# Data and Process Modeling

David Massimo

Course Project

Academic Year 2015/2016

## Table of Contents

## List of Figures

# 1 Collection of data

## 1.1 Textual description of the domain

Given the high complexity an individual has to deal with whenever he/she has to decide whether to eat a dish with a nutrition composition or another food with a certain other nutrition while needs to fulfill his needs according his lifestyle, a probable diet guideline and/or health indication for an allergy or food related diseases, with this work we aim to investigate the food domain in order to capture and model all the information that can improve individuals' life in the context of nutrition.

In this work we will, at first, identify authoritative resources in the area of food and nutrition, secondly, capture and explicitly bind their structure and relation, then, formalize and validate them according domain expert knowledge in order to build a suitable data model. Finally a process related a showcase application will be analyzed and modelled as a business process.

## 1.2 Description of the information sources

### 1.2.1 Recipes

The information about recipes our system will access are those that are available in two of the most important and largest online cooking communities: AllRecipes[1] and Food52[2]. The information source we refer to, and that we assume as the most authoritative, provides valuable information about recipes, their classification, their origin, as well as description in terms of cooking procedure and ground ingredients. (Moreover, we can access to relevant information related the impact of a recipe to the crowd.)

### 1.2.2 Nutrition Data

Nutrition is strictly related not only to individuals' culture and lifestyle, but also to how food is composed, either at a macro scale – a recipe – or at a fine grained level – ground ingredients. Eating an Italian pasta with eggplants or a more exotic Indian Biryani will provide an individual a set of nutrition values that can be more or less the same, even though the recipes are completely different in their cooking style, composition and origin

[1] http://allrecipes.com/
[2] http://food52.com/

of the ground ingredients. According the studies carried by the United States Department of Agriculture (USDA)[3] we aim to model their results in the bio-chemical analysis of food items in order to improve our data model about recipes and individuals requirements and energy expenditure. In the specific context of a nutrition system augmenting the ground knowledge with such data will be a good option to accomplish reasoning in terms of the constraint satisfaction problems that needs to be solved to consider health restriction given by an individual.

### 1.2.3 Physical Activity

As unique and authoritative information source about individuals' physical activity we refer to FitBit[4], a company that is leading the sport market segment with their technological product in the context of life logging. With their devices they provide access to a broad number of information about activity a human being can do. We will model them and find relations with the previously introduced resources according a literature that will be discussed later in this report.

# 2 Collection of Relevant Data

In this section we will extract information about specific domains according expert knowledge. The information elicitation process is based on the analysis of charts, tables and descriptions extracted from the information sources introduced in section (XX).

### 2.1.1 Recipe

Here follow a presentation of the recipe data that can be found on the two main food information sources we are referring to. We will show at first what can be extracted from foo52.com, outlining the relations that can be inferred from the various components of the community website. At a second instance, we will digg on the kind of data that comes from the allrecipes.com cookers community. Finally we will collapse common information in order to get a unique and uniform data model for recipes coming from the various sources.

In Figure 1 is showed the main user interface for recipe browsing inside the food52.com community. Generally a list of mixed popular and latest recipes is shown to new users,

[3] http://www.usda.gov/wps/portal/usda/usdahome
[4] http://www.fitbit.com

that can recognize the various food items visually, through a picture, and by means of a label, the recipe title. Moreover, given the huge amount of data that the website is serving, a user can filter recipes by directly quering the database via textual search or to narrow down the results by means of filters about the dish type, composition and avaiability of ground ingredients.

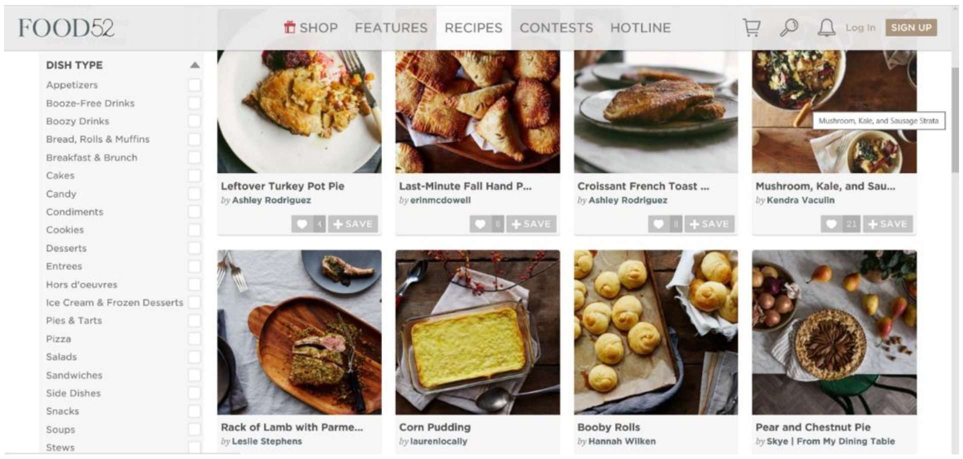
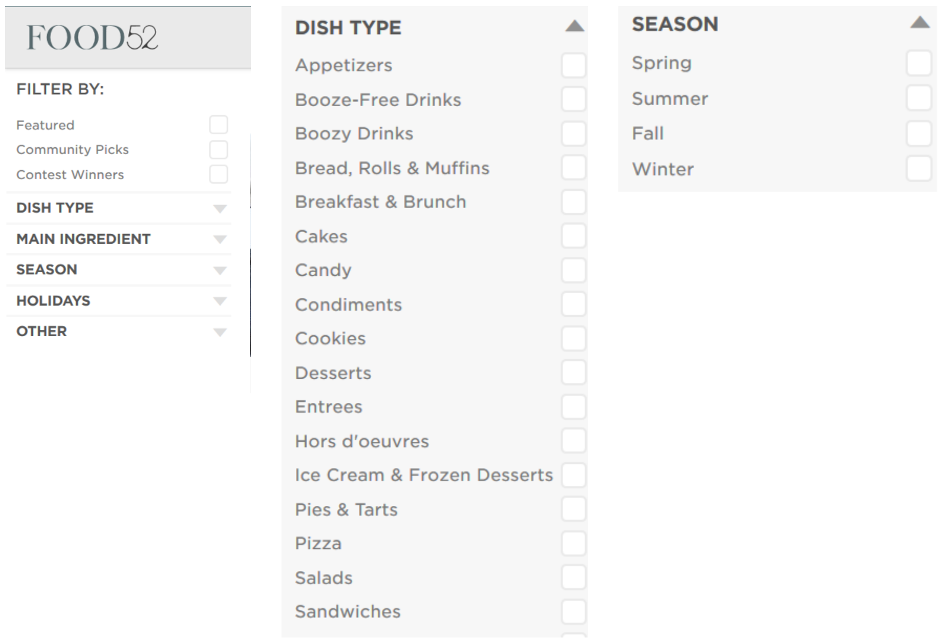


*Figure 1- food52.com recipe browsing web interface*

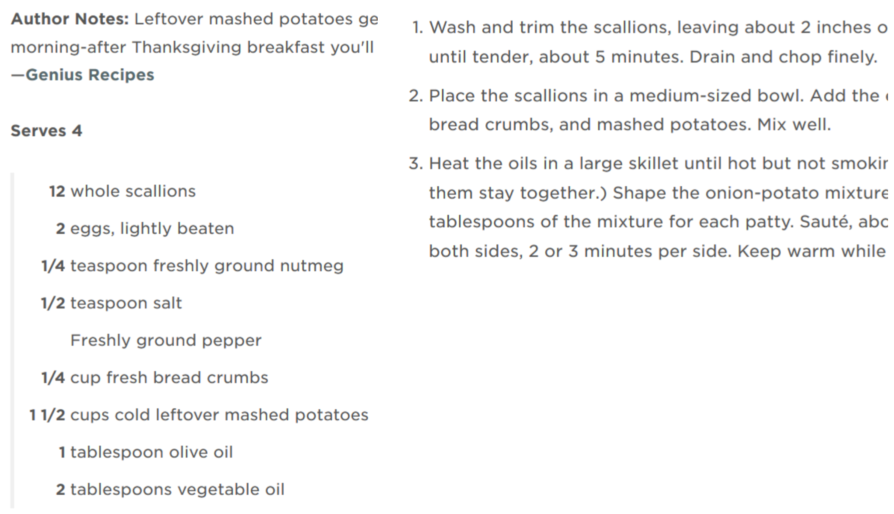
The filters that are employed in food52.com are more specifically: (1) recipes ranking according social influence (2) dish type (3) main ingredients (4) food availability in a specific time (5) special occasion like holydays, and so on. Dish type categorization reflects what can be found in a restaurant menu: filters acts limiting the view to appetizers, main dishes or desserts, as well as others food categories. Figure 2 shows the categorization details on food52.com.



(a) (b) (c)

*Figure 2 - avaibale filters in food52.com recipes browsing interface (a) overview (b) dish related (c) ingredients related*

From the recipes browsing user interface a user can access a recipe detail page by selecting an item of interest. A recipe detail page shows various data about an item, in particular can be found: a recipe image with a list of textual data that covers recipe author notes, serving units, a list of ingredients and the required quantities necessary for the food preparation, and finally preparation steps describing the methods and techniques as well timing for the cooking. In Figure 3 can be found the recipe details sections previously cited.



**Author Notes:** Leftover mashed potatoes ge
morning-after Thanksgiving breakfast you'll
—**Genius Recipes**

**Serves 4**

| | |
|---|---|
| **12** | whole scallions |
| **2** | eggs, lightly beaten |
| **1/4** | teaspoon freshly ground nutmeg |
| **1/2** | teaspoon salt |
| | Freshly ground pepper |
| **1/4** | cup fresh bread crumbs |
| **1 1/2** | cups cold leftover mashed potatoes |
| **1** | tablespoon olive oil |
| **2** | tablespoons vegetable oil |

1. Wash and trim the scallions, leaving about 2 inches o
   until tender, about 5 minutes. Drain and chop finely.
2. Place the scallions in a medium-sized bowl. Add the
   bread crumbs, and mashed potatoes. Mix well.
3. Heat the oils in a large skillet until hot but not smokir
   them stay together.) Shape the onion-potato mixture
   tablespoons of the mixture for each patty. Sauté, abc
   both sides, 2 or 3 minutes per side. Keep warm while

(a) (b)

*Figure 3 - deatils about (a) ingredients and (b) instruction steps in a food52 recipe webpage*

As done so far, we will now look at the data that comes from allrecipes.com. The browsing user interface is shown in Figure 4 and shows as for the case of food52.com recipes in terms of pictures and recipe titles, but with also a short explanation to enhance the system persuasiveness. To find recipes there are two ways: using a search box or browsing recipes by categories defined by dish types.

