

Report

FitFood

Dmai0914 | Group 18

Iliyan Stoev

Date: 6/6 2016

# Introduction

21st century is the century where computer technologies are involved in almost every daily life activity of the world’s population. With such a high availability, they have become inseparable part of almost everyone’s life, therefore the demand of software serving different purposes is really high, starting from something simple like

# Problem Statement

Fitness industry is becoming more and more popular in 21st century. It is already a consistent part of people’s daily life. As they are becoming more and more involved in it, at some point the interest of keeping track of the calorie intake during the day is growing. Often people don’t have the time to prepare their food, so they are ordering whether from restaurants or fast foods, where can’t be seen how many calories they are eating, therefore the track of the whole day/week is getting lost.

Considering the above mentioned information, this project is aiming on formulating and answering a problem statement that will improve the issue of computer technologies use in relation to the process of keeping track of what is going on with the food intake during people’s busy life, as well as staying informed about the macros while not having the time to prepare their own food.

The problem statement is formulated as follows: “How can people be informed in the area of concern and track their macros on a way, by using computer technologies? How can restaurants and fast foods be more attractive to people interested in staying fit?”

The idea is to design and develop an online platform that would allow people to keep track on how many calories they are putting in, when there is no time to prepare their own food. The platform’s main target group are people who are interested in improving their overall health and/or staying fit, as well as people highly involved in the fitness industry/life, allowing them to see the calorie/macros intake during a meal ordered from a restaurant or a fast food’s place. In what concerns the implementation of the platform, people should have an internet connection in order to access the website. The main purpose of this online platform is to give people the ability to see the macro nutrients / calories of the desired meal before even ordering it, which data is already being stored on a database. They will be able to choose product/s, add it/them to the basket, and finally place the order. All of the orders will be stored for account history purposes.

For the actual implementation of the software, the group decided to build the project in ASP.NET since web development is subject of interest for the members, and the framework is one of the subjects studied in the current semester.

Furthermore, additional software will be required on restaurant’s side in order to finalize the payment and complete the customer’s order, however it would be another project to be made and will not be covered in the current report.

# Domain Model

Domain Model is a starting point of every project, since it is representing the actual business classes going to be used, as well as the communication between them, in order to create working and useful software.

After reviewing customer’s demands the following domain classes where designed : \_User, Orders, OrderLine, Product, Restaurant, Contact.

The classes and their attributes were given meaningful names, so if a new developer appears in the current project, it will be easier and faster to integrate him/her.

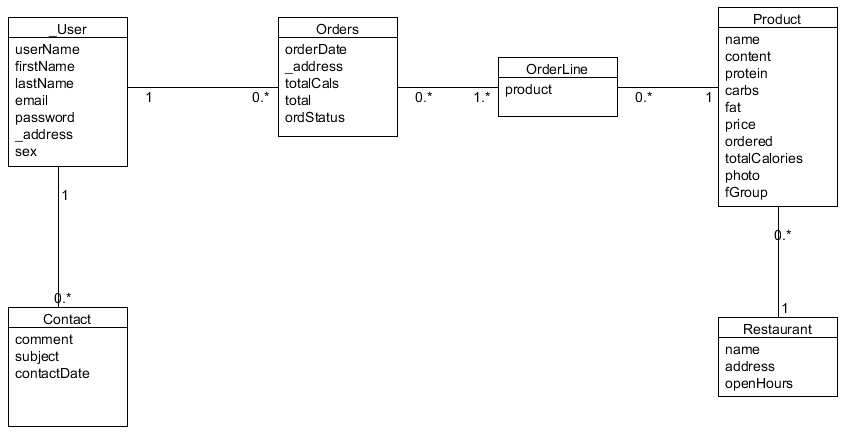


Figure 1 Domain Model

**\_User :** Creation of object of type \_User, containing a total of seven attributes : userName, firstName, lastName, email, password, \_address and sex.

* userName : used to allow the user to log in the system.
* firstName : stores the user’s first name.
* lastName : stores the user’s last name.
* email: stores the user’s email, also involved in sending contact email functionality.
* password: used in addition with userName to fullfil a pair of credentials needed for the login functionality.
* \_address: stores the user’s address, also involved as a parameter in the creation of an order
* sex: stores the user’s sex.

**Orders :** Creation of object of type Orders, containing a total of five attributes : orderDate, \_address, totalCals, total and ordStatus.

* orderDate : used to store the order’s creation date
* \_address : stores the address where the order should be delivered
* totalCals : stores the total calories of the order
* total : stores the total price of the order
* ordStatus : variable of type boolean which exposes the status of the order ( true = Delivered, false = InProcess )

**Product :** Creation of object of type Product, containing a total of ten attributes : name, content, protein, carbs, fat, price, ordered, totalCalories, photo and fGroup.

* name : stores the name of the product
* content : stores the content of the product
* protein : stores the protein in grams of the product
* carbs : stores the carbs in grams of the product
* fat : stores the fat in grams of the product
* price : stores the price of the product
* ordered : stores the times the product has been ordered, also used in the most ordered product functionality
* totalCalories : stores the product total calories
* photo : stores a photo of the product
* fGroup : exposes the food group the product belongs to.

**Restaurant :** Creation of object of type Restaurant, containing a total of three attributes : name, address and openHours.

* name : stores the name of the restaurant
* address : stores the location of the restaurant
* openHours : exposes the hours in between the restaurant is open

**Contact :** Creation of object of type Contact, containing a total of three attributes : comment, subject and contactDate.

