

Report

FitFood

Hand in Date:

06/06/2016

Participants:

Iliyan Stoev  
dmai0914 / Group 18  
  
  
Project Supervisor:   
Per Trosborg

Contents

[1. Introduction 2](#_Toc452931988)

[2. Problem Statement 2](#_Toc452931989)

[2.1 Delimitations 3](#_Toc452931990)

[3. Methodology 4](#_Toc452931991)

[3.1 Method Choice 4](#_Toc452931992)

[3.2 User-Stories 5](#_Toc452931993)

[4. Architecture 7](#_Toc452931994)

[4.1 Main Architecture 7](#_Toc452931995)

[4.2 Three Tier Pattern 9](#_Toc452931996)

[4.2.1 Business Logic Layer 9](#_Toc452931997)

[4.2.2 Data Access Layer 9](#_Toc452931998)

[4.2.3 Model Layer 9](#_Toc452931999)

[5. Domain Model 10](#_Toc452932000)

[5.1 Classes 11](#_Toc452932001)

[5.2 Associations 12](#_Toc452932002)

[6. Sprints 13](#_Toc452932003)

[6.1 Sprint 0 13](#_Toc452932004)

[6.2 Sprint 1 14](#_Toc452932005)

[6.3 Sprint 2 15](#_Toc452932006)

[6.4 Sprint 3 16](#_Toc452932007)

[7. Implementation 17](#_Toc452932008)

[7.1 User CRUD (Create, Read, Update, Delete) 17](#_Toc452932009)

[7.1.1 Create 17](#_Toc452932010)

[7.1.2 Update 19](#_Toc452932011)

[7.2 Login 23](#_Toc452932012)

[7.3 Order 25](#_Toc452932013)

[7.3.1 Select Restaurant and Product/s 26](#_Toc452932014)

[7.3.2 Add to Cart 27](#_Toc452932015)

[7.3.3 Remove item/s from cart 30](#_Toc452932016)

[7.3.4 Create the order 31](#_Toc452932017)

[8. Conclusion 34](#_Toc452932018)

[References 36](#_Toc452932019)

# 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.

The number of people concerned about their overall health and looks is growing every day. People nowadays have so busy life, so even though they want to stay healthy and have good looks, sometimes is nearly impossible to track what they are consuming.

How these two industries can be combined, so they can be in actual help of this part of the earth population?

# 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 own food, so they are ordering whether from restaurants or fast food places, where cannot be seen how many calories they are consuming, 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 the 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 consuming, 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 calories/macros intake during a meal ordered from a restaurant or a fast food place. In what concerns the implementation of the platform, the users 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 online shopping cart, and finally place the order. All of the orders will be stored for account history purposes.

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.

## Delimitations

The following chapter will expose what technologies were used in order for the project to be built, as well as why.

For the described project, the group chose to work with the .NET framework and more specifically ASP.NET. Even though the members were following Open Source course in the current semester, where basic knowledge of PHP has been gained, still ASP is the preferred choice, since it is a subject of interest, as well as having some, but not much experience with it from previous projects and semesters.

In what concerns the choice of database, a relational database has been chosen – SQL Server. The choice can be explained by the fact, that the group is quite familiar with the above mentioned database, as well as feeling comfortable working with it. Even though the team followed a database course through the current study year, where basic knowledge of Non-Relational databases has been gained, it was not enough to substitute the Relational one. It also has to be mentioned that the course took place at the very end of the semester, where the project was almost done, and it has been decided to stick with the original setup.

# Methodology

In order to make the system to be developed efficient within given period of time, fulfill all customer’s requirements, assure acceptable level of quality and etc., an appropriate software development methodology should be chosen:

* Light Methodology – SCRUM, Kanban etc.
* Heavy Methodology – Waterfall, Unified Process etc.

In order to select good method of work, some factors should be taken into consideration:

* Size of the project – Usually big projects adopt the Heavy Methodology, since documentation is a must, therefore the communication between the teams is always on point.
* Size of the team – Light Methodology is preferred among smaller teams for the purpose of being dynamic with clear responsibilities.
* Customer organizational culture – If the customer demands a close relationship with the development team – Lightweight Methodology would be better choice.
* Dynamics in requirements – Light Methodology is used where the requirements are more likely to change, since the main purpose of it is working agile, if the requirements are predefined – Heavy Methodology.

## Method Choice

Taking into consideration the above mentioned factors dictating the choice of method, Light Methodology has been chosen, more specifically – SCRUM and XP (Extreme Programming) combined, where SCRUM will cover the steps of development and XP will assure the quality of the code being developed.

Since the development team consist only of one person, it can be assumed that he is covering all the SCRUM roles within the project being developed – The development team, The Scrum Master and the Product Owner. Following the Scrum principles, the project has been divided into four Sprints, each with length of two weeks, as well as an additional Sprint “0”, which time was used in planning.

## User-Stories

“*A user story is a tool used in*[*Agile software development*](http://searchsoftwarequality.techtarget.com/definition/agile-software-development)*to capture a description of a software feature from an end-user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement.*”[[1]](#footnote-1)

After consultation with representatives from the main target group of the project, User-stories have been defined by the product owner. Since it has been mentioned that the developer within the team is covering all the SCRUM roles, it is also important to note that he is also a representative from the main target group, which has its advantages and disadvantages.

* Advantages: Knowing the upcoming changes required in design and/or features.
* Disadvantages: Not working with a real client, which could lead to gaining a lot of precious experience and knowledge.

Each user-story has its own acceptance criteria, written on the back of the user-story card, which is used by the development team to determine when an acceptable level of functionality has been achieved.