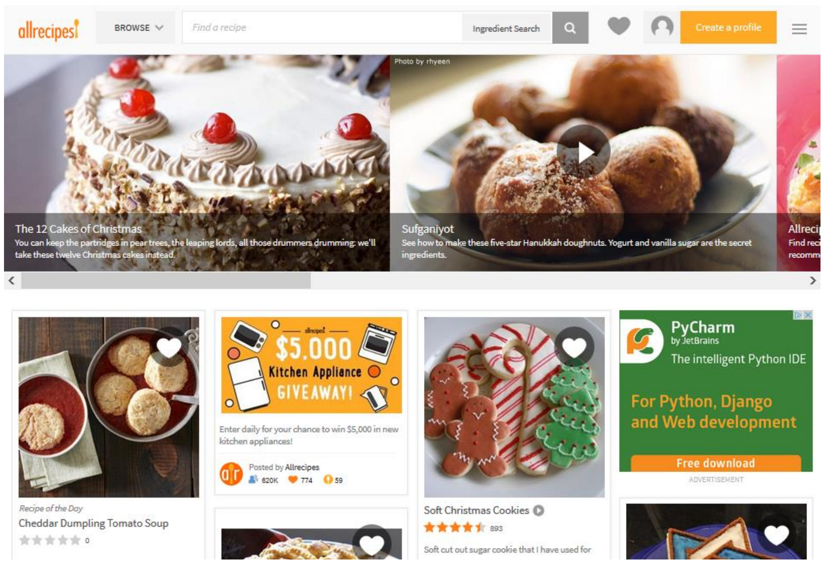

allrecipes
BROWSE
Find a recipe
Ingredient Search
Create a profile
The 12 Cakes of Christmas
You can keep the partridges in pear trees, the leaping lords, all those drummers drumming: we'll take these twelve Christmas cakes instead.
Sufganiyot
See how to make these five-star Hanukkah doughnuts. Yogurt and vanilla sugar are the secret ingredients.
$5,000 Kitchen Appliance GIVEAWAY!
Enter daily for your chance to win $5,000 in new kitchen appliances!
Posted by Allrecipes
PyCharm by JetBrains
The intelligent Python IDE
For Python, Django and Web development
Free download
Recipe of the Day
Cheddar Dumpling Tomato Soup
Soft Christmas Cookies
Soft cut out sugar cookie that I have used for

*Figure 4 -  allrecipes.com recipe browsing web interface*

Recipes classification relies on food serving type, covering appetizers to desserts in what a usual food menu could be. Figure 5 shows the browsing menu present in the allrecipes.com website.

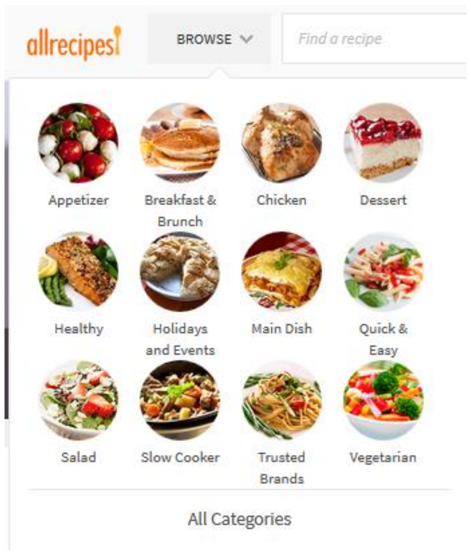
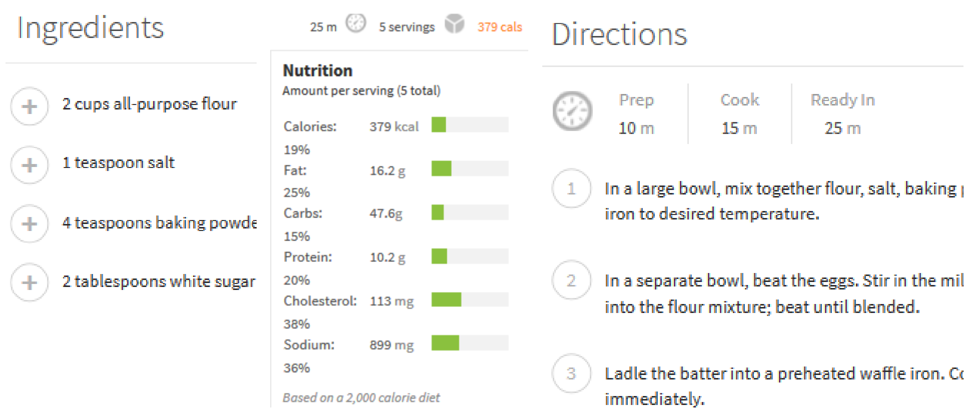


*Figure 5 - avaiable recipes categories on allrecipes.com*

A recipe detail page on allrecipes.com provides various information about the food item. Some of the kind of data that can be found, such as ingredients list and steps description are shared with food52.com, but some others are uniquely provided by allrecipes.com only. Uniquely served data are about recipe's nutrition facts, such as calories, percentage of fats, carbohydrates, cholesterol and sodium level. Moreover, timing about the preparation of the food, as well as cook time are provided to the service user. User interfaces components showing the aforementioned data can be found in Figure 6.



Ingredients

- 2 cups all-purpose flour
- 1 teaspoon salt
- 4 teaspoons baking powde
- 2 tablespoons white sugar

25 m 5 servings 379 cals

**Nutrition**

Amount per serving (5 total)

| | | |
|---|---|---|
| Calories: | 379 kcal | 19% |
| Fat: | 16.2 g | 25% |
| Carbs: | 47.6g | 15% |
| Protein: | 10.2 g | 20% |
| Cholesterol: | 113 mg | 38% |
| Sodium: | 899 mg | 36% |

*Based on a 2,000 calorie diet*

Directions

| Prep | Cook | Ready In |
|---|---|---|
| 10 m | 15 m | 25 m |

1. In a large bowl, mix together flour, salt, baking
   iron to desired temperature.
2. In a separate bowl, beat the eggs. Stir in the mil
   into the flour mixture; beat until blended.
3. Ladle the batter into a preheated waffle iron. C
   immediately.

(a) (b) (c)

*Figure 6 - Recipe details on allrecipes.com (a) ingredients list (b) Nutrition facts (c) timing and instruction steps*

From what we covered so far, according the data description of the two sources allrecipes.com and food52.com we can verbalize the following aspects to generate our recipe data model.

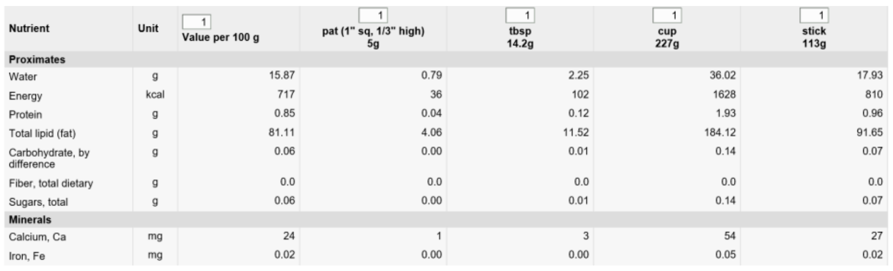
- Any recipe has an origin in terms of web resource (website url)
- A recipe comes from a particular location and is part of a culture, both location and culture determines a specific cuisine style.
- From the resources we refer to we can identify the following cuisine style: Italian, French, Spanish, Aisan and Indian.
- A recipe can be classified as part of a menu or serving time by identifying the dish type.
- The classes of dishes that are considered by our resources are exactly: Appetizer, Breakfast and Brunch, Chicken, Dessert, Healthy, Holidays and Events, Main Dish, Quick and Easy, Salad, Slow Cooker, Trusted Brands, Vegetarian. More classes that can be merged are the following: Appetizers, Booze-FreeDrinks, Boozy Drinks, Bread, Rolls & Muffins, Breakfast & Brunch, Cakes, Candy, Condiments, Cookies, Desserts, Entrees, Hors d'oeuvres, Ice Cream & Frozen Desserts, Pies & Tarts, Pizza, Salads, Sandwiches, Side Dishes, Snacks, Soups, Stews

- Each recipe can be measured in terms of time for its preparation and cooking time, both these measures lead to the total time for a given recipe.
- All the recipes can be measured in terms of amount of individuals that can be served.
- A recipe provides a set of instructions that describes the method to prepare and cook it. The method is a sequence of numbered steps that reports a textual description of the specific described task.
- A recipe is composed by ground food items called ingredients that are named with a specific food name and available in a one or more seasons.
- Ingredients in a recipe, most of the time, are associated to a numeric quantity and a unit of measurement (according the International System or the Imperial System).
- A recipe is described also by the nutrition facts it brings to an individual's energy requirement need. Nutrition facts are described by the fact name (calories, carbohydrates, fats, proteins, cholesterol and sodium), its amount given the recipe and a SI unit of measurement (kcal, g, mg)

### 2.1.2 Nutrition Data

From Table X we can derive the following textual information:

