UNIVERSITY OF CYPRUS
DEPARTMENT OF COMPUTER SCIENCE

EPL 343 SOFTWARE ENGINEERING

Specifications Document

Food Recommendation Agent

Valentinos Pariza
Marios Pafitis
Demetris Shimitras
Leonidas Achilleos
Stephanos Pantziaros

21.10.2019

Table of Contents

REVISION CHART

Version	Primary Author(s)	Description of Version	Date Completed
Draft	Valentinos Pariza Marios Pafitis	Initial draft for separating the jobs giving some ideas and recommendations to the rest of the team members as Project Managers.	11/10/19
Preliminary	Valentinos Pariza Marios Pafitis Demetris Shimitras Leonidas Achilleos Stephanos Pantziaros	Research and analysis of all the parts from each member of the team. Creation of all the diagrams and comment on them in a very sufficient state. Corrections from the feedback of the first deliverable.	19/10/19
Final	Valentinos Pariza Marios Pafitis	Final review of the content. Corrections and improvements for certain parts. Final touches and furthermore analysis in some cases.	21/10/19

1. Introduction

1.1 Purpose

This second specifications document serves the purpose to specify the data storing for the "Foody Recommendation Agent" system in order to provide to Foody's users a faster way to pick what they want by suggesting restaurants and specific food that would probably tend to, based on their preferences from their past usage of Foody. Moreover, different functionalities of the new system are designated along with the entities involved in them, as well as the model class diagram of the classes and objects that will be used for the operation of the system in communication with Foody's system. Lastly, an activity plan of the tasks and tests that need to be applied are described in detail.

The document is intended for all the members of the team of Foody to get awareness of the data that will be used in achieving the correct recommending feature to the users. With the Use Case diagram and Model Class diagram will better understand the functionalities of new system aside Foody's, as well as the API provided for Foody to communicate with the FRA.

1.2 Scope

The "Food Recommendation Agent" is a software system, which is going to provide services to the existing system of Foody (the "Foody Online Orders" system), and which services are associated with recommending targeted foods to its customers by estimating and using their designated or/and possible preferences. This system is going to be used not only for processing of data purposes (estimating user's preferences) but also as a data storage of data related to the offered services. Reaching at the functional part of the software system, the system is going to be built upon five main operations:

1. Collect data from Foody's database and feed them into the Recommendation System.
2. The Recommender System will calculate the recommendation attributes for each user and store them in our database.
3. The everyday collection of essential data from users like order-history, rankings and other information taken from websites which will let the users to show their preferences by checking what foods they like or dislike and by creating groups of their preferred combinations of foods and store them in a database.
4. The production of recommendation attributes through the Recommender System for the users of Foody, using the information gathered for them and store them in the Database.
5. The use of the recommendation attributes produced and stored in the Database to recommend targeted foods and drinks to users of Foody.

1.3 Definitions, Acronyms, and Abbreviations

- Foody Recommendation Agent (FRA): The system that we are implementing which is consisted of the Recommender System, Likes/Dislikes and Journeys features.
- Foody: A company in Cyprus in which you can order online from different restaurants,
- Recommender System: A system that is going to calculate the recommendation attributes of a user.
- Recommendation attributes: Values that define the preferences of a user.
- Likes: Foods that a user like.
- Dislikes: Foods that a user doesn't like.

- Journey: A collection of liked foods for a user.
- My Journeys: A collection of Journeys for a user.
- Food: Any kind of food or beverage. Products that Foody's restaurants sell.
- Item: Any sellable provided in Foody's website.
- Picture: An image of an item in the website.
- Database: A place to store the data calculated from our system.

1.4 References

- Material.io for User Interface Design: https://material.io/
- Foody's Website: https://foody.com.cy/gr/2202?filters=1
- Amazon's Website: https://www.amazon.com/
- User Interface Design: https://en.wikipedia.org/wiki/User_interface_design
- Responsive User Interface: https://en.wikipedia.org/wiki/Responsive_web_design
- Use Case Diagrams: https://www.uml-diagrams.org/use-case-diagrams.html
- UML Diagrams: https://tallyfy.com/uml-diagram/
- Modelio: https://www.modelio.org/
- Project Libre: https://www.projectlibre.com/
- UML Language: https://en.wikipedia.org/wiki/Unified_Modeling_Language
- Software Engineering Ian Sommerville: https://www.academia.edu/31140292/Ian_Sommerville_Software_Engineering_9th_Edition_2011_1_
- Using UML: software engineering with objects and components: http://homepages.inf.ed.ac.uk/perdita/Book/index.html
- Class diagrams: https://en.wikipedia.org/wiki/Class_diagram
- Use Case Diagrams: https://en.wikipedia.org/wiki/Use_case_diagram
- MySQL Data Type Storage Requirements : https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html & https://www.techonthenet.com/mysql/datatypes.php
- ERDplus : https://erdplus.com/
- Software Lifecycle Processes: https://standards.ieee.org/standard/12207-2017.html
- System Interfaces: https://deseng.ryerson.ca/dokuwiki/design:system_interface?fbclid=IwAR3M2mEcuOhjhTSC2gJjJcVaPzkBB5RgHTwCoSosNiRAbJ3VEiuoUCrhhhg
- Software Requirements: https://belitsoft.com/blog/software-requirements-specification-document-example-international-standard?fbclid=IwAR2nFX0f5kFs6hB7EvZBi3DEb83fhqXTMv8kW78q8JxqADt7gZtINejeq8Y
- Software Engineering Ian Sommerville: https://www.academia.edu/31140292/Ian_Sommerville_Software_Engineering_9th_Edition_2011_1_

2. Data store

At this section there is an analytical description of the database tables, attributes/columns as well as their types and sizes, the relationships between the tables and some estimations of the total space that is going to be needed for storing the data of the system Food Recommendation Agent.

2.1 *Data* definition

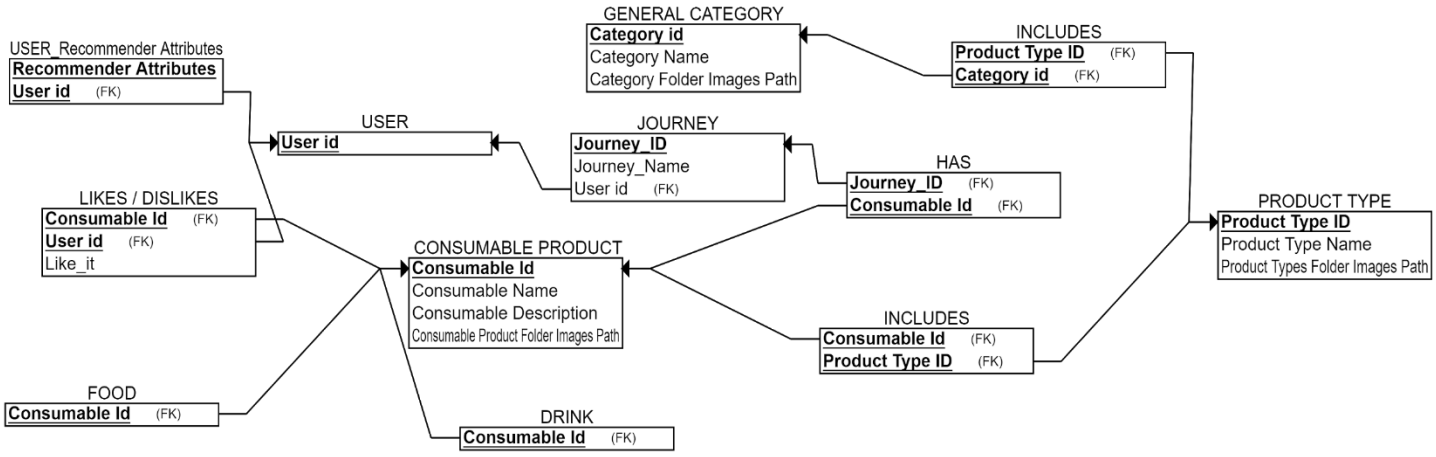


Figure 1 Logical Design of Relational Schema

This logical relational schema was created using the ERDplus which can be found in the **https://erdplus.com/ .**

On this logical design of relational schema - diagram it is possible to see the different tables of the Database that will exist, the attributes/columns in each table as well as the relationships between the tables. The types of the attributes/columns used in each table in the Database as well as their sizes are presented in the next subsection 2.2 Size Calculation.

2.2 Size calculation

In this section, first of all, the size of the individual data that are going to be stored in the database are described. Secondly, some estimations about the space-capacity needed for the data of the Food Recommendation Agent system to be stored are presented. Those estimations are based on some other estimations on the numbers of users, consumable products (consequently the number of product types and general categories that are going to be introduced) that the current Food Online Orders system has. Despite the fact that the estimations used are approximately near to reality, the estimations are a little bit higher than the actual, in order to represent space-capacity demands that might occur in the future when the complete system will be deployed.

GENERAL_CATEGORY	**Type**	**Size**	**Constrains - Descriptions**
Category_ID (P.K)	*INT*	8 Bytes	Every General Category has a unique ID.
Category_Name	*VARCHAR (80)*	max 80 Bytes	The name of the General Category.
Category_Folder_ Images_Path	*VARCHAR (300)*	max 300 Bytes	The path where the images of the general category exist in the server.
Total Bytes:	388 Bytes		

HAS	Type	Size	Constrains - Descriptions
Journey_ID (F.K)	*INT*	8 Bytes	The unique ID of a Journey.
Consumable_ID (F.K)	*INT*	8 Bytes	The unique ID of a Consumable product.
Total Bytes:	16 Bytes		

JOURNEY	Type	Size	Constrains - Descriptions
Journey_ID (P.K)	*INT*	8 Bytes	Every Journey has a unique ID.
Journey_Name	*VARCHAR (80)*	max 80 Bytes	The name of the Journey as given by a User.
User_ID (F.K)	*INT*	8 Bytes	The unique ID of a user.
Total Bytes:	96 Bytes		

PRODUCT_TYPE	Type	Size	Constrains - Descriptions
Product_Type_ID (P.K)	*INT*	8 Bytes	Every Product Type has a unique ID.
Product_Type_Name	*VARCHAR (80)*	max 80 Bytes	The name of the Product type.
Product_Types_Folder _Images_Path	*VARCHAR (300)*	max 300 Bytes	The path where the images of the product type exist in the server.
Total Bytes:	388 Bytes		

INCLUDES	Type	Size	Constrains - Descriptions
Consumable_ID (F.K)	*INT*	8 Bytes	The unique ID of the Consumable Product.
Product_Type_ID (F.K)	*INT*	8 Bytes	The unique ID of a user.
Total Bytes:	16 Bytes		

INCLUDES	Type	Size	Constrains - Descriptions
Product_Type_ID (F.K)	*INT*	8 Bytes	The unique ID of a Product Type.
Category_ID (F.K)	*INT*	8 Bytes	The unique ID of a General Category.
Total Bytes:	16 Bytes		