*“Acceptance criteria define the paramets of a user story and determine when a story is completed and working as expected. They add certainty to what the team is building”[[2]](#footnote-2)*

The acceptance criteria also cover the Functional Requirements of the system, which how the name suggests represent the desired functions the application must perform. However Non-Functional requirements are to be discussed with the product owner. Some examples of this type of requirements: Performance, Usability, Security etc. Unlike the Functional, Non-Functional requirements are often related to the system as a whole, and can be classified as critical. The user can always find a workaround if some functionality is not on point, but cannot bypass an unresponsive or slow performing system.

After a meeting with a few representatives of the main target group for the project, the product owner defined the following user-stories:

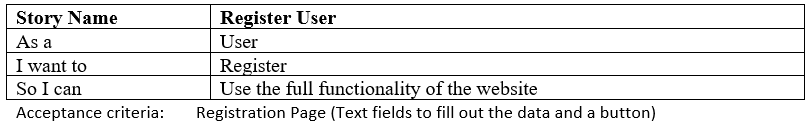


Figure User Story – Register

The above user-story is the very first functionality of the system, since it is representing the creation of a user, required to access the main functionalities of the application.

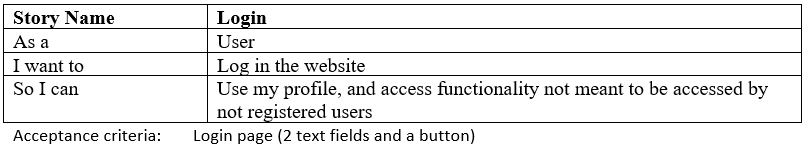


Figure User Story – Login

The above user-story represents the demand of the user to log in, in order to access functionality not meant to be accessed by not registered users.

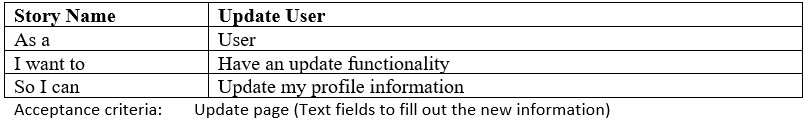


Figure User Story – Update

The above user-story represents the demand of the user to update his/her profile, if some data has been mistaken during the registration process.

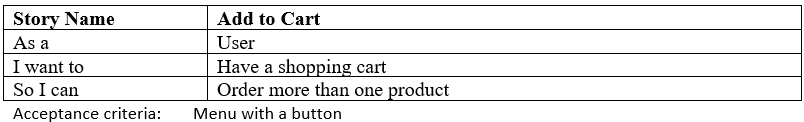


Figure User Story - Add to Cart

The above user-story represents the demand of the user to add products to an online shopping cart, so it can be possible to order more than one product at a time.

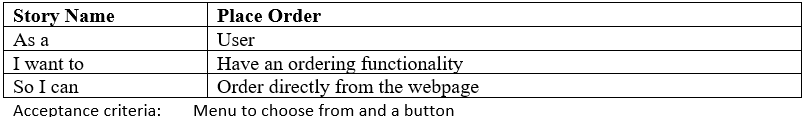


Figure User Story - Place Order

The above user-story represents the main functionality of the application, giving to the user the opportunity to place an order through the application.

The current chapter covered only the most important user-stories of the project, which stories are to be covered also in the implementation chapter.

# Architecture

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

## Main Architecture

Multiple options are available when it comes to choose the main system design, such as “Client – Server”, “Multi – Server”, “Proxies and caches” and “Peer to Peer”. Taking into consideration the lack of experience and knowledge of the development team about some type of architectures, as well as the scalability of the application developed, classical “Client – Server” model has been 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. Since the current project is meant for study purposes, the data about restaurant’s menus is being taken from a single local database. However, if the application was going to be published in the real world, it would be more appropriate to choose “Multi – Server” architecture, since each restaurant has its own database, which has to be reached by the main server in order to be provided to the client.

Down below can be seen a UML representation of the application’s architecture.

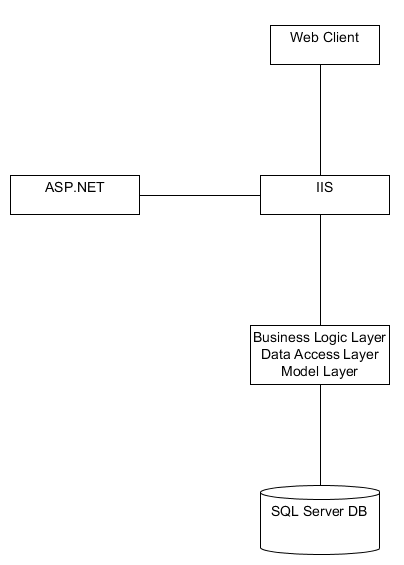


Figure 6 System Architecture

As can be seen on “**Figure 6**”, 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 a client for the database SQL server.

## Three Tier Pattern

Within the main architecture, a three tier pattern can also be noticed. The “**Web Client** and the **IIS**” are playing the role of the first tier, where “**BLL**, **DAL**, and **Model** **Layer**” and “**SQL** **Server** **DB**” are respectively the second and the 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.

# 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.

Down below can be found the UML representation of the Domain Model, showing the most important types of objects in the context of business (classes) and the communication between them by dint of associations.

