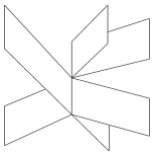


Creation of a Supermarket Web Application Based on ASP.NET

Iacopo Ermacora – 335141

Merlin Joshua Meyer – 335069

Allan Henriksen

14.162 Characters

IT-WEE1-S23

09.05.2023

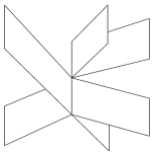
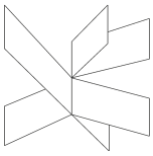
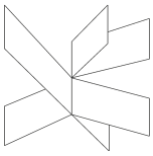


Table of Content



List of figures and tables

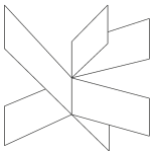


1 Introduction

The intention of this assignment is to develop a web application in ASP.NET, based on a fictional use case made by the students themselves. Through the identification of relevant stakeholders and the derivation of their requirements – functional as well as non-functional – models and diagrams shall be drafted. After the design of basic mock-ups and the attempt to translate those, along with the set requirements, into code, a functioning web application will be set up.

To exemplify the competencies gathered in the WEE course, the students built a web application with functionality similar to that of an online supermarket. Through the implementation of functionalities such as a product overview, a session-bound shopping cart and the possibility to create and administer a user profile, as well as many others, all competencies taught in this course are intended to be implemented.

In the outlined process, this report intends to act as an explanation of the thought processes and their inherent outcomes. Therefore, this report will start by introducing the reader in more detail to the stakeholders of the intended web application, as well as the requirements each of these user groups has. It will then go on to explain and showcase the models and diagrams that build the underlying basis during the implementation of the resulting code. Before describing the process of implementing the requirements into code and inherent challenges, a few examples of the intended design of the website, as well as a description of the underlying intentions, will be given.



2 Stakeholders and Requirements

To best be able to showcase as much functionality as possible with the competencies gathered and methods learned in the WEE course, three stakeholders were identified, that can be further distinguished into two groups.

2.1 Functional Requirements

On the side of the supermarket, there are the administrators who, when put into laymen's terms, have all the access needed to administer the other two target groups as well as themselves. Their functional requirements can be specified as follows:

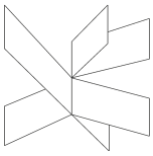
- As an administrator, I must be able to register and modify order pickers in the system.
- As an administrator, I must be able to implement new products into the system, as well as manage existing ones.
- As an administrator, I must be able to add, modify and delete existing consumer users.
- As an administrator, I must be able to look at all the orders that were placed and have the ability to delete them.

The second stakeholder on the supermarket's side is the group of order pickers. In other words, the employees responsible for picking the products the customer specified in their order and then getting them ready to be shipped at a later stage. Their functional requirements were identified as the following:

- As an order picker, I must be able to see the products of the order I'm packing.
- As an order picker, I must be able to see a product's alternative.
- As an order picker, I must be able to see where to find a product in the warehouse.

These are the stakeholders acting on behalf of the online supermarket. But, just as any good supermarket, this fictional online supermarket also has a platitude of customers to serve, which act on behalf of the group specifiable as consumers. These customers, the third and last target group in this assignment, have the following functional requirements:

- As a customer, I must be able to see the details and prices of products.
- As a customer, I must be able to see whether products are still in stock.
- As a customer, I must be able to add and delete products to my cart.
- As a customer, I must be able to submit an order on the website.



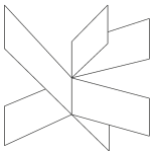
- As a customer, I must be able to view my past orders.

2.2 Non-Functional Requirements

Apart from the several functional requirements, the web application is supposed to meet certain non-functional requirements as well. These non-functional requirements mostly target all stakeholders to the same degree unless specified otherwise. The identified non-functional requirements look as follows:

- The system must require login to ensure only authenticated users can make actions
- The system must have different roles for different people to give different access to pages
- The system must present only the most recent data
- The system must not allow customers to order products registering stock unequal to the amount the customer intends to order.
- The system must have a shopping cart/session functionality.
- The system must have a registration function for users to register themselves within the system (incl. address).

