NutriLife

Software Design

Date: 12th September 2018
Author: Aravind Balaji

I. INTRODUCTION

NutriLife is a diet recommendation mobile app, which considers a user's current health, ailments, and allergies and also climatic conditions to suggest healthy food.

DEVELOPMENT REQUIREMENTS

- Platform Android
- IDE Android Studio v3
- Android OS Marshmallow (Android 6.0) | SDK 23
- Build System Gradle (included in Android Studio)

REQUIREMENT SPECIFICATION

Purpose
The purpose of the software is to accept a user's physical details and medical conditions so as to suggest the best food the user can consume to stay healthy and fit.

Inputs
The app will accept the user's height, weight, age, allergies and a few health conditions.
The user may also be asked his food preference, such as non-vegetarian, vegan etc.

Output
The user is shown a list of foods that are safe and healthy to consume based on the inputs given by him/her.

BASIC PRINCIPLE - The Meal Plan

The basic working of the app involves accepting the user's preferences, and running the Diet Algorithm against the food in the database. The food is selected based on the preference of the user and his other medical conditions. We shall call the output food list as a **"MEAL PLAN".** The algorithm, which we call the **Diet Algorithm,** prepares a Meal Plan by taking the above as input. The algorithm will be explained later.

We shall now see the various relationship between the entities in the application by making use of UML diagrams. This will give an overview of the application, its use cases and its working.

II. SYSTEM DESIGN

USE CASE

Entities: Food Database Admin, Customer/User

Steps

1. Food Database Admin prepares a list of all foods with nutritional information and prepares the database. We assume the initial list is fetched from the **USDA Food Composition Database.** We may need a consulting dietician to help tweak the Diet Algorithm.

2. User initially downloads and signs into the app
3. The user enters his details such as height, weight, age, medical conditions, allergies, and his location is read using the GPS.
4. The app runs its algorithm and generates a list of healthy foods
5. The suggestions are shown to the user

USE CASE DIAGRAM

A use case diagram at its simplest is the representation of a user's interaction with the diet recommendation app that shows the relationship between the user and the different use cases in which the user is involved.