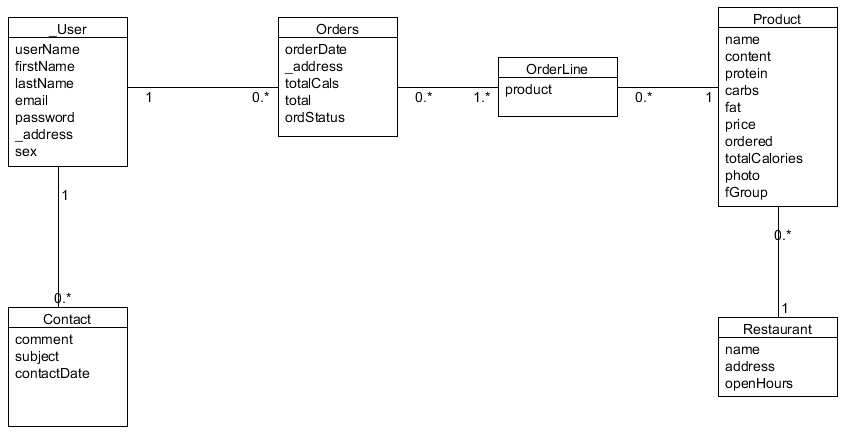


Figure 7 Domain Model

## Classes

**\_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.

# Sprints

The current chapter is going to cover the description of the work that has been done through the Sprints, in order for the project to be completed.

## Sprint 0

The most important Sprint in the whole project methodology, would be Sprint 0, where it is meant to discuss and agree on the project’s architecture, set up the development environment, populate the Product Backlog, creating the database, setting up the continuous integration software, prepare the Scrum board and burn down charts, as well as plan Sprint 1 where everything should be ready to start developing from day one of it.

The team spent Sprint 0 in populating the Product Backlog with all the User-Stories, which were going to be taken in Sprint 1, already ordered by their importance. The estimation of the user stories has been done based on tasks from previous projects, since the group consists only of one person. The rest of the Sprint was spent in setting up the code writing environment (Visual Studio), setting up repository (GitHub), preparing and populating the SCRUM board with the user-stories from the Product Backlog, burn down chart and the planning of Sprint 1.

Even though a lot of time has been spent on thinking about the design of the application to be developed, later on through the project has been realized that a lot of additional features have to be implemented, which lead to spending additional time and effort in rewriting and adding code in some of the sprints.

## Sprint 1

Although the group consists only of one person, the SCRUM Daily Standup Meeting, where the team is spending the time to split the tasks among its members, was held every day of the sprint at exactly 10:30 AM. From day one of the sprint, some of the XP (Extreme Programming) practices were put in use. The test first approach did not go as expected, since the group lacked the experience, and since it appeared to be time and resource consuming at that point, it has been decided to be held once a task has been completed. “On site Customer” practice has been used constantly within the duration of the project, since the group member has been a representative from the main target group of the project as well as the product owner himself.

The burndown chart for the sprint has been updated every day according to the points burned.

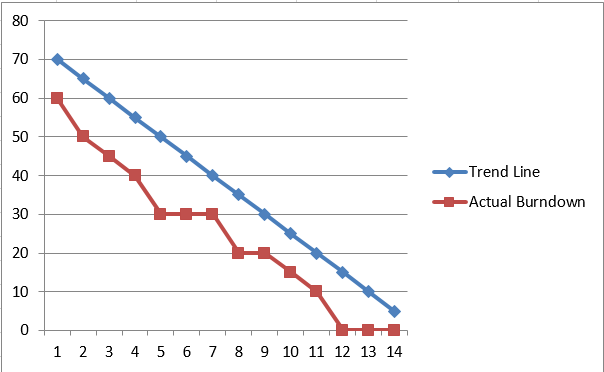


Figure 8 Burndown Chart Sprint 1

As can be noticed on “**Figure 8**”, the group has been done with the tasks meant to be completed in Sprint 1 earlier, which is a result of overestimating in the planning game, caused by the fact that a lot of the technologies used in the current project were not so well or not at all known to the developers within the team.

The sprint review with the customer and the development group, as well as the sprint retrospective, were held on the last days of Sprint 1, followed by planning of Sprint 2.

## Sprint 2

Due to an overestimation of the tasks in Sprint 1, causing early completion of which, it has been decided more points to be taken in the current sprint.

Sprint 2 was the point where the team started noticing that not enough time and effort were put in Sprint 0, where the design of the application should have been discussed and agreed on. This had negative effect on the overall performance during the sprint, due to the fact that a lot of new variables and features were presented, needed for the completion of the tasks planned. However, it did not affect meeting the Sprint deadline, due to the more than enough sprint length. On “**Figure 9**” (below) can be seen the burndown chart for Sprint 2, where clearly can be noticed how the poor design affect the work of the group. The days where no work has been done, were spent in adjustments and rethinking, in order for the meant user-stories to be completed.

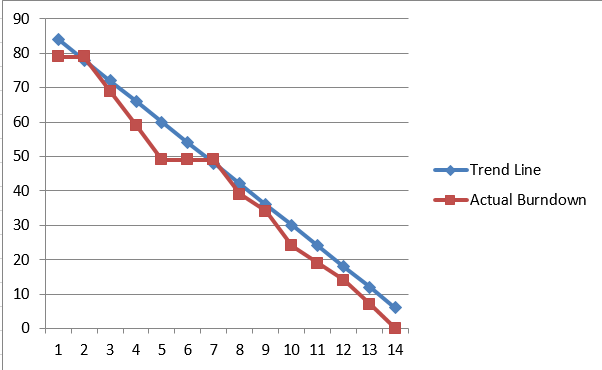


Figure 9 Burndown Chart Sprint 2

The sprint review and retrospective, as well as the planning for Sprint 3, were held on the last day of the current one.

## Sprint 3

Sprint 3 has been the time where the group decided to take the biggest and most important user-story, as well as a few smaller ones, in order to be done with the actual coding part. Even though there was one more sprint to follow, it has been decided in the last one to focus more on the technical documentation to be written, since it is as important as the product of development.