These requirements, functional or non-functional, will help not only design the databases holding all the information required for the website to operate fully but also in drafting the individual web pages and thereby giving every stakeholder the information that he needs within a few clicks.



3 Models and Diagrams

3.1 Use Case Diagram

For a superior understanding of the stakeholders and their respective requirements, a use case diagram was drafted. In this diagram also the “Anonymous” user is mentioned for the first time. Every user browsing the page without being logged in can be classified by the system as Anonymous, as there are no roles and therefore inheritances or rights attached to their person.

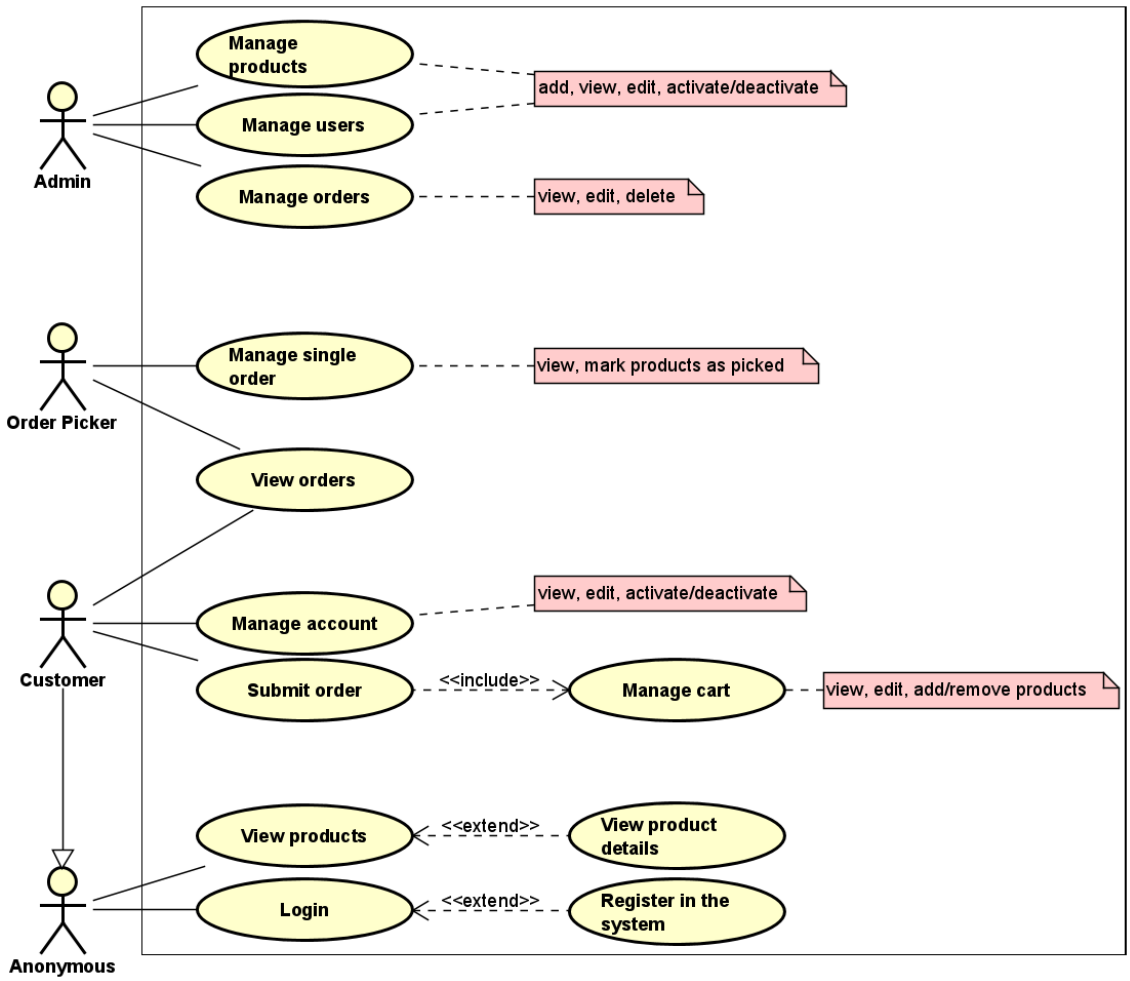
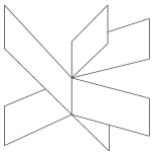


Figure 1, Use Case Diagram



3.2 Activity Diagram

As within the scope of the assignment, there are almost no flows, due to most actions within the web environment being atomic, only one Activity Diagram was drafted (s. Figure 2). It is supposed to give an insight into the steps of a customer finishing his order and where the action leads next once the order is submitted.

