Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores

Nutr.io - Multi-platform application for diabetics' nutritional choices

Beta release

Authors:

Pedro Pires	**Miguel Luís**	**David Albuquerque**
42206	43504	43566
A42206@alunos.isel.pt	A43504@alunos.isel.pt	A43566@alunos.isel.pt

Tutor:
Fernando Miguel Gamboa de Carvalho
mcarvalho@cc.isel.pt

May 4, 2020

Instituto Superior de Engenharia de Lisboa

Nutr.io - Multi-platform application for diabetics' nutritional choices

42206 - Pedro Miguel Sequeira Pires

Signature: _____

43504 - Miguel Filipe Paiva Luís

Signature: _____

43566 - David Alexandre Sousa Gomes Albuquerque

Signature: _____

Tutor: Fernando Miguel Gamboa de Carvalho

Signature: _____

Abstract

The idea that every field of study can be digitalized in order to ease monotonous tasks is continuously growing in the modern world. Our project aims to tackle the field of Type 1 diabetes, given its growing prevalence in the world.

One of those monotonous tasks is the count and measurement of carbohydrates in meals used to administer the correspondent amount of insulin, along with their blood levels, to maintain a healthy lifestyle. A task that heavily relies on having access to food databases and realize of how many portions a meal has - usually by using a digital balance or doing estimations.

Eating in restaurants is the perfect example that showcases a gap in this field, that our project, Nutr.io, aims to fill. Most nutritional applications do not provide data for restaurants' meals, such as MyFitnessPal, nor does the user bring his digital balance from home - resulting in a faulty carbohydrate count and therefore the administration of an incorrect insulin dose.

The main goal of this project is to design a system that offers a way to facilitate difficult carbohydrate measurement situations, like in restaurants. To that end, a system that stores meals' nutritional information will be developed, where users can use and calibrate its data with their feedback.

Contents

1 Introduction **1**

2 Project development **3**
2.1 Roadmap 3
2.2 Issues encountered and updates 3
2.2.1 Relational database 3
2.2.2 HTTP server 4
2.3 Roadmap updates 4

3 Results **5**
3.1 Relational database 5
3.2 HTTP server 6
3.3 Geolocation 6
3.4 Android application 7

4 Appendices **9**

Chapter 1

Introduction

This document is intended to give an update on the project's progress made to this date.

Its main goal is to report that the project is progressing according to the initial plan, previously stated in the project's proposal.

The report will also state the issues encountered during this time period, mentioning the decisions the group made to solve them. This might also include changes in the initial plan, that the group found relevant for the project's progress efficiency.

The diagrams and schemas developed for this project are shown when approaching the respective topic, however there is an appendix which contains additional information about the project, having references pointing to it when necessary.

Chapter 2

Project development

2.1 Roadmap

According to the proposed plan, the group managed to fulfill almost all the objectives that were planned to be accomplished until now - a relational database, a working HTTP server and a prototype Android aplication that can make requests to the server and use its information to display lists and detailed views.

However, as it will be detailed in the next section, the group faced some issues that caused slight drawbacks, such as the relational database conception, which had to be redesigned multiple times due to the project's requirements.

This report's appendix displays two project schedules: the initial one [Appendix A - Initial plan], also present in the project's proposal, and the actual one [Appendix B - Actual plan], which was updated to match the actual progress made until now.

2.2 Issues encountered and updates

This section describes the issues found and the decisions made to overcome them during development.

2.2.1 Relational database

The group had to redesign the database's models multiple times. One of the main issues that drove the redesign was the fact that the previous implementation only supported submissions from users, while it had to support submissions from both users and APIs.

Another issue is that our model offered no support to distinguish between API types and their submission, meaning that, for example, different restaurants from two different APIs (e.g.: Zomato and Yelp) with equal identifiers would generate data collisions.

It is worth mentioning that with every major addition, the database had to be renormalized, taking a considerable amount of time, which was not taken into account in the initial plan.

As such, this invested time ended up partially colliding with other schedules that were reserved for other modules.

2.2.2 HTTP server

The group initially struggled to implement the HttpServer as we were utilizing and learning a new framework - Spring MVC, meaning that implementing known patterns in server programing took more time than expected in the according to the initial project plan.

The group also recognizes that some functionalities are lacking as a result of the time invested in the database, namely endpoints that allow for resource creation.