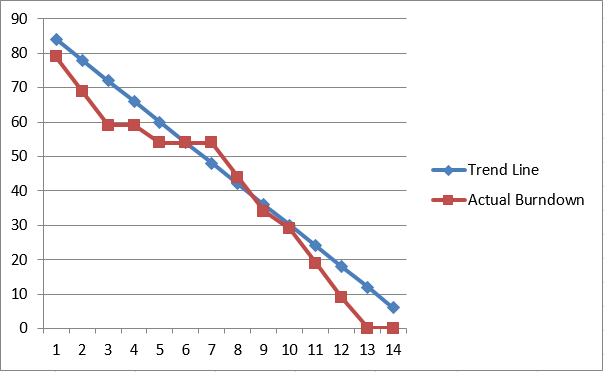


Figure 10 Burndown Chart Sprint 3

“**Figure 10**” represents the burndown chart for the current sprint. At this point the group was already prepared to face the consequences of the poor design, which was already a known problem from previous sprints, which readiness lead to dealing with the problem quickly and effectively from day one. However, after the good start, another issue has arisen. As can be seen on the burndown chart, there were a lot of days where no actual work has been done. This is a result of the lack of experience and knowledge of the team members about the framework that has been used, therefore days were needed to gain the understanding in order to solve the problem. Once the complication was eliminated, the work continued as expected, and the meant functionality for the sprint has been achieved even before the deadline. With the completion of the current sprint, the group was ready with all of the desired features from the product owner. The usual sprint review and retrospective were held again on the last day, as well as the planning of Sprint 4, where was decided to focus only on the technical documentation required for the project.

# Implementation

The current chapter is going to cover the implementation (actual code) of the most important features of the application. The functionalities to be described were chosen based on the business value of the user-stories standing behind them.

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

### Create

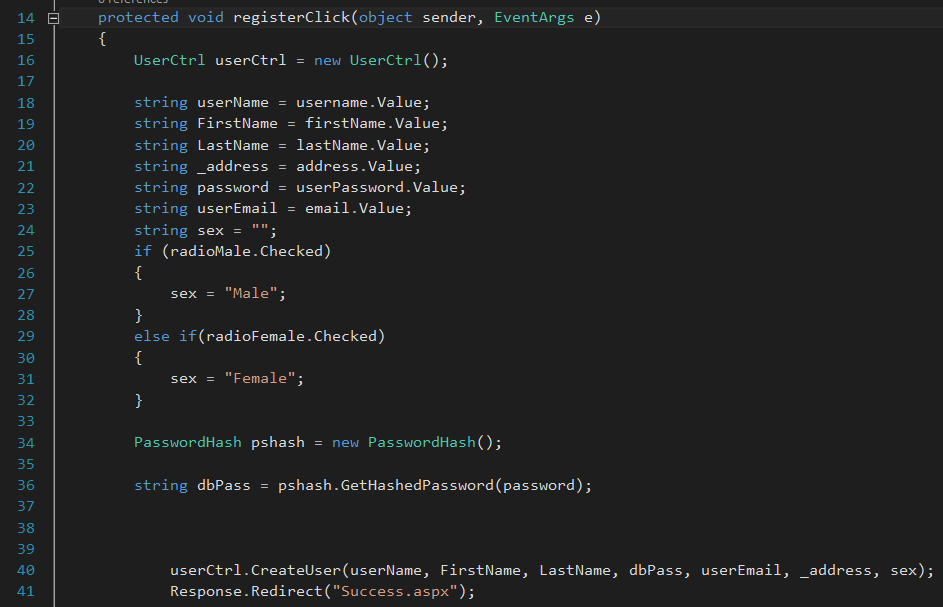


Figure 11 Register User UI

The code above (“**Figure 11**”) 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, 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.

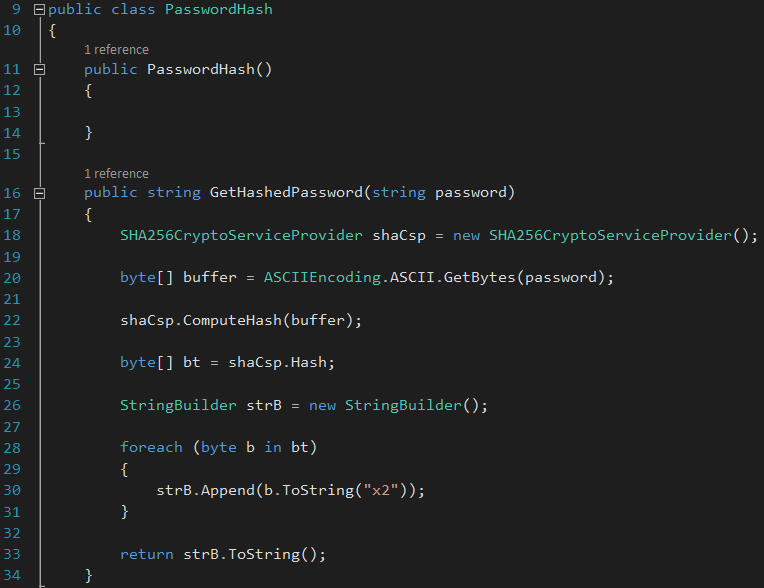


Figure 12 Password Hash

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 11**” on “**Line 40**” of the code, the “UserCtrl” has been called with the method “CreateUser”, passing all the parameters.