* comment : stores the actual message within the contact
* subject : stores the subject of the contact
* contactDate : stores the date when the contact has been sent

**OrderLine** : Creation of object of type OrderLine, which is used as an agreement between Order class and Product class giving the functionality to insert more than one product in a signle order. OrderLine has no attributes which are of any business value, therefore they are not shown in the domain model.

## Associations

Class **\_User** has associations with classes : **Orders** and **Contact**. User can have zero to many contacts, where contact can have only one user. User can have zero to many orders, and order can have only one user at the time.

Class **Orders** has association with class : **OrderLine**.Order can have one to many order lines, and order line can be in zero to many orders.

Class **Product** has association with classes : **OrderLine** and **Restaurant**. Product can be in zero to many order lines, where order line can have only one product. Product can be offered by only one restaurant, and restaurant can offer zero to many products.

Class **OrderLine** is in association with classes : **Order** and **Product**.The class has been used as an agreement between order and product, in order to allow more than one product to be ordered with a single order, also order line cannot exist without order.

# Architecture

In the software development cycle of an application the very first step after collecting all the requirements wanted by the customer, is to design an appropriate architecture. Well thought architecture will ensure high performance, flexibility, stability and excellent maintenance.

In the current project classical “Client – Server” architecture was chosen. This type of design allows clear separation of tasks between the providers (Servers) and consumers (Clients), where an appropriate amount of logic should reside on both sides.

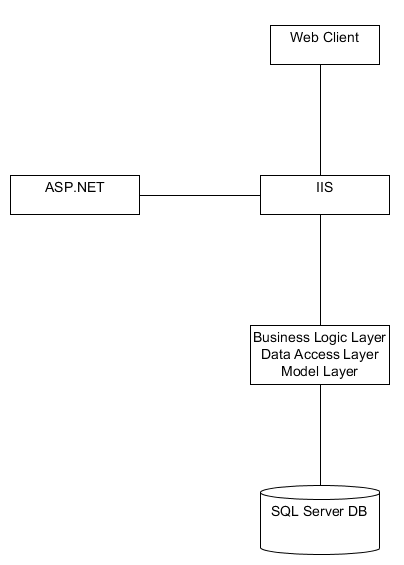


Figure 2 System Architecture

As can be seen on “Figure 2”, within the main architecture, there are also several other from the same type. For example : “Web Client” is a client to the “IIS” which is the server, “IIS” however is a client for “BLL, DAL, and Model Layer”, where “BLL, DAL, and Model Layer” is the client for the database SQL server.

Within the main architecture, a three tier pattern can also be noticed. The “Web Client and IIS” are playing the role of the first tier, where “BLL, DAL, and Model Layer” and “SQL Server DB” are respectively the second and third layers. The second pattern of the same type relies in “Business Logic Layer”, “Data Access Layer” and “Model Layer”.

Three tier type pattern is intended to segregate functionality between different parts, which can be placed on physically separate hardware. Even though in the current project all the tiers are located on the same machine, it is still possible for them to be separated. Because of the fact that each tier is independent and can be managed separately, changes in one of them does not affect the system as a whole, which gives excellent maintainability and flexibility, where at the same time assures low coupling.

## Business Logic Layer

Representing the business logic within the system. Main task of the layer is to receive information from either the Presentation Layer of the application or the Data Access Layer, do something with it and pass it respectively to one of both mentioned above.

## Data Access Layer

This layer of the three tier pattern deals with the database connection as well as querying the database with different statements : Insert, Update, Delete etc.

## Model Layer

Even though in the current project the Model Layer classes have been created by LINQ to SQL technology behind the scenes, it still serves the same purpose as it is intended to : representing the business entity classes.

# Implementation

The current chapter will cover the actual code (implementation) of the application developed.

## User CRUD (Create, Read, Update, Delete)

### Create

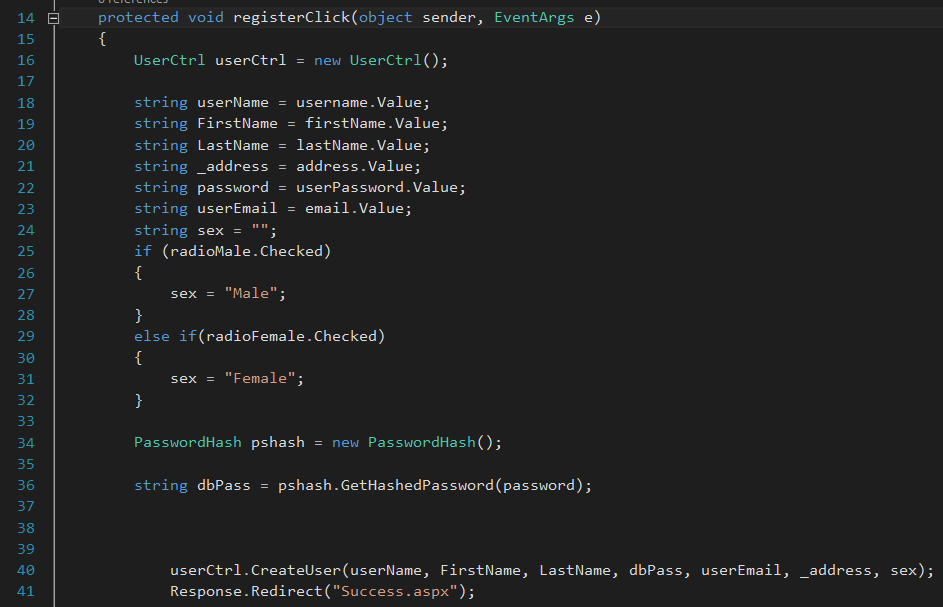


Figure 3 Register User UI

The code above is taken from the code behind class of the Registration Page. As can be seen when the user presses the “Register” button, the method above has been called.

