Delete</a>

</td>

<td>

<a href="#" data-id="@item.ID" class="edit-order btn btn-success">Edit</a>

</td>

</tr>

}

</tbody>

</table>

<a href="#" class="deliver-order btn btn-success">Finalise</a>

}

@section Scripts {

<script type="text/javascript">

$(document).ready(function () {

// Setup autocomplete search box

$("#productTypes").autocomplete({

source: function (request, response) {

$.ajax({

url: '/search',

data: { "term": request.term },

dataType: "json",

success: function (data) {

response($.map(data, function (item) {

return {

label: item.productType

}

}))

},

error: function (xhr, textStatus, error) {

alert(xhr.statusText);

alert(textStatus);

alert(error);

},

failure: function (response) {

alert("failure " + response.responseText);

}

});

},

select: function (e, i) {

GetProducts(i.item.label);

},

minLength: 3

});

// Routes to GetProducts action on selection from autocomplete

function GetProducts(term) {

var custId = $('#custId').val()

window.location.href = '@Url.Action("GetProducts", "Order")?term=' + term + '&custId=' + custId

}

// Finalise button – displays thank you page

var deliveryURL = '@Url.Action("Delivery")';

$(".deliver-order").click(function () {

var ordernumber = $("#orderNo").val();

location.href = deliveryURL + '?orderNumber=' + ordernumber;

});

// Delete button

var baseUrl = '@Url.Action("DeleteOrder")';

$(".delete-order").click(function () {

var id = $(this).data('id');

var custId = @ViewBag.CustomerId;

location.href = baseUrl + '?id=' + id + '&custId=' + custId;

});

// Edit button - diplays modal popup

var modalURL = '@Url.Action("EditOrder")';

$(function () {

$(".edit-order").click(function () {

//debugger;

var $buttonClicked = $(this);

var id = $(this).data('id');

var custId = $("#custId").val();

var options = { "backdrop": "static", keyboard: true };

$.ajax({

type: "GET",

url: modalURL,

contentType: "application/json; charset=utf-8",

data: { id: id, custId : custId },

datatype: "json",

success: function (data) {

//debugger;

$('#modalContent').html(data);

$('#editOrderModal').modal(options);

$('#editOrderModal').modal('show');

},

error: function () {

alert("Dynamic content load failed.");

}

});

});

$("#closbtn").click(function () {

$('#editOrderModal').modal('hide');

});

});

});

</script>

#### Search results page

@\*Search results page\*@

@model IEnumerable<ShoppingToYou.Models.ProductDetails>

@{

ViewData["Title"] = "ProductDetails";

}

<div>

<a **asp-action**="index">Return Home</a>

</div>

<h1 class="heading">Search Results</h1>

<input type="hidden" class="customer" id="custId" value="@ViewBag.CustomerId" />

<table class="table"> //table of products displayed after using search box

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.ID)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductDescription)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductPrice)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

<img **src**="@FileLocation.RetriveFileFromFolder@item.ImageName" class="img-thumbnail" height="80" width="80" **asp-append-version**="true" />

</td>

<td>

@Html.DisplayFor(modelItem => item.ID)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductDescription)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductPrice)

</td>

<td>

<input class="quantity" type="number" value="1" min="1" max="100" size="10" />

</td>

<td>

<a href="#" data-id="@item.ID" class="order btn btn-success">Order</a>

</td>

</tr>

}

</tbody>

</table>

@section Scripts {

<script type="text/javascript">

$(document).ready(function () {

var baseUrl = '@Url.Action("Place")';

$(".order").click(function()

{

var id = $(this).data('id');

var custId = $("#custId").val();

var quantity = $(this).closest('tr').find('.quantity').val();

location.href = baseUrl + '?customerId=' + custId + '&productId=' + id + '&quantity=' + quantity;

});

});

</script>

}

#### Edit order popup

All of my modal dialogue popups are jquery and have partial views loaded into them which contain forms for user input, each form input is validated and cant be left null

@\*Modal popup for editing an order\*@

@model ShoppingToYou.Models.OrderLineDetail

<h4>ProductDetails</h4>

<hr />

<div class="form-styling">

<div class="row">

<div class="col-md-4">

<form **asp-action**="EditOrder" enctype="multipart/form-data">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **asp-for**="ID" class="form-control" **type**="hidden" />

<input **asp-for**="CustomerId" class="form-control" **type**="hidden" />

<div class="form-group">

<label **asp-for**="ProductDescription" class="control-label"></label>

<input **asp-for**="ProductDescription" class="form-control" disabled />

<span **asp-validation-for**="ProductDescription" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ProductPrice" class="control-label"></label>

<input **asp-for**="ProductPrice" class="form-control" disabled />

<span **asp-validation-for**="ProductPrice" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="OrderQuantity" class="control-label"></label>

<input **asp-for**="OrderQuantity" class="form-control" />

<span **asp-validation-for**="OrderQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" style="background-color: #13921e; color: white; " />

</div>

</form>

</div>

</div>

<div>

<a **asp-action**="index">Back to List</a>

</div>

</div>

@{await Html.RenderPartialAsync("ValidationScriptsPartial");}

#### Finalise order page

@\*Page displayed after finalise button is clicked\*@

<div>

<a **asp-action**="index">Continue Shopping</a>

</div>

<h1 class="thanks">Thank You For Your Order!</h1>

### **Staff side**

#### Staff homepage

@\*Staff homepage\*@

@model IEnumerable<Database.Models.LowStock>

@{

ViewData["Title"] = "Low Stock";

}

<h1 class="heading">Low Stock</h1>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.ProductCode)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductDescription)

</th>

<th>

@Html.DisplayNameFor(model => model.StockQuantity)

</th>

<th>

@Html.DisplayNameFor(model => model.DefaultOrderQuantity)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.HiddenFor(modelItem => item.ID)

@Html.DisplayFor(modelItem => item.ProductCode)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductDescription)

</td>

<td>

@Html.DisplayFor(modelItem => item.StockQuantity)

</td>

<td>

@Html.DisplayFor(modelItem => item.DefaultOrderQuantity)

</td>

<td>

<a href="#" data-id="@item.ID" data-qty="@item.DefaultOrderQuantity" class="order btn btn-success">Order</a>

</td>

</tr>

}

</tbody>

</table>

<a href="#" class="manage-stock btn btn-success">Manage Stock</a>

@section Scripts {

<script type="text/javascript">

$(document).ready(function () {

// Manage Stock button

var baseUrl = '@Url.Action("Manage")';

$(".manage-stock").click(function () {

location.href = baseUrl;

});

});

// Order button

var baseUrl = '@Url.Action("Increase")';

$(".order").click(function () {

var id = $(this).data('id');

var dftOrderQty = $(this).data('qty');

location.href = baseUrl + '?id=' + id + '&dftOrderQty=' + dftOrderQty;

});

</script>

}

#### Manage stock page

@\*Page displayed after manage stock button is clicked\*@

@{

ViewData["Title"] = "Manage Stock";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<div>

<a **asp-action**="index">Return Home</a>

</div>

<h1 class="managestockheader">Manage Stock</h1>

<p>

<a href="#" class="add-product btn btn-success center">Add Product</a>

</p>

<input type="text" id="productDescriptions" placeholder="Search Products, Enter Description" style="margin-left: 420px; width: 270px; border: solid 1px #1b6aba; margin-top: 50px;"/>

