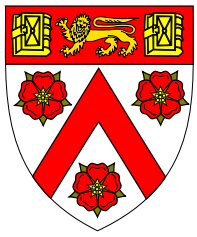
UNIVERSITY OF CAMBRIDGE

Building a Food Diary For a Mobile Device

Tiansi Jing



Trinity College

Supervisor: Professor Roberto Cipolla

Fourth year report submitted on 30 May 2018 in partial fulfilment of the requirements for the degree of Master of Engineering. I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signature:

Technical abstract

Building a Food Diary For a Mobile Device

Tiansi Jing

Trinity College

Bad eating habits cause health problems like obesity and can also originate life-threatening diseases such as cancer and diabetes. On the contrary, maintaining a healthy diet has many benefits. It not only provides sufficient energy to help people stay active during the day, but also helps prevent diet-related illness. Overall, maintaining a healthy lifestyle is important for everyone and what we eat is a big contributing factor to good health.

This report describes the design and development of a lifestyle mobile app called *Nutrient-Buddy*, which allows easy log of food items and provides useful nutritional feedback. The main focus of Nutrient-Buddy is promoting a healthy diet and assist people avoid or manage health conditions like obesity and diabetes.

People can log consumed food items in Nutrient-Buddy to track the the corresponding nutritional content. There are apps currently available which has similar functionalities, however Nutrient-Buddy is different since there are multiple types of interface for performing food search. The app interface is constructed to make food logging as easy as possible and it has the ability to perform text based searches as well as using speech recognition so the users can speak to the phone naturally. Nutrient-Buddy is also designed to be integrated with an image food recognition system in the future. Another important feature is the nutrient diary display for the six types of nutrient: energy, carbohydrate, protein, fat, water, vitamin c and sugar. A balance between carbohydrate, protein and fat has to be achieved daily while the energy consumption is limited. Also, at least eight glasses of water and sufficient vitamin C have to be consumed everyday. The users can also set a limit for the consumption of sugar. In addition, it is also possible to view the consumption of other types of nutrient. Furthermore, Nutrient-Buddy categorizes the food logs into

different meals like breakfast, lunch, dinner and snacks. All these features help users to easily keep track of what they eat and stay healthy.

Some tests have been carried out for the first version of Nutrient-Buddy. Tests to improve the technical aspect of the app are conducted by the developer to validate the performance of certain functions. The app was trialled by 19 users using exploratory and alpha test. Overall, the users were satisfied with Nutrient-Buddy and provided valuable feedback which helped develop a more user-friendly app.

Based on the feedback analysis, modifications were made on Nutrient-Buddy. Firstly, the color scheme was changed from six to three main colors. This makes the application look more professional. Later on, some display problems due to different sizes of iPhone screens were fixed. Moreover, the display of nutrient diary was adapted for an easier navigation experience and clarity of understanding. Some features were found not very beneficial to the user and further discussions are provided in the report as to why. Then, an *Amount Converter* that allows user to input food amount in terms of quantity, cup, tablespoons and volume were implemented. Finally, some guidance links were provided for limits and goals setting.

The focus of Nutrient-Buddy was on the user interface and the exploration of other forms of food logging (text search, speech search and image search). Once the backend search and database are implemented in a user friendly manner, additional extension to the app are easy. One example of extension is adding a different database or having a calendar view, all of which would make Nutrient-Buddy competitive with current food tracking applications.

It is believed that with the current development and additional features to be extended in the future, Nutrient-Buddy will be able to help people develop healthy eating habit in order to keep them strong and productive. Making Nutrient-Buddy intuitive, fast and enjoyable to use is the key factor to the success.

Risk Assessment

The risk of this project includes injury, fatigue, eye strain, upper limb problems and backache from over or improper computer use. The assessment submitted at the start of Michaelmas term reflects these hazards.

Sensible precautions were taken to control such risks. This includes ensuring comfortable desk set-up, large screens and keyboard. Short, frequent breaks were also taken to prevent strain on body and eyes.

Acknowledgement

A special thanks goes to my supervisor Professor Roberto Cipolla and Dr James Charles for guiding me through this project.