Firstly, a new instance of the UserCtrl class has been created, so its methods can be available for use. The following lines of code “18 – 32” are representing the storing of values, taken from the text boxes on the web page, and store them in local variables, which are used later on in the method body.

On line 34 a new instance of the PasswordHash class has been created.

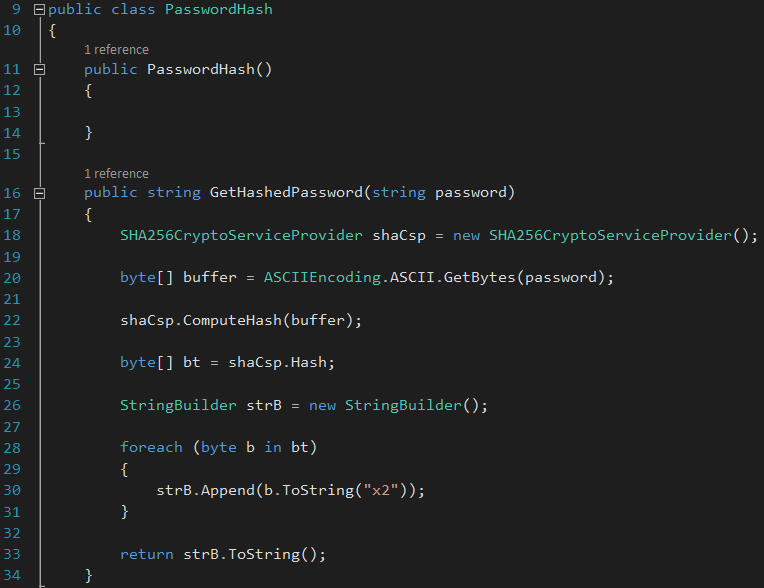


Figure 4 Password Hash

The PasswordHash class is a helper class with a single method inside “GetHashedPassword”. Using the SHA256, the method takes the user password as a parameter, and at the end returns the already hashed version of the password, which is going to be used as the one saved to the database for security purposes.

Back to “Figure 3” on line 40 of the code, the UserCtrl has been called with the method “CreateUser”, passing all the parameters.

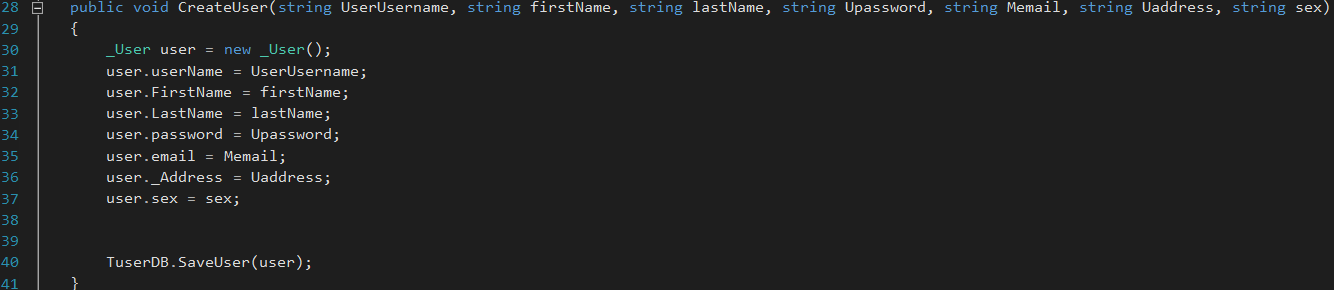


Figure 5 Create User (Controller)

The method above has the responsibility to create a new \_User object and fill it in with all the values passed from the user interface. At the end of the method body, a method from the according Data Access Layer has been called to finalize the user creation.

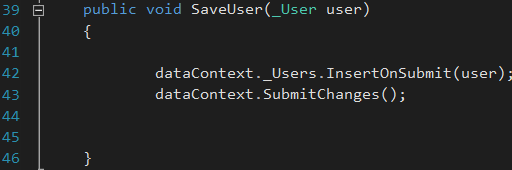


Figure 6 Save User ( User DB )

The task of the method above is to deal with the database insertion of the “user” object. Calling “InsertOnSubmit” on the current DataContext will insert the object in a pending state, which later on is finalized by the “SubmitChanges” method.

### Update

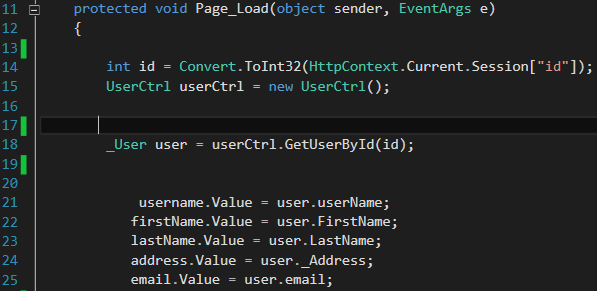


Figure 7 Update User (Page Load)

The Update user functionality starts firstly on the page load of the UpdateUser page, where is important to mention, that the page is not available, if a user is not logged in (which functionality is covered later). On Page Load, a local variable “id” is declared and assigned to the current Session “id” (saved in a session, after the user logs in, covered later in the Login Functionality), as well as the creation of new instance of the class “UserCtrl”. On line 18 the READ functionality occurs with the method “GetUserById”, which as the name suggests returns a user object from the database found by id and saves it in a local variable “user” of type “\_User”. Lastly the “user” object attributes values have been assigned to the according text boxes.