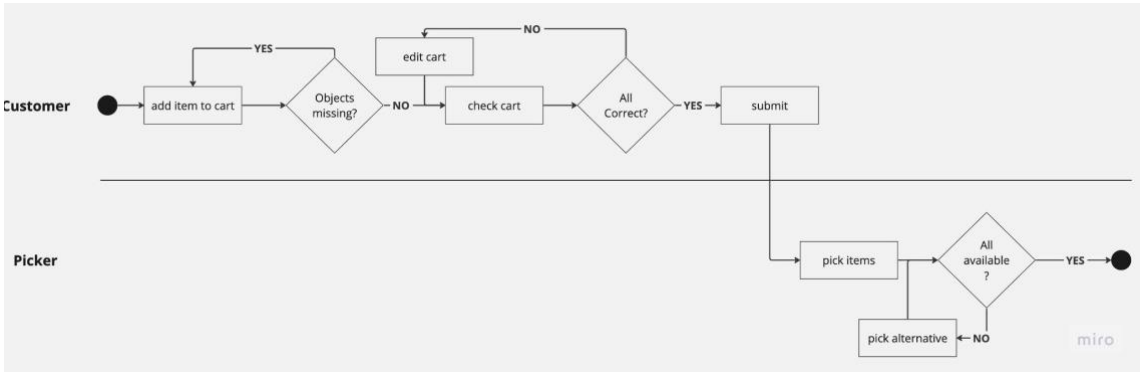


Figure 2, Activity Diagram

The flow starts from the point where a customer adds an item to their cart. Should after this step any items be missing that they might desire, they can add more items to their cart until they are satisfied with their potential order. This check if their cart is fulfilling their wishes, can already be seen as the next step: “check cart”. Should anything be unsatisfactory during the cart check, they can go back to edit their cart, such as deleting products from it or editing the quantity of products. Should all be correct, they can submit their order to the supermarket. After submission, the order will be forwarded to one of the supermarket’s order pickers. They will start to pick up the items from their respective location within the storage. Should not all items be available, they must pick up an alternative product. If all items are available, or the alternatives have been picked, the order is ready for further processing.

4 Design

This Chapter will handle the design of the web application, featuring topics such as database design with the help of an EER diagram, but also touch upon the UI design for the different users of the application. This short deep-dive into the UI decision, however, only serves as an explanation for the inherent design decision in regard to data security.

4.1 EER Diagram

As an overview and clearer explanation of how data flows within the constructed system and how single tables are connected to one another through keys, an EER Diagram was drawn up.