2.3 Roadmap updates

Given that the previously mentioned issues are only minor setbacks, the group agrees that every mandatory requirement made in the initial draft can still be implemented, however, the group expresses concerns that optional requirements will have to be reevaluated after the Beta release, if needed.

Chapter 3

Results

This chapter shows what has been achieved and developed to this date.

As previously written in the project's roadmap, this project has already a working relational database and a HTTP server, developed with postgreSQL and with kotlin and spring mvc, respectively. The group also managed to achieve, as planned, its first working plaform client - a prototype android mobile application, developed with kotlin.

3.1 Relational database

As a result of multiples redesigns, here is the database's final conceptual model.

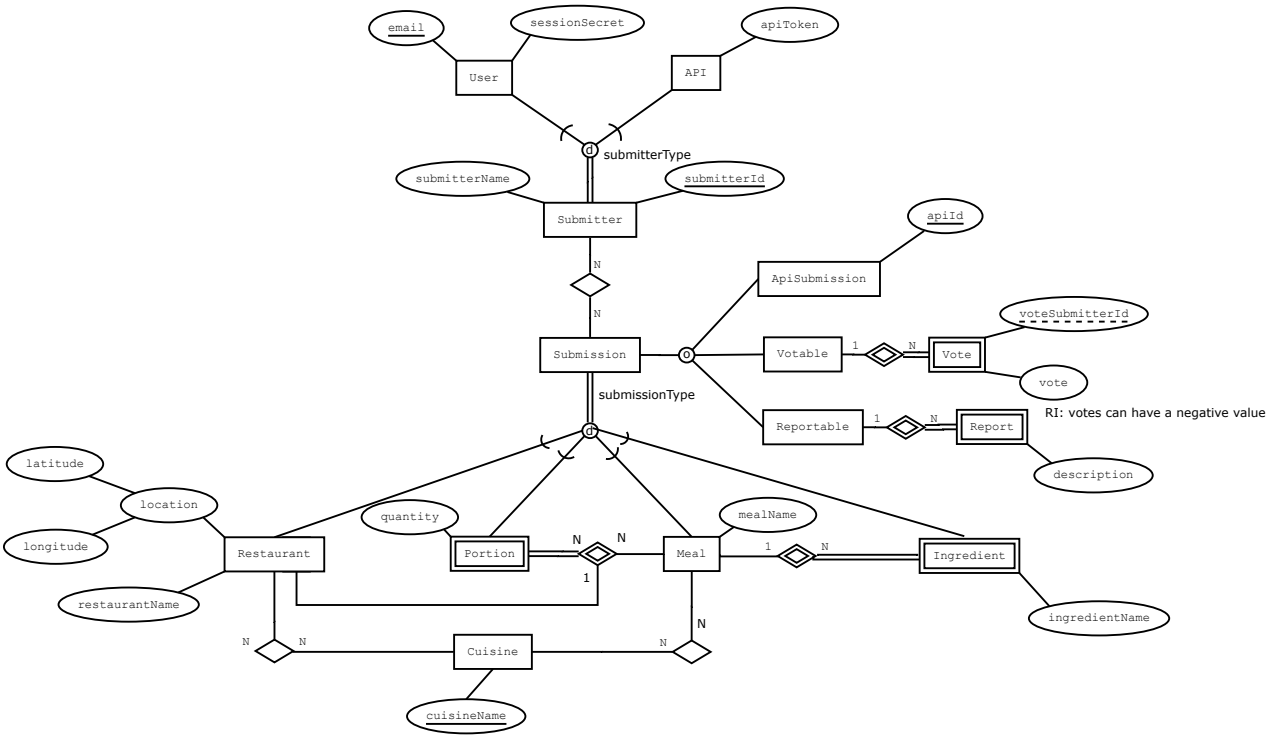


Figure 3.1: Database conceptual model

The database's relational model is present inside this report's appendix [Appendix C - Database relational model].

In the relational model there are tables which are not specified in the conceptual model. These are a product from associations between entities which will simplify queries' complexity.

The best feature of this redesigned model is that the database can filter user's submissions from API submissions. This is very useful because when a user tries to associate a meal that does not exist in the database to a restaurant that is already registered in the database, the system will need to search for that meal in an external API. When the meal is found, it will be inserted in the database as an API submission, because the data came from an external API.