Contents

Chapter 1

Introduction

"Everything in food works together to create health or disease" according to the famous biochemist T. Colin Campbell[1]. Food is like medicine which has the power to prevent and treat disease. Everything we eat is crucial to our health.

As a consequence of poor eating habit, one's nutrient intake can be affected. Poor nutrition impairs people's wellbeing and reduce one's ability to live an enjoyable and active life. In the short term, it leads to stress and tiredness as well as reduction of capacity at work. In the long term, it can cause illness or health problems such as obesity, high blood pressure and diabetes. Sometimes it could originate serious cancers and depression[2].

In contrast, a good eating habit builds up a healthier body, as well as leads to high working efficiency and reduction in risk of diseases. Studies found that cancer prevention is possible with the aid of diet with increased fruit and vegetable intake, balanced Omega 3 and 6 fats, vitamin D and reduced sugar intake, probiotics and enzymes[3]. Moreover, science has found no magic or miracle food[4]. There is no one single food item that is preventive or cure-all for diseases, but a combination of different food to provide various nutrient that the body needs. There is a trend for people to reply on technology to sustain a healthy lifestyle. Nowadays, mobile devices can provide convenient platform for keeping track on one's activities and help them achieve weight loss goals[5]. In addition, there is an increasing number of people relying on wearable smart devices to manage their diabetes[6].

To promote healthy diet, a few mobile phone applications have been designed and developed. Some good applications includes *MyFitnessPal*[7] and *Calorie Mama*[8]. However, users of MyFitnessPal are required to manually search for food entries. This can be tedious over time. CalorieMama appears to be less user-friendly since the amount of food the

users can enter is limited by the database. Existing apps automatically tracks motion and do not allow users to log exercise in. This will lead to inaccurate measurements because users might forget to turn on the app before their work out. For people who have time consuming job, these applications are not particularly useful.

Nutrient-Buddy is an iOS[1] app that is designed to help keep quick track of the type of nutrient as well as the amount of which people consume daily. The user can type, load photos, or simply speak to the phone to log food. Nutrient-Buddy creates a personal nutrient diary, keeps personal preferences and alerts when exceeding a certain nutrient goal or limit amount. With colorful design and playful features, NutrientBuddy will also encourage children to build up healthy dietary habits since the adoption of which has been shown to be beneficial to early cognitive development particularly for children and teenagers[9].

This report gives an outline of the usage of Nutrient-Buddy as well as the design concepts in chapter 2.3. Details of the database, search engine, Search Results, nutrient diary calculation, user personal settings can all be found in chapter 3. The first built version of Nutrient-Buddy was then tested and the feedback was collected by emails. The details of testing are presented in chapter 4. In chapter 5, some improvements were made in accordance with the test feedback. There are still future extension work that can be done to make Nutrient-Buddy easy to navigate (described in chapter 6).

[1] An operating system used for mobile devices manufactured by Apple Inc.

Chapter 2

Project Overall Design

For iOS application development, *Swift*, a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS[10], and *Xcode*, the integrated development environment, are used. Xcode provide *Model-View-Controller (MVC)* base classes that can be extended to synchronize the state of the *Model*, the *View*, and the *Controller*[11]. The MVC pattern defines the roles objects play in the application as well as the way objects communicate with each other[12]. T is also a way of organizing code and its goal was to provide the illusion of a direct connection from the user brain to the computer or devices[13]. There are three layers in MVC, the Model, the View and the Controller, the detailed definitions of which are listed below.

- The Model is where data gets created, manipulated and stored. It is also in charge of management of the logic and rules of the application directly.
- The View layers the user interface of the application. It interacts directly with the users, sends user updates to the Controller and displays update from the Controller.
- The Controller interacts with both the View and the Model. It sends updates to the Model from the View and gets notified and pass notification to the view.

Figure 2.1 shows the relationships between the three layers[14] explained above.