<div id="partialViewManageStock">

</div>

<div id='addProductModal' class='modal'>

<div class="modal-dialog">

<div class="modal-content">

<div id='addModalContent'></div>

</div>

</div>

</div>

<div id='editProductModal' class='modal'>

<div class="modal-dialog">

<div class="modal-content">

<div id='modalContent'></div>

</div>

</div>

</div>

@section Scripts {

<script type="text/javascript">

$(document).ready(function () {

// Setup autocomplete search box

$("#productDescriptions").autocomplete({

source: function (request, response) {

$.ajax({

url: '/find',

data: { "term": request.term },

dataType: "json",

success: function (data) {

response($.map(data, function (item) {

return {

label: item.productDescription, value: item.productID

}

//return item;

}))

},

error: function (xhr, textStatus, error) {

alert(xhr.statusText);

alert(textStatus);

alert(error);

},

failure: function (response) {

alert("failure " + response.responseText);

}

});

},

select: function (e, i) {

GetProducts(i.item.value);

},

minLength: 3

});

// Routes to GetProducts action on selection from autocomplete

function GetProducts(term) {

$.ajax({

url: '/Stock/GetProducts',

type: 'POST',

async: false,

data: { id: term },

success: function (result) {

$("#partialViewManageStock").html(result);

}

});

}

// Add button - diplays modal popup

var modalURL = '@Url.Action("Create")';

$(function () {

$(".add-product").click(function () {

var options = { "backdrop": "static", keyboard: true };

$.ajax({

type: "GET",

url: modalURL,

contentType: "application/json; charset=utf-8",

data: {},

datatype: "json",

success: function (data) {

$('#addModalContent').html(data);

$('#addProductModal').modal(options);

$('#addProductModal').modal('show');

},

error: function () {

alert("Dynamic content load failed.");

}

});

});

$("#closbtn").click(function () {

$('#addProductModal').modal('hide');

});

});

});

</script>

}

#### Search results page

@\*Search results page\*@

@model ShoppingToYou.Models.ProductDetails

<div class="productdetails">

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.ID)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductDescription)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductPrice)

</th>

<th>

@Html.DisplayNameFor(model => model.Quantity)

</th>

<th></th>

</tr>

</thead>

<tbody>

<tr>

<td>

@Html.DisplayFor(model => model.ID)

</td>

<td>

@Html.DisplayFor(model => model.ProductDescription)

</td>

<td>

@Html.DisplayFor(model => model.ProductPrice)

</td>

<td>

@Html.DisplayFor(model => model.Quantity)

</td>

<td>

<a href="#" data-id="@Model.ID" class="delete-product btn btn-success">Delete</a>

</td>

<td>

<a href="#" data-id="@Model.ID" class="edit-product btn btn-success">Edit</a>

</td>

</tr>

</tbody>

</table>

</div>

<script type="text/javascript">

$(document).ready(function () {

// Edit button - diplays modal popup

var modalURL = '@Url.Action("Edit")';

$(function () {

$(".edit-product").click(function () {

//debugger;

var $buttonClicked = $(this);

var id = $(this).data('id');

var options = { "backdrop": "static", keyboard: true };

$.ajax({

type: "GET",

url: modalURL,

contentType: "application/json; charset=utf-8",

data: { "id": id },

datatype: "json",

success: function (data) {

//debugger;

$('#modalContent').html(data);

$('#editProductModal').modal(options);

$('#editProductModal').modal('show');

},

error: function () {

alert("Dynamic content load failed.");

}

});

});

$("#closbtn").click(function () {

$('#editProductModal').modal('hide');

});

});

// Delete button

var baseUrl = '@Url.Action("Delete")';

$(".delete-product").click(function () {

var id = $(this).data('id');

location.href = baseUrl + '?id=' + id;

});

});

</script>

#### Add stock popup

@\*Modal popup for adding stock\*@

@model ShoppingToYou.Models.ProductDetails

<h4>ProductDetails</h4>

<hr />

<div class="form-styling">

<div class="row">

<div class="col-md-4">

<form **asp-action**="Create" enctype="multipart/form-data">

<div **asp-validation-summary**="ModelOnly" class="modal-body text-danger"></div>

<div class="form-group">

<label **asp-for**="ProductCode" class="control-label"></label>

<input **asp-for**="ProductCode" class="form-control" />

<span **asp-validation-for**="ProductCode" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ProductDescription" class="control-label"></label>

<input **asp-for**="ProductDescription" class="form-control" />

<span **asp-validation-for**="ProductDescription" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ProductPrice" class="control-label"></label>

<input **asp-for**="ProductPrice" class="form-control" />

<span **asp-validation-for**="ProductPrice" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Quantity" class="control-label"></label>

<input **asp-for**="Quantity" class="form-control" />

<span **asp-validation-for**="Quantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ReOrderQuantity" class="control-label"></label>

<input **asp-for**="ReOrderQuantity" class="form-control" />

<span **asp-validation-for**="ReOrderQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="DefaultOrderQuantity" class="control-label"></label>

<input **asp-for**="DefaultOrderQuantity" class="form-control" />

<span **asp-validation-for**="DefaultOrderQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ProductType" class="control-label"></label>

<input **asp-for**="ProductType" class="form-control" />

<span **asp-validation-for**="ProductType" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ImageFile" class="control-label"></label>

<input **asp-for**="ImageFile" accept="image/\*" />

<span **asp-validation-for**="ImageFile" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" style="background-color: #13921e; color: white; " />

</div>

</form>

</div>

</div>

<div>

<a **asp-action**="Manage">Back to List</a>

</div>

</div>

@{await Html.RenderPartialAsync("ValidationScriptsPartial");}

#### Edit stock popup

@\*Modal popup for editing stock\*@

@model ShoppingToYou.Models.ProductDetails

<h4>ProductDetails</h4>

<hr />

<div class="form-styling">

<div class="row">

<div class="col-md-4">

<form **asp-action**="Edit" enctype="multipart/form-data">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **asp-for**="ID" class="form-control" **type**="hidden" />

<div class="form-group">

<label **asp-for**="ProductDescription" class="control-label"></label>

<input **asp-for**="ProductDescription" class="form-control" />

<span **asp-validation-for**="ProductDescription" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ProductPrice" class="control-label"></label>

<input **asp-for**="ProductPrice" class="form-control" />

<span **asp-validation-for**="ProductPrice" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Quantity" class="control-label"></label>

<input **asp-for**="Quantity" class="form-control" />

<span **asp-validation-for**="Quantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ReOrderQuantity" class="control-label"></label>

<input **asp-for**="ReOrderQuantity" class="form-control" />

<span **asp-validation-for**="ReOrderQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="DefaultOrderQuantity" class="control-label"></label>

<input **asp-for**="DefaultOrderQuantity" class="form-control" />

<span **asp-validation-for**="DefaultOrderQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ImageFile" class="control-label"></label>

<input **asp-for**="ImageFile" accept="image/\*" />

<span **asp-validation-for**="ImageFile" class="text-danger"></span>

<img **src**="@FileLocation.RetriveFileFromFolder@Model.ImageName" class="img-thumbnail" height="80" width="80" **asp-append-version**="true" />