However if another user inserts this previously inserted meal to another registered restaurant, it will insert as an user's submission, because the data already exists in the database and the user just built the association between the specific the meal and the specific restaurant.

This process, which resembles how memory pointers work, provides a high scalability and, cooperatively with the database normalization, lowers the memory usage.

3.2 HTTP server

The HTTP server is being developed in Kotlin and Spring MVC and during the starting phase of development the group had to define its endpoints [Appendix D - Endpoints' table] and content negotiation, namely, the content type of a valid and invalid response.

After discussing between application/vnd.siren+json and application/json for valid responses, the group decided on application/json due to its simplicity and ease of use. As for invalid responses, the group preferred application/problem+json over application/json, in order to avoid the need to define new error response formats.

3.3 Geolocation

Given how all clients rely on obtaining nearby restaurants, there was a need to implement a geolocation function in the project's design.

Initial research showcased two possible solutions: Haversine distances and cartesian distances, where the latter returns a highly imprecise distances. As such, Haversine was selected.

The next step was to choose which system filters nearby restaurants: database or HTTP server. After some discussion, the group decided that database was the best option for two reasons:

- Given the large amount of existing restaurants, sending such data from the database to the HTTP server so that it could filter it would occupy too much memory;
- PostgreSQL already supplies extensions that add support for location queries, namely PostGIS.

3.4 Android application

Being a prototype, the mobile application has only the core features to work and interact with the HTTP server.

The group chose to create an application that offers a minimalist layout to the user, providing accessible menus and a menu drawer, which implies the use of fragments instead of activities.

The following weeks will focus on implementing a map displaying fragment, which the group expects to be challenging as the group needs to learn how to use a complex framework.

The volley framework was implemented to allow the application to make asynchronous server requests.

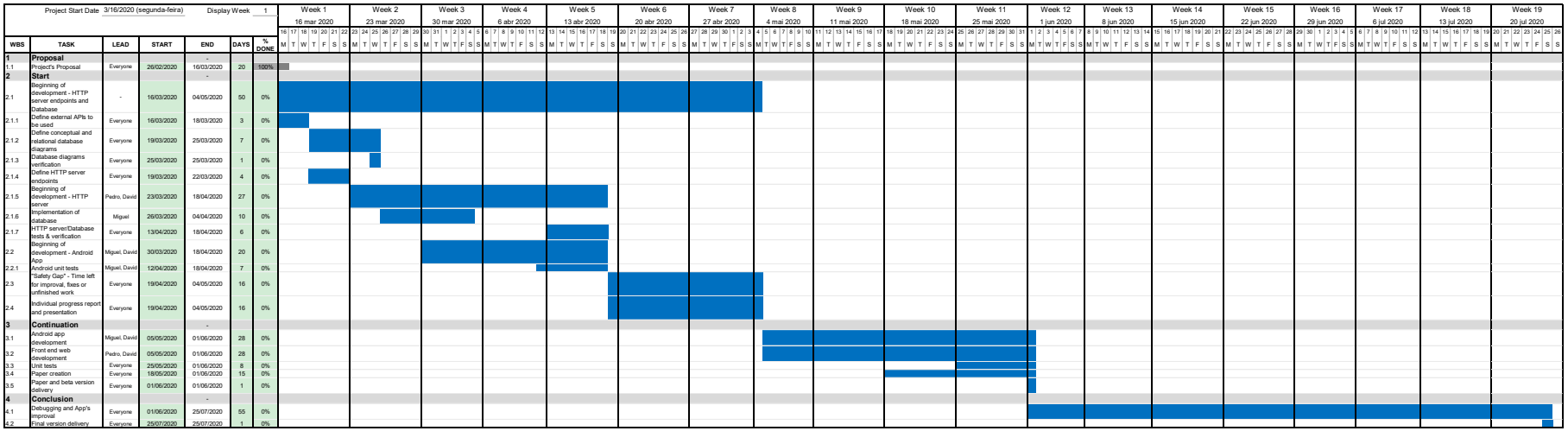
As all of the server endpoints send responses in JSON format, the application depends on the Jackson library to deserialize and serialize this format.

Chapter 4

Appendices

This chapter displays all the appendices referenced in this report.