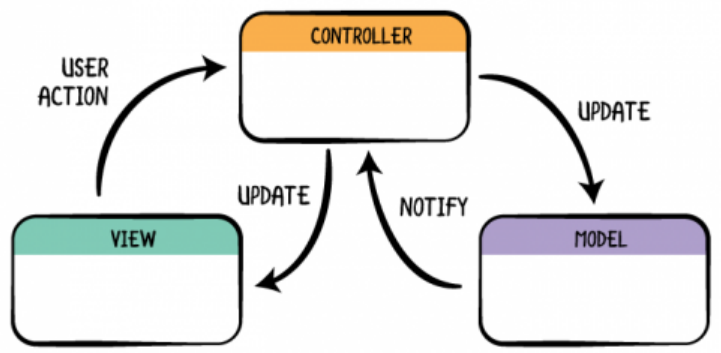


Figure 2.1: MVC Design Pattern

2.1 User Case

A list of actions that defines the interactions of the user with Nutrient-Buddy are illustrated using a user case diagram shown in figure 2.2. The user ("actor" in the diagram) has two options to interact with Nutrient-Buddy: *Log Food* and *View Diary*.

If the user chose to Log Food, *Search Food* function is included in this action. The Search Food function usually gives a list of possible items the users desire. The user can then choose to view detailed information of the food item (*View Food Information*) as well as add the food item into consumed food diary (*Add Food*). These two actions are marked as *extend* since the user could choose not to log the food to the diary. If the desired food item is found and the user decides to add the food into the diary, it is necessary to specify the amount consumed (*Specify Amount*) and save the food on the phone (*Save To Phone*).

The user could also choose to view the nutrient diary (View Diary). This function calls another function to fetch diary of the day from the phone (*Fetch Diary Today*) and load to the view (*Load Display*). *Setting* function can be selected by the users if required so that they can select more or fewer nutrient types to view (*Nutrient Type Selection*) as well as setting personal goals (*Nutrient Goal Setting*).