</div>

<div class="form-group">

<input type="submit" value="Save" style="background-color: #13921e; color: white;" />

</div>

</form>

</div>

</div>

<div>

<a **asp-action**="Manage">Back to List</a>

</div>

</div>

@{await Html.RenderPartialAsync("ValidationScriptsPartial");}

## **Controllers**

### Account controller

This class is responsible for routing account related requests

using Database.Models;

using Database.Repos;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using ShoppingToYou.Models;

using System;

using System.Collections.Generic;

using System.Security.Claims;

using System.Threading.Tasks;

namespace ShoppingToYou.Controllers

{

public class AccountController : Controller

{

private readonly IUserRepo userRepo;

public AccountController(IUserRepo userRepo) //grabs object of type IUserRepo from dependency injection container and puts it in userRepo

{

this.userRepo = userRepo;

}

[AllowAnonymous]

public async Task<IActionResult> Login(string returnUrl = null)

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme); //first line of code the program hits

return View(new LoginModel { ReturnUrl = returnUrl });

}

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> Login(LoginModel model)

{

var user = await userRepo.FindByNameAsync(model.Email); //when login button is clicked, checks database to check email exists already

if (user == null)

return Unauthorized();

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, user.UserName),

//in this block of code, 4 claims are made, these are added to a token which is added to requests for authentication

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.Role, user.Role),

};

var identity = new ClaimsIdentity(claims,

CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

await HttpContext.SignInAsync(

CookieAuthenticationDefaults.AuthenticationScheme,

principal,

new AuthenticationProperties { IsPersistent = model.RememberLogin });

if (user.Role == "Admin")

return LocalRedirect("/stock/index"); //display staff homepage

else if (user.Role == "User")

return LocalRedirect("/order/index"); //display customer homepage0

else

return Unauthorized();

}

[HttpGet] //called when the register link is clicked on login page

[AllowAnonymous]

public IActionResult Register(string returnUrl = null)

{

return View(new RegisterViewModel { ReturnUrl = returnUrl }); //displays register page

}

[HttpPost] //called from the register button on the register page

[AllowAnonymous]

public async Task<IActionResult> Register(RegisterViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.UserName, Email = model.Email, PasswordHash = model.Password, FavoriteColor = "blue", Role = "User" };

var result = await userRepo.CreateAsync(user);

if (result)

{

return LocalRedirect("/");

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return Redirect("/");

}

}

}

### Order controller

This class is responsible for routing order related requests

using Database;

using Database.Repos;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using ShoppingToYou.Database;

using ShoppingToYou.Models;

using System;

using System.Collections.Generic;

namespace ShoppingToYou.Controllers

{

[Produces("application/json")]

[Authorize]

public class OrderController : Controller

{

private readonly IStockRepo stockRepo;

private readonly IOrderRepo orderRepo;

private readonly ICustomerRepo customerRepo;

public OrderController(IStockRepo stockRepo, IOrderRepo orderRepo, ICustomerRepo customerRepo)

{

this.stockRepo = stockRepo;

this.orderRepo = orderRepo;

this.customerRepo = customerRepo;

}

[HttpGet]

public ActionResult Index()

{

int custID;

var userEmail = User.Identity.GetEmail(); //retrieves email address from tokens/claims

custID = customerRepo.CheckCustomerByEmail(userEmail); //calls customer repo

if (custID == 0)

{

var customer = new Customer { Email = userEmail };

return PartialView("CustomerDetailsPartial", customer); //calls partial view for customer to enter their address etc..

}

var orderNumber = orderRepo.GetOpenOrderNumberByCustomer(custID);//carries on with normal program and loads homepage

var currentOrders = orderRepo.GetOrderLines(orderNumber);

var results = currentOrders == null ? new List<OrderLines>() : currentOrders;

ViewBag.CustomerId = custID;

ViewBag.OrderNumber = orderNumber;

return View(results);

}

// Loads the autocomplete search box

[HttpGet("search")]

public IActionResult Search()

{

try

{

string term = HttpContext.Request.Query["term"].ToString();

var results = stockRepo.GetProductsTypesBySearchTerm(term);

return Ok(results);

}

catch

{

return BadRequest();

}

}

// Displays product list based on selection from autocomplete

public IActionResult GetProducts(string term, string custId)

{

try

{

var results = stockRepo.GetProductsDetailsByType(term);

ViewBag.CustomerId = custId;

return View("ProductDetails", results);

}

catch

{

return BadRequest();

}

}

// Called from the order button

public IActionResult Place(int customerId, int productId, int quantity)

{

try

{

// Check for an open order for this customer - if found return the order number else create it

string orderNumber = orderRepo.GetOrCreateOrder(1);

if (string.IsNullOrEmpty(orderNumber))

return BadRequest();

// add line to the order

orderRepo.InsertOrderLines(orderNumber, quantity, productId);

ViewBag.CustomerId = customerId;

ViewBag.OrderNumber = orderNumber;

var currentOrder = orderRepo.GetOrderLines(orderNumber);

return View("index", currentOrder);

}

catch

{

return BadRequest();

}

}

// Called from the finalise button

public IActionResult Delivery(string orderNumber)

{

try

{

orderRepo.MarkOrderAsOrdered(orderNumber);

return View("OrderThanks");

}

catch

{

return BadRequest();

}

}

public IActionResult CreateCustomer(Customer customer) //called when submit button is clicked on the CustomerDetails partial view

{

try

{

var custID = customerRepo.InsertCustomer(customer.Email, customer.FirstName, customer.Surname, customer.Address1, customer.Address2,

customer.Address3, customer.Address4, customer.Postcode, customer.Phone); //calls new action

var orderNumber = orderRepo.GetOpenOrderNumberByCustomer(custID);//carries on with normal program and loads homepage

var currentOrders = orderRepo.GetOrderLines(orderNumber);

var results = currentOrders == null ? new List<OrderLines>() : currentOrders;

ViewBag.CustomerId = custID;

ViewBag.OrderNumber = orderNumber;

return View("index", results);

}

catch

{

return BadRequest();

}

}

// Loads the edit popup

[HttpGet("EditOrder")]

public IActionResult EditOrder(int id, string custId)

{

try

{

var results = orderRepo.OrderLineById(id);

results.CustomerId = Convert.ToInt32(custId);

return PartialView("EditOrderPartial", results);

}

catch

{

return BadRequest();

}

}

// Saves changes made in edit popup

[HttpPost("EditOrder")]

public IActionResult EditOrder(OrderLineDetail order)

{

try

{

orderRepo.UpdateOrder(order);

var orderNumber = orderRepo.GetOpenOrderNumberByCustomer(order.CustomerId);//carries on with normal program and loads homepage

var currentOrders = orderRepo.GetOrderLines(orderNumber);

ViewBag.OrderNumber = orderNumber;

ViewBag.CustomerId = order.CustomerId;

return View("index", currentOrders);

}

catch

{

return BadRequest();

}

}

public IActionResult DeleteOrder(int id, string custId)

{

try

{

orderRepo.DeleteOrder(id);

var orderNumber = orderRepo.GetOpenOrderNumberByCustomer(Convert.ToInt32(custId));//carries on with normal program and loads homepage

var currentOrders = orderRepo.GetOrderLines(orderNumber);