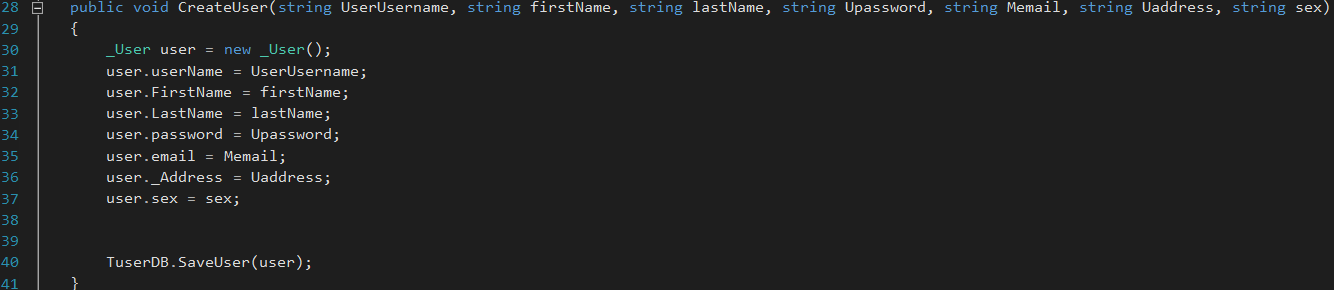


Figure 13 Create User (Controller)

The method above (**“Figure 13”**) 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 relevant “Data Access Layer” has been called to finalize the user creation.

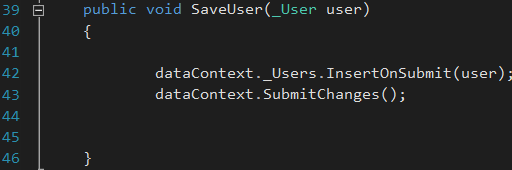


Figure 14 Save User ( User DB )

The task of the method on “**Figure 14**”, is to deal with the database insertion of the “user” object. Calling the “InsertOnSubmit” method on the current “DataContext” object will insert the object in a pending state, which later on is finalized by the “SubmitChanges” method.

### Update

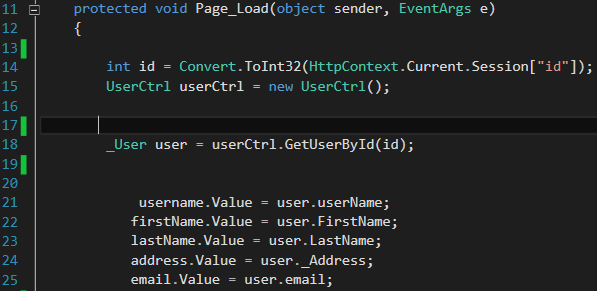


Figure 15 Update User (Page Load)

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 in the report). 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.

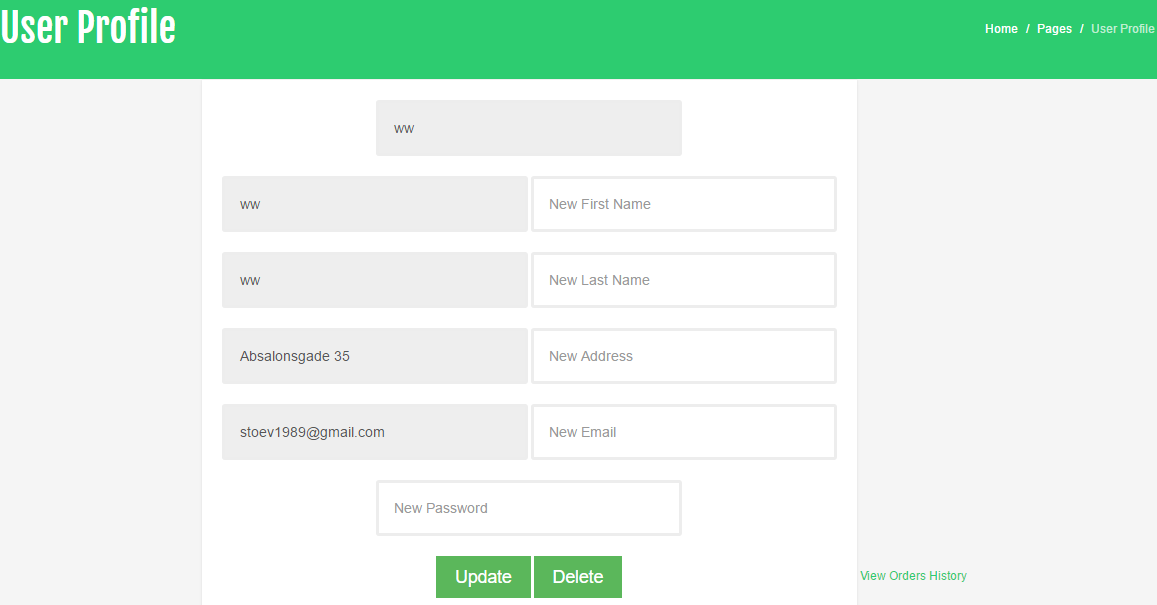
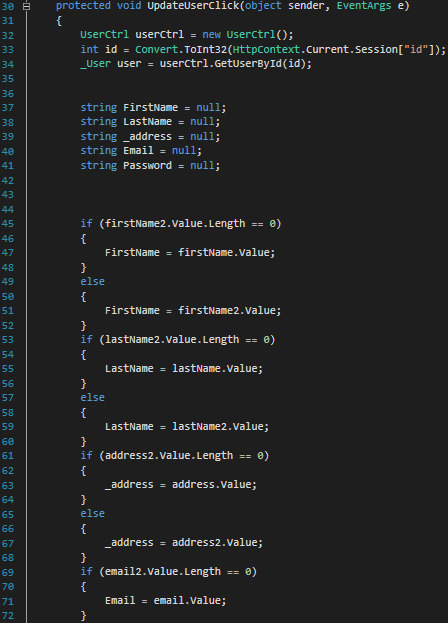


Figure 16 Update User UI View

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



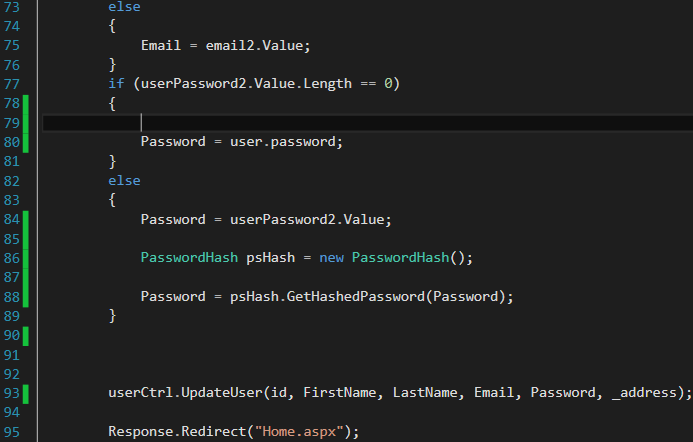


Figure 17 Update User Code Behind

After finding again the user by id, local variables for all the “user” properties have been declared (“**Figure 17**”). 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. Important 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.