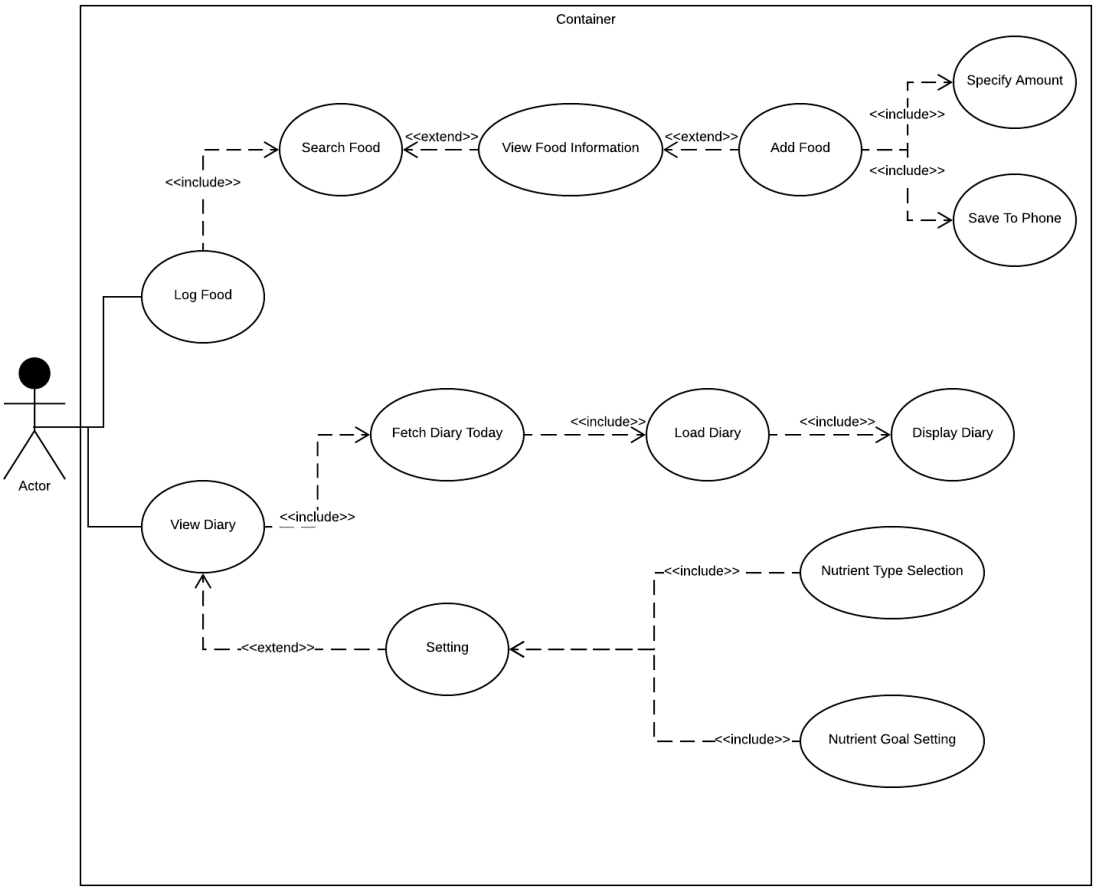


Figure 2.2: Use Case Diagram

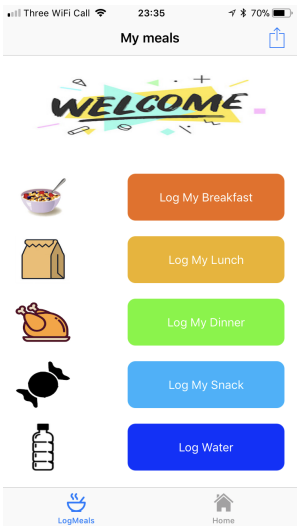
2.2 UI Flow

The user case is also explained with the aid of a flow of iOS user interfaces (UI). When the user clicks open the app, the *Launch Screen* (figure 2.3a) is presented while the on-device database (details in section 3.1) loads. The animated Launch Screen is implemented to entertain the user during the loading time. As it is shown in figure 2.3, the user can choose to switch between *Log Food Screen* (figure 2.3b) and *Home Screen* (figure 2.3c) using the tabs at the button of the screens.

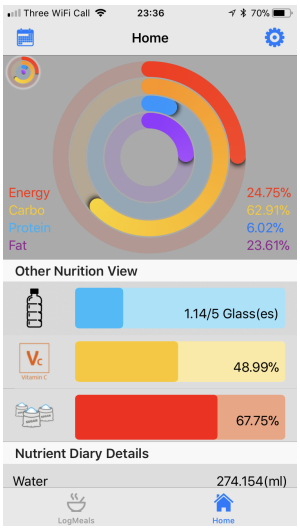
In the Log Food Screen, the user has a choice of logging breakfast, lunch, dinner, snacks and a glass of water. It can be demonstrated in figure 2.3d that the user simply needs to press the "Log Water" button in order to add one glass of water to the diary. The users will be directed to the Search Screen (shown in figure 2.3e) when tapping other buttons, where the user can type or speak to the phone to search for a food item. An example of the *Search Result* is shown in figure 2.3f when chicken curry was typed in the bar. The user can simply click the item from the result list to view nutrient information of the food. A *Food Information Screen* is shown in figure 2.3g. This page displays detailed information about specific foods, in addition to their nutrient information in tabulated form. The user can adjust the amount consumed, in grams, using the slider and press "+" on the top right to add it to their food diary.

Home Screen presents the nutrient diary. In figure 2.3c, the user can see graphics of the nutrient diary which will be further explained in section 3.3 later. The top right button on the Home Screen directs the user to the *Setting Page* shown in figure 2.3h where the user can specify their daily nutrient goals or limits for specified nutritions. The user could also follow the *"Nutrient Selection"* entry to *Nutrient Selection Screen* shown in figure 2.3i where they can select the type to view and monitor.