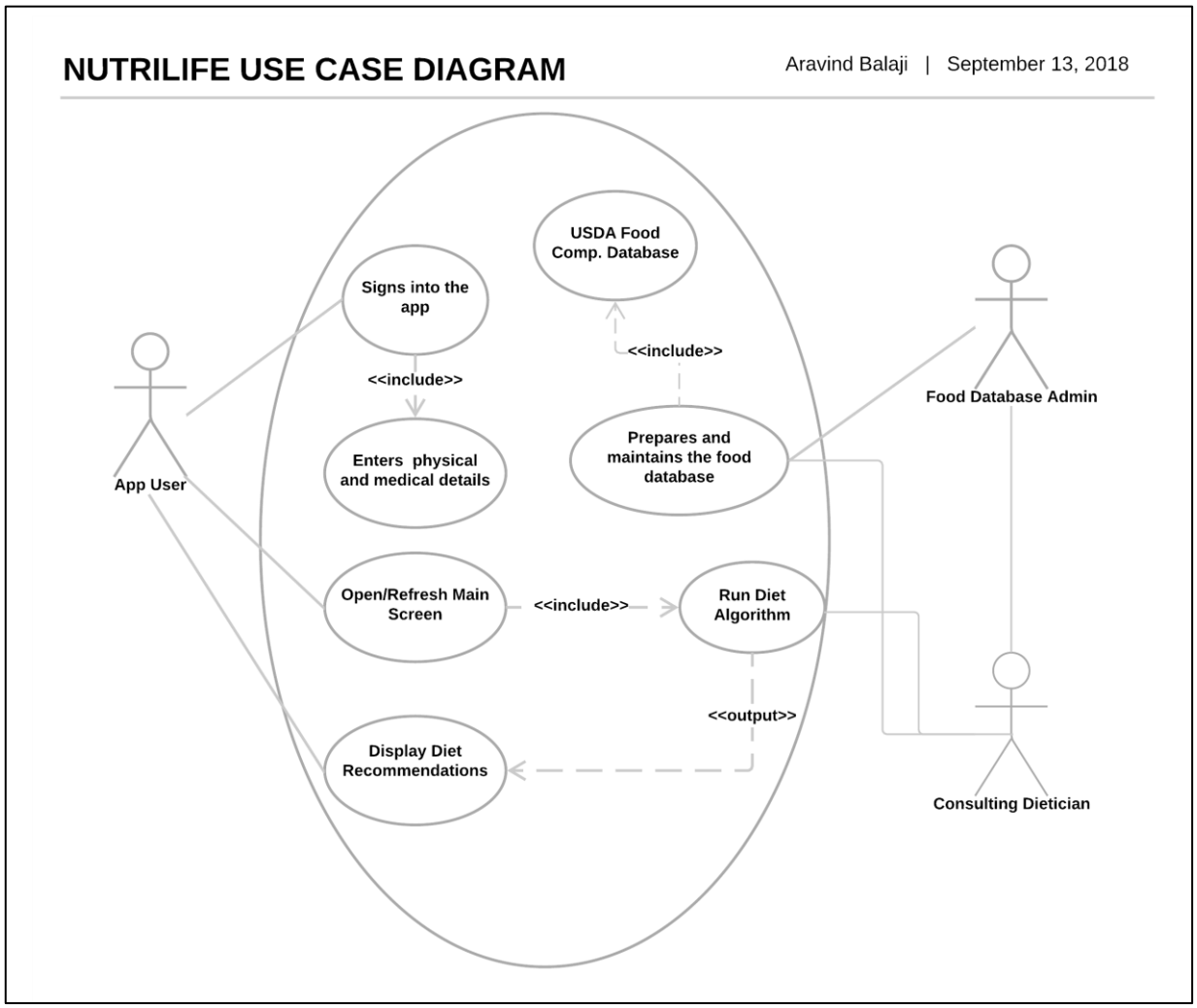

Fig. 1. NutriLife Use Case Diagram

SEQUENCE DIAGRAM

The sequence diagram shows the interaction of the user with the app and other entities that comprise the operation of the system

