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

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

Final release

Authors:

Pedro Pires
42206
A42206@alunos.isel.pt

Miguel Luís
43504
A43504@alunos.isel.pt

David Albuquerque
43566
A43566@alunos.isel.pt

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

September, 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.

This system will offer an Android application and a front end web application where users can search for restaurants and their respective meals and ingredients. By signing up, the user will be allowed to build its insulin profiles which, alongside with nutritional information provided by meals and ingredients, can calculate and provide an accurate insulin dosage that is individual for each user. It will also allow submissions from the user, that will be detailed in this report.

As security and privacy are revelant subjects nowadays, all user sensitive information will be encrypted when stored in the database, so only the user can have access to its information and to make its data unusable and intransmissible in case of security breaches.

Contents

Chapter 1

Introduction

1.1 Context

1.2 Objectives

1.3 Report structure

This document is related to the project's beta release, mentioning all progress made up to this date.

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

Requirements and project’s structure

2.1 Requirements analysis

2.1.1 Database

2.1.2 HTTP Server

2.1.3 Android application

2.1.4 Front web application

2.2 Project’s structure

Chapter 3

Project development

3.1 Roadmap

According to the proposed plan, the group managed to accomplish a more complete Android aplication that can make requests to the server and use its information to display lists, detailed views and calculate user's insulin values as store local cache such as user profiles and search history.

The HTTP server also had great progress, having now most of its planned features working and properly tested. In order to make this possible, the database suffered some changes that will be detailed later in this report.

However, as it will be detailed in the next section, the group faced some issues that caused delays and made the web browser application development not possible in this delivery, mainly due to API withdrawals and data models changes.

3.2 Issues encountered and updates

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

3.2.1 Relational database

The group had to redesign the database's models multiple times again, due to the previously mentioned API withdrawals and other encountered incoherences.

3.2.2 Food API's

Throughout development and with some research, the group concluded that no meal API exists that provides accurate nutritional - either for a recipe/meal or a base ingredient, such as rice.

This conclusion came from investigating three possible API's: Edamam, Nutrionix and Spoonacular; and comparing their provided nutritional values with corresponding values obtained from certified sources for Portugal. The results can be found in [Appendix D - API nutritional accuracy sheet].

The group assumes that this inaccuracy is due to the fact that mentioned API's automatically calculate a recipe's carbohydrates from it's ingredients without taking into consideration the cooking process, meaning, 100g of raw rice does not have the same carbohydrates has 100g of cooked rice.

Another assumption is that there is no international standard for nutritional values, meaning that the same meal (and it's ingredients nutritional composition) can have different carbohydrates between Portugal and the United States of America.

As a result, the group no longer relies on food API's and as such, inserts curated meals and ingredients in the project's database. This comes with the limitation that certain ingredients might have been missed, meaning that a user might not be able to create their desired meals.

3.2.3 Android client

The Android application's development progressed normally but it had to be put on hold sometimes, because of HTTP server's endpoints' completion, in which the application depends strongly. A major dto and model restructure had also to be made inside the mobile application in order to meet with the current HTTP responses.

3.3 Roadmap updates

Given that the previously mentioned issues, the group agrees that every mandatory requirement made in the initial draft and retified in the previous progress report can still be implemented until the project's final release, such as the web client apllication.

However the group will have to remove some optional features so the main ones can be delivered whole.

Chapter 4

Results

This chapter shows the final results of this project.

All options and decisions made by the group will be displayed in the sections below, as every relevant detail of each developed module.

4.1 Relational database

4.1.1 Used tecnologies

When the project was in a planning phase, the group decided that a relational database was the most suitable option for this project instead of a No-SQL database, because of the project's structure - there are many hierarchies between entities, which invalidated the no-SQL option.

After some discussion about the tecnology to be used, the group chose to use PostgreSQL rather than Microsoft SQL, because it has the PostGIS plugin which is convenient for geolocation proposes and it is supported by an Heroku plugin, where it is planned to deploy the front end web application alongside with the database model.