Appendix A - Initial plan



WBS	TASK	LEAD	START	END	DAYS	% DONE
1	**Proposal**			-		
1.1	Project's Proposal	Everyone	26/02/2020	16/03/2020	20	100%
2	**Start**			-		
2.1	Beginning of development - HTTP server endpoints and Database	-	16/03/2020	04/05/2020	50	0%
2.1.1	Define external APIs to be used	Everyone	16/03/2020	18/03/2020	3	0%
2.1.2	Define conceptual and relational database diagrams	Everyone	19/03/2020	25/03/2020	7	0%
2.1.3	Database diagrams verification	Everyone	25/03/2020	25/03/2020	1	0%
2.1.4	Define HTTP server endpoints	Everyone	19/03/2020	22/03/2020	4	0%
2.1.5	Beginning of development - HTTP server	Pedro, David	23/03/2020	18/04/2020	27	0%
2.1.6	Implementation of database	Miguel	26/03/2020	04/04/2020	10	0%
2.1.7	HTTP server/Database tests & verification	Everyone	13/04/2020	18/04/2020	6	0%
2.2	Beginning of development - Android App	Miguel, David	30/03/2020	18/04/2020	20	0%
2.2.1	Android unit tests	Miguel, David	12/04/2020	18/04/2020	7	0%
2.3	"Safety Gap" - Time left for improval, fixes or unfinished work	Everyone	19/04/2020	04/05/2020	16	0%
2.4	Individual progress report and presentation	Everyone	19/04/2020	04/05/2020	16	0%
3	**Continuation**			-		
3.1	Android app development	Miguel, David	05/05/2020	01/06/2020	28	0%
3.2	Front end web development	Pedro, David	05/05/2020	01/06/2020	28	0%
3.3	Unit tests	Everyone	25/05/2020	01/06/2020	8	0%
3.4	Paper creation	Everyone	18/05/2020	01/06/2020	15	0%
3.5	Paper and beta version delivery	Everyone	01/06/2020	01/06/2020	1	0%
4	**Conclusion**			-		
4.1	Debugging and App's improval	Everyone	01/06/2020	25/07/2020	55	0%
4.2	Final version delivery	Everyone	25/07/2020	25/07/2020	1	0%

Figure 1: Initial plan accorded in the project's proposal

Appendix B - Actual plan

Nutr.io Project Schedule

Instituto Superior de Engenharia de Lisboa

Project Start Date: 3/16/2020 (segunda-feira) Display Week: 1

WBS	TASK	LEAD	START	END	DAYS	% DONE
1	**Proposal**			-		
1.1	Project's Proposal	Everyone	26/02/2020	16/03/2020	20	100%
2	**Start**			-		
2.1	Beginning of development - HTTP server endpoints, Database and Android App prototype	-	16/03/2020	04/05/2020	50	100%
2.1.1	Define external APIs to be used	Everyone	16/03/2020	18/03/2020	3	100%
2.1.2	Define conceptual and relational database diagrams	Everyone	19/03/2020	23/04/2020	36	100%
2.1.3	Database diagrams verification	Everyone	25/03/2020	23/04/2020	30	100%
2.1.4	Define HTTP server endpoints	Everyone	19/03/2020	22/03/2020	4	100%
2.1.5	Beginning of development - HTTP server	Pedro, David	23/03/2020	04/05/2020	43	100%
2.1.6	Implementation of database	Miguel	26/03/2020	26/04/2020	32	100%
2.1.7	HTTP server/Database tests & verification	Everyone	13/04/2020	04/05/2020	22	100%
2.2	Beginning of development - Android App	Miguel, David	30/03/2020	04/05/2020	36	100%
2.2.1	Android unit tests	Miguel, David	12/04/2020	18/04/2020	7	100%
2.3	"Safety Gap" - Time left for improval, fixes or unfinished work	Everyone	19/04/2020	04/05/2020	16	100%
2.4	Individual progress report and presentation	Everyone	19/04/2020	04/05/2020	16	100%
3	**Continuation**			-		
3.1	Android app development	Miguel, David	05/05/2020	01/06/2020	28	0%
3.2	Front end web development	Pedro, David	05/05/2020	01/06/2020	28	0%
3.3	Unit tests	Everyone	25/05/2020	01/06/2020	8	0%
3.4	Paper creation	Everyone	18/05/2020	01/06/2020	15	0%
3.5	Paper and beta version delivery	Everyone	01/06/2020	01/06/2020	1	0%
4	**Conclusion**			-		
4.1	Debugging and App's improval	Everyone	01/06/2020	25/07/2020	55	0%
4.2	Final version delivery	Everyone	25/07/2020	25/07/2020	1	0%