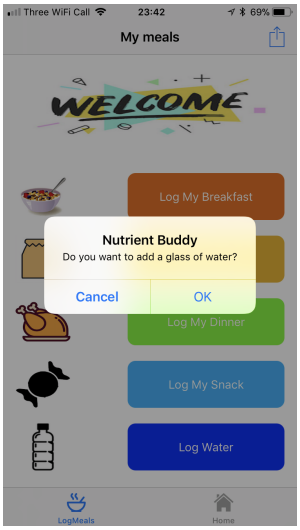
(a) Launch Screen



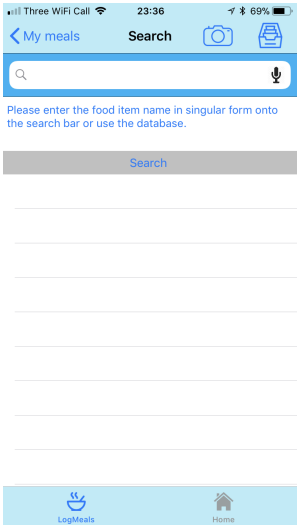
(b) Log Food Screen



(c) Home Screen

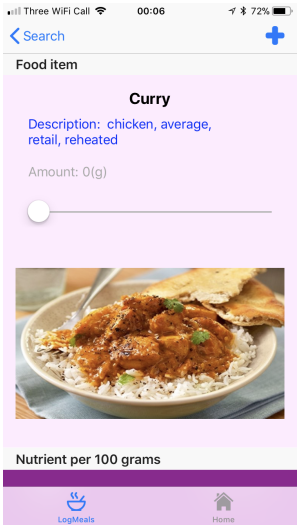


(d) Add Water

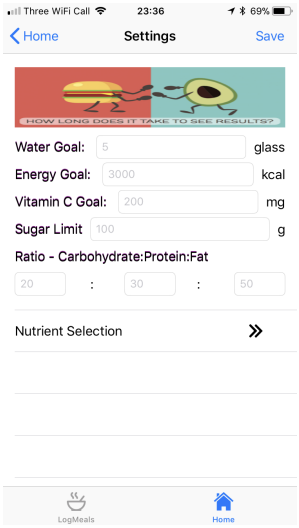


(e) Search Screen

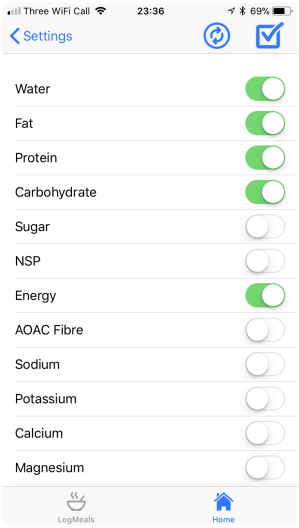
(f) Search Results



(g) Food Information Screen



(h) Setting Screen



(i) Nutrient Selection Screen

Figure 2.3: Nutrient-Buddy UI flow

2.3 View Controllers and Models

An overview of the relation controller class diagram of the main classes of the software is shown in figure 2.4. The view controllers are all implemented as separate classes in the code.

LaunchScreenViewController. This controller is linked to the view of the Launch Screen shown in figure 2.3a. The database is loaded when this screen is shown.

Database. The database of Nutrient-Buddy is stored in a *JavaScript Object Notation (JSON)*[1] file. The application reads the whole file at launching and passes the information down to the `LogFoodViewController` which is discussed next.

LogFoodViewController. This controller is related to Log Food Screen shown in figure 2.3b. This view appears after the Launch Screen and loads the food database during launch time. The food database information is passed to this controller.

SearchViewController. This controller controls the Search Screen shown in figure 2.3e. It calls the Model which contains the search engine logic and displays a list of Search Results. The food database information is also passed to this controller.

FoodInformationViewController. This controller controls the food information shown in figure 2.3g. The food information is also passed to this view controller. The user can choose to save the consumed food item in this view.

Stored data. If a consumed food is selected to be saved, the data are saved in a data storage which will be explained in section 3.5 later.

HomeViewController. This controller controls the view of the diary view shown in 2.3c, which display the nutrient diary of the day. HomeViewFunctions is the model of the class that support the controller class.

HomeViewFunctions. This class is a model and contains all functions needed to fetch diary information from the stored data and calculate the nutrient information of the day. The details are written in section 3.5.

[1] A data-interchange format which is easy for humans to read and write as well as for machines to parse and generate.