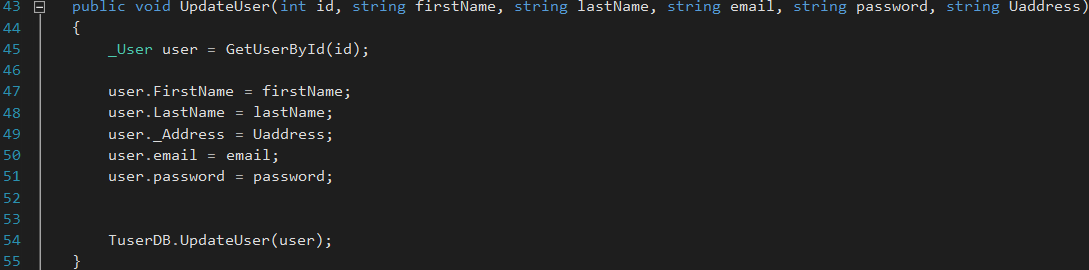


Figure 18 Update User ( User Controller )

In the “User Controller” (“**Figure 18**”), 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.

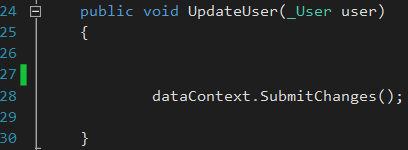
.

Figure 19 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.

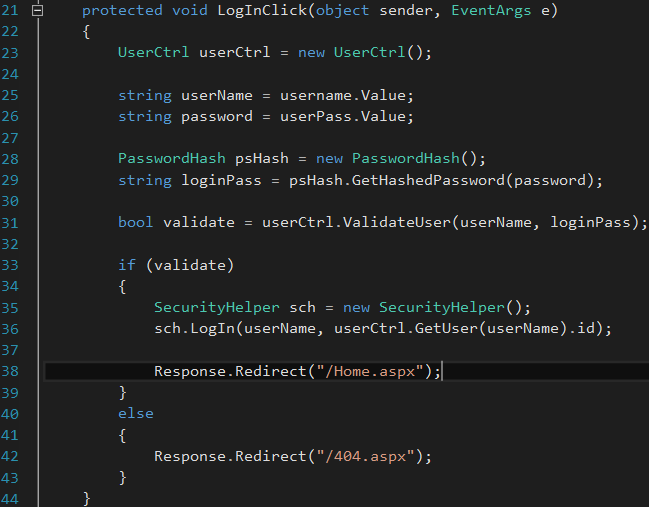


Figure 20 Login Page Code Behind

The process starts with the Login Button click. Firstly, a new instance of the “UserCtrl” class is being created. Taking the values for the user username 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**” (“**Figure 20**”) with the method “ValidateUser” taking the “userName and the “loginPass” as parameters.

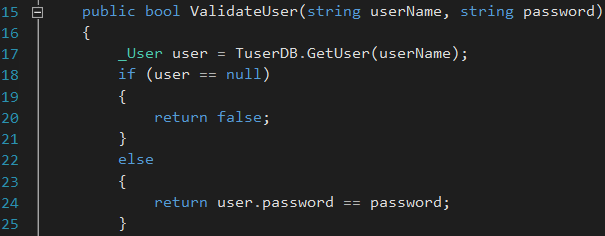


Figure 21 Validate User ( User Controller )

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.

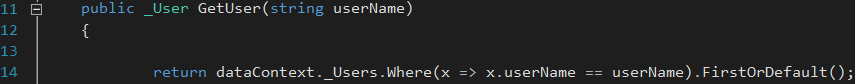


Figure 22 Get User ( T User Db )

Continuing on **Line 18** (“**Figure 21**”) 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 20**”, 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*”[[3]](#footnote-3)). On **Line 36**, the “Login” method is called on the “sch” object.

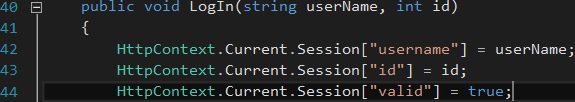


Figure 23 Security Helper Login

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 20**” **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 user-story 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 “Shop 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.

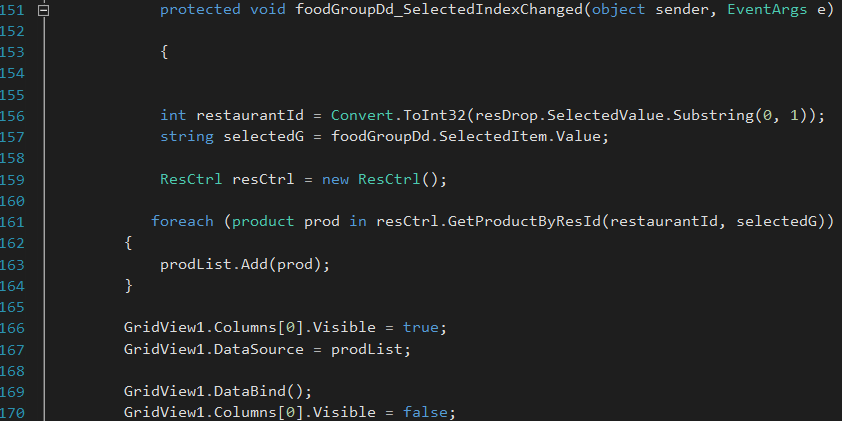


Figure 24 Menu Code Behind

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” (restaurant drop down) and saving it to a local variable called “restaurantId”, as well as taking the selected food group from “foodGroupDd” (food group drop down). 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”.