CONSUMABLE_PRODUCT	Type	Size	Constrains - Descriptions
Consumable_ID (P.K)	*INT*	8 Bytes	Every Consumable Product has a unique ID.
Consumable_Name	*VARCHAR (80)*	max 80 Bytes	The name of the Consumable Product.
Consumable_Description	*VARCHAR (300)*	max 300 Bytes	The description of the Consumable Product.
Consumable_Product_Folder, Images_Path	*VARCHAR (500)*	max 500 Bytes	The path where the images of Consumable product exist in the server.
Total Bytes:	888 Bytes		

DRINK	Type	Size	Constrains - Descriptions
Consumable_ID (F.K)	*INT*	8 Bytes	The unique ID of a Consumable Product.
Total Bytes:	8 Bytes		

FOOD	Type	Size	Constrains - Descriptions
Consumable_ID (F.K)	*INT*	8 Bytes	The unique ID of a Consumable Product.
Total Bytes:	8 Bytes		

LIKES_DISLIKES	Type	Size	Constrains - Descriptions
Consumable_ID (F.K)	*INT*	8 Bytes	The unique ID of a Consumable Product.
User_ID (F.K)	*INT*	8 Bytes	The unique ID of a user.
Like_it	*BOOLEAN*	1 Byte	Whether this product is liked (TRUE) by a user or not (FALSE).
Total Bytes:	17 Bytes		

USER	Type	Size	Constrains - Descriptions
User_ID (P.K)	*INT*	8 Bytes	The unique ID of a user.
Total Bytes:	8 Bytes		

RECOMMENDER_ATTRIBUTES	Type	Size	Constrains - Descriptions
User_ID (F.K)	*INT*	8 Bytes	The unique ID of a user.
Recommender_Attributes	*DOUBLE*	8 Bytes	A single attribute of a user. There will be many appearances of each ID of a user to this table, but for each user the number of appearances will be constant and equal to the number of appearances of any other user. This means that each user will have the same number of attributes. The number of attributes will be 40 for each user.
Total Bytes:	16 Bytes		

With all the information above we can define the total space needed for the data at a given time to be stored in the database as follows:

Total_Space = 388 * Number_of_Categories + 96 * Number_of_Journeys(In sum) + 16*Number_of_Journeys * Average_Foods_In_Journey*Number_of_users + 388 * Number_of_Product_Types + 16*Average_Product_Types_Related_to_a_Consumable_Product * Number_of_Consumable_Prodcucts + 16 * Average_General_Categories_Related_to_a_Product_Type * Number_of_Product_Types + 888 * Number_of_Consumable_Prodcucts +8*Number_of_Drinks + 8*Number_of_Foods + 17 * Likes_or_Dislikes_made + 8* Number_of_Users + Number_of_Users * (8 + Number_of_Attributes_each_user_has * 8) + (Number_of_Categories + Number_of_Consumable_Prodcucts + Number_of_Product_Types) * Size_of_The_Images

The last term (Number_of_Categories + Number_of_Consumable_Prodcucts + Number_of_Product_Types) * Size_of_The_Images shows the space needed for storing the images for the General Category, Product type and Consumable product in the Server. We know that the system will have only one image for each General Category, Product type and Consumable product. Despite this the structure of the system allows having more than one image for each of them.

Initially we estimate that the system will have approximately 30 General Categories, 500 Product Types (each of them refers to 3 general categories in average), 30,000 Consumable Products (7,000 drinks and 23,000 food), where each of

them refers to 5 Product Types in Average and approximately 300,000 users. → So, for these estimations we will need initially 388*30 + 388*500 + 16*5*30,000 + 16*3*500 + 888*30,000 + 8*7000 + 23000*8 + 300,000*8 + 300,000*(16+40*8) =132,709,640= 132.70964 MBytes → Approximately 133 MBytes

Suppose that each image that is going to be stored will have size 1 Mbyte.

The space allocated for the images stored in the Server will be → 30000+500+30 = **29.814GBytes**

Approximately both of them in total = 29.94GBytes

Note: These estimations are approximate estimations for near future possible Foody Online Orders system's needs in space, as the numbers are not supposed to indicate a current state of Foody current server and system but approximations of the space that will be needed when the Food Recommendation Agent system is going to be deployed.(So the numbers used and so the estimations got are a little bit higher than the current actual data).

Also, another good thing to calculate is the memory needed for the rest of the parts haven't been included in the previous calculations. It's actually valuable to make some estimations for the maximum space needed for the things that haven't been included in the previous estimations, because the behavior of a user is non-deterministic, we have to ensure that the amount of space needed in max in the worst case for the approximately 300,000 users is enough.

So, we can suppose that each user will be having 25 journeys, each of it having 30 foods in it and each user has made over all of the Consumable Products likes or dislikes. At that point we have → 96*25*300,000 + 16*25*30*300,000 + 17*30,000*300,000 = 1.5732 * 10^11 = **147.52 GBytes**

So, in total we will need initially approximately 147.52 Gbytes + 29.94GBytes = **177.46GBytes**

We can see that the data stored in the database are going to be somewhat static and not dynamic. That means that the data stored in the Database are going to be saved and not frequently deleted and so consequently the amount of data stored in the Database are not going to decrease day by day, but instead increase in average.

So, because our system is referring to customers in Cyprus and the population of Cyprus is approximately 1,000,000, we can see that with 1,000,000 users the previous equations become.

First Equation → 388*30 + 388*500 + 16*5*30,000 + 16*3*500 + 888*30,000 + 8*7,000 + 23,000*8 + 1,000,000*8 + 1,000,000*(16+40*8) =373,509,640= **356.21MBytes** → Approximately **356MBytes**

Second Equation → 96*25*1,000,000 + 16*25*30*1,000,000 + 17*30,000*1,000,000 = 5.244 * 10^11 = **488.39GBytes**

The space for the images stays the same.

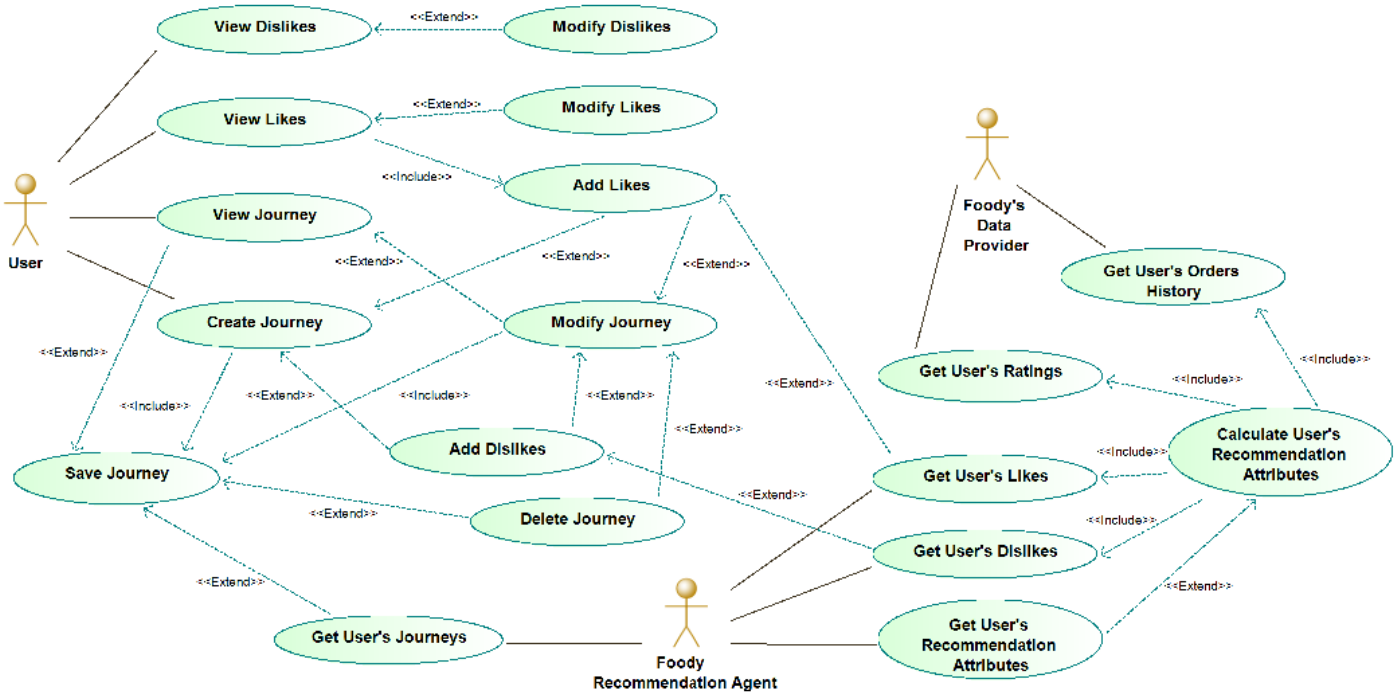
So, in total we will need 488.39GBytes + 356.21MBytes + 29.94= 518.68GBytes → Approximately **519GBytes**

As we can see, this is the worst case, where each user has 25 Journeys full of Consumable products in it. Even if all the Cyprus is using the Food Recommendation Agent system, the memory will be sufficient enough to store each of user's data and also the images of the system.

3. Use Cases

3.1 Actors

In the system there are 4 actors, who interact with it. The actors are: User, Foody's Data Provider, Recommender System, Foody Recommendation Agent. All the actors are independent, and no actor is the generalization of another.



3.1.1 Actors diagram

As we can see in the diagram above the User has the ability to, first of all, create, view, modify and save od delete a journey. The user can modify likes and dislikes, which means to delete them if he desires to through the Likes and Dislikes tab respectively. Moreover, through the creation or modification procedure of a journey, he can add liked and disliked foods.

Then we have the Foody Recommendation Agent. This actor can get all the data of each user (as long as they are saved in our Database), namely, get user's journeys, get user's likes, get user's dislikes and get user's recommendation attributes.

The Recommender System will be able to calculate User's recommendation attribute. Basically, calculate the ideal recommendations for the user. It is going to receive its input from the Foody's Data Provider for the user's order history and ratings, and from our Database for the Likes and Dislikes for each User.

Lastly, we have the Foody's Data Provider. This actor will provide to the Recommender System the user's order history and user's ratings which is going to collect from Foody's Database.

3.1.2 Actor descriptions

Description	The User is the actual person who will use the system. This actor is going to insert all of the data that the system will use in its calculations. Also, he is the purpose of existence of this system, because we are going to do the recommendations to him.
Aliases	None
Inherits	None
Actor Type	Active/Passive - Person

Description	Fooody Recommendation Agent is the actor that will take all that data from the user and parse them to the Recommender System.
Aliases	None
Inherits	None
Actor Type	Active/Passive - External System

Description	Another very important actor is the Recommender System. It's the actor that will do all the calculations in order to produce the recommendations of the user.
Aliases	None
Inherits	None
Actor Type	Active/Passive - External System

Description	*Foody's Data Provider is the actor that will get the data that are not contained in a journey, like the ratings of a restaurant and the other history.*
Aliases	None
Inherits	None
Actor Type	*Passive - External System.*

3.2 Use Case Descriptions

3.2.1 Create Journey

3.2.1.1 Diagram

For the use case Create Journey can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Create Journey is connected with the actor User and the use case Save Journey with **the include** condition, the use case Add Dislikes and Add Likes with the extend condition.