ViewBag.OrderNumber = orderNumber;

ViewBag.CustomerId = custId;

return View("index", currentOrders);

}

catch

{

return BadRequest();

}

}

}

}

### Stock controller

This class is responsible for routing all stock related requests

using Database.Repos;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using ShoppingToYou.Models;

namespace ShoppingToYou.Controllers

{

[Authorize]

public class StockController : Controller

{

private readonly IStockRepo stockRepo;

public StockController(IStockRepo stockRepo)

{

this.stockRepo = stockRepo;

}

public IActionResult Index() //called when the staff hompage is about to be displayed

{

var lowStock = stockRepo.GetLowStock();

return View(lowStock);

}

public IActionResult Manage() //called when manage stock button is clicked

{

return View("ManageStock");

}

// Loads the autocomplete search box

[HttpGet("find")]

public IActionResult Find()

{

try

{

string term = HttpContext.Request.Query["term"].ToString();

var results = stockRepo.GetProductsDescriptionsBySearchTerm(term);

return Ok(results);

}

catch

{

return BadRequest();

}

}

// Called on selection from autocomplete

public IActionResult GetProducts(int id)

{

try

{

var results = stockRepo.GetProductsDetailsById(id);

return PartialView("\_ProductDetailsPartial", results);

}

catch

{

return BadRequest();

}

}

[HttpGet("create")]

public IActionResult Create() //called when add stock button is clicked

{

return PartialView("\_CreateStockPartial");

}

[HttpPost("create")]

public IActionResult Create(ProductDetails product)

{

product.ImageName = product.ImageFile.FileName;

stockRepo.InsertProduct(product);

return View("ManageStock");

}

// Loads the edit popup

[HttpGet("Edit")]

public IActionResult Edit(int id)

{

try

{

var results = stockRepo.GetProductsDetailsById(id);

return PartialView("\_EditStockPartial", results);

}

catch

{

return BadRequest();

}

}

// Saves changes made in edit popup

[HttpPost("Edit")]

public IActionResult Edit(ProductDetails product)

{

try

{

if (product.ImageFile != null)

{

product.ImageName = product.ImageFile.FileName;

}

else

{

product.ImageName = "";

}

stockRepo.UpdateProduct(product);

return View("ManageStock");

}

catch

{

return BadRequest();

}

}

public IActionResult Delete(int id) //called when delete button is clicked

{

try

{

stockRepo.DeleteProductById(id);

return View("ManageStock");

}

catch

{

return BadRequest();

}

}

public IActionResult Increase(int id, int dftOrderQty) //called when order button is clicked

{

try

{

stockRepo.IncreaseProduct(id, dftOrderQty);

var lowStock = stockRepo.GetLowStock();

return View("Index", lowStock);

}

catch

{

return BadRequest();

}

}

}

}

## **Database**

### Customer repository

All functions in the repositories are ADO.Net and interact with the database

This class is responsible for managing the Customer table in the database

using Database;

using Microsoft.Extensions.Configuration;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Threading.Tasks;

namespace ShoppingToYou.Database

{

public class CustomerRepo : ICustomerRepo

{

private readonly ISqlServerConnection sqlServerConnection; //holds connection to database

public CustomerRepo(IConfiguration configuration, ISqlServerConnection sqlServerConnection)

{

this.sqlServerConnection = sqlServerConnection;

}

public int CheckCustomerByEmail(string email)

{

if (string.IsNullOrEmpty(email))

{

throw new ArgumentException($"'{nameof(email)}' cannot be null or empty.", nameof(email));

}

using SqlConnection conn = sqlServerConnection.GetConnection; //create connection to database

conn.Open(); //open connection to database

using SqlCommand cmd = new SqlCommand("CheckCustomer", conn); //calls the CheckCustomer stored procedure

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@email", email);

// execute query

object o = cmd.ExecuteScalar(); // returns first value in the first row in the table

if (o != null)

{

return (int)o; //if the customer is registered (they have already given their address etc..)

}

return 0;

}

public int InsertCustomer(string email, string firstName, string surname, string address1, string address2, string address3,

string address4, string postcode, string phone)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("InsertCustomerDetails", conn); //calls this stored procedure to transfer details entered customer details partial view to the database

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Email", email);

cmd.Parameters.AddWithValue("@FirstName", firstName);

cmd.Parameters.AddWithValue("@Surname", surname);

cmd.Parameters.AddWithValue("@Address1", address1);

cmd.Parameters.AddWithValue("@Address2", address2 ?? "");

cmd.Parameters.AddWithValue("@Address3", address3 ?? "");

cmd.Parameters.AddWithValue("@Address4", address4 ?? "");

cmd.Parameters.AddWithValue("@Postcode", postcode);

cmd.Parameters.AddWithValue("@Phone", phone);

cmd.Parameters.Add("@CustomerID", SqlDbType.Int).Direction = ParameterDirection.Output; //gets the new id generated by the database

try

{

// execute query

cmd.ExecuteNonQuery();

int custID = Convert.ToInt32(cmd.Parameters["@CustomerID"].Value);

if (custID > 0)

{

return custID; //returns new customer id

}

else

{

throw new ApplicationException("Failed To Insert Customer Details");

}

}

catch(Exception exception)

{

throw new ApplicationException("Failed To Insert Customer Details");

}

}

}

}

### Order repository

This class is responsible for managing all order related activities in the database

using Database;

using ShoppingToYou.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

namespace ShoppingToYou.Database

{

public class OrderRepo : IOrderRepo

{

private readonly ISqlServerConnection sqlServerConnection;

public OrderRepo(ISqlServerConnection sqlServerConnection)

{

this.sqlServerConnection = sqlServerConnection ?? throw new ArgumentNullException(nameof(sqlServerConnection));

}

public string GetOpenOrderNumberByCustomer(int customerId)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("GetOpenOrderNumberByCustomer", conn); //called to display any items in basket on customer homepage

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@CustomerId", customerId);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

return reader.GetValueOrDefault<string>("OrderNumber");

}

else

{

return string.Empty;

}

}

public string GetOrCreateOrder(int customerId)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("GetOrCreateOrder", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@CustomerId", customerId);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

return reader.GetValueOrDefault<string>("OrderNumber");

}

else

{

return null;

}

}

public List<OrderLines> GetOrderLines(string orderNumber)

{

List<OrderLines> OrderLine = new List<OrderLines>();

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("GetShoppingList", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@OrderNumber", orderNumber);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var line = new OrderLines

{

ID = reader.GetValueOrDefault<int>("ID"),

OrderNumber = reader.GetValueOrDefault<string>("OrderNumber"),

OrderLine = reader.GetValueOrDefault<int>("OrderLine"),

ProductID = reader.GetValueOrDefault<int>("ProductID"),

OrderQuantity = reader.GetValueOrDefault<int>("OrderQuantity"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription"),

ProductPrice = reader.GetValueOrDefault <double>("ProductPrice"),

ImageName = reader.GetValueOrDefault<string>("ImageName")

};

OrderLine.Add(line);

}

return OrderLine;

}

else

{

return OrderLine;