- Each food item has a list of nutrients
- Each food item belongs to a specific food group
- Each food item is identified by an internal id (USDA id)
- Each food item is described by a human readable string
- Each food item is described by a set of nutrition values
- Nutrition values refers to a specific nutrient, which can be related to others food items
- Each nutrient belongs to a specific nutrient class and is identified by a name
- Each food item's nutrient has a specific value per 100 grams expressed in International System scale
- Each food item's nutrient can have a corresponding value expressed in Imperial Quantity scale for its "value per 100 grams"



| Nutrient | Unit | 1 Value per 100 g | 1 pat (1" sq, 1/3" high) 5g | 1 tbsp 14.2g | 1 cup 227g | 1 stick 113g |
|---|---|---|---|---|---|---|
| **Proximates** | | | | | | |
| Water | g | 15.87 | 0.79 | 2.25 | 36.02 | 17.93 |
| Energy | kcal | 717 | 36 | 102 | 1628 | 810 |
| Protein | g | 0.85 | 0.04 | 0.12 | 1.93 | 0.96 |
| Total lipid (fat) | g | 81.11 | 4.06 | 11.52 | 184.12 | 91.65 |
| Carbohydrate, by difference | g | 0.06 | 0.00 | 0.01 | 0.14 | 0.07 |
| Fiber, total dietary | g | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Sugars, total | g | 0.06 | 0.00 | 0.01 | 0.14 | 0.07 |
| **Minerals** | | | | | | |
| Calcium, Ca | mg | 24 | 1 | 3 | 54 | 27 |
| Iron, Fe | mg | 0.02 | 0.00 | 0.00 | 0.05 | 0.02 |

*Figure 7- USDA Nutrition values for a specific food items (Butter, Salted)*

In Figure 7 it is possible to notice that the rows describing the conversion from the grams IS measure to any imperial quantity IQ correspondent unit provides too much information. Given that we introduce a set of data of the form shown in Table 1, where for each row description of Figure 7 there is a corresponding IQ measurement unit and a ratio for each pair of food item and nutrient. This is given by the fact that those conversion depends on the specific weight of the considered food – this weight is related to the volume and the density of a physical object. The column "Value" of Table 1 can be modeled by the following, generic formula:

$$Val_{i,n} = \frac{eq_{i,n} \cdot is_{i,n}}{100}$$

Where $eq_{i,n}$ is the equivalent value in grams of a given IQ unit for a food item, whereas $is_{i,n}$ refers to the quantity of a nutrient in a food item per 100 grams.

| Food | Nutrient | Unit (IQ) | IS Equivalent (g) | Value |
|---|---|---|---|---|
| Butter, Salted | Water | Table spoon | 14.2 | 2.25 |
| Butter, Salted | Water | Cup | 227 | 36.02 |
| Butter, Salted | Sugars | Table spoon | 14.2 | 0.01 |

Table 1 - IS to IQ conversion unit and ratio for a food item and nutrient

### 2.1.3 Physical Activity Data

As it has been explained in the introductory part of this work, we referred to activity information coming from FitBit devices. In order to access the information recorded by the device there is the need to access a public API[5], which documentation provides a clear overview of the data we can handle. From the FitBit documentation, we report relevant data describing the domain:

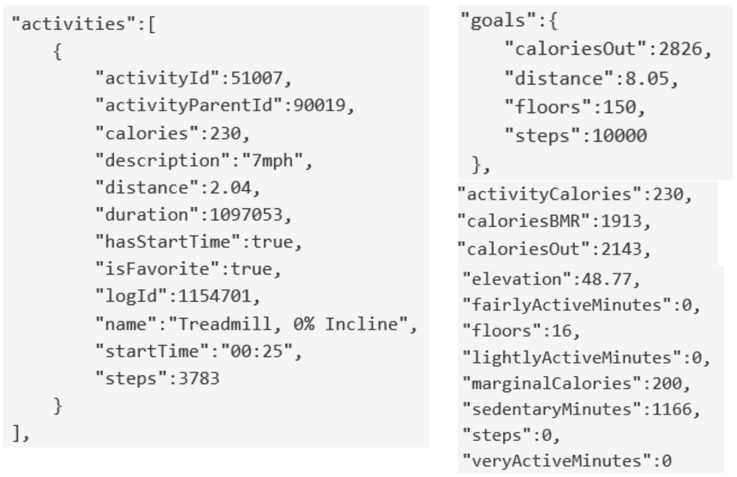
*"The Get Daily Activity Summary endpoint retrieves a summary and list of a user's activities and activity log entries for a given day in the format requested using units in the unit system which corresponds to the Accept-Language header provided.*

*Considerations*

1) *Daily summary data and daily goals for elevation (elevation, floors) only included for users with a device with an altimeter.*
2) *The steps field in activity log entires included only for activities that have steps (e.g. "Walking", "Running"); distance only included when it is relevant.*
3) *A false value for field means that activity log entry was created without explicit start time via some of our data import workflows. The field for such entry is commonly set to 00:00.*
4) *Calorie burn goal (caloriesOut) represents either dynamic daily target from the premium trainer plan or manual calorie burn goal. Goals are included to the response only for today and 21 days in the past."*

Fine grained detail about data we can collect are given in the form of API response example, it would be fair to say that many of the data can not always be collected given the dependece from the sensors on the various FitBit devices. Figure 8 shows the documentation discussed response example

[5] https://dev.fitbit.com/docs/activity



```
"activities":[
    {
        "activityId":51007,
        "activityParentId":90019,
        "calories":230,
        "description":"7mph",
        "distance":2.04,
        "duration":1097053,
        "hasStartTime":true,
        "isFavorite":true,
        "logId":1154701,
        "name":"Treadmill, 0% Incline",
        "startTime":"00:25",
        "steps":3783
    }
],
```

(a)

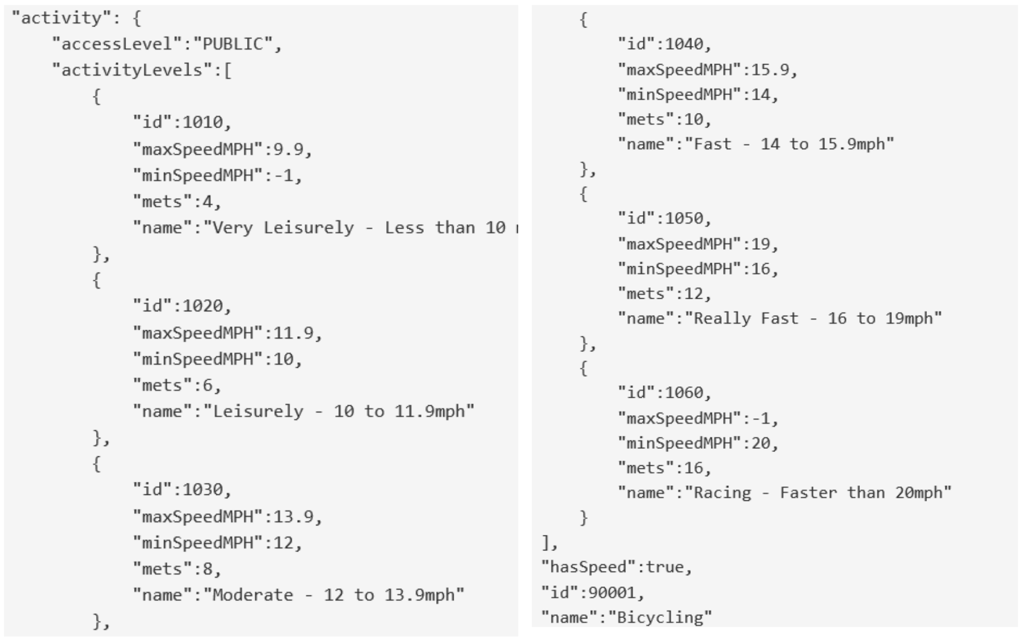
```
"goals":{
    "caloriesOut":2826,
    "distance":8.05,
    "floors":150,
    "steps":10000
 },
"activityCalories":230,
"caloriesBMR":1913,
"caloriesOut":2143,
"elevation":48.77,
"fairlyActiveMinutes":0,
"floors":16,
"lightlyActiveMinutes":0,
"marginalCalories":200,
"sedentaryMinutes":1166,
"steps":0,
"veryActiveMinutes":0
```

(b)

*Figure 8 – Example API response for daily activity summary (a) activities (b) goals and achievements*

*"Get Activity Type: Returns the details of a specific activity in the Fitbit activities database in the format requested. If activity has levels, also returns a list of activity level details."*

Example Response:



```
"activity": {
    "accessLevel":"PUBLIC",
    "activityLevels":[
        {
            "id":1010,
            "maxSpeedMPH":9.9,
            "minSpeedMPH":-1,
            "mets":4,
            "name":"Very Leisurely - Less than 10
        },
        {
            "id":1020,
            "maxSpeedMPH":11.9,
            "minSpeedMPH":10,
            "mets":6,
            "name":"Leisurely - 10 to 11.9mph"
        },
        {
            "id":1030,
            "maxSpeedMPH":13.9,
            "minSpeedMPH":12,
            "mets":8,
            "name":"Moderate - 12 to 13.9mph"
        },
        {
            "id":1040,
            "maxSpeedMPH":15.9,
            "minSpeedMPH":14,
            "mets":10,
            "name":"Fast - 14 to 15.9mph"
        },
        {
            "id":1050,
            "maxSpeedMPH":19,
            "minSpeedMPH":16,
            "mets":12,
            "name":"Really Fast - 16 to 19mph"
        },
        {
            "id":1060,
            "maxSpeedMPH":-1,
            "minSpeedMPH":20,
            "mets":16,
            "name":"Racing - Faster than 20mph"
        }
    ],
    "hasSpeed":true,
    "id":90001,
    "name":"Bicycling"
```

*Figure 9- FitbBit Activity Type Levels Example*

In Figure 9 is shown the fine grained level description of a recognized bicycling activity. For each level we can identify that the discriminators are given by the speed occurred in the activity performance of the user. Moreover, in order to explain the bounds of the activity levels it is given a human readable string that describes the activity intensity.

The knowledge about the physical activity domain can be expressed through the following points:

- A system user, namely a person, perform physical activities each day of his life.
- For each day a user can perform more than one activity.
- A physical activity in the fitbit system is identified via a numeric identifier and is also described by a human readable activity name.
- Activities that can be identified by FitBit devices are five, respectively “Bicycling", "Running", "Swimming", "Walking" and “Sleeping”
- For an activity it can be registered the time at which it started and also its duration.