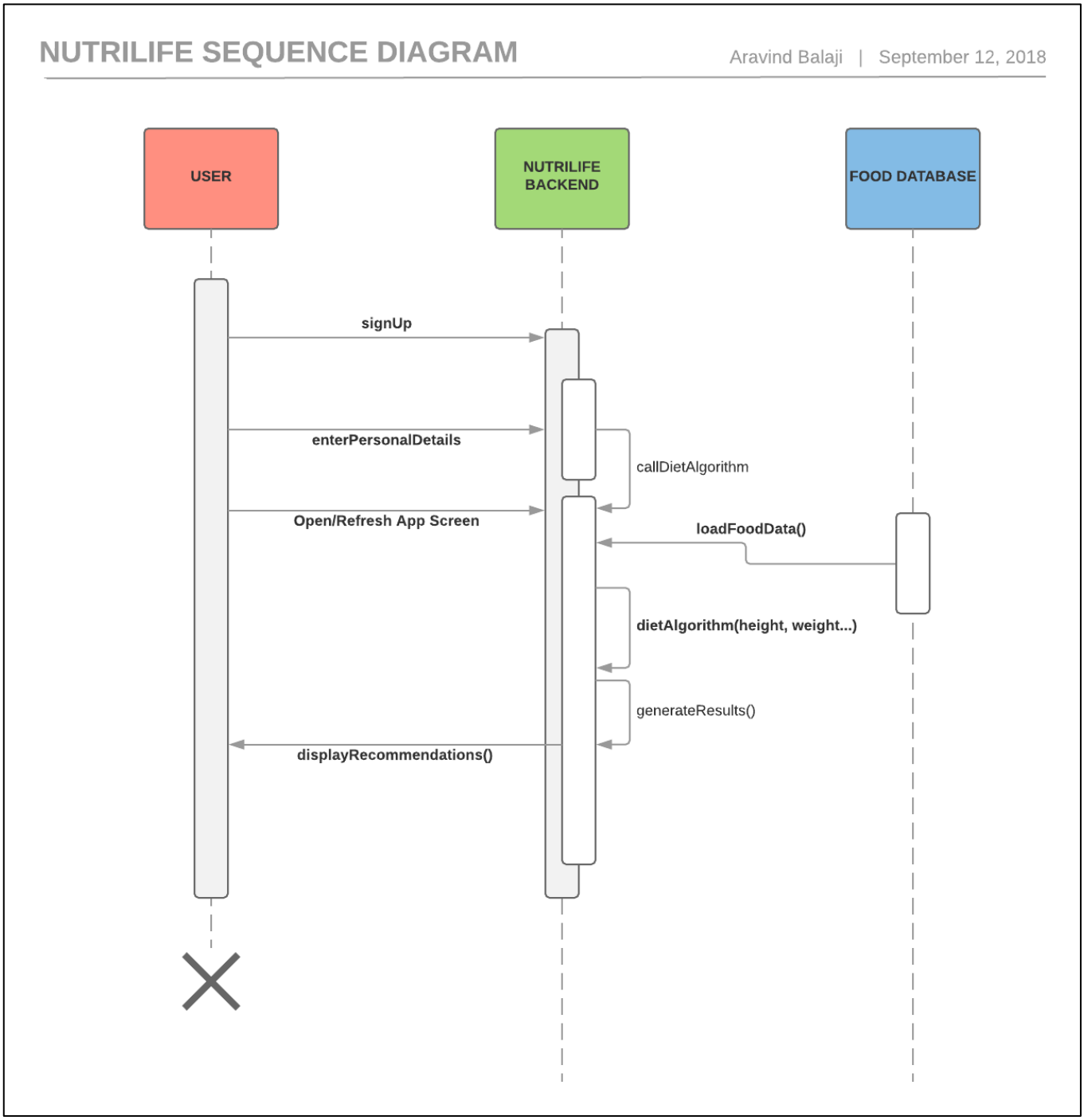

Fig. 2. NutriLife Sequence Diagram

III. IMPLEMENTATION

We shall now look, step by step, the implementation of the app starting with user registration.

1. **Welcome Screen & User Registration**
   The first step after the user downloads the app is the registration. This involves the user signing into the app via Facebook, Google, Email Address or via phone number.
   In our app, we shall use these methods to obtain basic information such as **Name, Age, Gender** and **Phone Number** (if automatic) of the user through their social profiles.

2. **Physical & Medical Details**
   Now that we have established the user and authenticated him in the previous step, we need to 'profile' the user by requesting the following details
   - Height in ft/in or in cms
   - Weight in kgs
   - Goal : Maintain/Gain/Lose Weight
   - Exercise : Rarely, 3,4,5 (times per week), Everyday
   - Preference
     - Vegetarian
     - Non-Vegetarian
     - Vegan
   - Allergies
     - Dairy
     - Seafood
     - Nuts
   - Medical Conditions, if any
     - Diabetes
     - Heart Disease
     - Pregnancy
     - Obesity

3. **Dashboard**
   The dashboard displays the list of recommended foods that the person requires based on information provided. The list is generated after running the Diet Algorithm. We shall discuss this in the next section.

   Additional features that could be added:
   - Specialist Consultation
   - Forums
   - Food Ordering
   - Notifications
   - External Trackers
   - Recipes
   - Grocery Shopping List
   - Premium Subscription

IV. CALCULATING CALORIE REQUIREMENTS OF USER

OVERVIEW

We have the user's details such as age, height, weight and medical conditions. We can use these to derive the calories burnt by a person per day. We use the **Mifflin St Jeor Equation,** and present a flowchart of the sequence of operations to be performed.