}

}

public void InsertOrderLines(string orderNumber, int orderQuantity, int productId)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("InsertOrderLines", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@OrderNumber", orderNumber);

cmd.Parameters.AddWithValue("@OrderQuantity", orderQuantity);

cmd.Parameters.AddWithValue("@ProductID", productId);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException($"Failed To Insert Order Line for Order {orderNumber}");

}

}

public void MarkOrderAsOrdered(string orderNumber)

{

if (orderNumber is null)

{

throw new ArgumentNullException(nameof(orderNumber));

}

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("MarkOrderAsOrdered", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@OrderNumber", orderNumber);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException($"Failed To Update Order {orderNumber}");

}

}

public void UpdateOrder(OrderLineDetail order)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("UpdateOrder", conn); //called when save button is clicked on the edit order popup

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ID", order.ID);

cmd.Parameters.AddWithValue("@OrderQuantity", order.OrderQuantity);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException( "Failed To Update order");

}

}

public void DeleteOrder(int id)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("DeleteOrder", conn); //called when delete button is clicked

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ID", id);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException("Failed To Delete Order");

}

}

public OrderLineDetail OrderLineById(int id)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("OrderLineById", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ID", id);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

var orderLine = new OrderLineDetail

{

ID = reader.GetValueOrDefault<int>("ID"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription"),

ProductPrice = reader.GetValueOrDefault<double>("ProductPrice"),

OrderQuantity = reader.GetValueOrDefault<int>("OrderQuantity")

};

return orderLine;

}

else

{

return null;

}

}

}

}

### Stock repository

This class is responsible for managing stock related activities in the database

using Database.Models;

using ShoppingToYou.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

namespace Database.Repos

{

public class StockRepo : IStockRepo

{

private readonly ISqlServerConnection sqlServerConnection;

public StockRepo(ISqlServerConnection sqlServerConnection)

{

this.sqlServerConnection = sqlServerConnection ?? throw new ArgumentNullException(nameof(sqlServerConnection));

}

public List<LowStock> GetLowStock()

{

List<LowStock> lowStockList = new List<LowStock>();

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("ProductMaster\_GetLowStock", conn); //called to retrieve low stock

cmd.CommandType = CommandType.StoredProcedure;

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var line = new LowStock

{

ID = reader.GetValueOrDefault<int>("ID"),

ProductCode = reader.GetValueOrDefault<string>("ProductCode"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription"),

StockQuantity = reader.GetValueOrDefault<int>("StockQuantity"),

DefaultOrderQuantity = reader.GetValueOrDefault<int>("DefaultOrderQuantity")

};

lowStockList.Add(line);

}

return lowStockList;

}

else

{

return lowStockList;

}

}

public List<Products> GetProductsTypesBySearchTerm(string searchTerm)

{

List<Products> products = new List<Products>();

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("ProductTypeSearch", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@SearchTerm", searchTerm);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var product = new Products

{

ProductType = reader.GetValueOrDefault<string>("ProductType")

};

products.Add(product);

}

return products;

}

else

{

return products;

}

}

public List<ProductDetails> GetProductsDetailsByType(string productType)

{

List<ProductDetails> products = new List<ProductDetails>();

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("ProductDetailsByType", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ProductType", productType);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var product = new ProductDetails

{

ID = reader.GetValueOrDefault<int>("ID"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription"),

ProductPrice = reader.GetValueOrDefault<double>("ProductPrice"),

ImageName = reader.GetValueOrDefault<string>("ImageName")

};

products.Add(product);

}

return products;

}

else

{

return products;

}

}

public List<Products> GetProductsDescriptionsBySearchTerm(string searchTerm)

{

List<Products> products = new List<Products>();

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("ProductDescriptionSearch", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@SearchTerm", searchTerm);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var product = new Products

{

ProductID = reader.GetValueOrDefault<int>("ID"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription")

};

products.Add(product);

}

return products;

}

else

{

return products;

}

}

public ProductDetails GetProductsDetailsById(int id)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new SqlCommand("ProductDetailsById", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Id", id);

using var reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

var product = new ProductDetails

{

ID = reader.GetValueOrDefault<int>("ID"),

ProductDescription = reader.GetValueOrDefault<string>("ProductDescription"),

ProductPrice = reader.GetValueOrDefault<double>("ProductPrice"),

ImageName = reader.GetValueOrDefault<string>("ImageName"),

Quantity = reader.GetValueOrDefault<int>("StockQuantity"),

ReOrderQuantity = reader.GetValueOrDefault<int>("ReorderQuantity"),

DefaultOrderQuantity = reader.GetValueOrDefault<int>("DefaultOrderQuantity")

};

return product;

}

else

{

return null;

}

}

public void UpdateProduct(ProductDetails product)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("UpdateProduct", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Id", product.ID);

cmd.Parameters.AddWithValue("@ProductDescription", product.ProductDescription);

cmd.Parameters.AddWithValue("@ReOrderQuantity", product.ReOrderQuantity);

cmd.Parameters.AddWithValue("@DefaultOrderQuantity", product.DefaultOrderQuantity);

cmd.Parameters.AddWithValue("@ProductPrice", product.ProductPrice);

cmd.Parameters.AddWithValue("@StockQuantity", product.Quantity);

cmd.Parameters.AddWithValue("@ImageName", product.ImageName);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException($"Failed To Update product ID {product.ID}");

}

}

public void DeleteProductById(int id)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("DeleteProductById", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Id", id);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException($"Failed To Delete product ID {id}");

}

}

public void InsertProduct(ProductDetails product)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("InsertProduct", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ProductCode", product.ProductCode);

cmd.Parameters.AddWithValue("@ProductDescription", product.ProductDescription);

cmd.Parameters.AddWithValue("@StockQuantity", product.Quantity);

cmd.Parameters.AddWithValue("@ReOrderQuantity", product.ReOrderQuantity);

cmd.Parameters.AddWithValue("@DefaultOrderQuantity", product.DefaultOrderQuantity);

cmd.Parameters.AddWithValue("@ProductType", product.ProductType);

cmd.Parameters.AddWithValue("@ProductPrice", product.ProductPrice);

cmd.Parameters.AddWithValue("@ImageName", product.ImageName);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException("Failed To Insert product");

}

}

public void IncreaseProduct(int id, int dftOrderQty)

{

using SqlConnection conn = sqlServerConnection.GetConnection;

conn.Open();

using SqlCommand cmd = new("IncreaseLowStock", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Id", id);

cmd.Parameters.AddWithValue("@DefaultOrderQuantity", dftOrderQty);

// execute query

int numberRecordsAffected = cmd.ExecuteNonQuery();

if (numberRecordsAffected <= 0)

{

throw new ApplicationException("Failed To Increase product");

}

}

}

}

### **User repository**

This class if responsible for managing user related database activities

using Dapper;

using Database.Models;

using Microsoft.Extensions.Configuration;

using System.Data.SqlClient;

using System.Threading.Tasks;

namespace Database.Repos

{

public class UserRepo : IUserRepo

{

private readonly string \_connectionString;

public UserRepo(IConfiguration configuration)

{

\_connectionString = configuration.GetConnectionString("DefaultConnection");

}

public async Task<bool> CreateAsync(ApplicationUser user)

{

using (var connection = new SqlConnection(\_connectionString))