- Several levels can be identified for an activity, which are described in terms of textual description.
- The levels of an activity are identified by two bounds in terms of speed. It can be identified an upper bound and a lower bound. The string that describes the activity refers to these value to give an understanding of the activity intensity.
- The physical activity done by a person has a series of objectives that are related to certain aspects of the individual's energy consumption. Such aspects are: calories to be burned, steps to be done, distance to be completed and floor climbed. Those objectives are intended within a day.
- Objectives about calories and distances are shown as decimal numbers, whereas steps and floors goals are indicated as natural numbers.
- A physical activity leads a user to consume energy. Such consumption refers to calories, steps, distance completed and floors climbed within the context of the activity.
- Activity's consumed energy features are counted with decimal numbers for calories and distance and with natural numbers for steps and floors done by the individual.
- A physical activity is estimated in terms of duration for three different activity threshold: active, light and sedentary. These three level corresponds to three condition over the speed an activity is reaching: sedentary < 1 km/h, light between 1 and 8 km/h, active over 8 km/h.

### 2.1.4 Glossary of terms

Recipe: A Recipe is a combination of ingredients and a method, created by a chef, that produces a food.

Ingredient: An Ingredient is the combination of a quantity and a food, giving the amount of something that should be used in the recipe.

Season: A Season is a period of time that recurs annually during which a food is typically available or at its best.

Step: A Step is the lowest level of instruction used in a recipe. A sequence of steps form a method.

Cuisine: A Cuisine is a macro classification of a recipe, which highlight the origin of a dish in terms of location and culture.

CookingTime: The CookingTime is the time, measured in ISO8069 format, that a recipe requires for its complete cooking process.

PreparationTime: The PreparationTime is the time, measured in ISO8069 format, that a chef needs to spend to prepare a recipe.

TotalTime: The TotalTime of a recipe is the time, expressed in ISO8069 format, within a recipe is ready. It is the sum of the PreparationTime and CookingTime.

Serving: The Serving is the number of indivuiduals that is intended to be served with a recipe.

DishType: The DishType specify a class in the context of the classification of a recipe in a Restaurant Menu context.

IQMeasurementSystem: refers to unit of measure and weights expressed according the International System

ISMeasurementSystem: refers to unit of measure and weights in the Imperial System expressed according

Nutrient: A nutrient is the bio-chemical property or descriptor associated to proximates in the context of a food item.

NutrientClass: is the class specifying the type of nutrient.

FoodGroup: identifies the group a food item is belonging to.

UsdaID: internal unique number in the United State Department of Agriculture – Database for Food Standard Reference 28

Summary: activity duration summary by date and per person

Goal: FitBit goal set for an activity in the FitBit domain.

StepGoal, FlorGoal, DistanceGoal: specification of the goal types in the FitBit domain. All of them can be measured and compared.

PhysicalActivity: is the physical activity a FitBit device or app is recording for a given user.

ActivityLevels: Aa Activity level is determined according its execution speed and participates to the formation of an activity. Many levels can compose an activity.

## 3 Structural Modelling

In this section we present the resulting data models which has been built according the domain knowledge covered in the previous chapter. For each data model it is shown an ORM schema with object types, fact types with readings, preferred identification schemes, uniqueness constraints, mandatory participations and complex constraint and derivation rules. For each schema it is discussed the translation of the raw data into the object types and where it is necessary proper adjustment to the schema will be discussed. Moreover, for each modeled domain it will be discussed a verification phase, aimed to check the inherent correctness of the diagram, and a validation phase, which is a confront of the schema with the collected data.

### 3.1 Recipe Domain

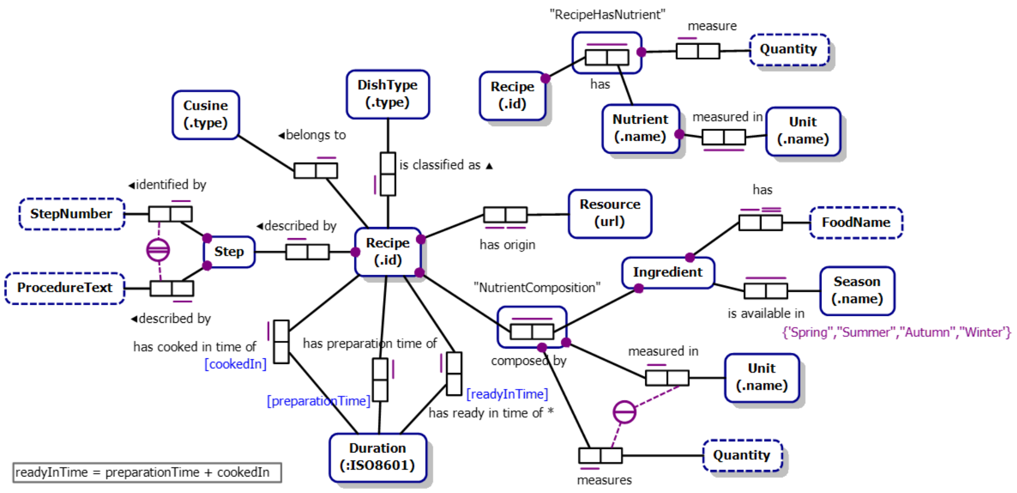


*Figure 10 - Recipe, first version of the ORM schema*

External uniqueness constraint spanning over the roles played by Unit(.name) and the value type Quantity that are attached to the objectified fact type “NutrientComposition” is redundant. The uniqueness constraints over the role played by “NutrientComposition” is enough to capture the uniqueness for a unit of measure and its value for an ingredient in a recipe.

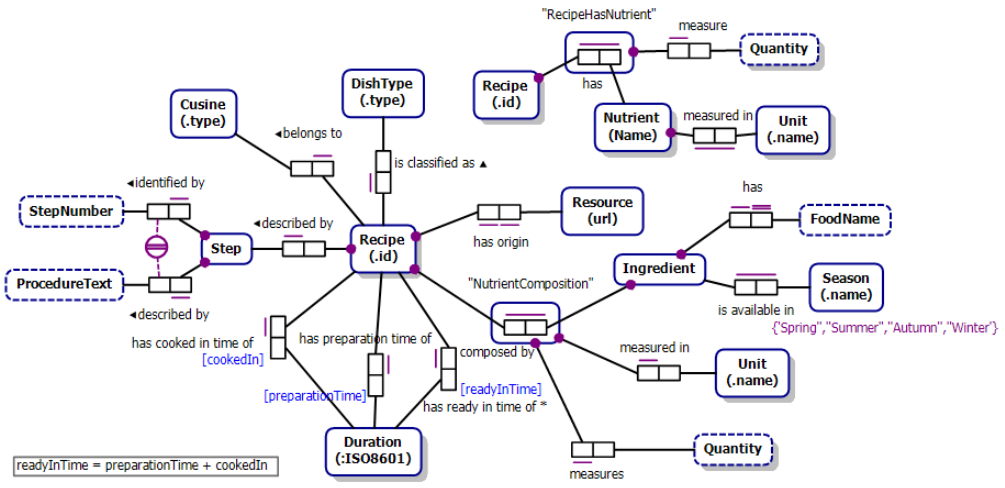


*Figure 11 - Recipe, full ORM*

### 3.1.1 Validation and Verification

| | |
|---|---|
| R1 | Any recipe has an origin in terms of web resource (website url) |
| R2 | A recipe comes from a particular location and is part of a culture, both location and culture determines a specific cuisine style. (Not really needed so far – not all the sources provides the origin of the dish) |
| R3 | From the resources we refer to we can identify the following cuisine style: Italian, French, Spanish, Aisan and Indian. |
| R4 | A recipe can be classified as part of a menu or serving time by identifying the dish type. |
| R5 | The classes of dishes that are considered by our resources are exactly: Appetizer, Breakfast and Brunch, Chicken, Dessert, Healthy, Holidays and Events, Main Dish, Quick and Easy, Salad, Slow Cooker, Trusted Brands, Vegetarian. More classes that can be merged are the following: Appetizers, Booze-FreeDrinks, Boozy Drinks, Bread, Rolls & Muffins, Breakfast & Brunch, Cakes, Candy, Condiments, Cookies, Desserts, Entrees, Hors d'oeuvres, Ice Cream & Frozen Desserts, Pies & Tarts, Pizza, Salads, Sandwiches, Side Dishes, Snacks, Soups, Stews *(Modeled as DishType withouth any constraints in order to allow future expansion via integration of other sources)* |

| | |
|---|---|
| R6 | Each recipe can be measured in terms of time for its preparation and cooking time, both these measures lead to the total time for a given recipe. |
| R7 | All the recipes can be measured in terms of amount of individuals that can be served. |
| R8 | A recipe provides a set of instructions that describes the method to prepare and cook it. The method is a sequence of numbered steps that reports a textual description of the specific described task. |
| R9 | A recipe is composed by ground food items called ingredients that are named with a specific food name and available in a one or more seasons. |
| R10 | Ingredients in a recipe, most of the time, are associated to a numeric quantity and a unit of measurement (according the International System or the Imperial System). |
| R11 | A recipe is described also by the nutrition facts it brings to an individual's energy requirement need. Nutrition facts are described by the fact name (calories, carbohydrates, fats, proteins, cholesterol and sodium), its amount given the recipe and a SI unit of measurement (kcal, g, mg) |

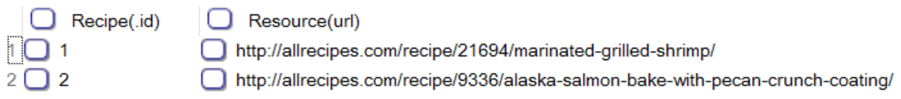
R2 is modeled just by means of the entity Cuisine, primarily identified by its type, given the fact that for the majority of the recipes the location and culture are not generally available. Instead, for many recipes a string identifying the cuisine type is available.

R5 can be better modeled by making the entity existing its own via the exclamation mark symbol, otherwise it can be modeled by constraining the possible values an instance can take. This can be done via the range notation on the schema diagram, i.e: {'Appetizer', 'Breakfast and Brunch', …}. In our case we opted to just refer to an entity dish type without constraining its value in order to ease future integration of external data sources.

### 3.1.2 Data verification

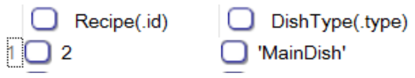
As introduced in the first section of this work recipes can come from different web sources, which are described by their url. Here follow an example for the model population.