3.2.1.2 Description

A user can create Journeys through the Explore tab of our Web Page. Through this procedure, the user will have the ability to add products in the Journey, like a product or dislike it. Then the user will have the choice to save the Journey which is going to be stored in our Database.

3.2.1.3 Actors.

The actor interacting with this scenario is the User, who is going to pass through the product selection procedure for creating the content of the Journey

3.2.1.4 Precondition

There is no precondition in association with the FRA system, except that a user needs to be logged in to be able to create Journeys.

3.2.2 View Journey

3.2.2.1 Diagram

For the use case View Journey can be found at the complete Use Case Diagram in section 3.1 Actors. The use case View Journey is connected with the actor User and the use case Modify Journey with the extend condition.

3.2.2.2 Description

A user can view one of its already created Journeys through the My Journeys tab of our Web Page. From there, the user can modify the Journey to add or remove products or even delete it.

3.2.2.3 Actors

The actor interacting with this scenario is the User, who can view the content of the Journey and decide whether he desires to modify it or delete it.

3.2.2.4 Precondition

There is no precondition in association with the FRA system, except that a user needs to be logged in to be able to view Journeys. Also, in order to View a Journey, the Journey has to already being saved.

3.2.3 Save Journey

3.2.3.1 Diagram

For the use case Save Journey can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Save Journey is connected with the use case Create Journey with the include condition, and the use case View Journey with the extend condition.

3.2.3.2 Description

The user can save a Journey if he is satisfied from its content in order to use it for future recommendations in his orders.

3.2.3.3 Actors

The actor interacting with this scenario is the User, who can save a Journey after finishing the exploration procedure for adding content to the Journey or modifying it.

3.2.3.4 Precondition

In order for a user to save their Journey(s), they have to be logged in and to create or modify and existing Journey.

3.2.4 Modify Journey

3.2.4.1 Diagram

For the use case Modify Journey can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Modify Journey is connected with the use case Save Journey with the include condition, the use case Add Dislikes, Add Likes. Delete Journey and View Journey with the extend condition.

3.2.4.2 Description

The user can modify a Journey that had already created, to add, remove products or just rename it. In order to modify a Journey first he has to view it.

3.2.4.3 Actors

The actor interacting with this scenario is the User, who can modify an already created Journey, to add or remove products

3.2.4.4 Preconditions

In order for a user to Modify a Journey he has to already create and save it. Then he as to navigate to the My Journeys tab to View a Journey and then Modify it.

3.2.5 Delete Journey

3.2.5.1 Diagram

For the use case Delete Journey can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Delete Journey is connected with the use case Save Journey and Modify Journey with the extend condition.

3.2.5.2 Description

The user can delete an already created Journey if he desires to. After modifying a Journey will have the ability to fully delete it.

3.2.5.3 Actors

The actor interacting with this scenario is the User, who can delete one of its already created Journeys.

3.2.5.4 Precondition

The users can delete a Journey only if he has already created and saved it before. Also, they have to be logged in to do so.

3.2.6 View Likes

3.2.6.1 Diagram

For the use case View Likes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case View Likes is connected with the actor User and the use cases Modify Likes with the extend condition and Add Likes with the include condition.

3.2.6.2 Description

A user can navigate in the Likes tab of the FRA to view the Liked items that they have liked in the past if any. Moreover, they can add more items or remove items there by pressing the add or modify button respectively.

3.2.6.3 Actors

The actor interacting with this scenario is the User, who can view his liked products.

3.2.6.4 Precondition

There is no precondition in association with the FRA system, except that a user needs to be logged in to be able to view their Liked items.

3.2.7 View Dislikes

3.2.7.1 Diagram

For the use case View Dislikes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case View Dislikes is connected with the actor User and the use cases Modify Dislikes with the extend condition.

3.2.7.2 Description

A user can navigate in the Dislikes tab of the FRA to view the Disliked items that he has disliked in the past. Moreover, a user can only modify the dislikes in order to remove any of that group. The user does not have the ability to add Dislikes as it is part of Foody's requests not to transferring a negative feeling to the user. (Section 6.1)

3.2.7.3 Actors

The actor interacting with this scenario is the User, who can view his disliked products.

3.2.7.4 Precondition

There is no precondition in association with the FRA system, except that a user needs to be logged in Foody to be able to view their Disliked items.

3.2.8 Add Likes

3.2.8.1 Diagram

For the use case Add Likes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Add Likes is connected with the use case View Likes with the include condition and with the use cases Create Journey, Modify Journey and Get User's Dislikes with the extend condition.

3.2.8.2 Description

A user through the exploration procedure for the creation or modification of a Journey can like products which are going to be added in his Likes group. Also, if any liked products had been added the Foody Recommendation Agent and the Recommender System have access to them

3.2.8.3 Actors

The actors interacting with this scenario is the User, the Foody Recommendation Agent and the Recommender System who have access to the liked products of the User.

3.2.8.4 Precondition

To add Liked products the user can either press the Add Likes button in the Likes tab which is going to bring him in the Explore window to find products to like. Another way to Add Likes is through the creation or modification procedure of a Journey.

3.2.9 Add Dislikes

3.2.9.1 Diagram

For the use case Add Dislikes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Add Dislikes is connected with the use case Create Journeys, Modify Journeys and Get User's Dislikes with the extend condition.

3.2.9.2 Description

The user can add Disliked items only through the creation or modification procedure for a Journey. Also, if any disliked products had been added the Foody Recommendation Agent and the Recommender System have access to them

3.2.9.3 Actors

The actors interacting with this scenario is the User, the Foody Recommendation Agent and the Recommender System who have access to the disliked products of the User.

3.2.9.4 Precondition

To add Disliked products the user has to be in the Explore tab for the creation of modification of a Journey. The user can dislike products that can find through the exploration procedure.

3.2.10 Modify Likes

3.2.10.1 Diagram

For the use case Modify Likes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Modify Likes is connected with the use case View Likes with the extend connection.

3.2.10.2 Description

A user can modify their Liked items in the Likes tab, where they can add more items by exploring in the Explore tab, or delete any liked products.

3.2.10.3 Actors

The actor interacting with this scenario is the User, who can modify his already liked products.

3.2.10.4 Precondition

In order for the user to be able to Modify his Likes, he has to View them first in the Likes tab.

3.2.11 Modify Dislikes

3.2.11.1 Diagram

For the use case Modify Dislikes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Modify Dislikes is connected with the use case View Dislikes with the extend connection.

3.2.11.2 Description

A user can modify their Disliked items in the Dislikes tab, where they delete any already disliked products.

3.2.11.3 Actors

The actor interacting with this scenario is the User, who can modify his already disliked products.

3.2.11.4 Precondition

In order for the user to be able to Modify his Dislikes, he has to View them first in the Dislikes tab.

3.2.12 Get User's Journeys

3.2.12.1 Diagram

For the use case Get User's Journeys can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Journeys is connected with the actor Foody Recommendation Agent and with the use case Save Journeys with the extend connection.

3.2.12.2 Description

The Foody's Recommendation Agent will be able to take a user's Journeys from our Databases based on the user's ID. Foody then can use the Journey to perform direct recommendation to user based on the Journey's content. Also, in order for Foody's Recommendation Agent to be able to get user's Journey, the user has to first create and save a Journey in our Database, through the Explore tab of our Web Page.

3.2.12.3 Actors

The actor interacting with the scenario of this use case is the Foody Recommendation Agent who requests from our Database the Journeys of a User.

3.2.12.4 Precondition

The procedure of this use case will be executed by the FRA only if the user has any Journeys saved to their account, that is if they have created and saved at least one Journey.

3.2.13 Get User's Orders History

3.2.13.1 Diagram

For the use case Get User's Orders History can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Orders History is connected with the actor Foody's Data Provide and the use case Calculate User's Recommendation Attributes with the include connection.

3.2.13.2 Description

The Recommender System will get the user's order history from the Foody's Data Provider to calculate the recommendation attributes of the user.

3.2.13.3 Actors

The actor interacting with the scenario of this use case is the Foody's Data Provider.

3.2.13.4 Precondition

There are no preconditions, provided that the user has registered to Foody and made any orders. If not, then the order history will not be considered as an input for the Recommendation System.

3.2.14 Get User's Ratings

3.2.14.1 Diagram

For the use case Get User's Ratings can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Ratings is connected with the actor Foody's Data Provide and the use case Calculate User's Recommendation Attributes with the include connection.

3.2.14.2 Description

The Recommender System will get the user's ratings from the Foody's Data Provider to calculate the recommendation attributes of the user.

3.2.14.3 Actors

The actor interacting with the scenario of this use case is the Foody's Data Provider.

3.2.14.4 Precondition

There are no preconditions, provided that the user has registered to Foody and made some ratings in restaurants. If not, then the Recommendation System will generate random values for the ratings based on other parameters of a user such as the orders history.

3.2.15 Get User's Likes

3.2.15.1 Diagram

For the use case Get User's Likes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Likes is connected with the actor Foody Recommendation Agent and the use case Calculate User's Recommendation Attributes with the include connection and the user case Add Likes with the extend connection.

3.2.15.2 Description

The Recommender System will get the user's Liked items to help calculate the recommendation attributes of the user. Also, the Foody's Recommendation Agent can request the Likes of a specific user from our Database.

3.2.15.3 Actors

The actor interacting with the scenario of this use case is the User, the Foody Recommendation Agent.

3.2.15.4 Precondition

If the user does not have any Liked items then they will not be considered during the calculation of the user's recommendation attributes.

3.2.16 Get User's Dislikes

3.2.16.1 Diagram

For the use case Get User's Dislikes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Dislikes is connected with the actor Foody Recommendation Agent, the use case Calculate User's Recommendation Attributes with the include connection and the user case Add Dislikes with the extend connection.

3.2.16.2 Description

The Recommender System will get the user's Disliked items to help calculate the recommendation attributes of the user. Also, the Foody's Recommendation Agent can request the Dislikes of a specific user from our Database.

3.2.16.3 Actors

The actor interacting with the scenario of this use case is the User, the Foody Recommendation Agent.

3.2.16.4 Preconditions

If the user does not have any Disliked items then they will not be considered during the calculation of the user's recommendation attributes.

3.2.17 Calculate User's Recommendation Attributes

3.2.17.1 Diagram

For the use case Calculate User's Recommendation Attributes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Calculate User's Recommendation Attributes is connected with the use cases Get User's Order History, Get User's Ratings, Get User's Likes and Get User's Dislikes with the include connection. Also, it extends the Get User's Recommendation Attributed use case.

3.2.17.2 Description

The Calculate User's Recommendation Attributes will collect the User's Ratings, Orders History, Likes and Dislikes to calculate the recommendation attributes, and store them in our Database.

3.2.17.3 Actors

The actors interacting with the scenario of this use case are the Foody's Data Provider and the Foody Recommendation Agent.

3.2.17.4 Precondition

For this use case the Recommender System has to get the user's order history, ratings, likes and dislikes in order to calculate the recommendation attributes for the user. If any of these data don't exist, random values will be generated as input for the calculation instead.