4.1.2 Conceptual model

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

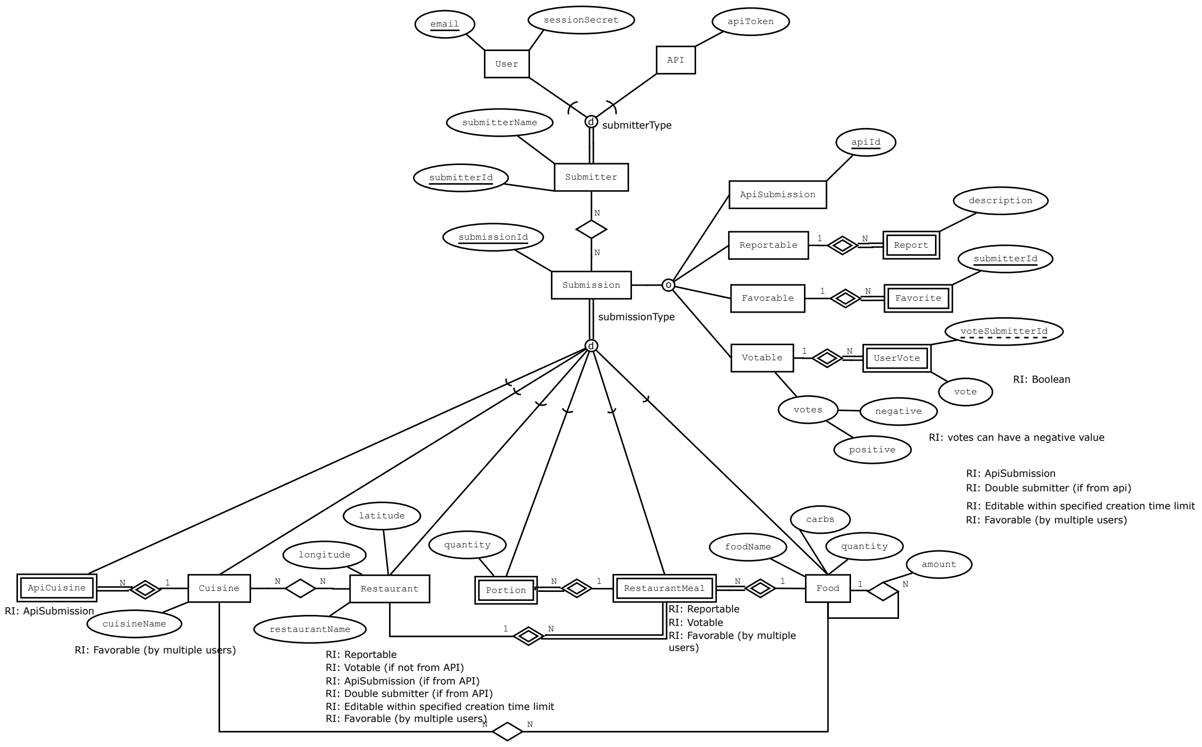


Figure 4.1: Database conceptual model

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

Now the submission can fall into 4 categories: ApiSubmission, Reportable, Favorable and Votable, in order to disguish between submissions that are from the user or from APIs and to separate which ones can be reportable, favorable and votable by the user.

The cuisine entity has now an associated entity called ApiCuisine, to save cuisine information provided by the Here API.

Meals and ingredients were now condensed into one entity called food - now each meal can have meals inside it that can also be considered ingredients in other contexts.

Therefore each meal possesses nutritional information, which is essential to the user especially to the insulin calculations. That information is composed by 'carbs' - meal's carbohydrates; and quantity - meal's quantity.

4.2 HTTP server

4.2.1 Used tecnologies

Kotlin

The group chose to use Kotlin for the HTTP server developed as it is a language that is being more adopted and used nowadays and because it is totally interoperable with Java.

It was also the language used during PDM, which is an optional course for Android application development inside the LEIC programme, making this a language the group felt confortable with.

Spring MVC

At the beginning of the project the group decided to use Spring MVC rather than Ktor, as the first one is taught in DAW, which is an optional course for Web applications development inside the LEIC programme. As Spring MVC has a better coverage inside the LEIC programme, the group considered it a more solid choice.

Used dependencies

TODO - eng. Félix says: "external dependencies used."