Recipe(.id) has origin Resource(.url)



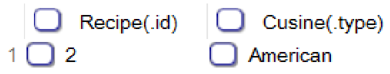
| | Recipe(.id) | Resource(url) |
|---|---|---|
| 1 | 1 | http://allrecipes.com/recipe/21694/marinated-grilled-shrimp/ |
| 2 | 2 | http://allrecipes.com/recipe/9336/alaska-salmon-bake-with-pecan-crunch-coating/ |

Recipe(.id) belongs to DishType(.type)



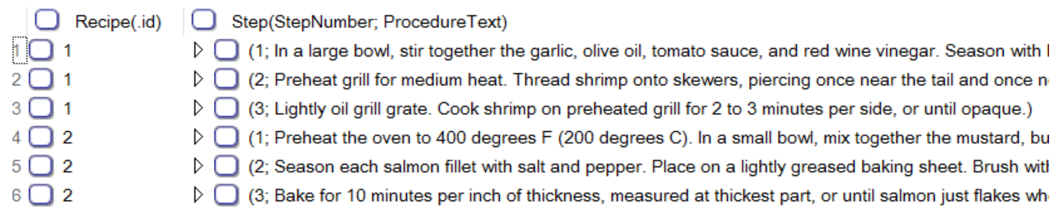
| | Recipe(.id) | DishType(.type) |
|---|---|---|
| 1 | 2 | 'MainDish' |

Recipe(.id) belongs to Cusine(.type)



| | Recipe(.id) | Cusine(.type) |
|---|---|---|
| 1 | 2 | American |

Recipe(.id) described by Step

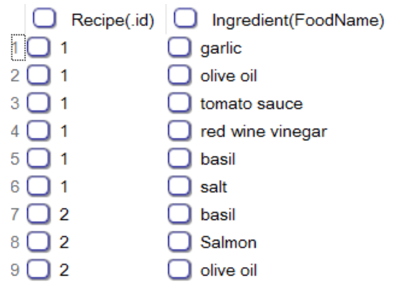


| | Recipe(.id) | Step(StepNumber; ProcedureText) |
|---|---|---|
| 1 | 1 | (1; In a large bowl, stir together the garlic, olive oil, tomato sauce, and red wine vinegar. Season with l |
| 2 | 1 | (2; Preheat grill for medium heat. Thread shrimp onto skewers, piercing once near the tail and once n |
| 3 | 1 | (3; Lightly oil grill grate. Cook shrimp on preheated grill for 2 to 3 minutes per side, or until opaque.) |
| 4 | 2 | (1; Preheat the oven to 400 degrees F (200 degrees C). In a small bowl, mix together the mustard, bu |
| 5 | 2 | (2; Season each salmon fillet with salt and pepper. Place on a lightly greased baking sheet. Brush wit |
| 6 | 2 | (3; Bake for 10 minutes per inch of thickness, measured at thickest part, or until salmon just flakes wh |

| Recipe ID | Step NR | Description |
|---|---|---|
| 1 | 1 | In a large bowl, […] 1 hour, stirring once or twice. |
| 1 | 2 | Preheat grill for medium heat. […] Discard marinade. |
| 1 | 3 | Lightly oil grill grate. Cook […]to 3 minutes per side, or until opaque. |
| 2 | 1 | Preheat the oven to 400 […] together the bread crumbs, pecans, and parsley. |
| 2 | 2 | Season each salmon fillet with salt and pepper[…] mixture. |
| 2 | 3 | Bake for 10 minutes per inch of […]Serve garnished with lemon wedges. |

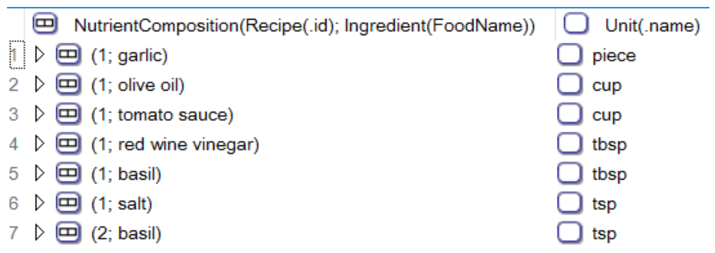
*Table 2 – Original source data format about recipe preparation steps*

Recipe(.id) composed by Ingredient(FoodName)



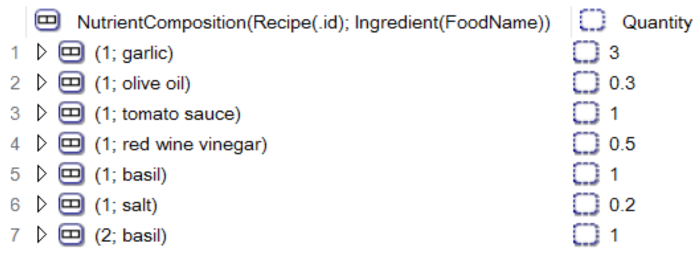
| | Recipe(.id) | Ingredient(FoodName) |
|---|---|---|
| 1 | 1 | garlic |
| 2 | 1 | olive oil |
| 3 | 1 | tomato sauce |
| 4 | 1 | red wine vinegar |
| 5 | 1 | basil |
| 6 | 1 | salt |
| 7 | 2 | basil |
| 8 | 2 | Salmon |
| 9 | 2 | olive oil |

NutrientComposition(Recipe(.id);Ingredient(FoodName)) measured in Unit(.name)



| | NutrientComposition(Recipe(.id); Ingredient(FoodName)) | Unit(.name) |
|---|---|---|
| 1 | (1; garlic) | piece |
| 2 | (1; olive oil) | cup |
| 3 | (1; tomato sauce) | cup |
| 4 | (1; red wine vinegar) | tbsp |
| 5 | (1; basil) | tbsp |
| 6 | (1; salt) | tsp |
| 7 | (2; basil) | tsp |

NutrientComposition(Recipe(.id);Ingredient(FoodName)) measures Quantity

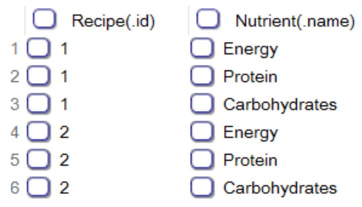


| | NutrientComposition(Recipe(.id); Ingredient(FoodName)) | Quantity |
|---|---|---|
| 1 | (1; garlic) | 3 |
| 2 | (1; olive oil) | 0.3 |
| 3 | (1; tomato sauce) | 1 |
| 4 | (1; red wine vinegar) | 0.5 |
| 5 | (1; basil) | 1 |
| 6 | (1; salt) | 0.2 |
| 7 | (2; basil) | 1 |

| Recipe ID | Ingredient | Measure | Quantity |
|---|---|---|---|
| 1 | Basil | 1 | tbsp |
| 1 | Olive oil | 0.3 | cup |
| 1 | Tomato sauce | 1 | cup |
| 2 | basil | 1 | tsp |

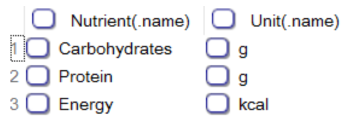
*Table 3- Extract of the original  data representation of the recipe igngredients*

Recipe(.id) has Nutrient(.name)



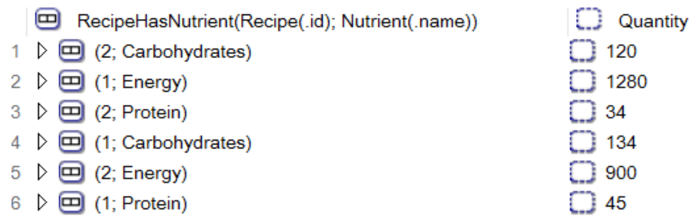
| | Recipe(.id) | Nutrient(.name) |
|---|---|---|
| 1 | 1 | Energy |
| 2 | 1 | Protein |
| 3 | 1 | Carbohydrates |
| 4 | 2 | Energy |
| 5 | 2 | Protein |
| 6 | 2 | Carbohydrates |

Nutrient(.name) measured in Unit(.name)



| | Nutrient(.name) | Unit(.name) |
|---|---|---|
| 1 | Carbohydrates | g |
| 2 | Protein | g |
| 3 | Energy | kcal |

RecipeHasNutrient() measure Quantity



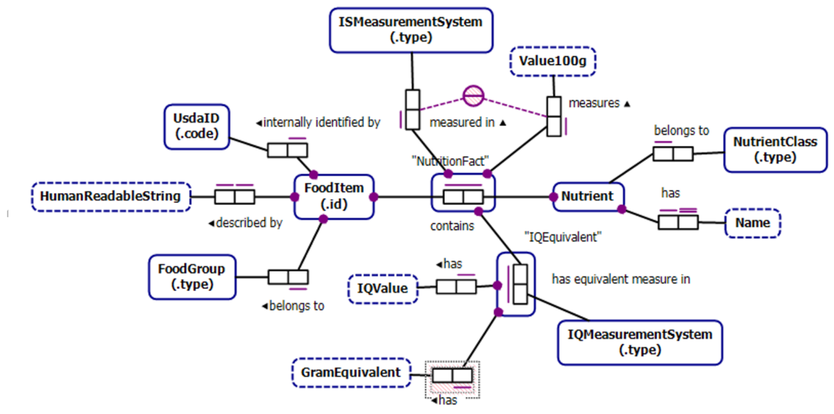
|  | RecipeHasNutrient(Recipe(.id); Nutrient(.name)) | Quantity |
|---|---|---|
| 1 | (2; Carbohydrates) | 120 |
| 2 | (1; Energy) | 1280 |
| 3 | (2; Protein) | 34 |
| 4 | (1; Carbohydrates) | 134 |
| 5 | (2; Energy) | 900 |
| 6 | (1; Protein) | 45 |

| Recipe ID | Nutrient | Quantity | Unit |
|---|---|---|---|
| 1 | Energy | 1280 | cal |
| 1 | Protein | 45 | g |
| 1 | Carbohydrates | 34 | g |
| 2 | Energy | 900 | cal |