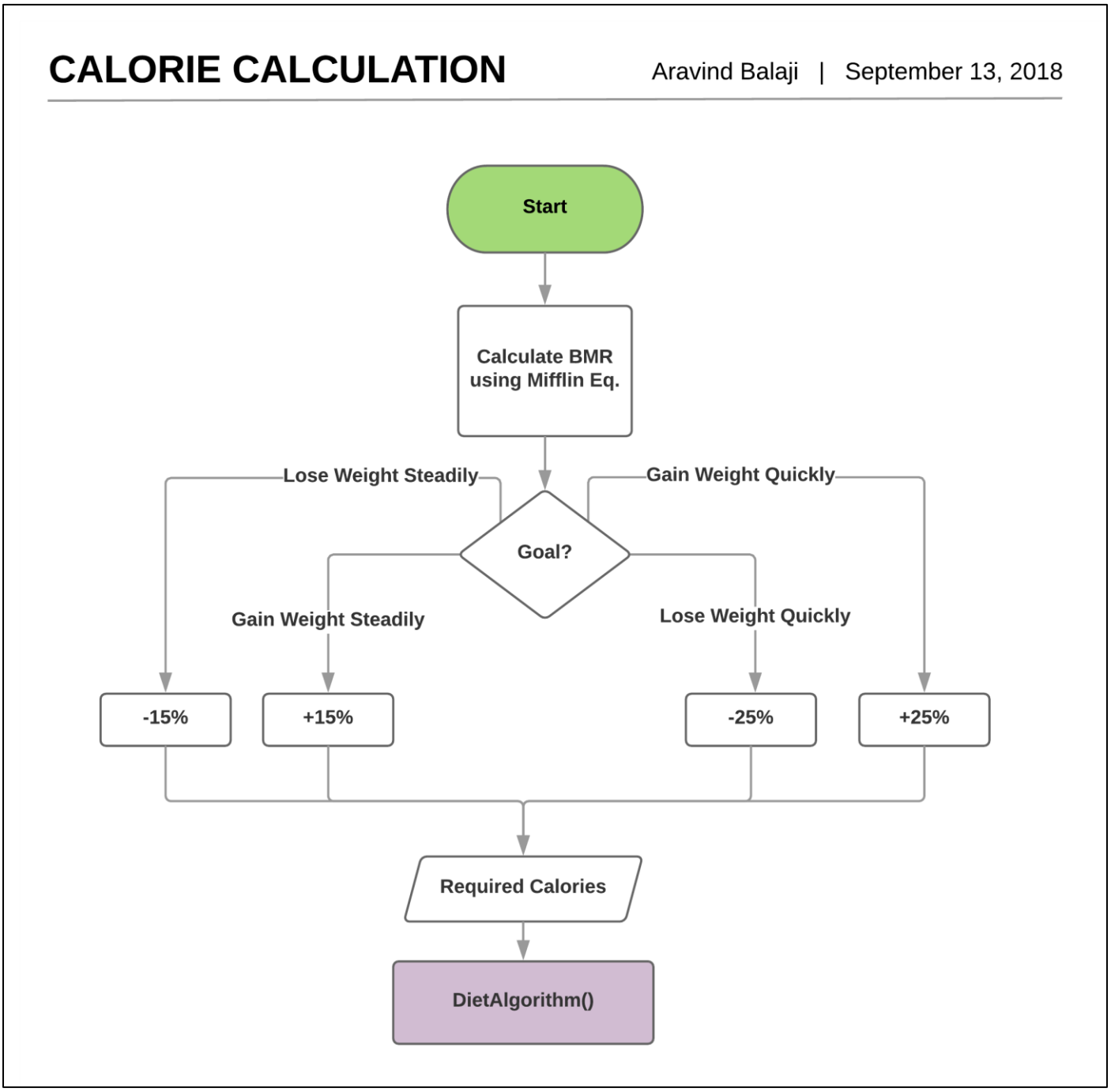

Fig. 3. Flowchart depicting the calculations

BASAL METABOLIC RATE

Metabolic rate is the rate of metabolism, the amount of energy used by an animal per unit of time. Basal metabolic rate (BMR) is the amount of energy used daily by animals at rest.
Processing food after eating uses chemical energy and produces some heat
About 70% of a human's total energy use is due to the basal life processes within the organs of the body. About 20% of one's energy use comes from physical activity and another 10% from the digestion of food after eating.

Several prediction equations exist, the most recent and most accurate equation so far is **The Mifflin St Jeor Equation:**

$$P = \frac{10\boldsymbol{m}}{1\ kg} + \frac{6.25\boldsymbol{h}}{1\ cm} - \frac{5\boldsymbol{a}}{1\ year} + \boldsymbol{s}$$

In the above formula,

P is total heat production at complete rest,
m is mass (kg),
h is height (cm), and
a is age (years),
s **+5** for males
-161 for females

PROCEDURE

1. First, we calculate the Basal Metabolic Rate (BMR):
   Men: 10 x weight (kg) + 6.25 x height (cm) - 5 x age (y) + 5
   Women: 10 x weight (kg) + 6.25 x height (cm) - 5 x age (y) - 161.