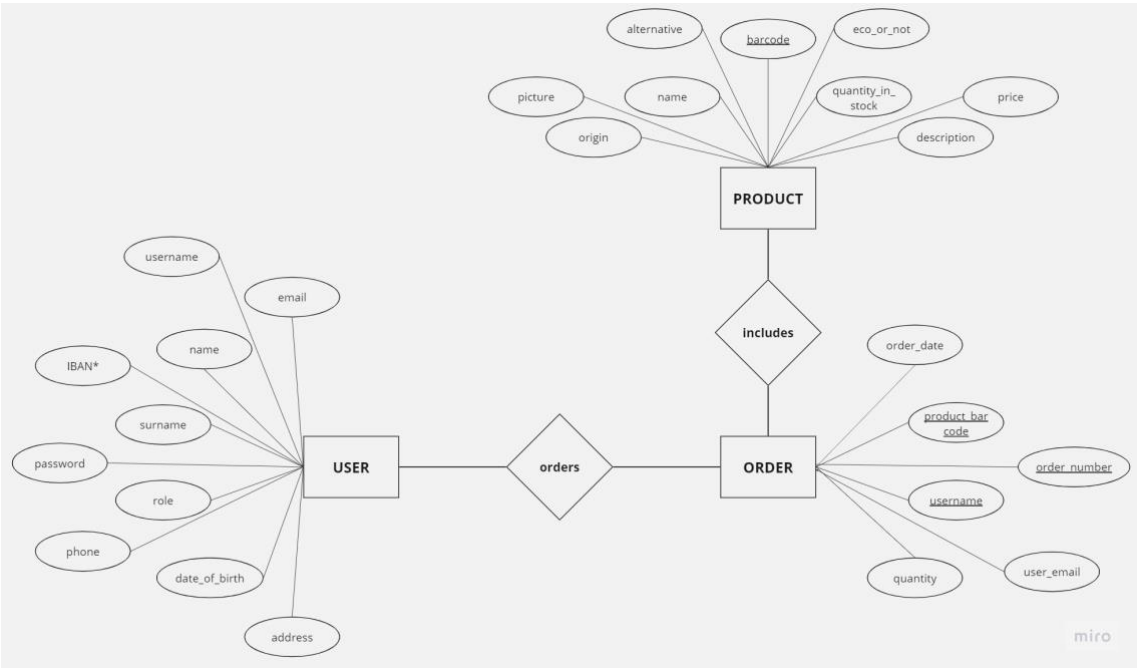
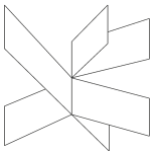


Figure 3, EER Diagram

Every user registered within the system holds a distinct amount of information, making them individual within the system. The key for the user table is their respective username. This username will also be represented within the order table, once they place an order, ultimately making the order traceable back to the user. The order table of course also holds information about the products included in the order. This is information such as the product barcode as well as the quantity ordered of each product with it. Furthermore, an order number and an order date are assigned to every order



within the table. The aforementioned product barcode is also the foreign key of the product table. This table, as the name suggests, holds further information about every product the supermarket offers. It includes information such as the alternative product, the quantity currently held by the supermarket, its name and price etc.

With these diagrams sorted out and a deeper understanding of the website to be designed, decisions on the front end can be made. Within the next chapter, a few examples of the design choices will be shown and explained. Additionally, a decision on which user can access what will be taken.

4.2 Security Design

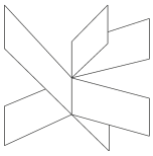
The design of the pages and the underlying principle are mostly based on securing the integrity of the website. For this, the simple rule of giving the respective user what they need without giving more information than really necessary is being followed. Otherwise, the non-relevant information could be used for unfavourable purposes by the users. The table below gives an extensive insight into which user is able to access and use what page. The given alphanumerical indicators are connected to Appendix A.

Table 1, Users and the Pages they can reach

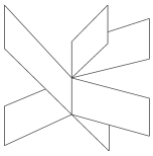
Admin	Order-Picker	Customer	Anonymous
A1, A2, A3, A4, A5		A2	C1, C2
B1, B2	B1, B2	B1, B2	E1, E2
C1, C2, C3		C1, C2	
		D1, D2	

As an example, while the admin naturally has access to almost all pages, they do not need access to a cart function or the subsequent affirmation of order submission. The order-picker is an especially good example of the underlying security principle. Even though they might be employees of the supermarket, they do not need any data about the customers, as they would have no means of processing it further. Therefore they only have an overview of the orders that were assigned to them within the system, as well as a detailed view of each order, to find the location of products and so on.

Furthermore, as was specified in Chapter 2.2, the web application must require login to ensure only authenticated users can make actions. As this requires the possibility



of registration, implying users of the application might not have a role assigned yet, this group of users is featured in Table 1 as Anonymous. As visible in Chapter 3.1 already, they will also be able to see the products page and check products individually. However, their actions are strictly limited to these two actions.