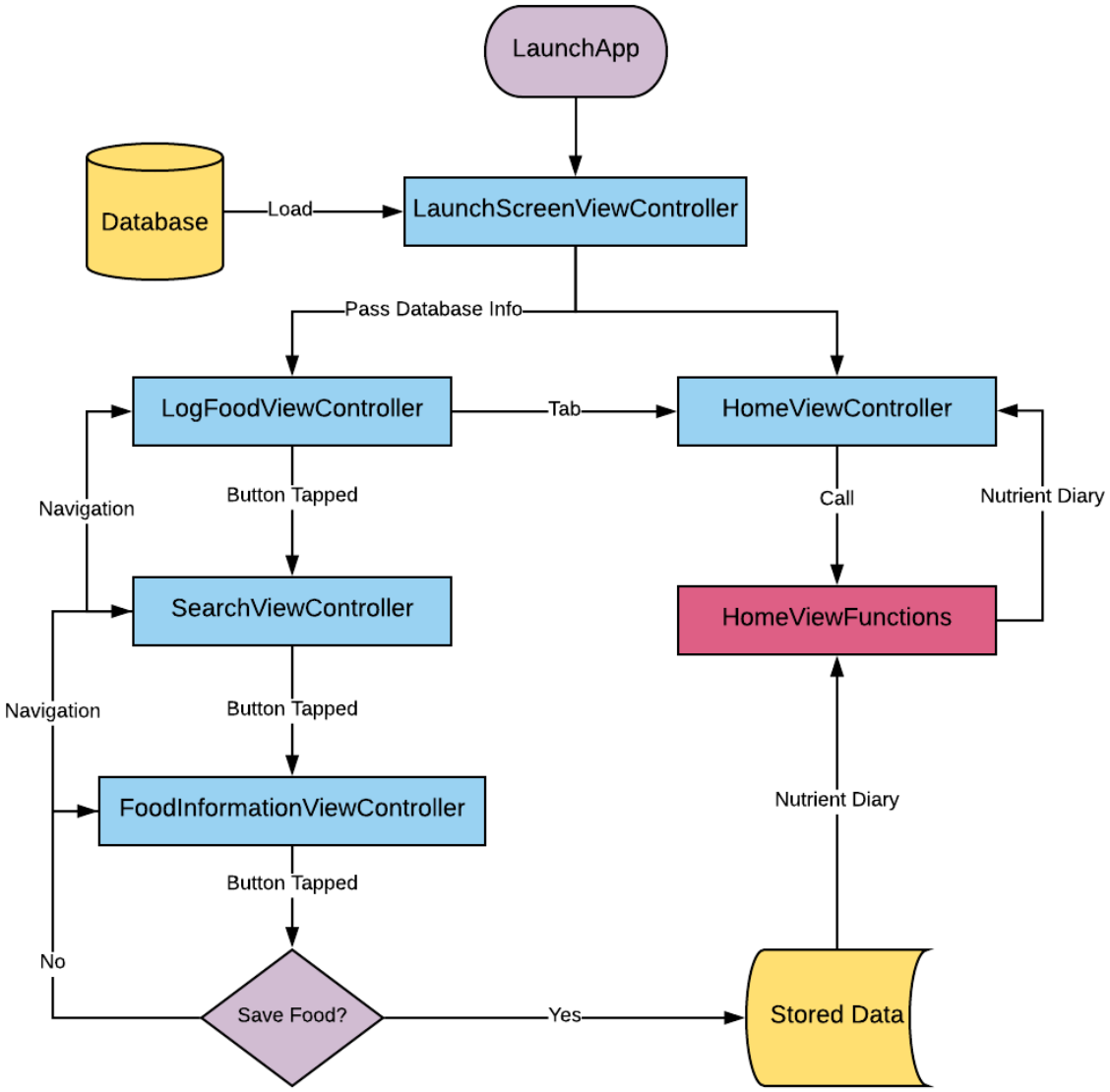


Figure 2.4: Overall Flow Diagram

Chapter 3

Detailed Design and Implementation

3.1 Database

Source of database. The food database used for Nutrient-Buddy comes from *McCance and Widdowsons The Composition of Foods Integrated Dataset*. This is provided by Public Health England which is responsible for maintaining up-to-date data on the nutrient content of the UK food supply in order to support the National Diet and Nutrition Survey[15]. The database was filtered; it now contains information about 2898 food items with 37 different nutrients. These nutrients are listed in table 3.1 in grams, milligrams and micrograms.