2. Next, we multiply base rate by between 1.2 and 1.9 depending on exercise level as a Total Daily Energy Expenditure (TDEE). The BMR of an individual is the total number of calories that an individual burns while at rest over a 24-hour period, the total daily energy expenditure (TDEE) value considers their BMR value plus all of their physical activities as well. We call this value as **Activity Level**

Rarely	= BMR X **1.2**	(little or no exercise, desk job)
Lightly Active	= BMR X **1.375**	(light exercise/sports 1-3 days/week)
Moderately Active	= BMR X **1.55**	(moderate exercise/sports 3-5 days/week)
Very Active	= BMR X **1.725**	(hard exercise/sports 6-7 days/week)
Extremely Active	= BMR X **1.9**	(hard daily exercise/sports & physical job or 2X day training, etc.)

3. The number of calories thus obtained is the amount currently being burned. To predict the calorie requirements for **weight maintaining, gain or loss**, we make the following adjustments to the TDEE. We call this the **Calorie Change Table**

Goal	Calorie Change
Maintain Weight	0%
Lose Weight Steadily	-15%
Lose Weight Quickly	-25%
Gain Weight Steadily	+15%
Gain Weight Quickly	+25%

Fig. Calorie Change Table

We can divide the calories proportionally depending on the type of diet with the following. We shall call this table as **Macro Ratio Table.** The following data must be obtained by consulting a dietician.

Proximates	Standard	Diabetes	Heart Disease	Pregnancy	…
Carbohydrates	50%	20%		…	…
Protein	20%	10%		…	…
Fat	30%	30%		…	…

Fig. Macro Ratio Table

USDA Food Composition Database *(United States Department of Agriculture)*

The TDEE as shown above represents how much calories are burned each day by the user. We can use this information to derive a diet plan that consist of 3 main components of every food: **Carbohydrates, Protein and Fats** *(Lipids).*

The USDA is a huge database consisting of common food items and products with their composition and nutrients. They also have an API that allows apps to query food items meeting certain conditions.

To convert the calculated calories to grams, we divide by 4 for carbs and proteins and divide by 9 for fats.

Sample details for the **Cadbury Snack Bar**

Proximate	kcal	1 value per 100g	Calories
Protein (/4)	g	4.44	1.11
Total Fat (/9)	g	17.78	1.97
Carbohydrate (/4)	g	37.78	9.44

We now have the entire database of foods with their macro nutrient values, we can use this information to fetch foods and their nutrients and develop a **Diet Algorithm** to add all foods that add up to the required calories.

V. THE DIET ALGORITHM

OVERVIEW

The following flowchart resumes from the previous step and proceeds to make recommendations to the user based on the required calories.