Here are all the dependencies injected inside HTTP server gradle settings file.

- **Kotlin base dependencies** - kotlin-reflect and kotlin-stdlib-jdk8;
- **Spring base dependencies** - spring-boot-starter and starter-web;
- **Mockito** - for tests with mocks;
- **Jackson** - for JSON serialization and deserialization;
- **JDBI** - the driver/interface for connecting with the relational database;
- **Spring Security** - for authentication and authorization proposes.

4.2.2 Code structure

TODO - eng. Félix says: "Needs more information about the backend organization, such as: intermediaries, controllers, services, database access method, external dependencies used"

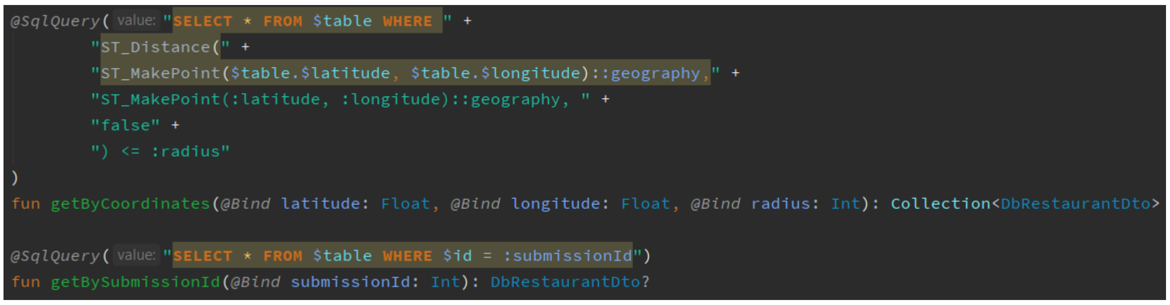
4.2.3 JDBI

After some discussion of which driver should be used to allow communication between the server and database, the group decided that the JDBI was best option as it is a library built on JDBC.

The library also exposes two different API's styles: a fluent style and sql object style (used during development), as shown below.

```
// using in-memory H2 database
DataSource ds = JdbcConnectionPool.create("jdbc:h2:mem:test",
                                          "username",
                                          "password");
DBI dbi = new DBI(ds);
Handle h = dbi.open();
h.execute("create table something (id int primary key, name varchar(100))");

h.execute("insert into something (id, name) values (?, ?)", 1, "Brian");

String name = h.createQuery("select name from something where id = :id")
                    .bind("id", 1)
                    .map(StringColumnMapper.INSTANCE)
                    .first();

assertThat(name, equalTo("Brian"));

h.close();
```

Figure 4.2: JDBI using a fluent API style



```
@SqlQuery( value: "SELECT * FROM $table WHERE " +
        "ST_Distance(" +
        "ST_MakePoint($table.$latitude, $table.$longitude)::geography," +
        "ST_MakePoint(:latitude, :longitude)::geography, " +
        "false" +
        ") <= :radius"
)
fun getByCoordinates(@Bind latitude: Float, @Bind longitude: Float, @Bind radius: Int): Collection<DbRestaurantDto>

@SqlQuery( value: "SELECT * FROM $table WHERE $id = :submissionId")
fun getBySubmissionId(@Bind submissionId: Int): DbRestaurantDto?
```

Figure 4.3: JDBI using a SQL object API style (example from HTTP server)

TODO - eng. Félix says: "Usage of JDBI and the declarative API needs justification and perhaps a brief introduction."

4.3 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.

4.4 Android application

4.4.1 Used tecnologies

Kotlin

The group chose to use Kotlin for the mobile application development, as it is now the official programming language for Android development, according to Google.

External dependencies

Here are the dependencies that were included in the mobile application which gave more functionalities to it.

- **Volley** - an HTTP library for Android networking;
- **Jackson** - JSON serialization, deserialization and handling;
- **Room** - A framework to store data locally;
- **MapBox** - A framework to provide maps and geolocation tools;
- **MPAndroidChart** - provides custom graphs inside the application;
- **Glide** - a framework for image loading.

4.4.2 Code structure

Drawing pattern

The mobile application code structure follows the **repository pattern**, which is a code architecture recommended by the **Android Jetpack** for this type of applications.