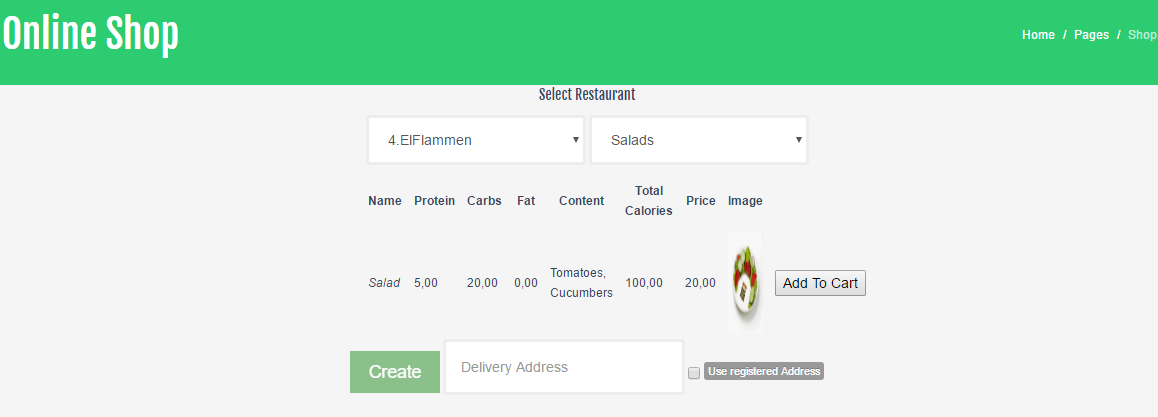


Figure 25 Online Shop Webpage

On “**Figure 25**” can be seen the user interface of the “Shop Page”, after the execution of the methods described above.

### Add to Cart

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

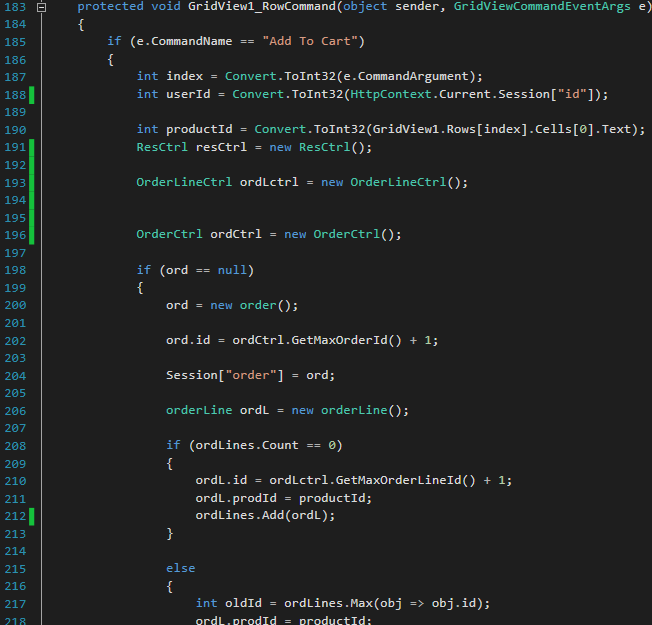




Figure 26 Add To Cart CS

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, “Figure 26”**). 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” classes), 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”, as well as 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.

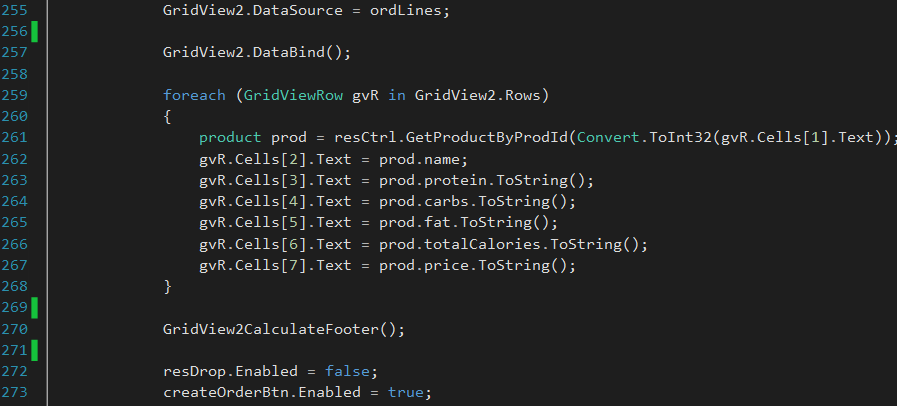


Figure 27 Add to Cart CS

On “**Figure 27**” 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 by 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 is able to 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, to check if there is enough amount of the product/s, 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, in a future version of the application this kind of functionality will be implemented.