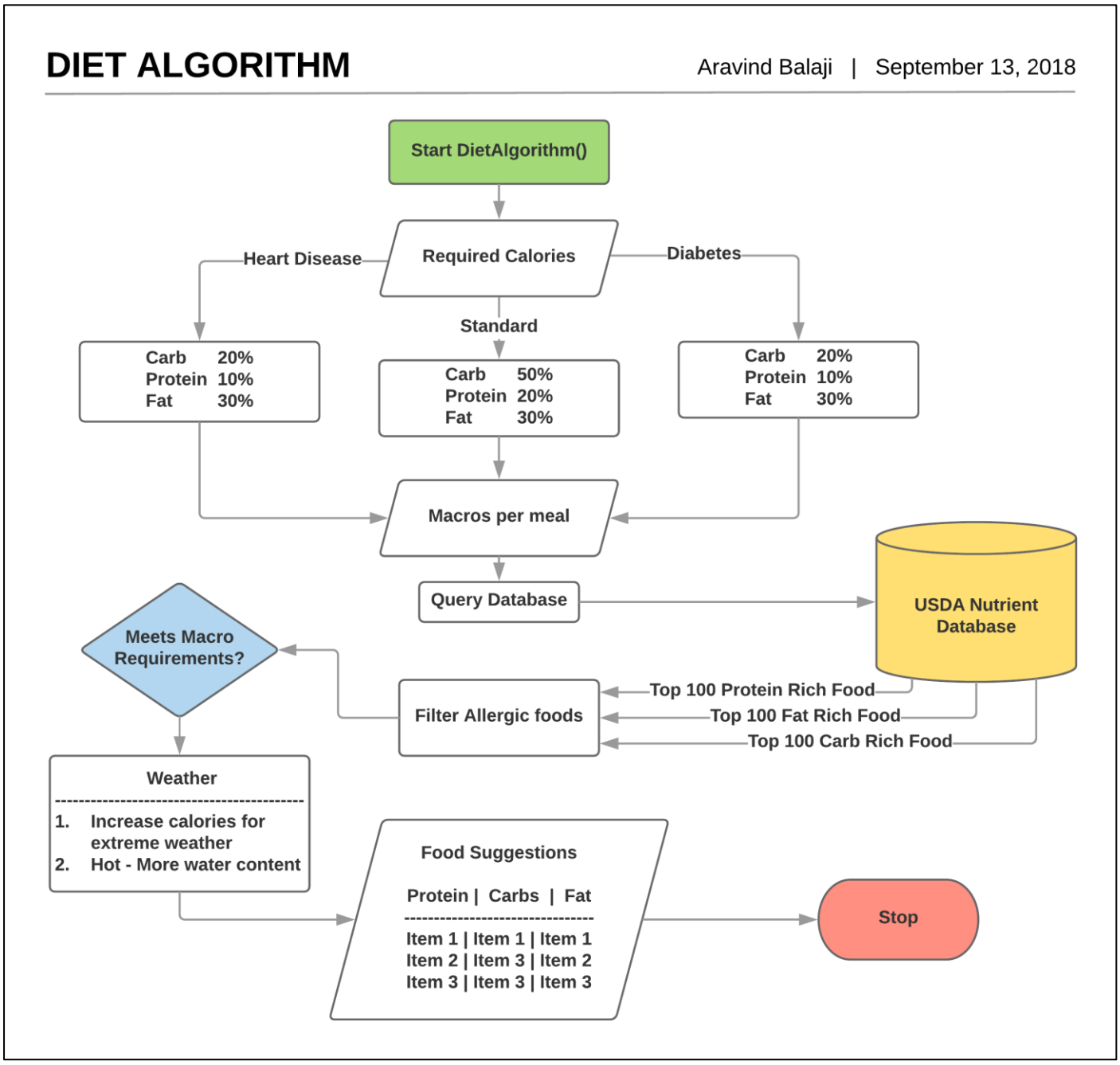

Fig. 4. Flowchart of the Diet Algorithm

Thus, the above flowchart presents an overview of the Diet Algorithm. We shall now try to formulate the above plan in terms of pseudo-code to understand it stepwise.

NOTE: The Algorithm DOES NOT check for weather conditions, reasons for which will be explained in the end. The above flowchart contains an intuition of how a weather-checking component could be included to suggest a diet based on temperatures.

We shall now proceed to prepare pseudo-code for the above procedure. To begin, we define certain notations that are used to represent entities in the algorithm.

cur_cal	Represents the BMR. Calories burned by user per day (TDEE)
req_cal	Represents the target TDEE that the user needs to achieve per day
mass	Weight of the user in Kilograms
height	Height of the user in Centimetres
age	Age of the user in years
GEN	Constant, **+5** for males and **-161** for females
eat_freq	Number of times the person eats per day. Default is 3.
act_val	Represents the activity level of the user. It holds the value of range 1.2 - 1.9 or can be represented as **enum** datatype of size 5 *(Ref. Activity Level Table)*
delta_cal	Represents the change in calorie to obtain **req_bmr** so as to achieve the goal set by the user. Example, Weight maintain, loss, gain. *(Ref. Calorie Change Table)*
req_protein	Represents the total protein requirement for the day based on health
req_carb	Represents the total carbohydrates requirement for the day based on health
req_fat	Represents the total fat requirement for the day based on health
ratio_protein	Ratio of protein required of the total required calories depending on health
ratio_carb	Ratio of carbs required of the total required calories depending on health
ratio_fat	Ratio of fat required of the total required calories depending on health *(Ref. Macro Ratio Table)*
allergies[]	A set of strings which represent the various allergies the user has. This was selected during sign up. The values are different categories of groups which the user likes to avoid, such as ‘dairy products’, or ‘nuts’.

We shall have the first function that calculates the total calorie requirement for a day for a user

```
function getReqCalorie(mass, height, age, act_val, delta_cal)
{
     //Caculating BMR
     cur_cal = (10 * mass) + (6.25 * height) – (5 * age) + GEN;
     cur_cal = cur_cal * act_val;        //Calculating TDEE
     req_cal = cur_call * delta_cal;     //Total Required TDEE

}
```

As we can see from the above pseudocode, we obtain the proportions of carbs, proteins and fats required for the user to maintain a diet based on his preferences. We can use this information to begin querying for foods in the USDA database that satisfy the above requirements.