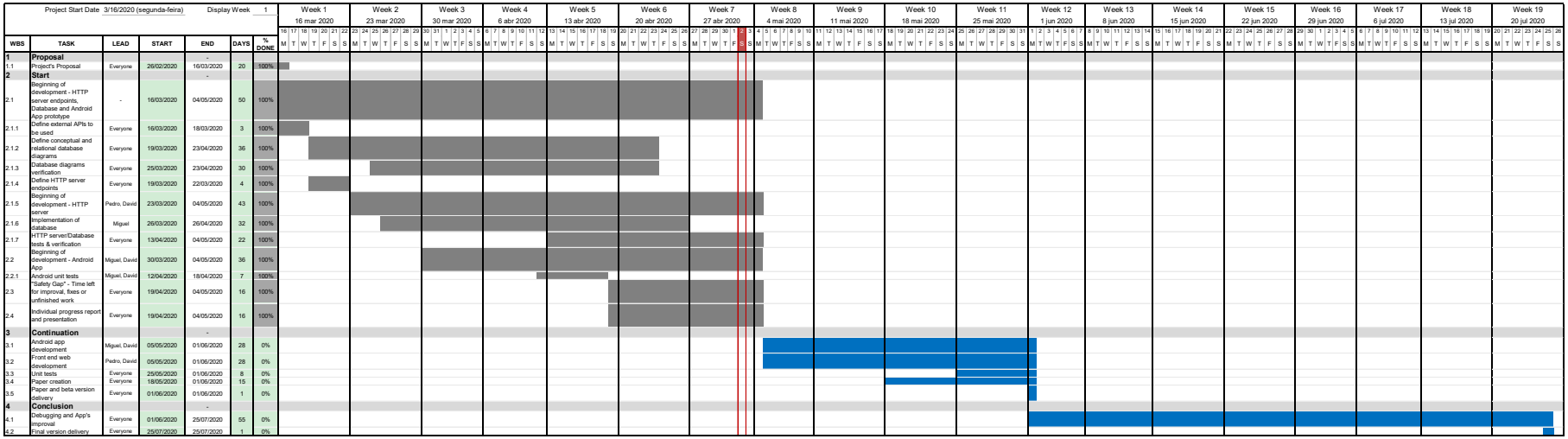


Figure 2: Actual plan with the progress updated as of today

Appendix C - Database relational model

- **Submitter**
  - Attributes: submitterId, submitterName, submitterType
  - Primary Key(s): submitterId
  - Foreign Key(s): -
  - Not null: submitterName, submitterType
- **User**
  - Attributes: *submitterId*, email, sessionSecret
  - Primary Key(s): *submitterId*, email
  - Foreign Key(s): *submitterId* references Submitter(submitterId)
  - Not null: sessionSecret
- **API**
  - Attributes: *submitterId*, apiToken
  - Primary Key(s): *submitterId*
  - Foreign Key(s): *submitterId* references Submitter(submitterId)
  - Not null: apiToken
- **Submission**
  - Attributes: submissionId, submissionType
  - Primary Key(s): submissionId
  - Not null: submissionType
- **ApiSubmission**
  - Attributes: *submissionId*, apiId, submissionType
  - Primary Key(s): *submissionId*, apiId
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: submissionType
- **SubmissionSubmitter**
  - Attributes: *submissionId*, *submitterId*, submissionDate
  - Primary Key(s): *submissionId*, *submitterId*
  - Foreign Key(s):
    - *submissionId* references Submission(submissionId)
    - *submitterId* references Submitter(submitterId)
  - Not null: *submitterId*
- **SubmissionContract**
  - Attributes: submissionId, submissionContract
  - Primary Key(s): submissionId, submissionContract