3.2.18 Get User's Recommendation Attributes

3.2.18.1 Diagram

For the use case Calculate User's Recommendation Attributes can be found at the complete Use Case Diagram in section 3.1 Actors. The use case Get User's Recommendation Attributes is connected with the actor Foody Recommendation Agent and the use case Calculate User's Recommendation Attributes.

3.2.18.2 Description

The Recommender System has already collected the data, order history, ratings, likes and dislikes of a user from the Foody's Database and calculated the Recommendation Attributes. Then, Foody's Recommendation Agent can request the Recommendation Attributes for each user from our Database, in order to provide valid recommendations to its customers.

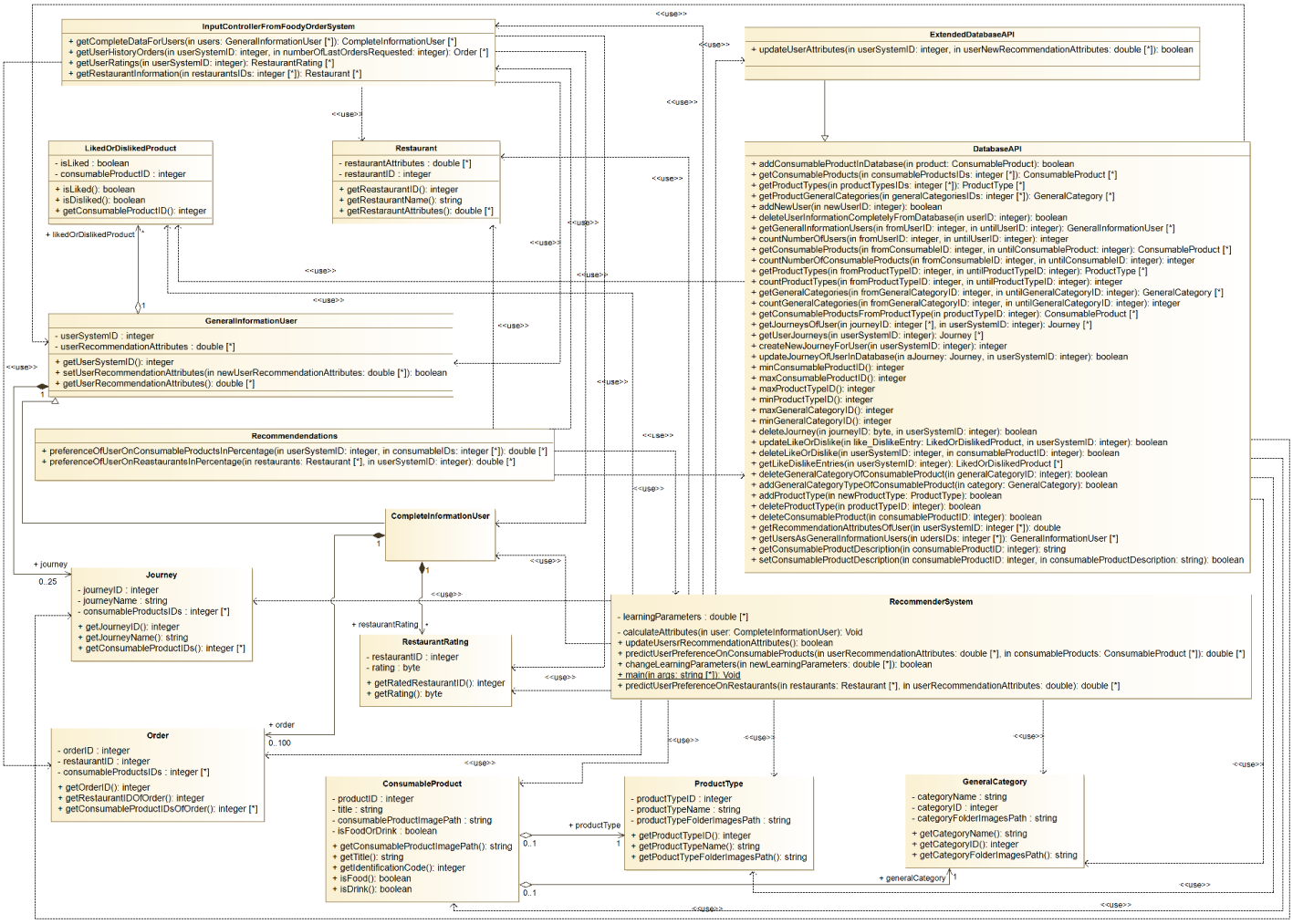
3.2.18.3 Actors

The actors interacting with the scenario of this use case are the Foody's Data Provider and the Foody Recommendation Agent.

3.2.18.4 Precondition

The recommendation attributes for the user have to be already calculated before the Foody Recommendation Agent being able to request them.

4. Domain Model Class Diagram



Notes: The classes Restaurant, LikedOrDislikedProduct, GeneralInformationUser, Journey, CompleteInformationUser, Order, RestaurantRating, ProductType, ConsumableProduct, GeneralCategory are used as "wrap" classes in order to store information in structured ways and to be used as a temporary container of information which can give access to the data when it is needed. As it is clear from the UML class diagram and also from the description below, no one class (except the GeneralUserInformation which offers setter method for the Recommendation Attributes of a user) offers setter methods. That's because there is no need for them, after the creation of the objects of the classes. More specifically each object of each of the classes mentioned above are initialized with values at their creation and as these objects are used in the scope of the Food Recommendation Agent system and for the purpose of extracting information from these objects mainly; changing information on the objects is not needed.

Brief descriptions of the classes, their attributes and their methods are the following:

GeneralInformationUser	
Description	This class represents the general information of a user. The information that it stores comes from the FoodRecommendationAgent Database. It is a container used in the program for a better structure of the user's information.
Attributes	
-userSystemID : integer	It is the ID of a user, as it is given by the FoodyOnlineOrders system.

-userRecommendationAttributes : double[]	The attributes of a user in an array that describe the user's preferences on Foods and Drinks (what he/she likes and what doesn't). ((I.e. attributes of a user might be {0.2,0.9,0.6} and might represent that the user likes 0.2 from 1 salad, 0.9 from 1 chicken and 0.6 from 1 fruit))
Aggregation relation - Attribute: LikedOrDislikedProduct[]	This attribute represents the products that the user with userSystemID likes or dislikes. There is no limit on the amount of products store in this array.
Composition relation - Attribute: Journey[]	This attribute represents the Journeys that a user with userSystemID has. The maximum number of Journeys that a user can have are 25.
Methods	
+getUserSystemID() : integer	This method returns the user System ID (userSystemID) of this object.
+setUserRecommendationAttributes(double newUserRecommendationAttributes[]): boolean	This method changes carefully the recommendation attributes of a user, stored in an object of this class. It takes as an argument an array of double, which are the new recommendation attributes and returns whether the attributes have been successfully changed.
+getUserRecommendationAttributes(): double[]	This method returns the recommendation attributes stored in the object from which the method is invoked. If there aren't any attributes, it returns null.

CompleteInformationUser	
Description	This class is a subclass of the GeneralInformationUser. It contains more information than its parent-class as it embeds information taken from FoodyOnlineOrders system's database. It represents a quite complete collection of information for a user. It is a container used in the program for a better structure of the user's information.
Attributes	
Composition relation - Attribute: Order []	This attribute represents the last orders of a user. They can be from 0 to 100.
Composition relation – attribute: RestaurantRating[]	This attribute represents the ratings of a user on restaurants.
Methods	
There are not any methods in this class.	

Journey	
Description	This class is used to describe a journey of a user. This class is used as a container class to build better structures for the information, in order to be easier to pass many information in an organized way.
Attributes	
-journeyID: integer	The id of a journey.
-journeyName: string	The name of the journey. It is public because the name of the journey can be changed some websites.
-ConsumableProductsIDs: integer[]	Every journey has up to 30 foods/drinks (consumable products). This attribute represents the different consumables products that the journey has.
Methods	
+getJourneyID() : integer	This method returns the id of the journey associated with this object.
+getJourneyName(): string	This method returns the name of the Journey, associated with the object from which the method is invoked.

+getConsumableProductIDs(): integer[]	This method returns all the Foods/Drinks (Consumable Products) that are stored in this Journey.

Order	
Description	This class is used to describe the necessary information needed by the FoodRecommendationAgent system for an order made by a user. This class is used as a container class to build better structures for the information, in order to be easier to pass many information in an organized way.
Attributes	
-orderID: integer	The id of the order as given by the FoodyOnlineOrders system.
-restaurantID: integer	The id of the restaurant from which the order occurred.
-consumableProductsIDs:integer[]	The Foods/Drinks that were ordered in the order associated with the orderID.
Methods	
+getOrderID() : integer	This method returns the orderID stored in the object of this class from which the method is invoked.
+getRestaurantIDOfOrder	This method returns the ID of the restaurant stored in the object of this class from which the method is invoked.
+getConsumableProductIDsOdOrder(): integer[]	This method returns the Foods/Drinks (Consumable Products) associated with the Order and stored in the object of this class from which the method is invoked.

RestaurantRating	
Description	This class describes the rating of a user on a restaurant.
Attributes	
-restaurantID : integer	This attribute represents the ID of the restaurant that was rated.
-rating: byte	This attribute represents the rating made on the restaurant with ID as designated by the attribute restaurantID. The size of this variable is byte because the rating is only 5 different numbers.
Methods	
+getRatedRestaurantID() : integer	This method returns the restaurantID stored in the object of this class from which the method is invoked.
+getRating() : byte	This method returns the rating on a restaurant stored in the object of this class from which the method is invoked.

LikedOrDislikedProduct	
Description	This class represents objects that show that a Consumable Product is liked or disliked by a user.
Attributes	
-isLiked : boolean	This attribute stores a Boolean value which shows whether a consumable product is liked or disliked. If it is liked, then it has a true value. If it is disliked, then it has a false.
-consumableProductID : integer	This is the ID of a Food/Drink which is liked or disliked as showed by the isLiked attribute.
Methods	
+isLiked() : boolean	This method returns whether the consumable product stored in the object of this class from which the method is invoked is liked.

+isDisliked() : boolean	This method returns whether the consumable product stored in the object of this class from which the method is invoked is disliked.
+getConsumableProductID() : integer	This method returns the ID of the consumable product associated with this object.

Restaurant	
Description	This class represents the essential and necessary by the FoodRecommendationAgent system; information of a restaurant, for learning the recommendation Attributes of a user.
Attributes	
-restaurantAttributes : double[]	This attribute shows the attributes of a restaurant. That is some values which specify what the restaurant offers and what the restaurant's variety of foods is. (I.e. {0.4,0.9,0.1} might represent that the restaurant has 0.4 from 1 salad, 0.9 from 1 chicken and 0.1 from 1 fruit)
-restaurantID : integer	This attribute represents the Identification code of a restaurant as it is given by the FoodyOnlineOrders system.
Methods	
+getRestaurantID() : integer	This method returns the ID of the restaurant stored in the object of this class from which the method is invoked.
+getRestaurantName() : string	This method returns the name of the restaurant associated the object of this class from which the method is invoked.
+getRestaurantAttributes() : double[]	This method returns the Restaurant Attributes of the restaurant associated with the object of this class from which the method is invoked.