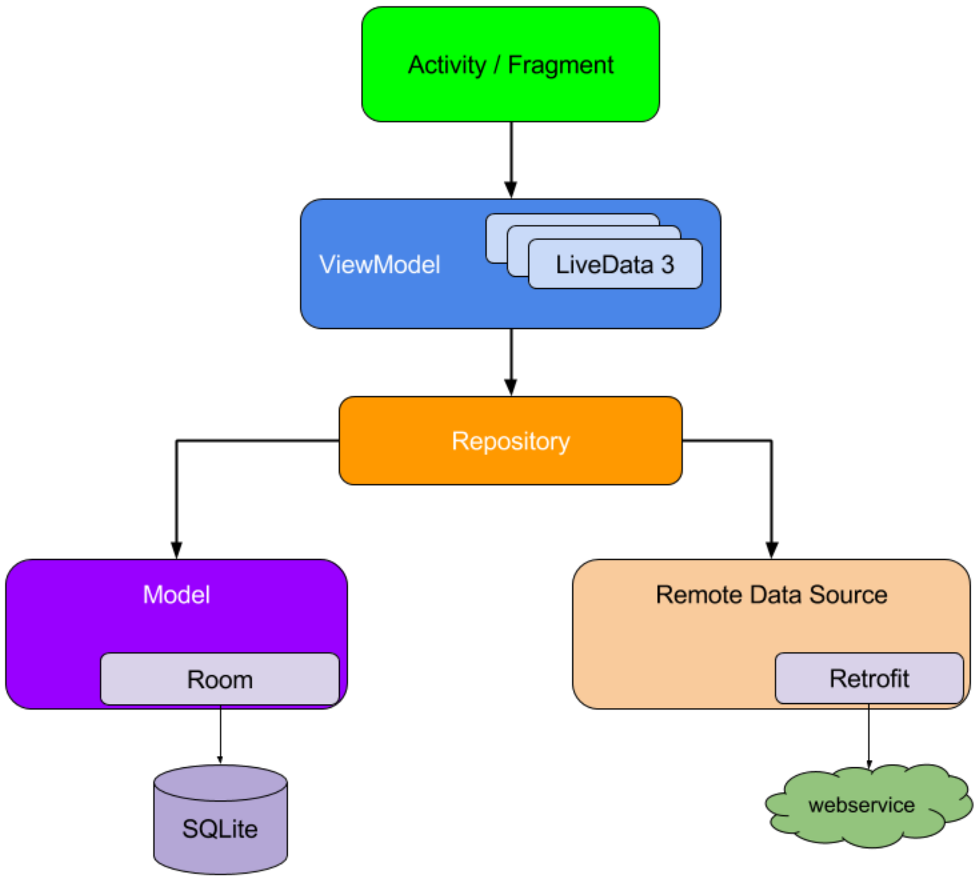


Figure 4.4: The repository pattern diagram

Above there is the pattern's diagram provided by the Android Jetpack.

The idea behind this architecture is that each Activity or fragment has its own ViewModel and each one calls the needed functions present inside the repository. The repository corresponds to a layer that manages where the information should be retrieved from.

The 'DTO to model' mapping also occurs inside the repository, following the rule where ViewModels should only manipulate models and the layers below should only use DTOs.

The application follows this diagram with the only difference that Volley is used instead of Retrofit.

By following this pattern, the code becomes segmented and organized, allowing a good comprehension and code maintainability.

Local data storing

As mentioned in the dependencies, the mobile application utilizes Room to store data locally. This is convenient for multiple reasons:

- To allow using the application in offline situations;
- To save data in order to avoid unnecessary requests to the server;
- To help data synchronization, that will be detailed later in this section.

User authentication and authorization

The user has the ability to sign up or sign in the mobile application. Besides being the server responsable for this matter, the mobile application has also some intervention here, because after a successful login or register, the HTTP server will return a jwt (JSON Web Token) that will authenticate and authorize the user in future requests.

This token will be stored in the Android SharedPreferences, which is the safest way to store a token inside Android applications. The token will also to be renewed from time to time due to expiration time.