*Table 4 - Extract of the original Recipe's Nutrition Fact table*

## 3.2 USDA Domain

V1



| | |
|---|---|
| U1 | Each food item has a list of nutrients |
| U2 | Each food item belongs to a specific food group |
| U3 | Each food item is identified by an internal id (USDA id) |
| U4 | Each food item is described by a human readable string |
| U5 | Each food item is described by a set of nutrition values |

|  |  |
|---|---|
| U6 | Nutrition values refers to a specific nutrient, which can be related to others food items |
| U7 | Each nutrient belongs to a specific nutrient class and is identified by a name |
| U8 | Each food item’s nutrient has a specific value per 100 grams expressed in International System scale |
| U9 | Each food item’s nutrient can have a corresponding value expressed in Imperial Quantity scale for its “value per 100 grams” |

Gram equivalent should not be in Food-Nutrient but in FoodItem

At this stage:

[(salt,protein), cup] -> 30

[(salt,energy), cup] -> 30

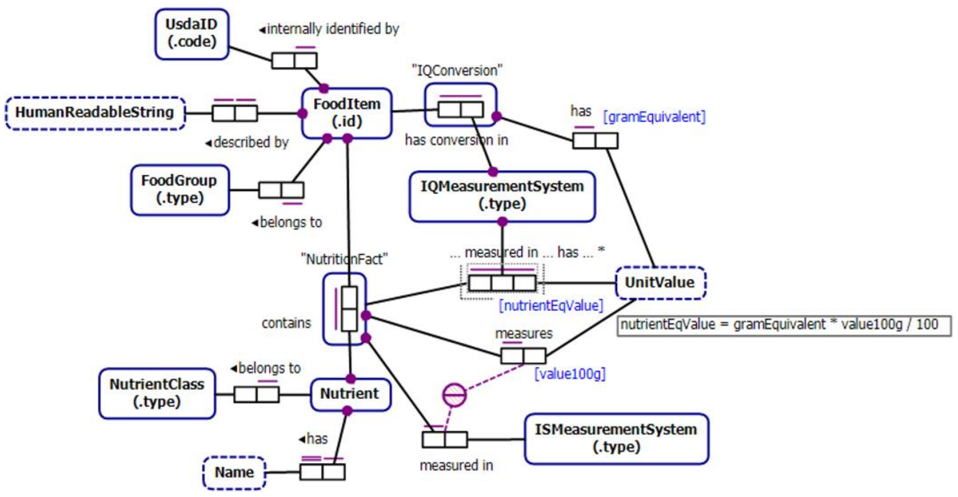
It should be

(salt,cup) -> 30g

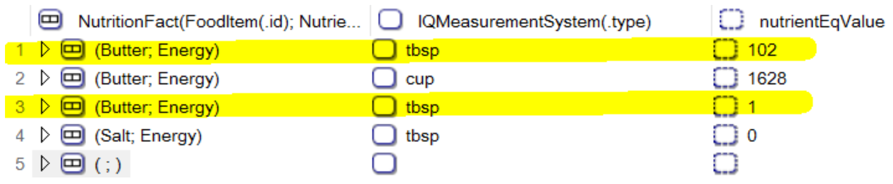
(salt,tbsp) -> 6g

So we can change the schema in such way:

V2



*Measured...in..has** violates a domain constraint over the uniqueness of Food, Nutrient and IQUnit to a unique value. Look at the example data shown below.



| | NutritionFact(FoodItem(.id); Nutrie... | IQMeasurementSystem(.type) | nutrientEqValue |
|---|---|---|---|
| 1 | (Butter; Energy) | tbsp | 102 |
| 2 | (Butter; Energy) | cup | 1628 |
| 3 | (Butter; Energy) | tbsp | 1 |
| 4 | (Salt; Energy) | tbsp | 0 |
| 5 | ( ; ) | | |

The schema needs to be modified as follows,

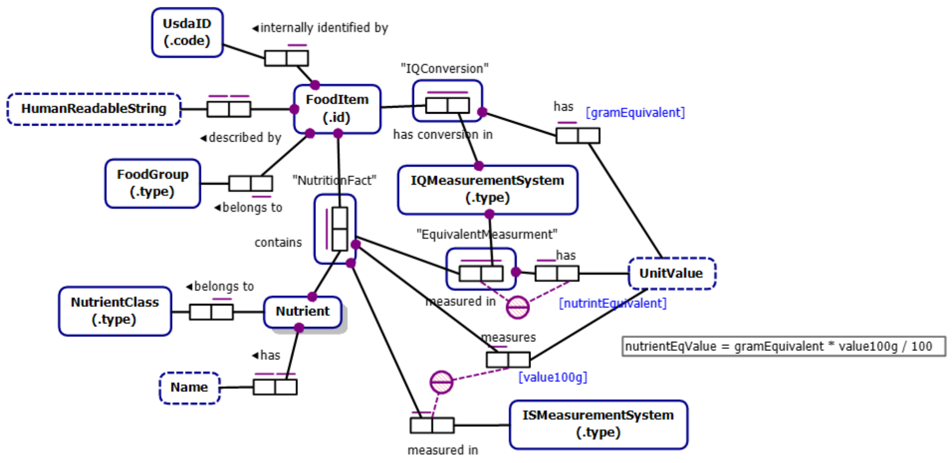


*Figure 12-Nutrients full ORM*

The external uniqueness constraint spanning from the roles involved in the fact type “EquivalentMeasurement has UnitValue” are redundant given (1) that the role attached to “NutritionFact” refers to an objectified fact type, (2) the population of EquivalentMeasurement is given by the combination of its attached roles (note that IQMeasurementSystem is mandatory). It follows that the uniqueness constraint in the first role of “has” identifies itself exactly one [nutrientEquivalent].

The external uniqueness constraint spanning from “NutritionFact expressed in ISMeasurementSystem” and “NutritionFact measures UnitValue” is reduntand given the uniqueness constraints on both facts for the role played by “NutritionFact”.

V3

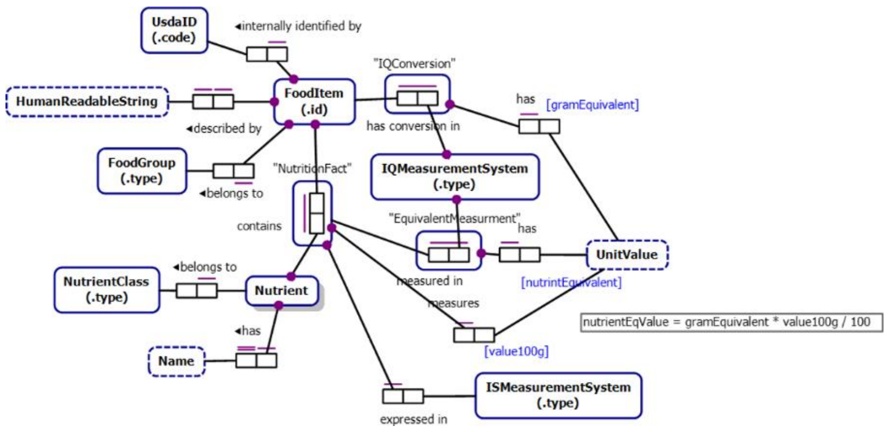


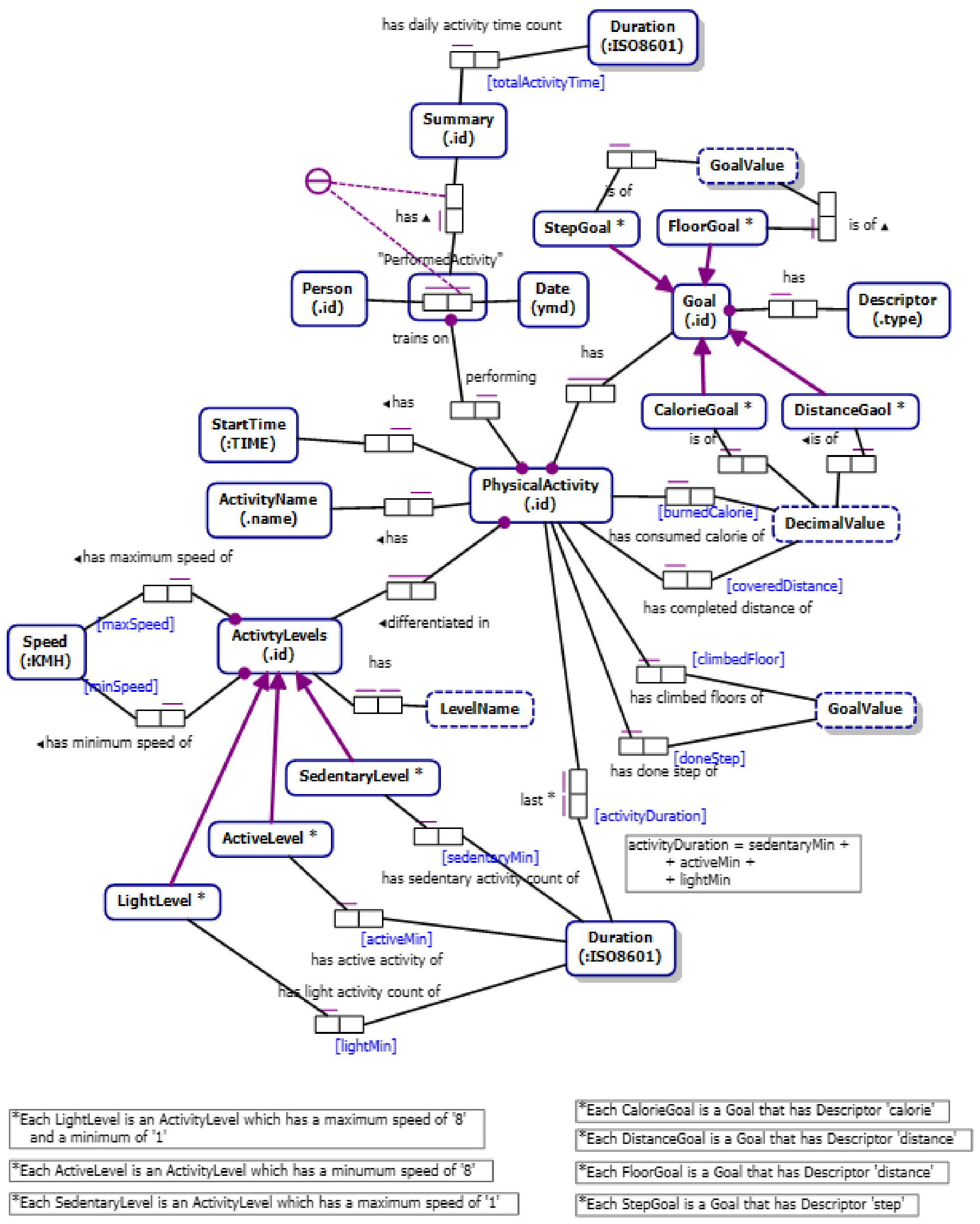
*Figure 13 - Nutrients full ORM*

### 3.2.1 Data validation

Please look at the population sample included in the model file, it clearly shows the data sample and the schema consistency. Given the complexity of the model there is not enough space to explicitly report the population sample here.