GeneralCategory	
Description	This class represents general categories of Foods and Drinks (Consumable Products) such as Vegetables, meat, Fruits and so on.
Attributes	
-categoryName : string	This attribute represents the name of a general category of Foods/Drinks.
-categoryID : integer	This attribute represents the ID of the general Category of Foods/Drinks. It is an identification code that distinguishes this Foods/Drinks general category from other.
-categoryFolderImagesPath : string	This attribute shows where in a server the images about a general category; specified by an object of this class; are placed.
Methods	
+getCategoryName() :string	This method returns the name of the category represented by the object invoking this method.
+getCategoryID() : integer	This method returns the identification code of the category represented by the object invoking this method.
+getCategoryFolderImagesPath() : string	This method returns the path in the server where the images of the category associated with the object invoking the method reside.

ProductType	
Description	This class describes objects which specify more specialized types of foods than General Categories objects.
Attributes	
-productTypeName : string	This attribute represents the name of the product type of Foods/Drinks.

-productTypeID: integer	This attribute represents the ID of a product type of Foods/Drinks. It is an identification code that distinguishes this Foods/Drinks product type from other.
-productTypeFolderImagesPath : string	This attribute shows where in a server the images about a product type; specified by an object of this class; are stored.
Methods	
+getProductTypeName() :string	This method returns the name of the product type represented by the object invoking this method.
+getProductTypeID() : integer	This method returns the identification code of the product type represented by the object invoking this method.
+getProductTypeFolderImagesPath() : string	This method returns the path in the server where the images of the product type associated with the object invoking the method reside.

ConsumableProduct	
Description	This class describes a Food or a Drink and its necessary information needed for the system to use it combined with other information in order to learn the recommendation attributes of the users.
Attributes	
-productID: integer	An identification code that distinguishes this Food/Drink from other.
- Title: string	This attribute represents the name of the Food/Drink represented by an object of this class.
-isFoodOrDrink : boolean	This attribute shows whether this consumable product is a food or a drink. If it is a food then it is true, otherwise it is false.
-consumableProductImagePath : string	This attribute shows where in a server the image about a consumable product; specified by an object of this class; is stored.
Methods	
+getConsumableProductImagepath() : string	This method returns a string representing where in a server the image, associated with this consumable product is stored.
+getTitle() : string	This method returns the title-name of the consumable product represented by the object invoking the method.
+getIdentificationCode() : integer	This method returns the ID of the consumable product represented by the object invoking this method.
+isFood() : Boolean	This method returns true if the consumable product is a Food, otherwise it returns false.
+isDrnk() : Boolean	This method returns true if the consumable product is a Drink, otherwise it returns false.

Recommendations	
Description	This class represents the functionalities that are offered by the FoodRecommendationAgent system to the FoodyOnlineOrders system, associated with the recommendation of Foods/Drinks and restaurants to customers.
Attributes	
There are not any attributes in this class.	
Methods	
+preferenceOfUserOnConsumableProductsInPercentage(integer userSystemID, integer consumableIDs[]) :double[]	This method takes as parameters the user ID as defined by the system and an array of the IDs of aome consumable products. It returns an array of doubles of the same length as the array of consumable products' IDs. Each position of the former array represents a possibility that that the food/drink at the

	corresponding position in the consumable product IDs array is to be liked by the user with ID userSystemID. (i.e. userSystemID=7 and consumableIDs={1,2} -> Method returns {0.4, 0.8} . That means that user with ID 7 would possibly like the product with ID 1 with 0.4 possibility and the product with ID 2 with 0.8 possibility).
+preferenceOfUserOnRestaurantsInPercentage(Restaurant[] restaurants, integer userSystemID) : double[]	This method takes as parameters the user ID as defined by the system and an array of some restaurants. It returns an array of doubles of the same length as the array of restaurants. Each position of the former array represents a possibility that that the restaurant at the corresponding position in the restaurants array is to be liked by the user with ID userSystemID. (i.e. userSystemID=7 and restaurantIDs= {restaurantObject1, restaurantObject2} -> Method returns {0.4, 0.8} . That means that user with ID 7 would possibly like the restaurant restaurantObject1 with 0.4 possibility and the restaurant restaurantObject2 with 0.8 possibility).

InputControllerFromFoodyOrderSystem	
Description	This class is used in order to load data to the FoodRecommendationAgent system from the database of the FoodyOnlineOrders system. Although these data are information not stored in the FoodRecommendationAgent system's Database, are needed in order to calculate the recommendation attributes of a user This information includes user's last N orders (where N can be from 0 to 100, inclusive), user's restaurant ratings, restaurants' attributes(attributes which describe the type of the restaurant and what it offers).
Attributes	
There are not any attributes for this class.	
Methods	
+getCompleteDataForUsers(GeneralInformationUser[] users) :CompleteInformationUser[]	This method takes an array of objects of type GeneralInformationUser, which represents the general information of each user, as described in the description of the GeneralInformationUser class, and returns an array of the same length as the array taken as input, which in each position in this array, a CompleteInformationUser object is placed which corresponds to the GeneralInformationUser object at the same position from the input array. Each of the objects in the array returned, has more information than its corresponding object in the input array. The new information used to create each of the new objects is taken from the FoodyOnlineOrders systen's database.
+getUserHistoryOrder(integer userSystemID, integer numberOfLastOrdersRequested) : Order[]	This method collects the necessary information of the last numberOfLastOrdersRequested Orders of the user with ID userSystemID and returns an array with objects of type Order, which each of the objects contains the necessary information needed by the FoodyRecommendationAgent system to calculate the recommendation attributes of the user with ID userSystemID.
+getUserRatings(integer userSystemID) : RestaurantRating[]	This method collects the necessary information about the restaurant's ratings of the user with ID userSystemID from the FoodyOnlineOrders system's database and it returns an array which each object in the array represents a rating on a restaurant (RestaurantRating object).
+getRestaurantInformation(integer restaurantsIDs[]) : Restaurant[]	This method returns the information of the restaurants that are denoted by the restaurants IDs in the array given as an input to the method. The array returned is an array of Restaurant objects in it. The ith Restaurant object in the array returned has ID restaurantIDs[i] .

RecommenderSystem

Description	This class represents the part of the system that is going to be executing the algorithms for learning the recommendation Attributes of a user.
Attributes	
-learningParameters : double	This attribute represents some special parameter values that are used by the algorithms of this class (the algorithms in the methods) for calculating the attributes of a user and for predicting the preferences of a user on Foods/Drinks and restaurants.
Methods	
-calculateAttributes(CompleteInformationUSer user) : Void	This method is private because it is used as an auxiliary method for the objects of this class. Moreover, it takes as an argument an object of type CompleteInformationUser and it updates the recommendation attributes of that user int the Database of the FoodRecommendationAgent system by invoking the updateUserAttributes of the class ExtendedDatabaseAPI.
+updateUsersRecommendationAttributes() : boolean	This method calculates and updates the recommendation attributes of all the users in the Database of the FoodRecommendationAgent system. It returns whether the calculation and update were successful.
+predictUserPreferenceOnConumableProducts(ConsumableProduct consumableProducts[], double userRecommendationAttributes[]) : double[]	This method predicts whether some Foods/Drinks are likely to be liked by a user with recommendation attributes as indicated by the array userRecommendationAttributes. It returns an array of the same length as the array consumableProducts, given as an input, and the ith number stored in the array returned represents the possibility that the ith consumable product(represented by the ith ConumableProduct object) in the array given as an input is liked by the user with recommendation attributes as indicated by the userRecommendationAttributes.
+predictUserPreferenceOnConumableProducts(Restaurant restaurants[], double userRecommendationAttributes[]) : double[]	This method predicts whether some restaurants are likely to be liked by a user with recommendation attributes as indicated by the array userRecommendationAttributes. It returns an array of the same length as the array restaurants, given as an input. The ith number stored in the array returned represents the possibility that the ith restaurant (represented by the ith Restaurant object) in the array given as an input is liked by the user with recommendation attributes as indicated by the userRecommendationAttributes.
+changeLearningParameters(double[] newLearningParameters) : double[]	This method changes the values of the learning parameters stored in the object from which the method is invoked, to the new values passed as argument to the method.
+main(string[] args):Void	This method is the main method of the program that supports the algorithms for learning the recommendation attributes of each user.

DatabaseAPI	
Description	This class describes objects, that provide with an application programming interface for other programs to use the Database. This is the general Database API that allows the FoodyOnlineOrders system (as well as the FoodRecommendationAgent system) to apply general changes to some data in the database, delete some data and retrieve data from it.
Attributes	
This class does not have any attributes.	
Methods	
+addNewUser(integer newUserID) : boolean	This method creates a new user in the Database and returns true if it was created successfully, otherwise returns false. The desired ID of

	the user is given as an argument. If the ID exists, then it won't create the user and it will return false.
+deleteUserInformationCompletelyFromDatabase(integer userID): boolean	This method deletes all the information about a user with ID userID and returns true if the deletion was successful, or false otherwise.
+getGeneralInformationUsers(integer fromUserID, untilUserID) : GeneralInformationUser[]	This method returns all the users with IDs between fromUserID and untilUserID inclusive.
+countNumberOfUsers(integer fromUserID, integer untilUserID) : integer	This method returns the number of users, with ID fromUserID to untilUserID inclusive, that there is information about them in the database.
+getConsumableProducts(integer fromConsumableProductID, untilConsumableProductID) : ConsumableProduct[]	This method returns all the consumable products with IDs between fromConsumableProductID and until ConsumableProductID inclusive.
+countNumberOfConsumableProducts(integer fromConsumableProductID, untilConsumableProductID) : integer	This method returns the number of consumable products, with ID fromConsumableProductID to untilConsumableProductID inclusive, that there is information about them in the database.
+addConsumableProductInDatabase(ConsumableProduct product) : boolean	This method adds a new Consumable product to the Database. If the insertion was successful, then it returns true, otherwise false. Reasons for not inserting the new consumable product is because its ID is not unique.
+getConsumableProducts(integer[] consumableProductsIDs) : ConsumableProduct[]	This method returns objects of type ConsumableProduct that have ID one of the integers in the array consumableProductsIDs from the Database.
+getConsumableProductsFromProductType(integer productTypeID) : ConsumableProduct[]	This method takes all the ConsumableProduct objects (consumable products foods or drinks) that belong to one of the product Type which have IDs as denoted by the array productTypeID.
+minConsumableProductID() : integer	This method returns the minimum ID number of a consumable Product that is stored in the Database.
+deleteConsumableProduct(integer consumableProductID) : boolean	This method deletes the consumable product from the database which has ID the consumableProductID. If the deletion was successful it returns true, otherwise it returns false.
+maxConsumableProductID() : integer	This method returns the maximum ID number of a consumable Product that is stored in the Database.
+getProductTypes(integer[] productTypesIDs) : ProductType[]	This method collects from the Database and returns the product types of Foods/Drinks (Product Type objects) in an array that correspond (have) as ID one of the integers stored in the array productTypesIDs.
+getProductTypes(integer fromProductType, integer untilProductType) : ProductType[]	This method returns all the Product Types with IDs between fromProductType and untilProductType inclusive.
+ countProductTypes(integer fromProductType, integer untilProductType) : integer	This method returns the number of all the Product Types with IDs between fromProductType and untilProductType inclusive.
+getProductGeneralCategories(integer[] generalCategoriesIDs) : GeneralCategory[]	This method collects from the Database and returns the General Categories in an array of Foods/Drinks (General Category objects) that correspond (have) as ID one of the integers stored in the array generalCategoriesIDs.
+minProductTypeID() : integer	This method returns the minimum Product Type ID number of consumable Products that is stored in the Database.
+maxProductTypeID() : integer	This method returns the maximum Product Type ID number of consumable Products that is stored in the Database.
+addProductType(ProductType newProductType) : boolean	This method adds a new Product type and its associated data to the Database. Takes an object of type newProductType, takes the information inside of it and tries to create a new ProductType entry