Data synchronization

Background data synchronization will happen after a successful login or register. The only user data that will be synchronized are:

- Insulin profiles;
- Custom meals made by the user;
- Favorites.

When logged in, the data can be synchronized in two ways: the user forces the synchronization by triggering a button inside the specific list; The Android WorkManager will make sure that the data is synchronized at least once a day when the phone is inactive and connected to the internet.

Android version compatibility

In order to garantee a global support by most of the Android devices nowadays, the mobile application is supported since **Android 7** (API level 24) up to **Android 10** (API level 29). It is recommended to use the last one as the Android emulator used this version.

TODO

As mentioned in the progress report, the group managed to implement in the mobile application a fragment that displays a map with a list of restaurants nearby the user.

The user can also search for restaurants, meals and cuisines providing the associated name or identifier.

The core feature of the application was also finished during this time period - the insulin calculator. This feature calculates how many insulin doses should be injected in order to maintain the blood glucose levels stabilized according to the user's planned glucose objective for that period of the day.

The blood glucose objective is set inside the user's profile, by creating multiples insulin profiles, each one has a limited time period to give the user freedom to map its own insulin routine throughout the day.

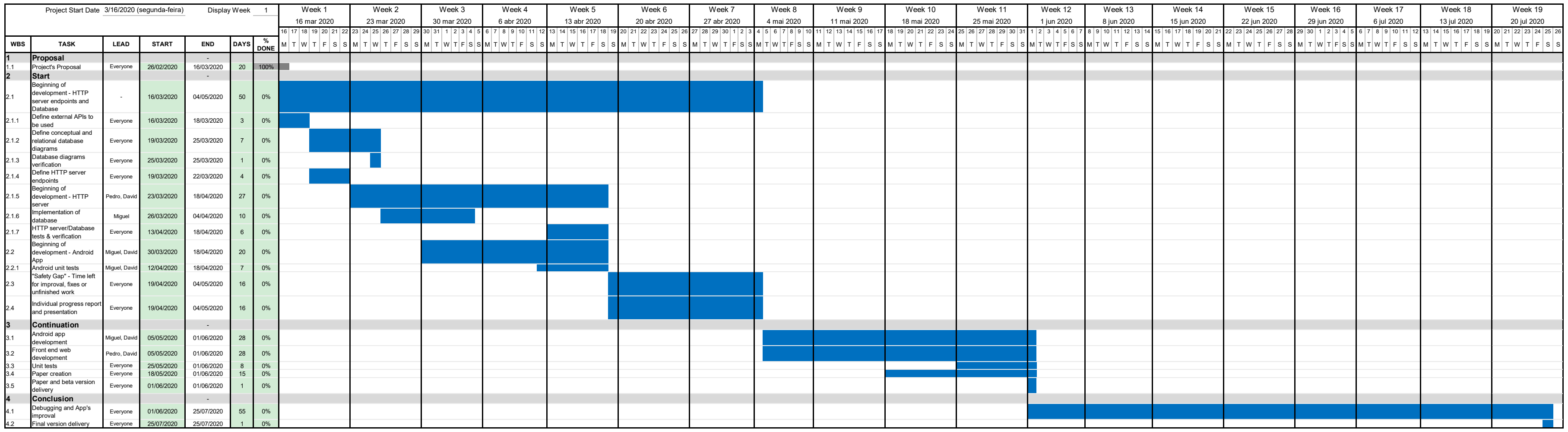
This is due to the fact that user's insulin sensitivity factor varies along the day, so the user has the ability to specify its own values in order to the calculator

Chapter 5

Appendices

This chapter displays all the appendices referenced in this report.