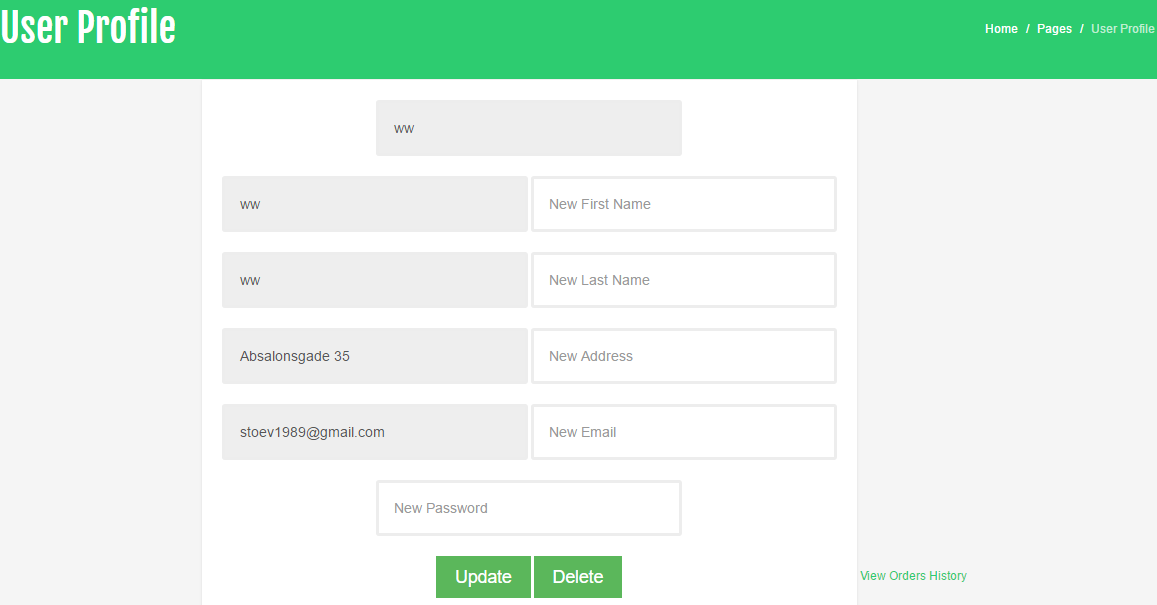
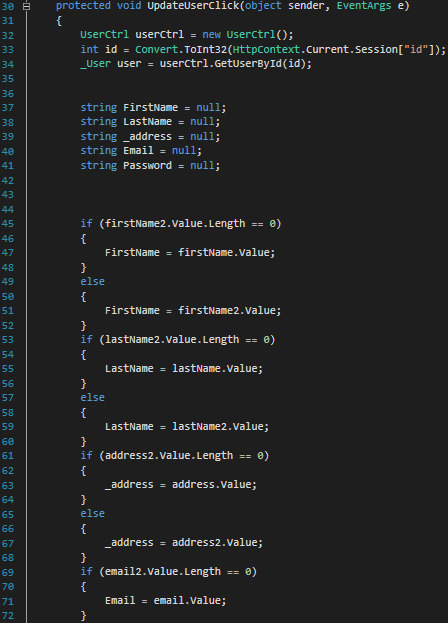


Figure 8 Update User UI View

The update functionality continues when the “Update Button” click has been performed.



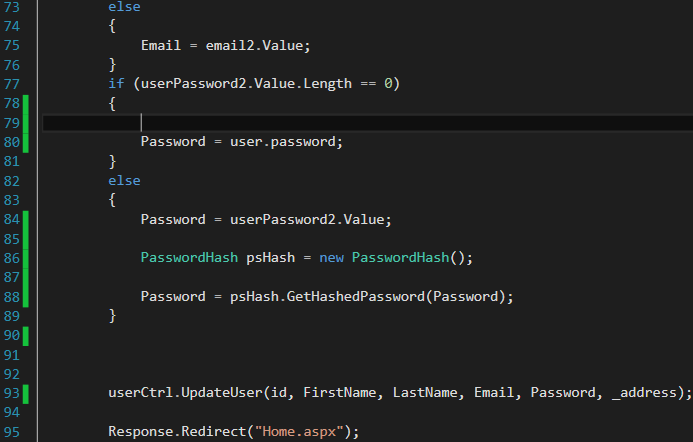


Figure 9 Update User Code Behind

After finding again the user by id, local variables for all the “user” properties have been declared. The method continues with performing a check for each text box value, if it is empty the according local variable is assigned the value of the original one (which was loaded from the “user” object on Page Load), else the new value has been taken. This was created, in case the user wants to change only a few fields, not all of them. The interesting thing here is the password, which is the main reason why the “user” object was loaded again in the beginning of the method. Since SHA256 is not reversible and it is used in the hashing of the user’s password, if a user does not want to change it, it has to be the same as it was before, therefore it must be loaded directly from the object. If the user wants to change the password, the order of its hashing is the same as in the Registration. The method body finishes again with hashing of the new password, and calling the “UpdateUser” method on “UserCtrl”, with all the parameters.