{

await connection.OpenAsync();

user.Id = await connection.QuerySingleAsync<int>($@"INSERT INTO [ApplicationUser] ([UserName], [Email], [PasswordHash], [FavoriteColor], [Role])

VALUES (@{nameof(ApplicationUser.UserName)}, @{nameof(ApplicationUser.Email)}, @{nameof(ApplicationUser.PasswordHash)},

@{nameof(ApplicationUser.FavoriteColor)}, @{nameof(ApplicationUser.Role)});

SELECT CAST(SCOPE\_IDENTITY() as int)", user);

}

return true;

}

public async Task<ApplicationUser> FindByNameAsync(string email)

{

using (var connection = new SqlConnection(\_connectionString))

{

await connection.OpenAsync();

return await connection.QuerySingleOrDefaultAsync<ApplicationUser>($@"SELECT \* FROM [ApplicationUser]

WHERE [Email] = @{nameof(email)}", new { email });

}

}

}

}

### **SQL server connection**

using Microsoft.Extensions.Configuration;

using System.Data.SqlClient;

namespace Database

{

public class SqlServerConnection : ISqlServerConnection

{

private readonly IConfiguration Configuration;

public SqlServerConnection(IConfiguration configuration)

{

Configuration = configuration;

}

public SqlConnection GetConnection

{

get

{

return new SqlConnection(Configuration.GetConnectionString("DefaultConnection")); //creates a new SQL connection based on the configuration

}

}

}

}

### **SQL stored procedures**

#### **Save details from customer details page**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[InsertCustomerDetails] Script Date: 08/03/2023 09:32:52 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[InsertCustomerDetails]

(@Email nvarchar(256)

,@FirstName nvarchar(50)

,@Surname nvarchar(50)

,@Address1 nvarchar(50)

,@Address2 nvarchar(50)

,@Address3 nvarchar(50)

,@Address4 nvarchar(50)

,@Postcode nvarchar(10)

,@Phone nvarchar(20)

,@CustomerID int output)

as

insert into Customers

(

UserName,

Email,

FirstName,

Surname,

Address1,

Address2,

Address3,

Address4,

Postcode,

Phone

)

values

(' ',

@Email,

@FirstName,

@Surname,

@Address1,

@Address2,

@Address3,

@Address4,

@Postcode,

@Phone)

set @CustomerID = SCOPE\_IDENTITY();

#### **Check if customer is registered**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CheckCustomer] Script Date: 08/03/2023 09:15:34 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[CheckCustomer]

(@email nvarchar(256))

as

select ID from Customers where Email = @email

#### **Get any open order from a customer**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetOpenOrderNumberByCustomer] Script Date: 08/03/2023 09:27:33 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[GetOpenOrderNumberByCustomer]

(@CustomerId int)

as

select OrderNumber from OrderHeader where CustomerID = @CustomerId and Status = 0

#### **Get products in the open order**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetShoppingList] Script Date: 08/03/2023 09:30:44 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[GetShoppingList]

(@OrderNumber nvarchar(50))

as

Select

ol.ID,

ol.OrderNumber,

ol.OrderLine,

ol.ProductID,

ol.OrderQuantity,

pm.ProductDescription,

pm.ProductPrice,

pm.ImageName

FROM

OrderLines ol join ProductMaster pm on ol.ProductID = pm.ID

WHERE

ol.OrderNumber = @OrderNumber

#### **Get or create an order**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetOrCreateOrder] Script Date: 08/03/2023 09:34:59 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[GetOrCreateOrder]

(@CustomerId int)

as

if exists(select OrderNumber from OrderHeader where CustomerID = @CustomerId and Status = 0)

begin

select OrderNumber from OrderHeader where CustomerID = @CustomerId and Status = 0

end

else

begin

insert into OrderHeader (

OrderNumber,

CustomerID,

Status) values (

(select max(ID) from OrderHeader) + 1,

@CustomerId,

0)

select OrderNumber from OrderHeader where CustomerID = @CustomerId and Status = 0

end

#### **Update order**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UpdateOrder] Script Date: 08/03/2023 09:36:07 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[UpdateOrder]

(

@ID int,

@OrderQuantity int

)

as

update OrderLines

set

OrderQuantity = @OrderQuantity

where ID = @ID

#### **Delete order**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DeleteOrder] Script Date: 08/03/2023 09:36:47 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[DeleteOrder]

(@ID int)

as

delete from OrderLines where ID = @ID

#### **Inserting new order lines**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[InsertOrderLines] Script Date: 08/03/2023 09:38:12 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[InsertOrderLines]

(@OrderNumber nvarchar(50)

,@OrderQuantity int

,@ProductID int)

as

insert into OrderLines(

OrderNumber,

OrderLine,

OrderQuantity,

Status,

ProductID) values (

@OrderNumber,

isnull((select max(OrderLine) from OrderLines where OrderNumber = @OrderNumber) + 1, 1),

@OrderQuantity,

0,

@ProductID)

#### **Mark order as ordered**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MarkOrderAsOrdered] Script Date: 08/03/2023 09:41:13 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[MarkOrderAsOrdered]

(@OrderNumber nvarchar(50))

as

update pm

set pm.StockQuantity = pm.StockQuantity - ol.OrderQuantity

from ProductMaster pm

join OrderLines ol on pm.ID = ol.ProductID

where ol.OrderNumber = @OrderNumber

update OrderHeader

set Status = 1

where OrderNumber = @OrderNumber

#### **Get order line from ID**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[OrderLineById] Script Date: 08/03/2023 09:46:05 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[OrderLineById]

(@Id int)

as

select ol.ID, ol.OrderQuantity, pm.ProductDescription, pm.ProductPrice from OrderLines ol

join ProductMaster pm on ol.ProductID = pm.ID

where ol.ID = @Id

#### **Get product types related to search term**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProductTypeSearch] Script Date: 08/03/2023 09:41:49 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[ProductTypeSearch]

(@SearchTerm nvarchar(100))

as

select distinct(ProductType) from ProductMaster where ProductType like '%'+@SearchTerm+'%'

#### **Get products under the product type**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProductDetailsByType] Script Date: 08/03/2023 09:41:42 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[ProductDetailsByType]

(@ProductType nvarchar(100))

as

select ID, ProductDescription, ProductPrice, ImageName from ProductMaster where ProductType = @ProductType

#### **Get product details from ID**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProductDetailsById] Script Date: 08/03/2023 09:44:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[ProductDetailsById]

(@Id int)

as

select ID, ProductDescription, ProductPrice, ImageName, StockQuantity, ReorderQuantity, DefaultOrderQuantity from ProductMaster where ID = @Id

#### **Get products low on stock**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProductMaster\_GetLowStock] Script Date: 08/03/2023 09:45:23 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[ProductMaster\_GetLowStock]

as

select \*

from

ProductMaster

where

StockQuantity < ReorderQuantity

#### **Order products with low stock**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[IncreaseLowStock] Script Date: 08/03/2023 09:48:16 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[IncreaseLowStock]

(@Id int, @DefaultOrderQuantity int)

as

update ProductMaster

set

StockQuantity = StockQuantity + @DefaultOrderQuantity

where ID = @Id

#### **Update product**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UpdateProduct] Script Date: 08/03/2023 09:49:27 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[UpdateProduct]