- **Report**
  - Attributes: *submitterId*, *submissionId*, description
  - Primary Key(s): *submitterId*, *submissionId*
  - Foreign Key(s):
    - *submissionId* references Submission(submissionId)
    - *submitterId* references Submitter(submitterId)
  - Not null: description
- **Vote**
  - Attributes: *submissionId*, *voteSubmitterId*, vote
  - Primary Key(s): *voteSubmitterId*, *submissionId*
  - Foreign Key(s):
    - *submissionId* references Submission(submissionId)
    - *voteSubmitterId* references Submitter(submitterId)
  - Not null: vote
- **Restaurant**
  - Attributes: *submissionId*, restaurantName, latitude, longitude
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: restaurantName
- **Cuisine**
  - Attributes: cuisineName
  - Primary Key(s): cuisineName
- **Meal**
  - Attributes: *submissionId*, mealName
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: mealName
- **Portion**
  - Attributes: *submissionId*, quantity
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: quantity

- **Ingredient**
  - Attributes: *submissionId*, ingredientName
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: ingredientName
- **MealIngredient**
  - Attributes: *mealSubmissionId*, *ingredientSubmissionId*
  - Primary Key(s): *mealSubmissionId*, *ingredientSubmissionId*
  - Foreign Key(s):
    - *mealSubmissionId* references Meal(submissionId)
    - *ingredientSubmissionId* references Ingredient(submissionId)
- **RestaurantMealPortion**
  - Attributes: *mealSubmissionId*, *portionSubmissionId*, *restaurantSubmissionId*
  - Primary Key(s): *mealSubmissionId*, *portionSubmissionId*
  - Foreign Key(s):
    - *mealSubmissionId* references Meal(submissionId)
    - *portionSubmissionId* references Portion(submissionId)
    - *restaurantSubmissionId* references Restaurant(submissionId)
- **RestaurantCuisine**
  - Attributes: *restaurantSubmissionId*, *cuisineName*
  - Primary Key(s): *restaurantSubmissionId*, *cuisineName*
  - Foreign Key(s):
    - *restaurantSubmissionId* references Restaurant(submissionId)
    - *cuisineName* references Cuisine(cuisineName)
- **MealCuisine**
  - Attributes: *mealSubmissionId*, *cuisineName*
  - Primary Key(s): *mealSubmissionId*, *cuisineName*
  - Foreign Key(s):
    - *mealSubmissionId* references Meal(submissionId)
    - *cuisineName* references Cuisine(cuisineName)

Appendix D - Endpoints' table

Method	Path	Query String	Body parameters	Description
GET	restaurant	int lat, int lon, int skip, int count, String name		Search for restaurants and their cuisines, based on location or named search
GET	restaurant\:id			Restaurant's name and cuisines
GET	restaurant\:id\meal	int skip, int count		Restaurant's meals with votes
GET	cuisines	int skip, int count		List possible cuisines
GET	ingredients	int skip, int count		Get all possible meal ingredients
GET	meal\:id			Meal's portions and ingredients
GET	report\:id			GET user's report on a submission
GET	vote\:id			GET user's vote on a submission
POST	restaurant		String name, float lat, float lon, String[] cuisines	Create a new restaurant from location
POST	meal		String name, MealIngredient[], amount, carbs, image, unit	Create a new restaurant meal, allowing for multiple ingredients (optional), first portion submission (optional)
POST	restaurant\:restaurantMeal\meal\:mealId		Object portion	Associates existing meal with existing or api restaurant
POST	restaurant\:id\meal\:mealId\portion		Object portion	Creates a restaurant's meal portion
POST	report\:id		String reason	Create a report on a restaurant's meal
POST	vote\:id		boolean isPrimitive	Create a vote on a restaurant's meal
PUT	restaurant\:id		Object portion	Updates a restaurant
PUT	meal\:mealId		String name, Object portion, int[] ingredients, String[] cuisines	Update a restaurant's meal
PUT	restaurant\:id\meal\:mealId\portion		Object portion	Creates a restaurant's meal portion
PUT	vote\:id		boolean isPositive	Update a vote on a restaurant's meal
DELETE	meal\:id			Delete user created meal
DELETE	restaurant\:id			Delete user created restaurant, if it wasn't used by the community yet
DELETE	restaurant\:id\meal\:mealId			Delete user created restaurant meal, if it wasn't used by the community yet
DELETE	restaurant\:id\meal\:mealId\portion			Delete user's restaurant's meal portion
DELETE	vote\:id			Delete user's restaurant meal vote