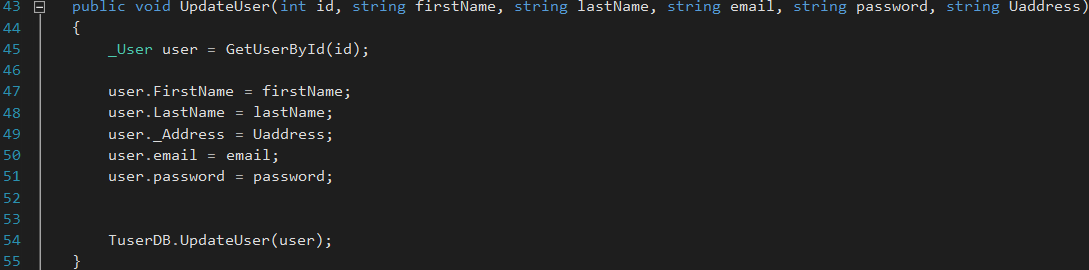


Figure 10 Update User ( User Controller )

In the User Controller, the method starts again with finding the “user” object with the passed from the User Interface “id”. Once found all the object properties have been assigned to the ones passed. At the end, the User Data Access Layer has been called with the method “UpdateUser”, where the “user” object has been passed.

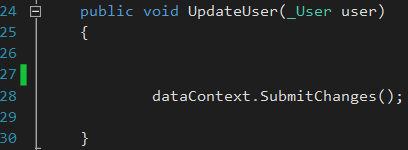
.

Figure 11 Update User ( User DAL )

Since there is no new record to insert in the database, only “SubmitChanges” method has been called, in order to save changes been made on the passed object.

Assuming the fact that, READ and DELETE functionality are using basically the same syntax and order of functions as CREATE and UPDATE, they are not going to be covered in the following on any other chapter.

## Login

The login functionality gives the users permission to visit pages which are forbidden for non-registered users.

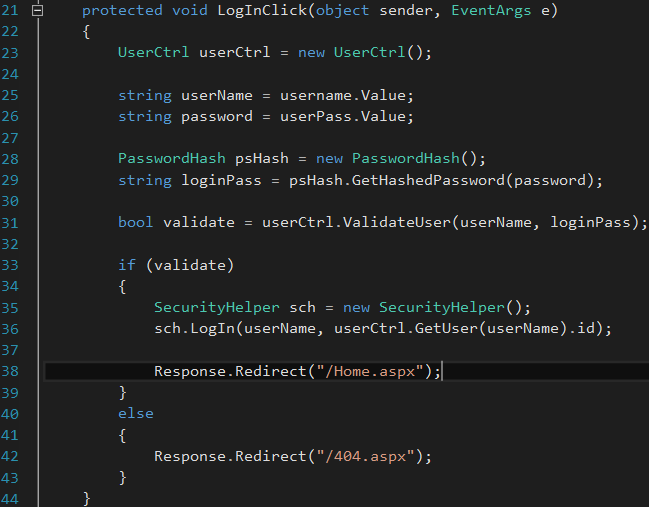


Figure 12 Login ASPX Code Behind

The process starts with the Login Button click. Firstly, a new instance of the “UserCtrl” class is created. Taking the values for the user user name and password from the text boxes and saving them into local variables followed by creating a new instance of the “PasswordHash” class, which as it has been described in the CRUD is used to return the hash value of a string ( the user password ), and saving the hashed version into a local variable “loginPass”. The “userCtrl” has been called on line 31 with the method “ValidateUser” taking the “userName and the “loginPass” as parameters.

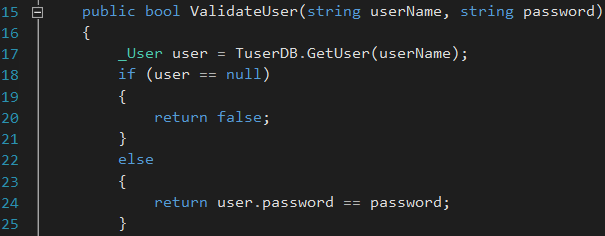


Figure 13 Validate User ( User Controller )

The method starts with calling the “TuserDB” with the “GetUser” method which takes “userName” as a parameter, and simply returns an object of type “\_User” where the user name property matches the one passed.

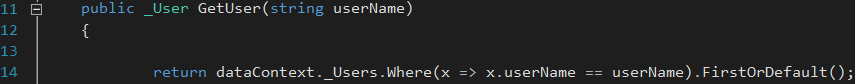


Figure 14 Get User ( T User Db )

Continuing on Line 18 (Figure 13) can be seen an if statement, checking if the “user” object returned is null (does not exists), where it returns false if it is, else a password comparison is being executed comparing the “user” object password with the one passed. At the end a “false” or “true” value is returned depending on the comparison.

Going back to Figure 12, on line 33 a check is performed, determine whether if “validate” value is true or false, if it is “true” a new instance of the class “Security Helper” is created. “SecurityHelper” is a helper class which comes in handy when value/s need to be saved into a Session object (Since the HTTP does not maintain state, “*ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the user's computer and it contains information that identifies the user. This interface is called the Session object*”). On Line 36, the “Login” method is called on the “sch” object.

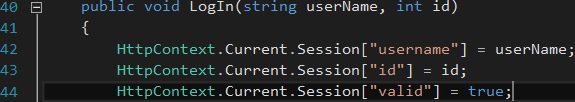


Figure 15 Security Helper Login

The method is used to store the “userName” and “id” into the current Session object, so they can be used later on for various purposes, as well as changes a session variable called “valid” to true – used to determine whether there is a logged in user or not.