### 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.

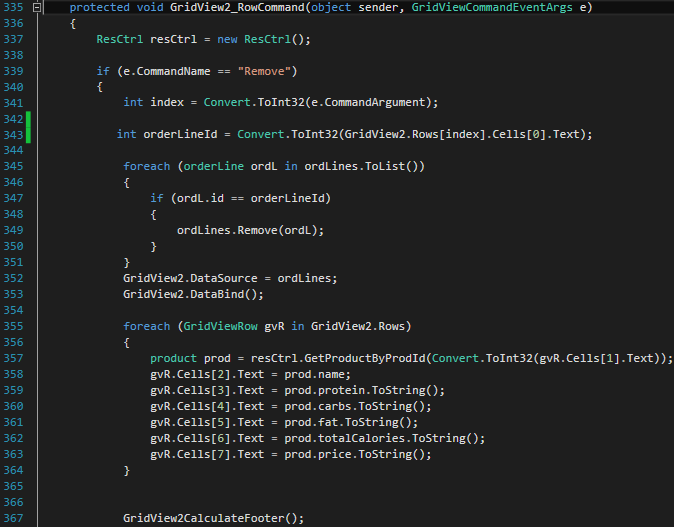


Figure 28 Remove Order Line Code Behind

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** “**Figure 28**”, 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 object’s 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 stating 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.

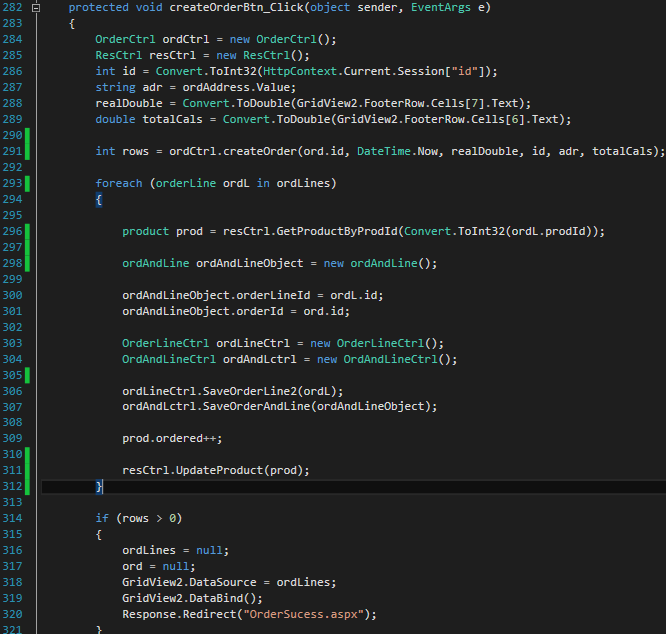


Figure 29 Create order Button Click

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 “Figure 29”**, 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).

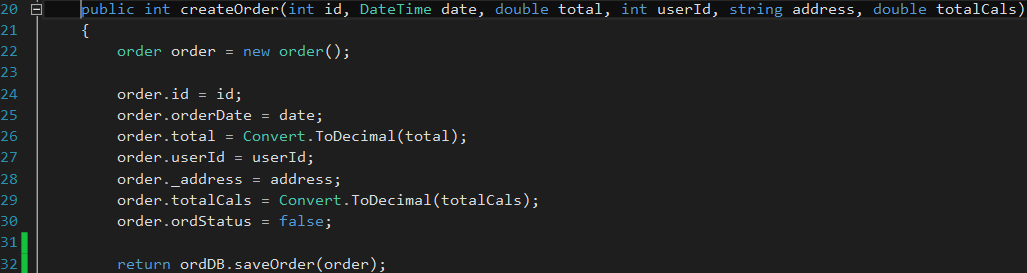


Figure 30 Create Order - Order Controller

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 database, 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.

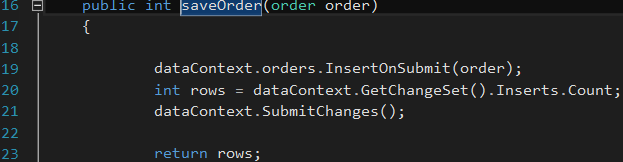


Figure 31 Insert Order - Order DB

As can be seen on “**Figure 31**”, 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 29**, the method continues with a foreach loop on **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” classes.
* 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” is 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, so a new order can be placed without interfering with the one being placed.

# Conclusion

The main goal for the current project, developed over the given time, was to answer the question of the proposed problem statement: “***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 answer of the above questions, was found in creating a new online platform called FitFood, made within the given deadline and following the right choices of system architecture, design, as well as the right choice of system development method. The creation of FitFood made it possible for the people interested in the fitness industry or interested in overall healthy lifestyle, to be able to stay informed about their calorie intake while they are not at home, or whenever they are forced to order food from a restaurant or a fast food place. The platform gives them the opportunity to do the whole cycle online, which makes it so comfortable and easy to use.

For the creation of the whole project, ASP.NET was used, which is a part of the .NET framework. “ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding”[[4]](#footnote-4). Current project was intended to give the development team, the experience of working with ASP.NET, as well as gain knowledge in the same area, since the developers within the team are interested in web development.

The choice of the right system development method also played a big role in the successful implementation of the product, as well as meeting the project’s deadline. By choosing to work agile, in the face of SCRUM and XP, it has been possible to be dynamic and flexible in order to react to the constant changes. As a bonus, the team gained valuable knowledge and experience in working with agile development methods.

According to the final result, it can be said that the desired outcome has been met. The group has implemented fully functional online platform, gained precious experience and knowledge in developing web applications.

# References

[1]. <http://searchsoftwarequality.techtarget.com/definition/agile-software-development>

[2]. <http://www.boostagile.com/user-stories-part-2-acceptance-criteria/>

[3]. <http://www.w3schools.com/asp/asp_sessions.asp>

[4]. <https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx>

GitHub Repository Link: <https://github.com/IliyanStoev/dmai0914Gr18_4Sem>

1. <http://searchsoftwarequality.techtarget.com/definition/agile-software-development> [↑](#footnote-ref-1)
2. <http://www.boostagile.com/user-stories-part-2-acceptance-criteria/> [↑](#footnote-ref-2)
3. <http://www.w3schools.com/asp/asp_sessions.asp> [↑](#footnote-ref-3)
4. <https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx> [↑](#footnote-ref-4)