Continued..

DIET ALGORITHM

We have a value **eat_freq,** that defines how many times the user eats per day. For each time per day the user has to consume (**req_cal/eat_freq)** number of calories. Each time the person needs to have a meal, we suggest him/her at least 4-5 items that all sum up to the required calories for that meal per day into 3 categories, Carbs, Protein and Fatty foods.

Let us consider an example:

Required Total Calories per day: **1800 kcal**

Assuming user eats thrice a day,

Breakfast 600 kcal < The user wishes to have breakfast
Lunch 600 kcal
Dinner 600 kcal

Next, we assume that the user is a standard profiled user who has no diseases. So, his distribution of calories per meal will be as follows

Carbs 50%	**Protein 20%**	**Fat 30%**
300 kcal	120 kcal	180 kcal

For the purpose of illustration, we will assume that the user wishes to have **breakfast** and he's looking into the app for suggestions. The user will be shown three sections, Protein Rich foods, Carb foods, and Fatty foods. In each of these sections, foods with highest quantity of the respective macro ingredient is suggested, which all sum up to the total requirement of each macro.

Protein Rich (300 kcal)	Carbohydrates (120 kcal)	Fatty Food (180 kcal)
Item1 (**100 kcal**)	Item1 **(45 kcal)**	Item1 **(55 kcal)**
Item2 (**100 kcal**)	Item2 **(35 kcal)**	Item2 **(65 kcal)**
Item3 (**100 kcal**)	Item3 **(35 kcal)**	Item3 **(50 kcal)**

Each section has items, whose calories add up to the total calories required by the section per meal. The following things must be considered to suggest the items.

1. The items are selected at **random**.
2. Number of items suggested can be anything between **4-5** depending on the screen size.
3. The calorie per item is decided by the **number of items divided by per section calories.**
4. The amount of protein, carbs and fat per item per section need not exactly sum up to the total requirement. Calories can never be exact, so we shall have an error window of, let's say **± 7 kcal.** The error can be distributed unequally among items.
5. Our goal is to let the user decide what he wants to eat, we're just suggesting him food based on distribution of the macros

Let us assume that for each food in the database, we have a class named **FoodObject** with the following structure in the USDA database.

```
class FoodObject()
{
     int    food_id;
     double food_protein;
     double food_carb;
     double food_fat;
     Set<String> food_group;
}
```

food_group[] A set of strings representing the various groups that a given food belongs to. A food may belong to multiple categories.
Example: “Nuts”, “Milk Products” etc.
We use these categories to **avoid suggesting food to people who are allergic** to a particular group or type of food.

protein_plan[] A set of IDs of all recommended protein foods. This is initially empty.
carb_plan[] A set of IDs of all recommended carb foods. This is initially empty.
fat_plan[] A set of IDs of all recommended fat foods. This is initially empty.

food_db[] A set of all **FoodObject** present in the database. Basically**, this is the entire database.**

STEPS

1. Calculate proportions of protein, carbs and fat to be eaten per day
2. Calculate proportions of the 3 macros to be eaten per meal
3. Fetch top 100 protein, carbs and fat foods and store in 3 sets.
4. From each set, remove all foods which belong to a food group mentioned by the user as allergic.
5. For every food in each of the three sets, fetch their macro content and use an error value to arrive at a range between which the foods may belong to.
6. If a food’s macro content fall within the range, add it to its respective macro plan.
7. Check the weather conditions. If weather is on either extremity *(too hot/cold),* we increase the required calories *(explained later).* If the weather is too hot, we also prioritize the suggestions of cool drinks or food that contains more water.
8. The three sets as output contain IDs of all foods that are suggestions.
9. We can display the suggestions in the user interface by further fetching food details, images etc… using the food ID.

Why is the weather component abstract?

The BMR works on the basic principle of the number of calories required to be functioning and considers the energy consumed by the body to warm/cool, digestion etc.

So, the BMR is proportional to the amount of body heat or internal body temperature.