## 3.3 Physical Activity

V1



activityDuration = sedentaryMin +
+ activeMin +
+ lightMin

*Each LightLevel is an ActivityLevel which has a maximum speed of '8' and a minimum of '1'

*Each ActiveLevel is an ActivityLevel which has a minumum speed of '8'

*Each SedentaryLevel is an ActivityLevel which has a maximum speed of '1'

*Each CalorieGoal is a Goal that has Descriptor 'calorie'

*Each DistanceGoal is a Goal that has Descriptor 'distance'

*Each FloorGoal is a Goal that has Descriptor 'distance'

*Each StepGoal is a Goal that has Descriptor 'step'

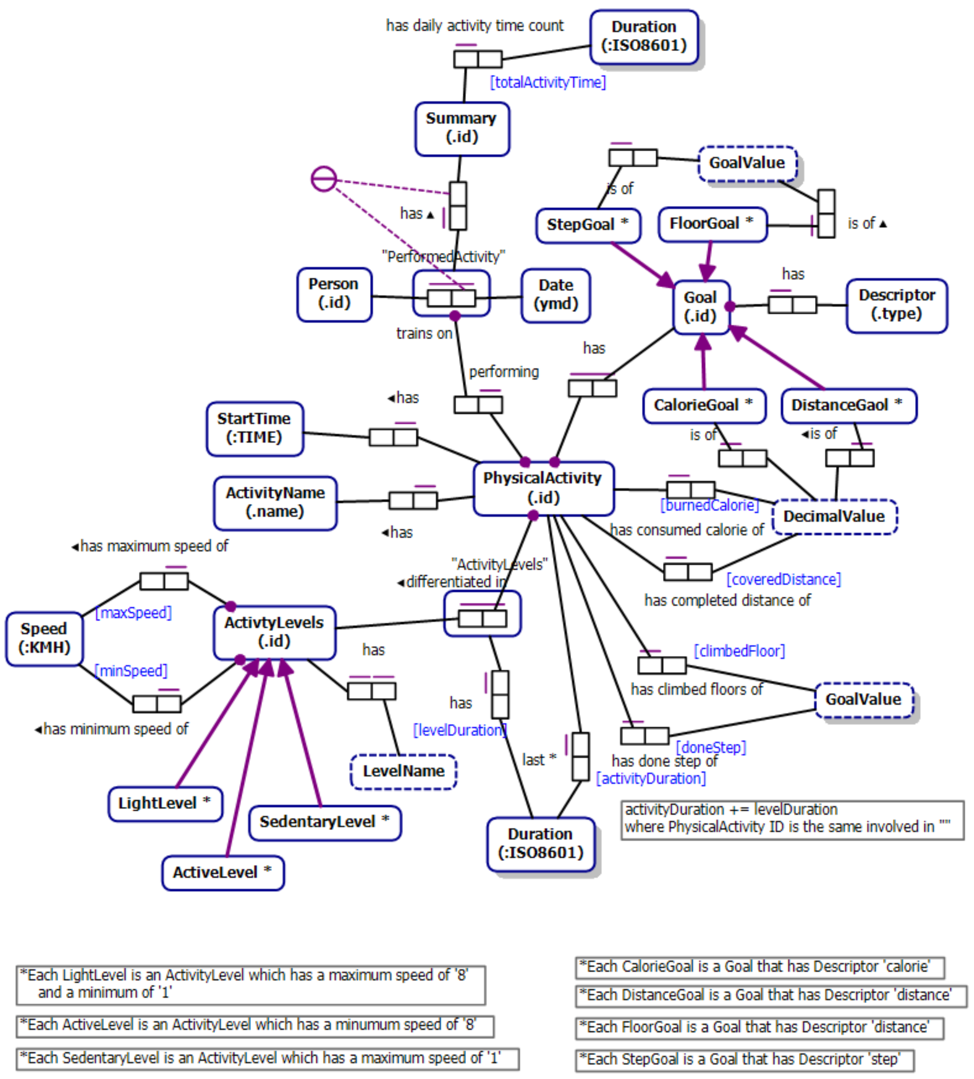
*Figure 14 - Physical Activity ORM*

ActivityDuration does not capture properly the meaning of the derivation. This is given by the fact that the three subtypes SedentaryLevel, ActivityLevel and LightLevel are, correctly attached to roles identifying their duration, but unfortunately the PhysicalActivity object, even thought conneted to the superclass of the aforementioned subtypes, cannot refer to the roles [sedentaryMin], [activityMin] and [lightMin].

In order to deal with this problem the model can be modified by pushing the duration information to an upper level where the fact type “PhysicalActivity differentiated in ActivityLevels” is played. By means of objectification the semantics can be captured referring to a unique role [activityDuration] and a complex arithmetic derivation rule for the role [activityDuration].

Derived object types can be taken for taxonomical purposes.

V2



activityDuration += levelDuration
where PhysicalActivity ID is the same involved in ""

*Each LightLevel is an ActivityLevel which has a maximum speed of '8'
and a minimum of '1'

*Each ActiveLevel is an ActivityLevel which has a minumum speed of '8'

*Each SedentaryLevel is an ActivityLevel which has a maximum speed of '1'

*Each CalorieGoal is a Goal that has Descriptor 'calorie'

*Each DistanceGoal is a Goal that has Descriptor 'distance'

*Each FloorGoal is a Goal that has Descriptor 'distance'

*Each StepGoal is a Goal that has Descriptor 'step'

*Figure 15- PhysicalActivity full ORM*

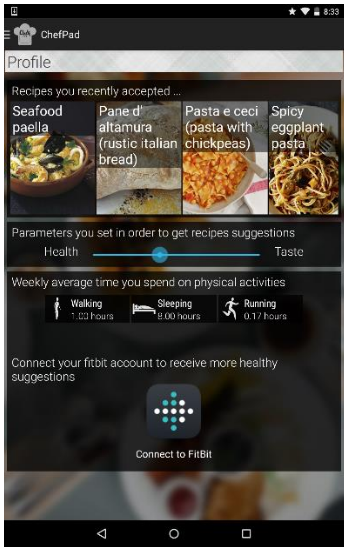
### 3.3.1 Data validation

Please show the population sample included in the file, it clearly shows the data sample and the schema consistency.

# 4 Process

In this section we identify tasks and sub-processes of the human computer interaction between a user and a mobile system as well as the system to system interaction involved in the act of linking a third party account into the system user profile.

The process we modeled is extracted by the following scenario: a user of a mobile application wants to link his FitBit account to the user profile that he owns in the mobile app he is using. The system user request the linkage of his FitBit account through a user interface that shown a button called “Connect to FitBit”. Figure 16 shows the user interface for this interaction. Once the user clicked the app redirects the user to the fitbit user authentication webpage where the user can confirm the connection to the FitBit account and the mobile app. Once the user confirmed the fitbit server redirects a response to the mobile app. This is done via the app server side. The mobile app is part of a lager system composed by a mobile client and a server application.



Recipes you recently accepted ...
Seafood paella
Pane d' altamura (rustic italian bread)
Pasta e ceci (pasta with chickpeas)
Spicy eggplant pasta
Parameters you set in order to get recipes suggestions

Weekly average time you spend on physical activities

Connect your fitbit account to receive more healthy suggestions

*Figure 16 – User interface to start the FitBit account linking*

Details about the interaction between an app and the FitBit endpoint follows are extracted from the FitBit official authenticatio doc[6]:

*"Obtaining Consent*

*Fitbit supports the Authorization Code Grant and Implicit Grant flows as defined in RFC 6749.*

*The Authorization Code Grant flow is recommended for applications that have a web service. This flow requires server-to-server communication using an application's client secret.*

*Note: Never put your client secret in distributed code, such as apps downloaded through an app store or client-side JavaScript.*

*Applications that do not have a web service should use the Implicit Grant flow.*

*WARNING - DO NOT embed the Authorization Page*

*Any attempt to embed the OAuth 2.0 authentication page will result in your application being banned from the Fitbit API.*

*For security consideration, the OAuth 2.0 authorization page must be presented in a dedicated browser view. Fitbit users can only confirm they are authenticating with the genuine Fitbit.com site if they have they have the tools provided by the browser, such as the URL bar and Transport Layer Security (TLS) certificate information.*

*For native applications, this means the authorization page must open in the default browser. Native applications can use custom URL schemes as callback URIs to redirect the user back from the browser to the application requesting permission.*

*iOS applications may use the SFSafariViewController class instead of app switching to Safari. Use of the WKWebView or UIWebView class is prohibited.*

*Android applications may use Chrome Custom Tabs instead of app switching to the default browser. Use of WebView is prohibited.*

*For web applications, do not use an iframe. Web applications may use a pop-up window, so long as the URL bar is visible.*

[6] https://dev.fitbit.com/docs/oauth2/

*Authorization Code Grant Flow*

*Fitbit follows the OAuth 2.0 Authorization Code Grant as specified in RFC 6749. Fitbit strongly recommends that you review the specification and use an OAuth 2 client library for your programming language.*

*The Authorization Code Grant flow has the following steps:*

*Your application redirects the user to Fitbit's authorization page. See Authorization Page below.*

*Upon user consent, Fitbit redirects the user back to your application's callback URL with an authorization code as a URL parameter.*

*Your application exchanges the authorization code for an access token and refresh token. See Access Token Request below.*

*Your application stores the access token and refresh token. It will use the access token to make requests to the Fitbit API. It will use the refresh token to obtain a new access token when the access token expires without having to re-prompt the user. (More information on the refresh token is below in the "Refreshing Tokens" section.)"*

From the user perspective the process starts autonomously when the user wants to bind his FitBit account to the client app we mentioned before. The user send request via the client via a proper user interface and then waits until on the screen it will be shown the official FitBit account connection user interface. At this step the user can confirm or reject the binding action. After this task the process ends.