5 Implementation

For the actual implementation of all aforementioned ideas and considerations, a platitude of different coding concepts was utilized, trying to get the most out of ASP.net and finding workarounds for whatever posed a challenge within the software's boundaries. Within the course of this chapter, some of the challenges encountered and the concepts used will be explained in more detail. The coding concepts implemented for this assignment are the following:

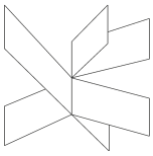
- Login and register through ASP memberships
- Masterpage
- Treeview and sitemaps
- Webconfig settings
- Cart implementation via sessions and the use of a custom class
- Different data displayed thanks to "login view role groups"
- Data displayed with listview (with custom queries also based on user roles and other factors)
- URL Query-based pages
- Custom error pages
- Data creation, modification and deletion through forms and database interaction (including image management)
- Implementation of charts for data display

5.1 Cart Implementation & Session Function

The cart function for the online supermarket was implemented based on sessions, making it necessary to put the data about each product placed in the cart into the database before being ordered. Therefore, a custom class called "Cart_item" was created, defining the model of the data that the session would store. This "Cart_item"-class is composed of four attributes:

- product_barcode (long)
- product_name (string)
- product_quantity (integer)
- product_price (integer)

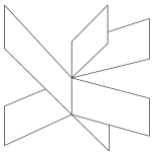
It would have been easier to just include the barcode and the quantity within the "Cart_item". However, it was decided to include all the data that had to be displayed, making a call to the database for every row of the cart every time it is to be displayed



obsolete. Within the “cart” session, a list of “Cart_items” can be found, which is updated every time an item is added to the cart by the customer.

The implementation of this cart function required consideration of the non-functional requirement to always display the most current data. This of course brings up the question of quantity in stock of a specific product. As a supermarket stock can never be infinitive, a quantity in stock was defined for every item offered. This of course means that a user must not be allowed to add more products of a kind to their cart than the supermarket has in stock at that moment. This problem can arise in three different situations. In all cases, it was treated similarly, but at the same time a bit differently:

- Once a user adds a product to their cart. If a user added more than one piece of a product at a time, the items put within the cart could exceed the amount in stock. → Inside the function that adds a new item to the cart once the “AddToCart” button is pressed, a check was implemented. It retrieves the quantity of a product currently held in stock and then does not allow the user to increase the quantity anymore. (See Appendix 2)
- Once a user incrementally increases the quantity of a product in their cart page. This could lead to them adding a quantity of a product that the supermarket does not hold a sufficient amount of. → Here, the solution is implemented in the same way as with the problem before. (See Appendix 2)
- Even with the previous checks regarding the quantity held in stock, there could be a change in the quantity within the database, leaving a product out of stock, that someone else might have in their cart. → In this case, once the customer clicks “Order”, trying to insert the data from the cart into the database, another check of each item's quantity in the cart and in the stock of the supermarket is performed to see if it is still available. All the items that were not inserted into the database and thereby technically not ordered by the user, are displayed to them on the confirmation page right after the order. The user could then decrease the quantity in the cart and try to order again.