Appendix A - Initial plan



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 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 - Database relational model

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

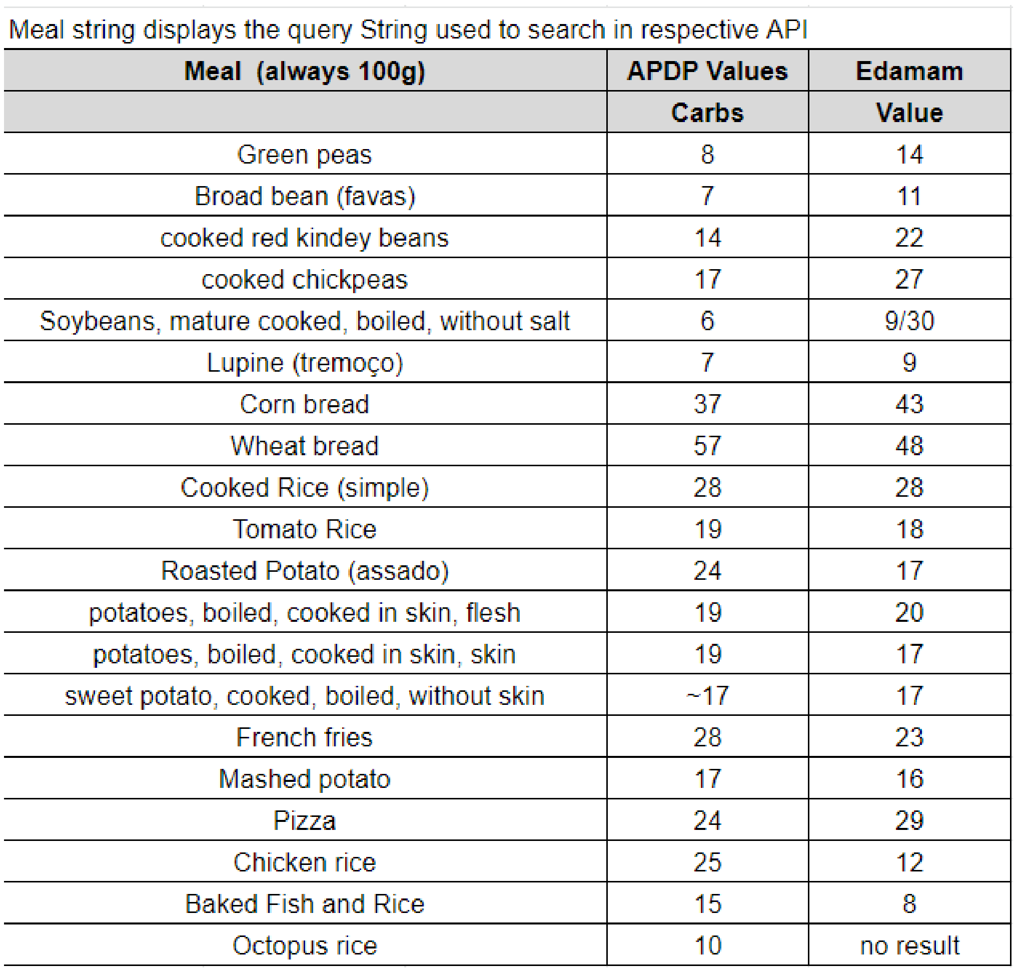
- **Report**
  - Attributes: *submissionId*, *submitterId*, description
  - Primary Key(s): *submissionId*, *submitterId*
  - Foreign Key(s):
  -
    - *submissionId* references Submission(submissionId)
    - submitterId references Submitter(submitterId)
  - Not null: description
- **Votes**
  - Attributes: *submissionId*, positiveCount, negativeCount
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: submissionType
- **UserVote**
  - Attributes: *submissionId*, voteSubmitterId, vote
  - Primary Key(s): *submissionId*, voteSubmitterId
  - Foreign Key(s):
    - *submissionId* references Submission(submissionId)
    - voteSubmitterId references Submitter(submitterId)
- **Restaurant**
  - Attributes: *submissionId*, restaurantName, latitude, longitude
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: restaurantName
- **Cuisine**
  - Attributes: submissionId, cuisineName
  - Primary Key(s): cuisineName
- **ApiCuisine**
  - Attributes: *submissionId*, *cuisineSubmissionId*
  - Primary Key(s): cuisineName
  - Foreign Key(s):
    - *submissionId* references Submission(submissionId)
    - *cuisineSubmissionId* references Cuisine(submissionId)
- **Meal**
  - Attributes: *submissionId*, mealName, carbs, quantity, unit
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: mealName