	in the database. It returns true if the insertion was successful or false otherwise.
+deleteProductType(integer productTypeID) : boolean	This method deletes all the information about the product Type that has as ID the productTypeID, from the database. It returns true if the deletion was successful or false otherwise.
+getGeneralCategories(integer fromGeneralCategoryID, integer untilGeneralCategoryID) : GeneralCategory[]	This method returns all the General Categories with IDs between fromGeneralCategoryID and untilGeneralCategoryID inclusive.
+countGeneralCategories(integer fromGeneralCategoryID, integer untilGeneralCategoryID) : integer	This method returns the number of all the General Categories with IDs between fromGeneralCategoryID and untilGeneralCategoryID inclusive.
+minGeneralCategotyID() : integer	This method returns the minimum General Category ID number of consumable Products that is stored in the Database.
+maxGeneralCategotyID() : integer	This method returns the maximum General Category ID number of consumable Products that is stored in the Database.
+deleteGeneralCategoryOfConsumableProduct(integer generalCategoryID) : boolean	This method deletes all the information about the General Category that has as ID the generalCategoryID, from the database. It returns true if the deletion was successful or false otherwise.
+addGeneralCategoryTypeOfConsumableProduct(GeneralCategory category) : boolean	This method adds a new General Category and its associated data to the Database. Takes an object of type newProductType, takes the information inside of it and tries to create a new GeneralCategory entry in the database. It returns true if the insertion was successful or false otherwise.
+getJourneysOfUser(integer journeyID[], integer userSystemID) : Journey[]	This method returns all the Journeys (array of Journey objects)of the user with ID userSystemID, that have ID one of the byte numbers stored in the array journeyID, which was passed as an argument to the method.
+getUserJourneys(integer userSystemID) : Journey[]	This method returns all the Journeys (array of Journey objects) of the user with ID userSystemID.
+createNewJourneyForUser(integer userSystemID) : integer	This method creates a new Journey for the user with ID userSystemID and returns the ID of the Journey if the creation was successful. If it wasn't it (i.e. user has already 25 journeys, it) returns a negative number.
+updateJourneyOfUserInDatabase(Journey aJourney, integer userSystemID) : boolean	This method takes as a parameter an object Journey and the ID of a user and it updates the information of the Journey of the user with ID userSystemID in the Database that corresponds to the Journey passed as an argument. The update is made using the information in the Journey object passed as a parameter. The ID of the Journey passed as an argument should be an actual Journey ID of the user which corresponds to an actual Journey of the user. Then the information in the object passed as an argument updates the information in the Database.
+deleteJourney(byte JourneyID, integer userSystemID) : boolean	This method deletes the Journey of the user with ID userSystemID, that has ID JourneyID. It returns true if the deletion was succesful or false otherwise.
+updateLikeOrDislike(LikedOrDislikedProduct like_DislikeEntry, integer userSystemID) : boolean	This method updates the Like or Dislike state of a consumable product for a user in the Database. It takes an object which specifies a consumable product and whether the user with ID userSystemID likes it or not and it updates the Database with that information. If the consumable product being liked/disliked by the user (entry in the database about like or dislike of the consumable product by the user) doesn't exist, then it is added in the database. Otherwise the information which is described on the object passed as an argument,

	updates the information of the liked/disliked by the user with ID userSystemID entry in the database. Furthermore, it returns true if the update was successful or false otherwise.
+deleteLikeDislikeEntries(integer userSystemID, integer consumableProductID) : boolean	This method deletes entirely a like/dislike entry which corresponds to the consumable product with ID consumableProductID of the user with ID userSystemID from the Database. It returns true if the deletion was succesful or false if it wasn't.
+getLikeDislikeEntries(integer userSystemID) : LikedOrDislikedProduct[]	This method retrieves from the database and returns all the entries of liked/disliked consumable products as an array of LikedOrDislikedProduct objects.
+getRecommendationAttributesOfUser(integer userSystemID[]) : double	This method returns the recommendation attributes as an array of doubles for the user with ID userSystemID which are stored in the database.
+getUsersAsGeneralInformationUsers(integer usersIDs[]) : GeneralInformationUser[]	This method returns the GeneralInformationUser objects (as an array) that correspond to the integers stored in the usersIDs. That is it returns a collection of the general information of each user that has as user-system ID one of the integers exist in the array of integers usersIDs. It returns an array of GeneralInformationUser objects. The ith GeneralInformationUser object in the array has as user-system ID the usersIDs[i] integer.
getConsumableProductDescription(integer consumableProductID): string	This method returns the description of the consumable product with ID consumableProductID, as given as argument.
setConsumableProductDescription(integer consumableProductID, string consumableProductDescription): boolean	This method takes the ID of a consumable product and its description and tries to update the consumable product's description in the database with the new passed as an argument. If the update was successful it returns true, otherwise it returns false.

5. Activities Plan

5.1 Development Phases

5.1.1 Requirement Gathering and Analysis

Collect the requirements for our system from research and personal interviews with Foody. We observe that Foody's recommendations are more general, and very often they recommend things that the users don't desire. We decided that there is a need of a Recommender System for Foody's website in order to improve sales by making more appropriate recommendations to the users. We made some research about this technique and we discovered that other big tech companies such as Amazon uses a recommender system, in order to improve sales up-to 30%.

5.1.2 Design

Decide the main components of the system based on the requirements gathering and analysis phase. We decided to design a website were the user can Like or Dislike foods and create personal Journeys which are groups of foods. Through this input and some other parameters collected from Foody's Database, the user will be able to receive more appropriate recommendations. For these results a machine learning Recommender algorithm is going to be used to create the recommendation attributes for each user.

5.1.3 Implementation

Implementation is being split into two major sections. In the first phase of the implementation will take place between November and December of 2019 and the second phase between June and July of 2020.

In the first phase we will implement the Journeys and Likes/Dislikes Web page. We will create some extra functionalities to help the Testing phase such as a Login screen and a kind of Database that will represent Foody's Database. In this phase we are going to implement our Database too, for storing user's data from the Journeys and Likes/Dislikes.

In the second phase we will implement the Recommender System. The Recommender System is going to use the Likes/Dislikes and the Foody's Database for its inputs. So, the implementation of the Database that represents Foody's Database will be used for testing in this phase too. Moreover, we are going to use our Database, that had been created in the first phase, to store data that have been produced through the Recommender System.

5.1.4 Testing

In the Testing phase we are going to use our temporary Database that will represent Foody's Database to get information for some users and see how the Recommender System performs. Also, we are going to use fake accounts to test the login and how the Journeys and Likes/Dislikes Web page adapts for each user. each user should be able to see the Journeys that had created and the Likes or Dislikes that had select. We will test the API for collecting data from our Database, which Foody is going to use. Finally, we will use JSON files to temporary import some images and titles for foods to be used in the Journeys and Likes/Dislikes Web page.

5.1.5 Deployment

We will remove the Login screen that had been created for testing, because we will authenticate the user through Foody's website. The Journeys and Likes/Dislikes Web page will be embedded to Foody' website in order to become part of it.

We will change our method of collecting data for the Recommender System from our temporary Database to Foody's Database. Also, the Journeys and Likes/Dislikes systems will be collecting data from Foody's database for the pictures and titles of foods.

We will provide an API to Foody for our System, in order for them to gain access in our Database which will contain data from users Journeys, Likes/Dislikes and the Recommender System.

We will install our Database that is required for the Journeys Web page and Recommender System to work. We will install the Recommender System in one of Foody's server and execute it in order to start collecting data from Foody's Database.

5.2 Activity Plan

5.2.1 Milestones

- First meeting with Foody to find out their requirements and discuss about our project idea for improving their current System at 25 September 2019
- First deliverable with System Analysis, Specific Requirements, Dependencies and Constraints at 11 October 2019
- Second meeting with Foody to finalize the details of the System and show examples of wireframes at 15 October 2019
- Second deliverable with UML, Use Case and Gantt diagrams at 21 October 2019

- Third meeting with Foody to discuss about the process and for any improvements or changes during the implementation period at 1 November 2020
- Third deliverable with Architecture Analysis, Analytical Class Diagrams and Sample Scenarios at 4 November 2020
- Finish all the basic HTML structures for the pages My Journeys, Explore and Likes/Dislikes at 20 November 2020
- Deliver the implementation of the First Phase that is consisted of the Journeys and Likes/Dislikes Web Pages at 29 November 2019
- Fourth meeting with Foody to discuss about the first phase and the requirements of the second Phase 1 June 2020
- Finish basic structure of the Second Phase and the Recommender System Machine Learning algorithm at 30 June 2020
- Fourth deliverable with Implementation Analysis for the Second Phase at 7 July 2020
- Fifth deliverable with Description of the Final System's API which combines the First and Second Phase together at 15 July 2020
- Fifth meeting with Foody to present the merged version of the System consisted of the first and second phase. Request for real data to test the system and collect statistics at 20 July 2020
- Deliver a Beta version of the System with the First and Second Phase merged at 4 August 2020
- Sixth meeting with Foody to present the Beta version of the product and request for feedback to implement the final optimizations at 7 August 2020
- Deliver the Final version of the System with the First and Second Phase merged at 11 August 2020

5.2.2 Deliverables

- First deliverable with System Analysis, Specific Requirements, Dependencies and Constraints at 11 October 2019
- Second deliverable with UML, Use Case and Gantt diagrams at 21 October 2019
- Third deliverable with Architecture Analysis, Analytical Class Diagrams and Sample Scenarios at 4 November 2020
- Deliver the implementation of the First Phase that is consisted of the Journeys and Likes/Dislikes Web Pages at 29 November 2019
- Fourth deliverable with Implementation Analysis for the Second Phase at 7 July 2020
- Fifth deliverable with Description of the Final System's API which combines the First and Second Phase together at 15 July 2020
- Deliver a Beta version of the System with the First and Second Phase merged at 4 August 2020
- Deliver the Final version of the System with the First and Second Phase merged at 11 August 2020

5.2.3 Tasks

5.2.3.1 Implementation Phase 1

1. Learn required technologies to implement the Database, the Web Page and the Recommender System (Material.io, JavaScript, HTML, CSS, Python, MySQL)
2. Top Bar implementation with the ability of changing HTML pages.
3. Image Grid for the Likes/Dislikes and Explore HTML pages.
4. Log In implementation for temporary use until Phase 2.
5. JSON Parser for importing the Pictures with their IDs, Category and Title to use it in the Likes/Dislikes and Journeys Web Page.
6. Like/Dislike HTML page to view your Liked and Disliked products respectively.
7. Database design and implementation for storing the data of the Web Page and the Recommender System.