(

@Id int,

@ProductDescription nvarchar(100),

@ProductPrice float,

@ReOrderQuantity int,

@DefaultOrderQuantity int,

@StockQuantity int,

@ImageName nvarchar(100)

)

as

update ProductMaster

set

ProductDescription = @ProductDescription,

ProductPrice = @ProductPrice,

ReorderQuantity = @ReOrderQuantity,

DefaultOrderQuantity = @DefaultOrderQuantity,

StockQuantity = @StockQuantity,

ImageName = case when @ImageName = '' then ImageName else @ImageName end

where ID = @Id

#### **Delete product**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DeleteProductById] Script Date: 08/03/2023 09:49:57 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[DeleteProductById]

(@Id int)

as

delete from ProductMaster where id = @Id

#### **Add new product**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[InsertProduct] Script Date: 08/03/2023 09:50:20 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[InsertProduct]

@ProductCode nvarchar(100)

,@ProductDescription nvarchar(500)

,@ProductPrice float

,@StockQuantity int

,@ReOrderQuantity int

,@DefaultOrderQuantity int

,@ProductType nvarchar(30)

,@ImageName nvarchar(100)

as

insert into ProductMaster

(

ProductCode,

ProductDescription,

ProductPrice,

StockQuantity,

ReorderQuantity,

DefaultOrderQuantity,

ProductType,

ImageName

) values (

@ProductCode,

@ProductDescription,

@ProductPrice,

@StockQuantity,

@ReOrderQuantity,

@DefaultOrderQuantity,

@ProductType,

@ImageName

)

#### **Get products related to search term**

USE [ShoppingToYou]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProductDescriptionSearch] Script Date: 08/03/2023 09:50:52 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[ProductDescriptionSearch]

(@SearchTerm nvarchar(100))

as

select ID, ProductDescription from ProductMaster where ProductDescription like '%'+@SearchTerm+'%'

# **Testing**

## **Test 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 1 | 1.2.1  1.2.3 | Check that the login capabilities of the website work correctly and only authorise correct details and correct details are navigated to correct homepage (customer or staff) | Existing customer login: [customer@shoppingtoyou.com](mailto:customer@shoppingtoyou.com)  New customer login:  [newcustomer@shoppingtoyou.com](mailto:newcustomer@shoppingtoyou.com)  Staff login:  staff@shoppingtoyou.com | Existing customer is navigated to customer homepage. New customer is navigated to a customer details page. Staff is navigated to staff homepage. | Existing customer is navigated to customer homepage. New customer is navigated to a customer details page. Staff is navigated to staff homepage. |

Video - <https://www.youtube.com/watch?v=gpCpI3UOPEc>

## **Test 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | | Actual result |
| 2 | 1.2.2 | Check that any details entered for a new account are validated and any saved details are saved in the database | Username:  NewCustomer123 | Input boxes will display error messages if the input doesn’t follow validation requirements. If validated, the details will be saved to the ApplicationUser table in the database. | Input boxes display error messages if the input doesn’t follow validation requirements. The validated details are saved to the database and the customer is taken to the next page | |

Video - <https://www.youtube.com/watch?v=07onErWjyoE>

## **Test 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 3 | 1.3.1  1.3.2  1.3.3 | Check there is a list of products displayed for Product of The Week, check the search bar works and returns correct products with their images and information | Product types entered in search box:   * Soda * Chocolate | There will be products of the week displayed on the customer homepage, entering a valid product type in the search box will result in products under that type being displayed with their information and images. The search box will have autocomplete features. | Rotating carousel of product of the week images displayed on the customer homepage with their prices. Entering valid product types in the search box results in products under that type being returned and displayed along with their information and images. The search box has autocomplete features. |

Video - <https://www.youtube.com/watch?v=pIXDPUFr1p4>

## **Test 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 4 | 1.3.4 | Check that a customer is able to order a product and add to a shopping list that can be finalised with the click of a button | Product type entered in search box:  Dairy  Item ordered:   * Baby Bells * Milk   Amount ordered:   * Baby Bells: 3 packs * Milk: 2 bottles | Customer can enter the quantity they want to order of the desired product. Clicking the order button will result in the product being added to a shopping list, this shopping list can be completed by clicking the finalise button where the customer will be taken to an order confirmation page. | Customer enters the quantity they want to order of the desired product. Clicking the order button results in the product being added to a shopping list with the correct quantity, this shopping list is completed by clicking the finalise button and the customer is taken to an order confirmation page. |

Video - <https://www.youtube.com/watch?v=jrtEnGBCV6Q>

## **Test 5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 5 | 1.3.5 | Check that customers can edit any outstanding orders and delete products off the order | Change quantity of sausage to: 1 pack  Delete: Bacon | Clicking the edit button displays a modal popup giving the ability to edit the order, the modal popup will be validated. Clicking the delete button removes the product from the order. | Clicking the edit button displays a modal popup that gives the ability to only edit the product quantity, price and description fields are displayed but they aren’t changeable. The input box is validated. Clicking the delete button removes the product from the list and removes it from the OrderLines table in the database |

Video - <https://www.youtube.com/watch?v=t3J3d4wSqjo>

## **Test 6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 6 | 1.4.1  1.4.3 | Check that a list of products that are low on stock are displayed on the staff homepage and that staff has the ability to order more of the stock. | Order: pint of milk | Products with quantities less than their re-order quantity are displayed on the staff homepage. Clicking on the order button increases the quantity of the product by its default order quantity and the product is removed from the low stock list | Products with quantities less than their re order point are displayed on the staff homepage. Clicking on the order button increases the quantity of the product by its default order quantity and the product is removed from the low stock list |

Video - <https://www.youtube.com/watch?v=g23snBmzA4M>

## **Test 7**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 7 | 1.4.2 | Check that staff can use the search bar to find any product in the database | Input: Coke  Input: Cheese | When staff click the manage stock button, they will be taken to the search bar where the staff will enter ‘coke’ and all products related to that description will be displayed in a drop-down box (autocomplete feature). An input of ‘cheese’ will result in no output. | When the manage stock button is clicked, staff are taken to the search bar where they enter ‘coke’ and all products related to that description are displayed in a drop-down box (autocomplete feature). When ‘cheese’ is entered, nothing is returned because it is not in the database. |

Video - <https://www.youtube.com/watch?v=84Eyd3N4P-w>

## **Test 8**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 8 | 1.4.4 | Check staff is able to add a new product to the database by entering all the new products details in a modal popup | Product code: ChedCheese50g  Product description: Cheddar Cheese 50g  Price: £1.99  Quantity: 300  Re order quantity: 100  Default order quantity: 150  Product type: dairy  Image name: cheddar.jpg | When staff clicks the add product button, a modal popup will be displayed allowing staff to enter the new product’s details in the boxes, which are validated. Once the create button is clicked, the product’s details will be saved to the database. | When the add product button is clicked, a modal popup is displayed which allows staff to enter a new products details into a form, each input box is validated. Once the save button is clicked, providing the inputs are validated, the product’s details are saved to the database. |