1. Cold environments raise BMR because of the energy required to maintain the standard body temperature.
2. Likewise, too much external heat can raise BMR as the body spends energy to cool off internal organs.
3. Studies have shown that, the BMR increases approximately 7% for every increase of 1.36 degrees Fahrenheit in the body's internal temperature.

Why not use the external temperature?

Temperature can be very warm outside and a person who stays indoors most of time would have a cooler body. So, suggesting him foods to cool him down is not logical.

Alternative to the weather component?

We can make a general assumption based on seasons and average weather over a month.

1. During summers, we could tweak the algorithm to suggest foods which have more water content, including cool drinks. The water content of food is available in the USDA database.
2. During extreme weathers, such as when it's too hot or too cold, we can also increase the required calories per day upto a certain small percentage, say 5%.

VI. Improvements

Several improvements can be added to the application depending on the budget and system requirements.

- **Specialist Consultation** – The app can allow users to contact specialists or dieticians to provide custom recommendations tailored to their needs.
- **Forums** – Users may be allowed to interact with other users sharing ideas, opinions, tips and other useful information to achieve their goals.
- **Food Ordering** – Food may be ordered when a user clicks on recommendations.
- **Notifications** – Constant need for reminders, inspiration to reach their target calories and possible 'gamification' of the app will inspire users to follow their diet.
- **External Trackers** – External trackers, such as wrist watches, heart monitors may provide live information and can be imported into the app to optimize recommendations.
- **Recipes** – If you're suggesting what to eat, why not tell the user how to make them?
- **Grocery Shopping List** – A small to-do list feature would be very helpful.
- **Premium Subscription** – The app can be monetized by providing extra features to people who subscribe to the app. Example would be any of the above improvements or, another feature where a user can enter custom requirements of nutrients he needs and modifying the algorithm to fetch suitable foods.

The actual algorithm follows.

```
ALGORITHM  :    DietAlgorithm()
Input      :    food_db[], req_cal, eat_freq, ratio_protein, ratio_carb,
                ratio_fat, allergies[]
Output     :    protein_plan[], carb_plan[], fat_plan[]

function DietAlgorithm()
{
      req_protein = req_cal * ratio_protein;
      req_carb   = req_cal * ratio_carb;
      req_fat     = req_cal * ratio_fat;

      //Per meal requirement of proteins, carbs, and fat
      meal_protein = req_protein/eat_freq;
      meal_carb = req_carb/eat_freq;
      meal_fat = req_fat/eat_freq;

      //Choose 4 items to be displayed at a time.
      n_items = 4;

      //Display items that add up each macro
      item_protein = meal_protein/n_items;
      item_carb    = meal_carb/n_items;
      item_fat     = meal_fat/n_items;

      //The data is huge, so we use categories like vegetarian,
      //south-indian etc to avoid slow-down
      //API allows results to be sorted.

      p_temp[] = API_query("Top random 100 foods rich in protein");
      c_temp[] = API_query("Top random 100 foods rich in carbs");
      f_temp[] = API_query("Top random 100 foods rich in fat");

      //After fetching these, remove foods that are allergic
      //and suggest those that sum up the requirements

      protein_rich[] = filterSugg(p_temp,1,item_protein);
      carb_rich[]    = filterSugg(c_temp,2,item_carb);
      fat_rich[]     = filterSugg(f_temp,3,item_fat);
}
```

Continued…

```
function Set<String> filterSugg(Set<String> list,int type, int item_cal)
{
    //Choose an error of 7kcal
    //Each item will have plus or minus 7kcal
    error = 7;
    Set<String> temp;
    for(FoodObject food in list[])
    {
        //Return true if two lists/sets are disjoint
        //Takes O(m + n) on sets
        if(Collections.disjoint(food.food_group,allergies))
        {
            //This food is not allergic, now we shall check if
            //amount of nutrients adds up to the required level
            //for each macro nutrient

            max = item_cal + error;
            min = item_cal - error;
            if(min<0)   min=0;
            switch(type)
            {
                case 1:
                       if(food.protein>min && food.protein<max)
                            temp.add(food);
                        break;
                case 2:
                       if(food.carb>min && food.carb<max)
                            temp.add(food);
                        break;
                case 3:
                       if(food.fat>min && food.fat<max)
                            temp.add(food);
                        break;
            }
        }
    }
    return temp;
}
```

</Thank You>

Aravind Balaji
13th September 2018
aravind.balaji@live.com