- **RestaurantMeal**
  - Attributes: submissionId, *restaurantSubmissionId*, *mealSubmissionId*
  - Primary Key(s): submissionId, *restaurantSubmissionId*, *mealSubmissionId*
  - Foreign Key(s):
    - ∗ *restaurantSubmissionId* references Restaurant(submissionId)
    - ∗ *mealSubmissionId* references Meal(submissionId)
- **Favorite**
  - Attributes: *submissionId*, submitterId
  - Primary Key(s): *submissionId*, submitterId
  - Foreign Key(s): *submissionId* references Submission(submissionId)
- **Portion**
  - Attributes: *submissionId*, restaurantMealSubmissionId, quantity
  - Primary Key(s): *submissionId*
  - Foreign Key(s): *submissionId* references Submission(submissionId)
  - Not null: quantity
- **MealIngredient**
  - Attributes: *restaurantSubmissionId*, *ingredientSubmissionId*, quantity
  - Primary Key(s): *restaurantSubmissionId*, *ingredientSubmissionId*
  - Foreign Key(s):
    - ∗ *mealSubmissionId* references Meal(submissionId)
    - ∗ *ingredientSubmissionId* references Ingredient(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 C - Endpoints' table

Method	Path	Query String	Body parameters	Description
GET	\restaurant	float latitude, float longitude, optional String name, optional int radius, optional String name		Search for restaurants and their cuisines, based on location or named search
GET	\restaurant\:restaurantId			Obtain specific restaurant's full information by given restaurantId
GET	\restaurant\:restaurantId\meal			Obtain all suggest and user inserted restaurant meals for given restaurant
GET	\restaurant\:restaurantId\meal\:mealId	int skip, int count		Obtain specific restaurant meal for given restaurantId and mealId
GET	\cuisines	optional int skip, optional int limit		List possible cuisines
GET	\ingredients	optional int skip, optional int limit		Get all possible ingredients
GET	\meal	optional string[] mealTypes, optional int skip, optional int count, optional string[] cuisines		Get all suggested meals
GET	\meal\:mealId			Obtain specific meal's full information by given mealId
POST	\restaurant		RestaurantInput	Create a new restaurant around given geolocation
POST	\meal		MealInput	Create a user meal with at least one ingredient
POST	\restaurant\:restaurantId\meal\:mealId		PortionInput	Insert a new portion for given restaurant meal
PUT	\restaurant\:restaurantId\vote		VoteInput	Add or update your vote on a user restaurant
PUT	\restaurant\:restaurantId\:mealId		VoteInput	Add or update your vote on a restaurant meal created by an user
PUT	\restaurant\:restaurantId\meal		RestaurantMealInput	Creates a restaurant meal from given user meal
DELETE	\restaurant\:restaurantId			Delete user created restaurant
DELETE	\restaurant\:restaurantId\vote			Delete user's vote on an user's restaurant
DELETE	\restaurant\:restaurantId\meal\:mealId			Delete user's portion submission for given restaurant meal
DELETE	\restaurant\:restaurantId\meal\:mealId\portion			Delete user's restaurant's meal portion
DELETE	\restaurant\:restaurantId\meal\:mealId\vote			Delete user's restaurant's meal vote
DELETE	\meal\:mealId			Delete an user created meal, along with any associations with a restaurant the meal might have

Appendix D - API nutritional accuracy sheet



Meal string displays the query String used to search in respective API

Meal (always 100g)	APDP Values	Edamam
	Carbs	Value
Green peas	8	14
Broad bean (favas)	7	11
cooked red kindey beans	14	22
cooked chickpeas	17	27
Soybeans, mature cooked, boiled, without salt	6	9/30
Lupine (tremoço)	7	9
Corn bread	37	43
Wheat bread	57	48
Cooked Rice (simple)	28	28
Tomato Rice	19	18
Roasted Potato (assado)	24	17
potatoes, boiled, cooked in skin, flesh	19	20
potatoes, boiled, cooked in skin, skin	19	17
sweet potato, cooked, boiled, without skin	~17	17
French fries	28	23
Mashed potato	17	16
Pizza	24	29
Chicken rice	25	12
Baked Fish and Rice	15	8
Octopus rice	10	no result

Figure 2: API nutritional accuracy sheet