Video - <https://www.youtube.com/watch?v=yQtE4k_e4qM>

## **Test 9**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 9 | 1.4.5 | Check staff is able to edit a products details successfully | Product to edit: Cheddar Cheese 50g  Change price: £2.99  Change default order quantity: 300 | When the edit button is clicked, a modal popup will be displayed giving the option to change the products details, these input boxes will be validated. Clicking the save button will save any changes to the database and staff will be taken back to the manage stock page | When the edit button is clicked, a modal popup is displayed giving the option to change the products details, these input boxes are validated. Clicking the save button saves changes to the database and staff is taken back to the manage stock page |

Video - <https://www.youtube.com/watch?v=d5GMTDocbkc>

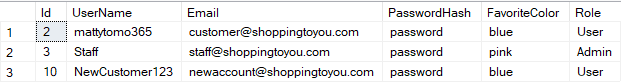
## **Test 10**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test  No | Objective No | Purpose of test | Test Data | Expected result | Actual result |
| 10 | 1.4.6 | Check staff can delete products from the database | Product to delete: Cheddar Cheese 50g | When the delete button is clicked, the product will be deleted from the database and removed from the website | When the delete button is clicked, the product is deleted from the database and the website, as proven when entering ‘cheese’ in the search box returns nothing |

Video - <https://www.youtube.com/watch?v=ZRWM9zBqUfw>

## **Database tables**

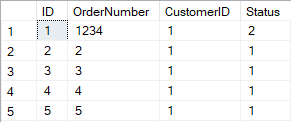
### ApplicationUser



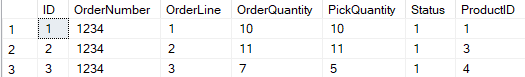
### Customers



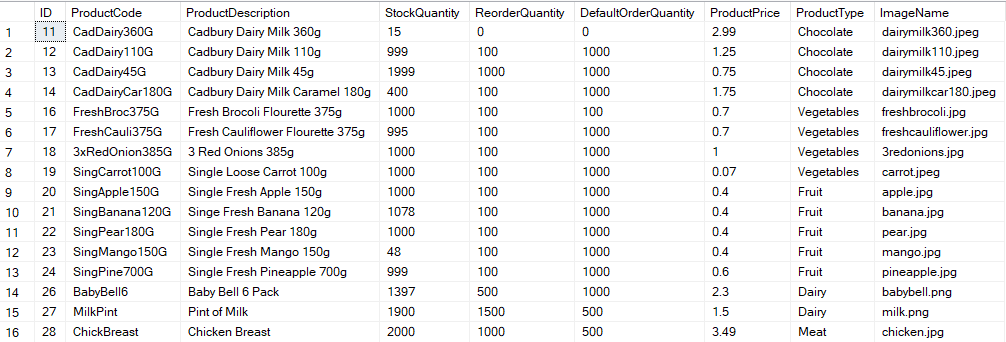
### OrderHeader



### OrderLines



### ProductMaster



# **Evaluation**

## **Objectives achieved**

1. **Create an online shopping website**
   1. User-friendly interface
      1. single theme
      2. easily identifiable buttons to give more information on products
      3. simple/clean design

All 1.1 objectives have been completely met, my website implements a single theme, the interface is simple and clean. Buttons are easily identifiable, as they are a different colour to the rest of the page. This was achieved using CSS styling.

* 1. Log in capabilities
     1. login if customer already has an account
     2. ability to create a new customer account

This objective was fully achieved through the creation of a login page, which is a form, allowing customers to enter details for a new account, these form inputs are validated prevent nulls. The data entered is then submitted and stored in the ApplicationUser table in the database using SQL stored procedures.

* + 1. ability to log in as a customer or a staff member, giving access to different features

This objective and 1.2.1 were fully achieved by authenticating using claims based access tokens to authenticate in subsequent requests until logged out

* 1. Customer accounts
     1. a special feature such as: products of the week

This objective was partially achieved through the implementation of a bootstrap image slider that displays random products. This could be improved by turning the images into buttons and allowing customers to order products displayed on the slider

* + 1. pictures of products

This was fully achieved through storing images in the file system and storing the path to these images in the ProductMaster table in the database, they were displayed using HTML

* + 1. the ability to search for products using a search box

This objective was fully achieved using jquery autocomplete. I have 2 of these with different behaviour, on the staff side of the website, the autocomplete filters on product descriptions whereas on the customer side of the website, it filters on product type. This

* + 1. ability to add products to a list and click on a finalise button to complete order

This was fully achieved through implementing the search bar which returns a list products with order buttons and quantity input boxes next to them, the order button submits the product information into a new order line which is added to the OrderLines table in the database this is done through an SQL stored procedures which create new order lines and check for any open orders before creating a new one. When the finalise button is clicked, an SQL stored procedure is used to change the status of the order. The customer is then directed to a thank you page.

* + 1. ability to edit/delete any outstanding orders

This was fully achieved through implementing a modal dialogue popup displaying a user input form, where the user can input and edit order quantity, the form input is validated to prevent nulls, there are also two other form inputs which are disabled and are used to display information about the order line being edited. Once submitted, the details are saved to the OrderLines table using SQL stored procedures. Delete buttons are next to each item in the open order, once clicked, the order line ID is passed into an SQL stored procedure which removes the order line from the OrderLines table in the database

* 1. Staff accounts
     1. can view products that are currently low on stock

This was fully achieved using an SQL stored procedure to return a list of products with quantities less than their re order quantity

* + 1. ability to search for products using a search box

This objective was fully achieved using jquery autocomplete. I have 2 of these with different behaviour, on the staff side of the website, the autocomplete filters on product descriptions whereas on the customer side of the website, it filters on product type. Both autocomplete features return a drop down box with returned products, once the products in this drop down box are clicked, a partial view is used to display the details of the products.

* + 1. ability to increase stock levels of a certain product

This was fully achieved by implementing an order button next to each low stock product on the staff homepage, when clicked, an SQL stored procedure is used to add the products default order quantity to its quantity

* + 1. ability to add a new product
    2. ability to edit current products

These were fully achieved through implementing a modal dialogue popup displaying a user input form, where the user can input product details, the form inputs are validated to prevent nulls, once submitted, the details are saved to the ProductMaster table using SQL stored procedures

* + 1. ability to delete products

This is fully achieved by implementing delete buttons next to each product returned from using the search box, once the delete button is clicked, the product ID is passed into an SQL stored procedure which removes the product from the ProductMaster table in the database

1. **Create database**
   1. add password protection
   2. create tables containing all data such as accounts and orders
   3. create procedures to maintain and list data

Achieved through implementing a series of SQL stored procedures.

## **Feedback from end user**

Strengths:

“Nice and simple user interface, with fast search ability. Items I have ordered are easy to change with the buttons next to them, making it easy to delete items that I ordered, I liked the fact that the interface was free of distracting adverts. There was no prepopulation of the basket.”

Improvements:

“I would like to select multiple products from the search results before returning to the homepage, I would like a wider selection of products. I would like to see price per unit of measurement, ingredients, allergens and nutritional information.”

End user signature:

## **Summary**

My application has many strengths, all of my objectives were met (fully and partially), the user interface is clean and simple, there are no distractions to the customer, making it easier to navigate. Ordering products can be easily achieved through the click of a button, and any open orders are displayed to the customer.

## **Improvements**

The search boxes on both the customer and staff side of the website could be extended to make them more intelligent. I could also have displayed more information about products, with the addition of an extra page when the customer clicks on the product which could display ingredients, allergens etc.