Back on Figure 12 Line 38, a simple redirect to the home page is being performed. If the “validate” value is false, the user is redirected to the 404 page.

## Order

The Order Functionality is the most important and biggest part of the project, as well as takes main part in answering the problem statement. The platform offers an order of a chosen product from a restaurant’s menu to be placed, where further on it will be processed and taken care of on restaurant’s side software, which can be a good theme for further projects.

### Select Restaurant and Product/s

Starting from the Order Page, the user can choose from a drop box filled with predefined restaurants. Next step is to select the desired food group, and based on the choice the application would display a grid view with the available products from the chosen restaurant and food group.

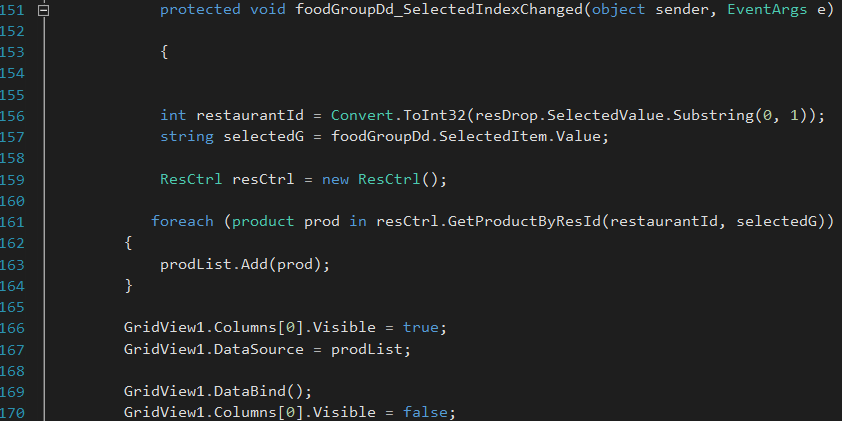


Figure 16 Menu Code Behind

Above is the method responsible to fill out the Grid View with the products from the database offered from the selected restaurant, belonging to the selected food group. Starting with extracting the restaurant’s id from the “resDrop” and saving it to a local variable called “restaurantId”, as well as taking the selected food group from “foodGroupDd”. A foreach loop has been executed on a collection of “Product” objects (result from the method “GetProductByResId”, taking as parameters the local variables described above), where foreach “product” object in the List, it is added to a new List called “prodList”. The new list is a Session Variable, and it is used as data source for the Grid View.

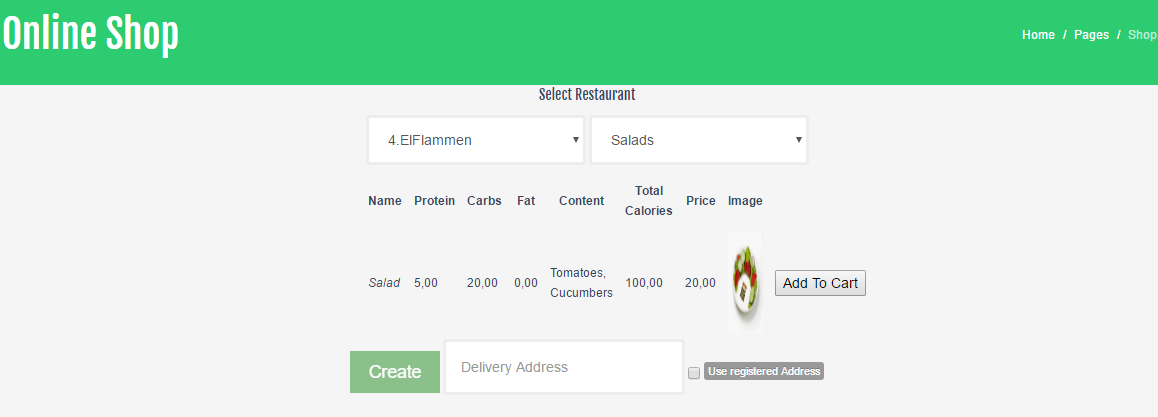
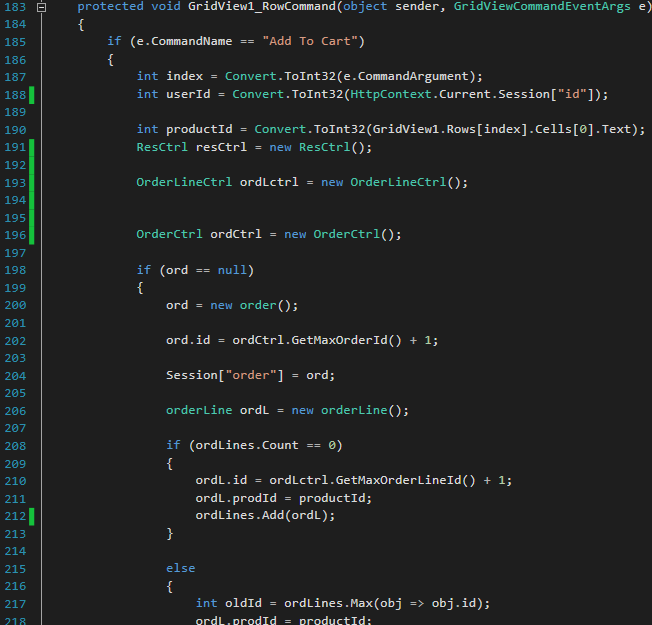


Figure 16 Online Shop Webpage

### Add to Cart

Next step for the user would be to Add a product to his/her cart.