8. Explore HTML page with the basic components to develop on top the Expansion Mechanism and the Side Menu.
9. Side Menu in the Explore page showing the current selected products to be added in the new Journey that the user creates.
10. Likes/Dislikes functionality when you select a food in the Explore HTML page
11. My Journeys HTML page presenting the Journeys that a user had already created.
12. Save and Load from Database functionality.
13. Test responsiveness to be supported in different screen ratios and sizes.
14. Expansion Mechanism for the Explore page as described in the First Deliverable.
15. Combine all the parts together and test for any incompatibilities.

5.2.3.2 Implementation Phase 2

1. Foody's Database Reader for collecting the data from Foody's Database.
2. API for our Database in order to give access to Foody for any requests.
3. Algorithm optimizations and improvements from Phase 1 to match the requirements of Phase 2.
4. Recommender System Machine Learning algorithm implementation.
5. Replace Log-In page with the automatic authentication of Foody's website.
6. Modify JSON Parser to use also Foody's Database for content validation.
7. Intensive test and improve of the Recommender System algorithm.
8. Merge Phase 1 and Phase 2 together, in order to use the Web Page as input for the Recommender's System Machine Learning algorithm.
9. Use data from Foody's real users as input to the Recommender System and collect statistics for performance issues and potential improvements.
10. Final optimizations in the algorithm and the system in general after examining the collected statistics.

5.2.4 Time plan

5.2.4.1 Analysis

Activity	Duration	Resources
Find problems in Foody's System	2 days	All Team
Research for the idea	6 days	All Team
Fill First Deliverable	10 days	All Team
Fill Second Deliverable	10 days	All Team
Fill Third Deliverable	10 days	All Team
Fill Fourth Deliverable	10 days	All Team
Fill Fifth Deliverable	10 days	All Team

5.2.4.1 Implementation Phase 1

Activity	Duration	Resources
Team Meeting for clarifications	1 days	All Team
Top Bar Implementation	4 days	Leonidas Achilleos
Image Grid	4 days	Valentinos Parizza

Log In	7 days	Demetris Shimitras
JSON Parser	4 days	Marios Pafitis
Like/Dislike HTML	6 days	Stephanos Pantziaros
Database Implementation	6 days	Leonidas Achilleos
Explore HTML	6 days	Marios Pafitis
Side Menu	5 days	Stephanos Pantziaros
Likes/Dislikes Functionality	5 days	Valentinos Parizza
My Journeys HTML	5 days	Demetris Shimitras
Save and Load in Database	5 days	Leonidas Achilleos
Test Responsiveness	3 days	Marios Pafitis
Expansion Mechanism	5 days	Valentinos Parizza
Combine all parts and Test	4 days	All Team

5.2.4.3 Implementation Phase 2

Activity	Duration	Resources
Foody's Database Reader	10 days	Marios Pafitis
API for our Database	15 days	Leonidas Achilleos
Algorithm Optimizations in Phase 1	7 days	Stephanos Pantziaros
Recommendations ML algo.	20 days	Valentinos Parizza
Replace Log-In page	7 days	Demetris Shimitras
Modify JSON Parser	7 days	Marios Pafitis
Test & Improve Recommendations algo.	10 days	Stephanos Pantziaros
Merge Phase 1 & Phase 2	5 days	Leonidas Achilleos
Test Foody's Real Data & Statistics	20 days	All Team
Final Optimizations	7 days	Demetris Shimitras

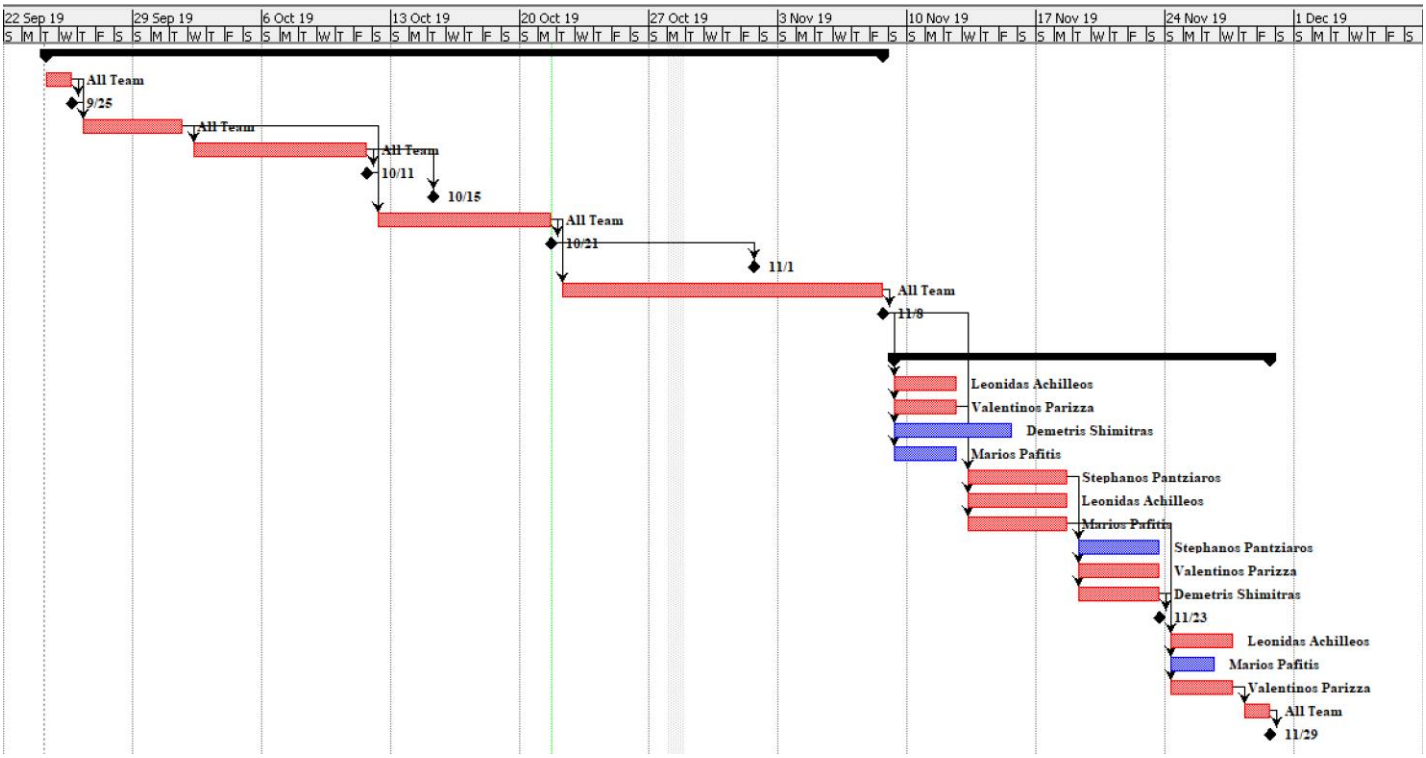
5.2.4.4 Maintenance

Activity	Duration	Resources
Handle any Issues	7 days	All Team

5.2.5 Gantt Diagram

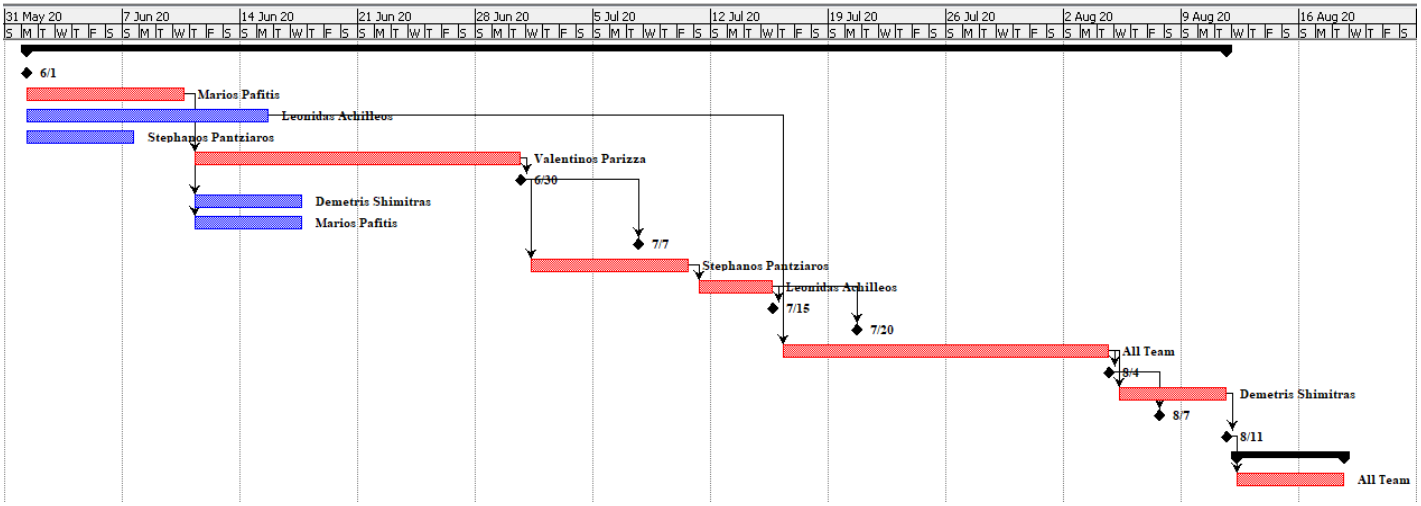
With the red color we can see the critical path of our Project. The Implementation of our System is being split into two Phases. The First Phase will take place from 24 September 2019 to 29 November 2019. The Second Phase will take place from 1 June 2020 to 11 August 2020. In the First Phase we will implement the Journeys, Likes/Dislikes Web Pages and the Database. During the Second Phase we will implement the Recommender System Machine Learning algorithm and combine the First and Second Phase together.