From the mobile client perspective the process starts when a user request, specific for the FitBit and app account binding, is received. The first task is to send via POST action user and app information to the Fitbit server via callback procedure. The aforementioned task is a sub-process that has to handle the following events:

- Request error, after which the app has to show an error notification and then stop the request action. This event is blocking and led to termination.
- No internet, the app has to show an error notification and then stop the request action. This event is blocking and led to termination.
- Notify delay, event that is non blocking and when triggered led to user notification via the refresh of a progress bar on the user interface.

When the task is completed the process continues by waiting for two events, namely (1) 90 seconds – timeout set to abort the process and notify that no message is received – and (2) trigger of an inbox message coming from FitBit server with the user authentication information to be connected with the app account.

In the case of FitBit message is received the process enters a new task which is to show on the app user interface the account binding webpage FitBit employs to let a third party app to access its API for a given FitBit account. After this task the client process waits for two events: the first one is (1) timeout of 90 seconds that lead to a timeout notification and the stop of the process, the second event is (2) trigger of message from the app server describing the status of the account binding. Finally the system checks whether the received message contains an error or not. Whatever the status of the app server is the system will always return to the previous screen in the app user interaction scheme. In the case the app server contains an error the system will show a notification in parallel. After this step the process regarding the app ends.

With regard the app server process we have that it starts when a POST request from the paired client is received.  The first task is to check the validity of the request, this task is a sub-process which has to handle an invalid error request. When the invalid error event is catched the process end by sending the client app a send request rejection notification. In the case the validation task ends properly, a message about the acceptance of the request is sent to the client app. It follows a task regarding the redirection of the communication channel to the FitBit internal server. After the redirection task the process waits for two events: (1) it waits for 60 seconds and then aboirt the whole process, (2) wait for response from the FitBit internal server. Having received the response a parse task is executed to understand if error messages has been received from the FitBit backend. If errors are present the process continues by logging them and them generating an error status code to be sent to the client app, otherwise – no errors – the process continues by logging the success and generating a success status code to be sent to the client app. The process ends after sending the status code to the mobile app.

Fitbit server, its process starts when an auth request is collected. The initial task is to generate an account linking confirmation page to be shown to an end user. The page is rendered directly on a user device via a callback function set on the third party app server. The process continues by waiting for the user input (confirmation message). When the message is received the process continues with a send response to the app server and then ends.

### 4.1.1 Sub-processes Description

The sub-process “send post to app server” part of the process related to the app client starts with a first task about the collection of the user token (of the client user) present in the system, then pack the token and request data to forward a request to FitBit backend. The next task is to chack availability of an internet connection, if it is not present the system a “no internet” error is triggered, otherwise the request is sent and then the system waits for one of the the following events:

- Wait for 30 seconds and then rise an escalation event to notify a delay.
- If a request refused message is received the system rise a request error
- If the request is accepted the process ends normally

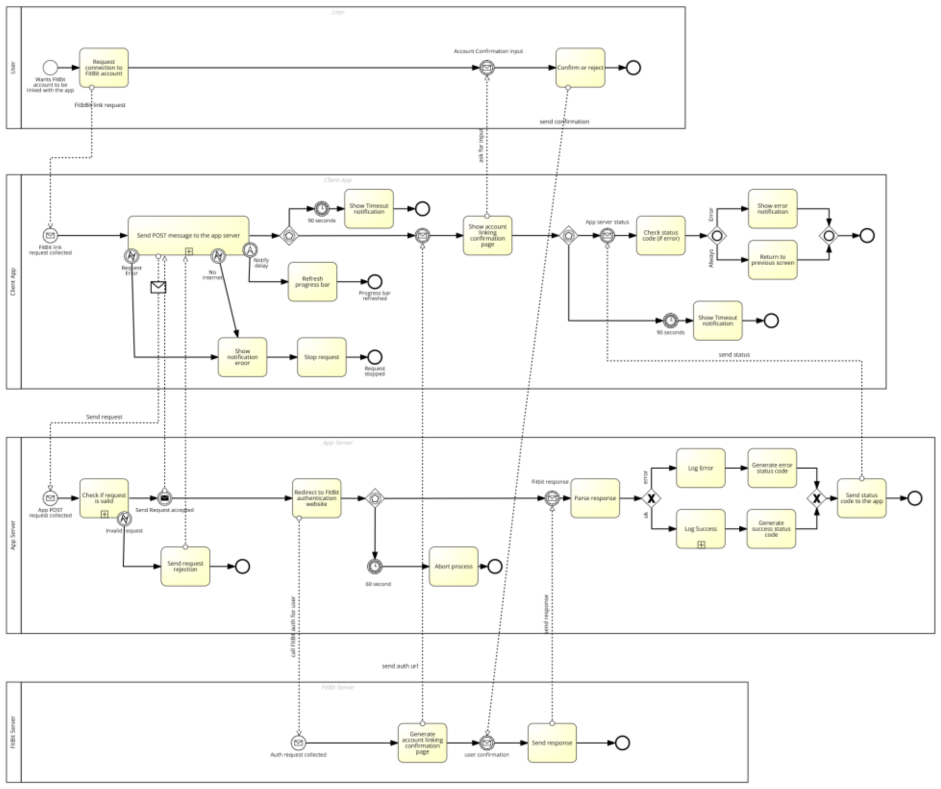
The sub-process “Check if request id valid” part of the app server process starts with a request to be checked with the task “check validity”. If the request is not valid the failure is logged and an invalid request event is triggered, in the other case the success is logged and an acceptance message is generated (then the process ends normally).

With regard the sub-process “Log success” part of the process describing the app server we can say that it starts whenever a passed response is present. The process starts directly with a parallel flow of tasks: a first one about the storage of the user refresh token received from the FitBit backend and a second one that is articulated as follows: (1) storage of user access token to access data on FitBit backend and (2) the setting of a token expiration date. After the two flows are completed the system fetch user FitBit data and store them.

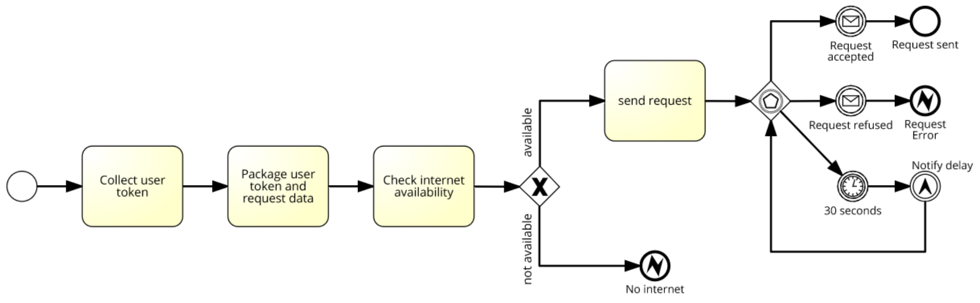
## 4.2 Identification of Process Participants

In the process we identify the following participants: (1) the system user who owns a FitBit account, (2) the mobile app – client, (3) the server side of the app and (4) the FitBit endpoint. Both user and FitBit backend processes are shown with interaction details even though they are out of the control of the system maintainer/developer. A proper representation can be the adoption of empty lanes and message exchange shown attached to lane borders. We opted for a fine grained view only for sake of clarity.

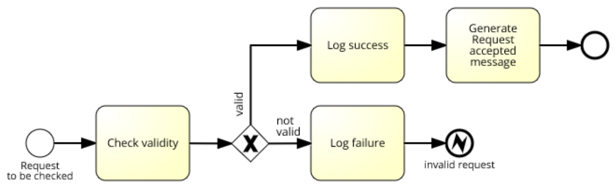
## 4.3 BPMN Diagrams


User
Wants FitBit account to be linked with the app
Request connection to FitBit account
FitbBit link request
Account Confirmation input
Confirm or reject
send confirmation
ask for input
Client App
FitBit link request collected
Send POST message to the app server
Request Error
No internet
Notify delay
90 seconds
Show Timeout notification
Refresh progress bar
Progress bar refreshed
Show notification eroor
Stop request
Request stopped
Show account linking confirmation page
App server status
Check status code (if error)
Error
Always
Show error notification
Return to previous screen
90 seconds
Show Timeout notification
send status
Send request
App Server
App POST request collected
Check if request is valid
Invalid request
Send Request accepted
Send request rejection
Redirect to FitBit authentication website
60 second
Abort process
Fitbit response
Parse response
error
ok
Log Error
Generate error status code
Log Success
Generate success status code
Send status code to the app
call FitBit auth for user
send auth url
send response
FitBit Server
Auth request collected
Generate account linking confirmation page
user confirmation
Send response

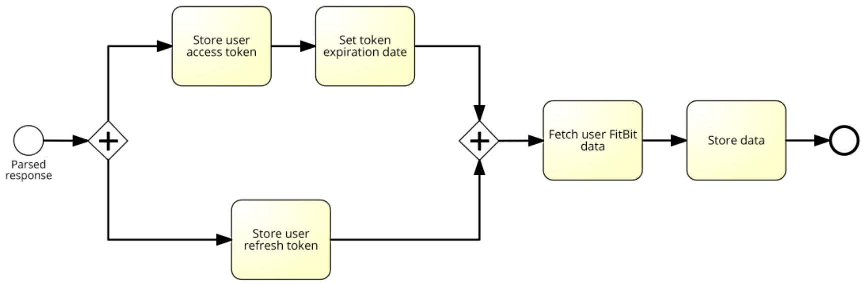
### 4.3.1 Send post to app server



### 4.3.2 Check if request is valid



### 4.3.3 Log Success


Parsed response
Store user access token
Set token expiration date
Store user refresh token
Fetch user FitBit data
Store data