Accessing the Grid View’s Row Command Event, firstly a check occurs determining if the command is equal to “Add to Cart”, if it is the method continues, with saving needed data into local variables – the user id, the product id etc.

Another check occurs later on, checking if the “ord” is null, where “ord” is a Session Variable of type “Order”. If the order does not exist, a new object of type “Order” is created, and it has been filled with an Id (Line 202). Here has to be mentioned that a previous version of the application was creating the order first, saving it in the database, and returning the corresponding object, just to have a valid order Id (used for the creation of the “OrdAndLine” object, due to the many to many relationship between “Order” and “OrderLine”), where it was a problem, because the order was inserted whether or not the method executes successfully. However, now it is possible to get a valid order id with the method “GetMaxOrderId”, which is returning the biggest id from the database, adds one to the result, so duplicates can be avoided and later on is used for its intended purpose.

Next the Session variable “order” is associated with the local one “ord”, also checking if the session list of “OrderLine” objects “ordLines” is empty. If the list is empty, the “OrderLine” object “ordL” created on Line 206, is filled with an id and a product id, where the “ordL” id is taken the same way as described above for the “order” object. However, if the list has more than zero objects, the biggest value of “orderLine” id is found in the list and saved into a local variable – “oldId” – where later on is assigned as an id for the “ordL” object plus one , to avoid duplicate id’s. In both cases the “orderLine” object is added to the session list – “ordLines”.

The matching *else* statement to the *if* one on Line 186 is executing the same code as the described above, but assuming that the “ord” object is not null, it has been assigned to the Session “ord”, so the changes can be kept until the Session expires, the order is created or the user logs of.

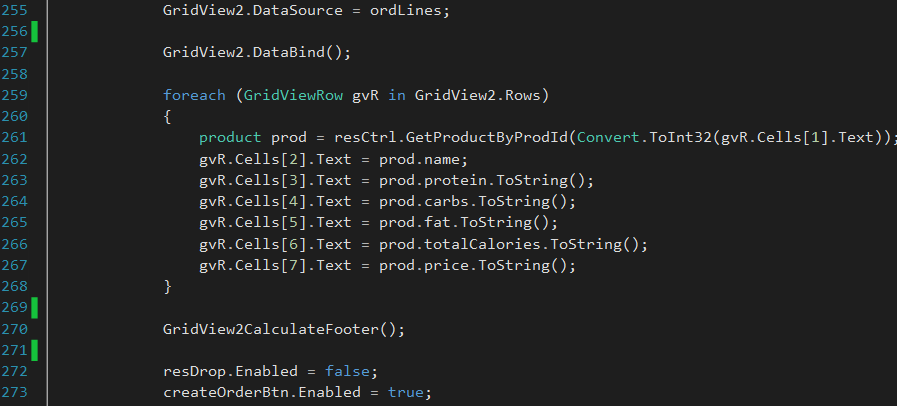


Figure 17 Add to Cart CS

On Figure 17 can be seen the final lines of code of the method explained. After assigning the List of “OrderLine” objects to the Grid View as its data source, a foreach loop is following. Its purpose is to loop through each row, find the “product” object from the product id (taken from the Grid View cell with cell id of 1), and fill in the object’s properties into the rest of the cells. The described foreach loop has two purposes:

* the user can see in the shopping cart the product macros and calories as well as all other properties
* some of them are used to calculate the total values of the order.

The method body ends with another method call to “GridView2CalculateFooter”, which is simply a method taking all the values from the cells filled above, calculate the sum of each, and display them in the Grid View’s footer, so the sums can be used when the creation of the order occurs.

Currently the application does not have a “quantity” attribute, since the group assumes that a restaurant is always filled up with products, and is hard or nearly impossible to get out of stock. However, if an actual customer wants that kind of functionality, it would not take much of an effort to be implemented in the current project.

### Remove item/s from cart

In case the user made a mistake with adding an undesired product, there is a functionality allowing the removal of an order line.

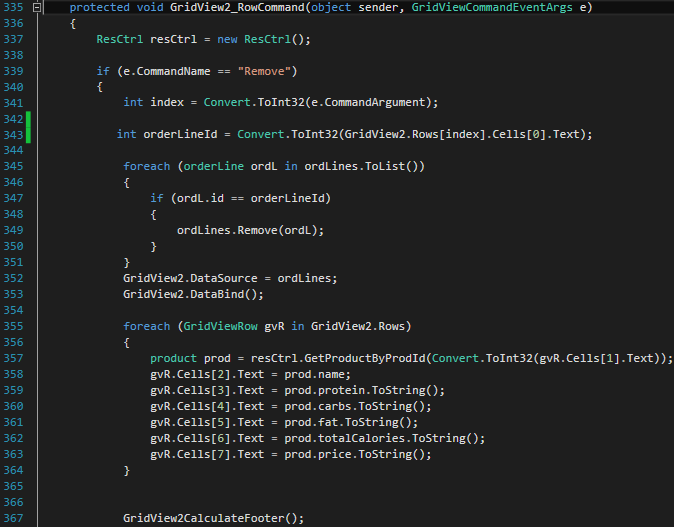


Figure 18 Remove Order Line Code Behind

Method starts with determining if the command name is equal to “Remove”, if it does, the execution continues further on. The index of the row selected is taken and stored in a local variable “index”, so it can be used on Line 343, to take the order line object id from the cell with index of zero. A foreach loop occurs next, looping through the Session List of “OrderLine” objects, checking if the objects id is equal to the one saved in “orderLineId”, if it is the object is removed from the list. Important to mention here is the “ordLines.ToList()”. Iterating through the original list “ordLines” and removing items, was producing an error saying the collection has been modified, which is not permitted. This has been solved with simply calling the “ToList” method, which creates a copy of the List to be iterated on, that way it can be assured that the iteration would happen only on the copy, but the remove operation would be called on the original List.