5.2.5.1 Gantt Phase 1



Name	Duration	Start	Finish	Predecessors	Resource Names
Analysis	45 days	**9/24/19 8:00 AM**	**11/8/19 5:00 PM**		
Find problems in Foody's System	2 days	9/24/19 8:00 AM	9/25/19 5:00 PM		All Team
First meeting with Foody	0 days	9/25/19 5:00 PM	9/25/19 5:00 PM	2	All Team
Research for the idea	6 days	9/26/19 8:00 AM	10/1/19 5:00 PM	2;3	All Team
Fill First Deliverable	10 days	10/2/19 8:00 AM	10/11/19 5:00 PM	4	All Team
Deliver First Deliverable	0 days	10/11/19 5:00 PM	10/11/19 5:00 PM	5	All Team
Second meeting with Foody	0 days	10/15/19 8:00 AM	10/15/19 8:00 AM	5	All Team
Fill Second Deliverable	10 days	10/12/19 8:00 AM	10/21/19 5:00 PM	4;6	All Team
Deliver Second Deliverable	0 days	10/21/19 5:00 PM	10/21/19 5:00 PM	8	All Team
Third meeting with Foody	0 days	11/1/19 6:00 PM	11/1/19 5:00 PM	9	All Team
Fill Third Deliverable	17 days	10/22/19 8:00 AM	11/8/19 5:00 PM	8	All Team
Deliver Third Deliverable	0 days	11/8/19 5:00 PM	11/8/19 5:00 PM	11	All Team
Implementation Phase 1	21 days	**11/9/19 8:00 AM**	**11/29/19 5:00 PM**		
Top Bar Implementation	4 days	11/9/19 8:00 AM	11/12/19 5:00 PM	12	Leonidas Achilleos
Image Grid	4 days	11/9/19 8:00 AM	11/12/19 5:00 PM	12	Valentinos Parizza
Log In	7 days	11/9/19 8:00 AM	11/15/19 5:00 PM	12	Demetris Shimitras
JSON Parser	4 days	11/9/19 8:00 AM	11/12/19 5:00 PM	12	Marios Pafitis
Like/Dislike HTML	6 days	11/13/19 8:00 AM	11/18/19 5:00 PM	16	Stephanos Pantziaros
Database Implementation	6 days	11/13/19 8:00 AM	11/18/19 5:00 PM	12	Leonidas Achilleos
Explore HTML	6 days	11/13/19 8:00 AM	11/18/19 5:00 PM	12	Marios Pafitis
Side Menu	5 days	11/19/19 8:00 AM	11/23/19 5:00 PM	21	Stephanos Pantziaros
Likes/Dislikes Functionality	5 days	11/19/19 8:00 AM	11/23/19 5:00 PM	19	Valentinos Parizza
My Journeys HTML	5 days	11/19/19 8:00 AM	11/23/19 5:00 PM	21	Demetris Shimitras
Finish the basic HTML Pages Structure	0 days	11/23/19 5:00 PM	11/23/19 5:00 PM	24	All Team
Save and Load in Database	4 days	11/24/19 8:00 AM	11/27/19 5:00 PM	24	Leonidas Achilleos
Test Responsiveness	3 days	11/24/19 8:00 AM	11/26/19 5:00 PM	24	Marios Pafitis
Expansion Mechanism	4 days	11/24/19 8:00 AM	11/27/19 5:00 PM	21	Valentinos Parizza
Combine all parts and Test	2 days	11/28/19 8:00 AM	11/29/19 5:00 PM	28	All Team
Deliver Phase 1	0 days	11/29/19 5:00 PM	11/29/19 5:00 PM	29	All Team

5.2.5.2 Gantt Phase 2



Name	Duration	Start	Finish	Predecessors	Resource Names
Implementation Phase 2	72 days	**6/1/20 8:00 AM**	**8/11/20 5:00 PM**		
Fourth meeting with Foody	0 days	6/1/20 8:00 AM	6/1/20 8:00 AM		All Team
Foody's Database Reader	10 days	6/1/20 8:00 AM	6/10/20 5:00 PM		Marios Pafitis
API for our Database	15 days	6/1/20 8:00 AM	6/15/20 5:00 PM		Leonidas Achilleos
Algorithm Optimizations in Phase 1	7 days	6/1/20 8:00 AM	6/7/20 5:00 PM		Stephanos Pantziaros
Recommendations ML algo.	20 days	6/11/20 8:00 AM	6/30/20 5:00 PM	34	Valentinos Parizza
Finish Basic Structure of Phase 2	0 days	6/30/20 5:00 PM	6/30/20 5:00 PM	37	All Team
Replace Log-In page	7 days	6/11/20 8:00 AM	6/17/20 5:00 PM	34	Demetris Shimitras
Modify JSON Parser	7 days	6/11/20 8:00 AM	6/17/20 5:00 PM	34	Marios Pafitis
Deliver Fourth Deliverable	0 days	7/7/20 5:00 PM	7/7/20 5:00 PM	38	All Team
Test & Improve Recommendations algo.	10 days	7/1/20 8:00 AM	7/10/20 5:00 PM	38	Stephanos Pantziaros
Merge Phase 1 & Phase 2	5 days	7/11/20 8:00 AM	7/15/20 5:00 PM	42	Leonidas Achilleos
Deliver Fifth Deliverable	0 days	7/15/20 5:00 PM	7/15/20 5:00 PM	43	All Team
Fifth meeting with Foody	0 days	7/20/20 5:00 PM	7/20/20 5:00 PM	43	All Team
Test Foody's Real Data & Statistics	20 days	7/16/20 8:00 AM	8/4/20 5:00 PM	35;43	All Team
Deliver Beta	0 days	8/4/20 5:00 PM	8/4/20 5:00 PM	46	All Team
Final Optimizations	7 days	8/5/20 8:00 AM	8/11/20 5:00 PM	46	Demetris Shimitras
Sixth meeting with Foody	0 days	8/7/20 5:00 PM	8/7/20 5:00 PM	47	All Team
Deliver Project	0 days	8/11/20 5:00 PM	8/11/20 5:00 PM	48	All Team
Maintenance	7 days	**8/12/20 8:00 AM**	**8/18/20 5:00 PM**		
Handle any Issues	7 days	8/12/20 8:00 AM	8/18/20 5:00 PM	50	All Team

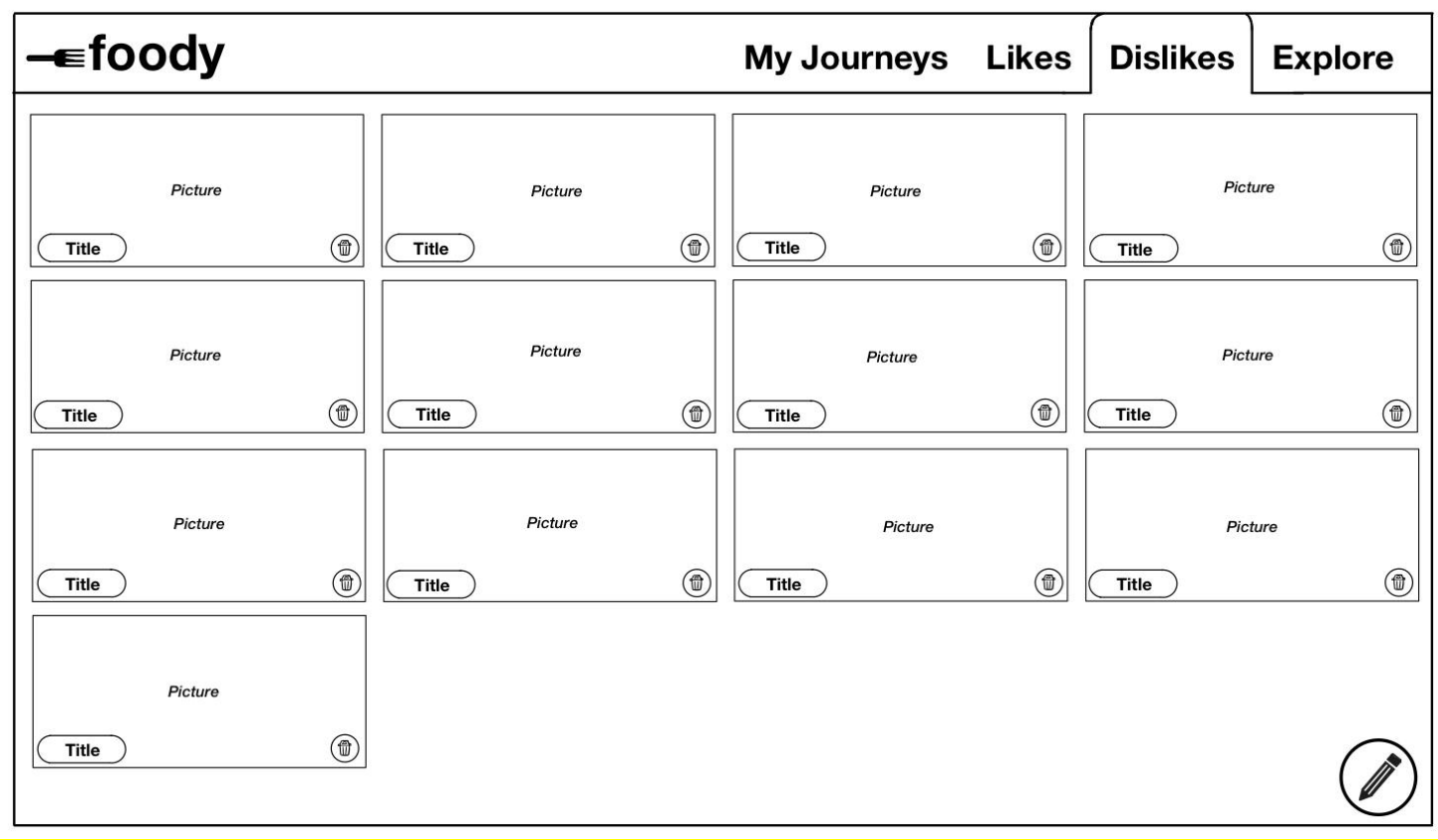
6. Appendices

6.1 The need of Journeys (Section 2.1.2.2.1 and Section 3.1.2.1, First Deliverable)

Journeys are something that Foody does not have at the moment but they would like to have as part of their Recommendation Agent. A user will create some Journeys based on his desire to categorize products. For example, a user can create a Journey based on the products that his mother likes, or the products that he prefers to eat for lunch, etc. Then, the user can select one of its Journeys, when he searches for food and get direct recommendations based on the content of the Journey that he has been selected. This method of recommendation is more direct, as the user had already defined a specific group of products, so based on that group we are going to recommend him Restaurants that offer this kind of products.

6.2 Changes in Dislikes tab (Section 2.1.2.2.2, First Deliverable)

At the second meeting with Foody, we asked the User Interface Designer of Foody to give us some feedback about our wireframes. He commented that we should remove the Add Dislikes Button, which provides a negative feeling to the user and it is not appropriate suggesting them to dislike more products. So, we created a new wireframe for the Dislikes Web Page and we removed the Add New Dislikes button.



6.3 The categories of products for the Journeys (Section 2.1.2.2.3, First Deliverable)

Those categories will be imported as a JSON file which is going to describe the basic content of the Web Page. We do not know the exact categories that Foody will want to use but this is an advantage for us, as we are going to implement an abstract parser which is going to transform the Web Page dynamically based on this JSON file. To be clear, this JSON file should be the only part that the content administrator of Foody should change and the content of our Web Page should transform accordingly.

6.4 Portability of the Food Recommendation Agent (Section 3.3.5, First Deliverable)

The Web Page is not going to be limited based on the browser because it is going to be using only Material.io components which does not limit us. So, in general, our system will be able to run, wherever the Foody's website already works as it uses the same components for its development.