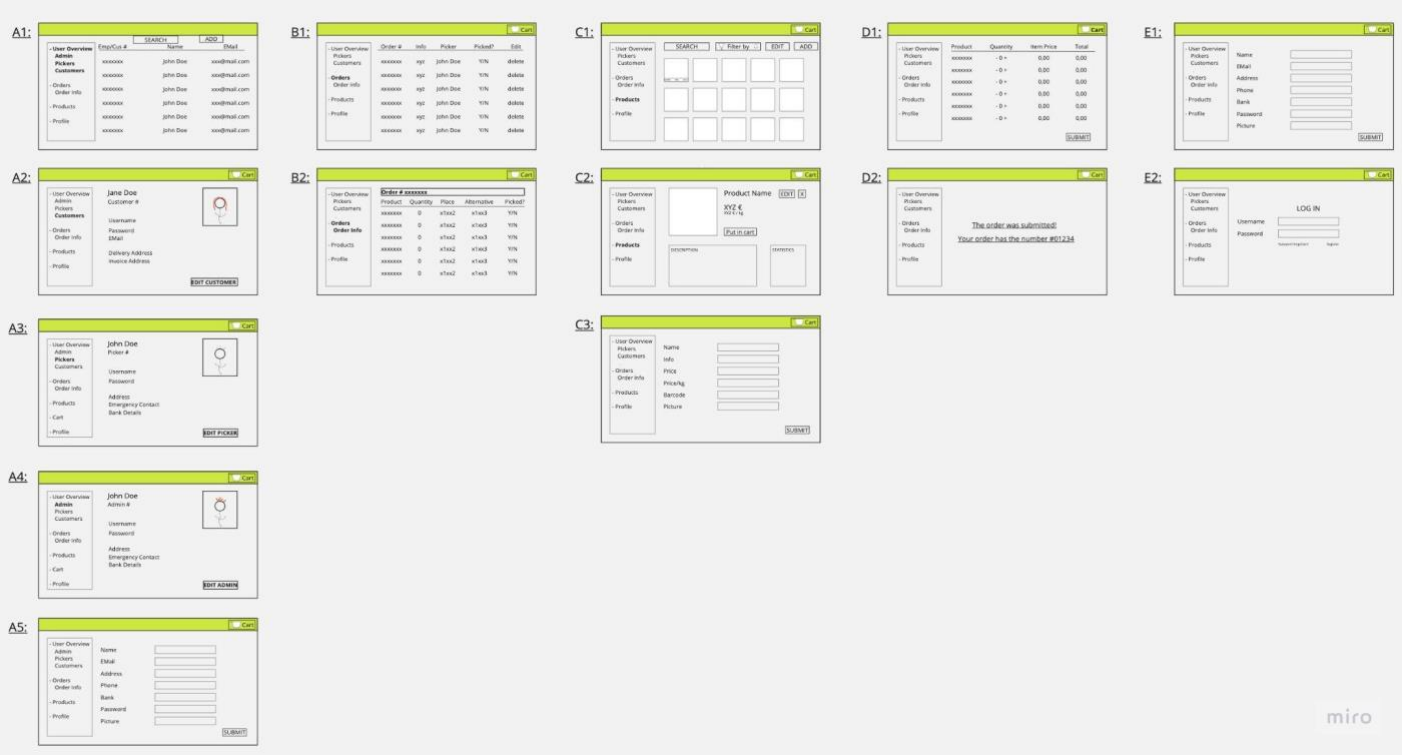


5.2 URL Query-based pages

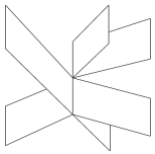
To be able to display data based on requested conditions, URL parameter queries were utilised. The implementation of user profiles can be used as an example for this: the username is passed to the system as a query and the correct user data then gets displayed. A security issue often connected to this approach, as the page is accessible for both customers and admins, is that the user, who should only be entitled to see their own data by knowing the username of another user, would be able to get a look at their data too.

To work around this obvious security issue, a check before querying the data from the database is implemented in the code. Making it only possible for users with the role of an admin to read the query parameter. Otherwise, the query searches for the username of the current user, independently from the parameter that might be specified in the URL.

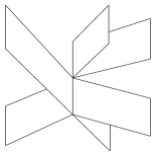
Appendices



Appendix A: Page Overview (please also refer to Table 1)



```
<!-- Connects to database ->

string sqlStr_check_stock = "SELECT quantity_in_stock FROM Product WHERE barcode = " +
found_item.product_barcode;

<!-- Open connection to database →

int stock_quantity = 0;
//Reads data from database
 if (DR1.Read())
   {
      stock_quantity = Convert.ToInt16(DR1.GetValue(0).ToString());
   }

<!-- Close all the connections ->

//If there are enough products in stock, add the quantity requested to the list (later inserted in the
session), otherwise just add the item that was removed from the list to increase their quantity

if (found == 1 && stock_quantity > found_item.product_quantity)
     {
         cartItem.product_quantity = found_item.product_quantity + 1;
         cartItem.product_name = found_item.product_name;
```



```
        cartItem.product_barcode = found_item.product_barcode;
        cartItem.product_price = found_item.product_price;
        cart_list.Add(cartItem);
    }
     else if (found == 1 && stock_quantity <= found_item.product_quantity)
    {
        cart_list.Add(found_item);
    }
        else
    {
    //if there was no item already in the list, add a new one
    }
```

Appendix B: Code-solution for the quantity in stock problem