Back to the main method, the next and final step is to bind again the “ordLines” to the Grid View, in order the user to see the changes. Here again is required to fill the product’s field’s values in the Grid View, since its source has been modified, as well as calling again the “CalculateFooter” method, to calculate again the total sums.

### Create the order

Once the user has finally chosen the desired product, he/she has to finalize (place) the order by clicking the “Create” button on the corresponding web page.

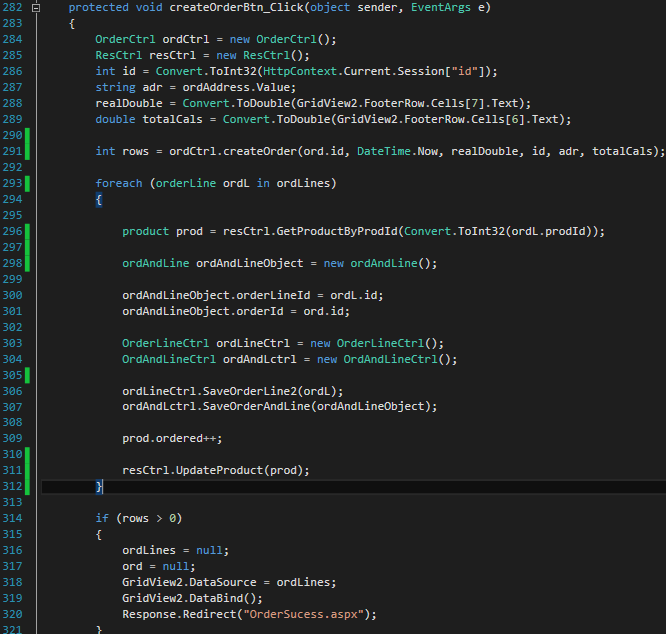


Figure 19 Create order Button Click

The method starts with the usual creation of controller objects, as well as filling local variables with values used later on to create the order object. On Line 291 can be seen that the “ordCtrl” (Order Controller) has been called and saved in a local variable of type “int”, with the method “createOrder” which takes as parameters, the order id, current date, “realDouble” (total sum of the order), the user id, address and “totalCals” (total calories of the order).

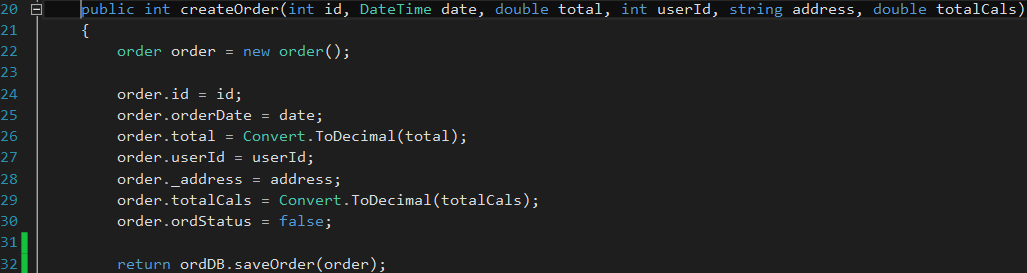


Figure 20 Create Order - Order Controller

The Order Controller has the responsibility to create the “order” object and fill it up with the passed from the user interface values. Here the attribute “ordStatus” is of type “bit” in the SQL Server, and is used to determine if the order is in process (false) or is completed (true). In the current project the order will always be “In Process”, since it can be completed only when it is being handed to the customer, which would require software on the restaurant’s side as has been mentioned earlier in the report.

The execution continues with a call to the Order Database Layer class with “saveOrder” method, taking the “order” object as a parameter, so it can be inserted in the database.

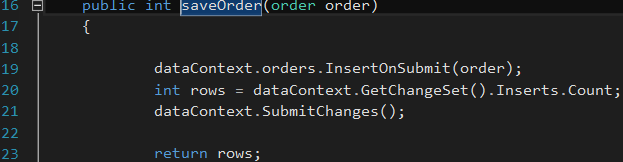


Figure 21 Insert Order - Order DB

As can be seen on Figure 21, as usual the order object is being inserted in pending state in the corresponding table. On Line 20, a local variable of type “int” is presented, saving the value of “GetChangeSet().Inserts.Count” method, which returns the number of rows affected. At the end of the method “rows” is returned up to the controller and finally to the user interface.

Back to Figure 19, the method continues with a foreach loop (Line 293), where for each “orderLine” object in the List of “orderLine” object – “ordLines” :

* Product object has been found from the “ordL” by id.
* A new “OrdAndLine” object has been created and filled with the order id and the order line id, because of the many to many association between “Order” and “OrderLine”.
* Saving the “ordL” and the “ordAndLineObject” in the corresponding tables.
* Incrementing the found product attribute – “ordered” (used for statistics)

The creation of the order finishes with performing a check whether or not the local variable “rows” bigger than zero, so it can be determined if the insertion of the “order” object was successful. If it is, the List “ordLines” and the Order Object “ord” are cleared (set to null), as well as the user being redirected to the “OrderSucess” web page.