Table 3.1: Nutrients Listed for Each Food Item

Nutrient Content in Database		
Water(g)	Protein(g)	Fat(g)
Carbohydrate(g)	Sugar(g)	NSP(g)
Energy kcal	AOAC fibre(g)	Sodium(mg)
Potassium(mg)	Calcium(mg)	Magnesium(mg)
Phosphorus(mg)	Iron(mg)	Copper(mg)
Zinc(mg)	Chloride(mg)	Manganese(mg)
Selenium(µg)	Iodine(µg)	Retinol(µg)
Carotene(µg)	Retinol Equivalent(µg)	Vitamin D(µg)
Vitamin E(mg)	Vitamin K1(µg)	Thiamin(mg)
Riboflavin(mg)	Niacin(mg)	Tryptophan P60(mg)
Niacin equivalent(mg)	Vitamin B6(mg)	Vitamin B12(µg)
Folate(µg)	Pantothenate(mg)	Biotin(µg)
Vitamin C(mg)		

Class diagram for database. The original dataset comes in comma-separated values (CSV) form, it was filtered and converted into JSON format. The class digram is shown in figure 3.1. Each food item has one JSON entry which contains the food item name, nutrient type and the corresponding amount of each type per 100 grams of food item. This is then read into a struct called `FoodInfo`. Each `FoodInfo` has 38 attributes (food name plus all nutrition types). There are in total 2898 food entries and therefore 2898 FoodInfo instances. They are all combined together into an array of struct called *Database*. When the Log Food Screen is loading, `FoodData` class is instantiated. The database is loaded on launching of Nutrient-Buddy and is passed to `LogFoodViewController` and the view controllers that come after.

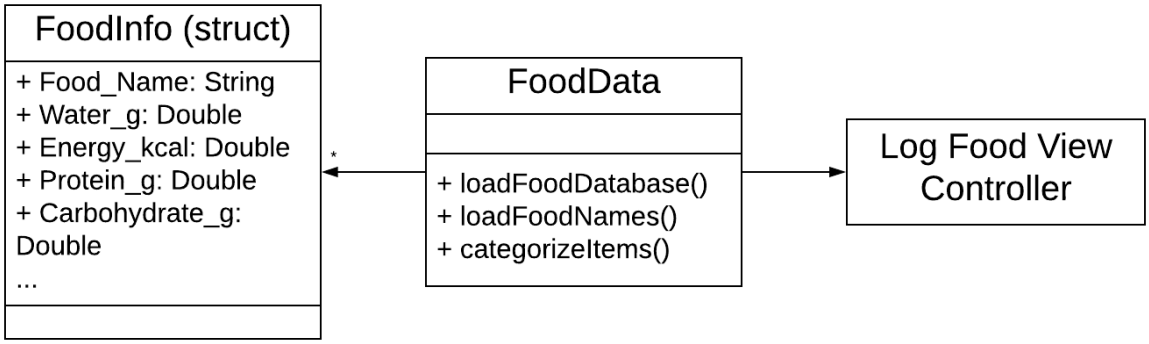
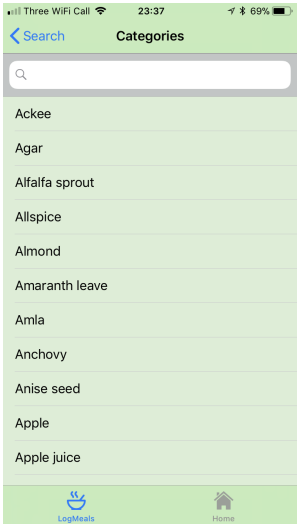
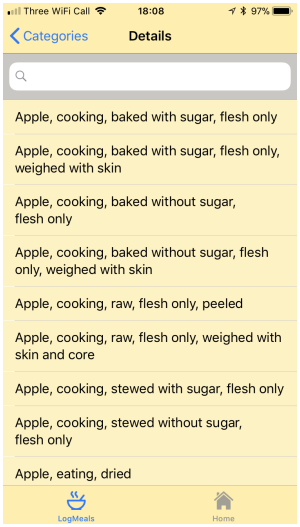


Figure 3.1: Database Class Diagram

Database view. By clicking the database button at the top right of the Search Screen (figure 2.3e), the users are able to visualize and search in the database from the app as shown in figure 3.2. One category of food material can be processed in different ways. For example, "Apple cooking baked with sugar flesh only" and "Apple cooking baked without sugar flesh only" both belong to category "Apple". A class that implements view controller `CategoryViewController` controls the view for all categories in the database shown in figure 3.2a. And `CategoryDetailsViewController` controls the view for all items in the selected category shown in figure 3.2b. As shown in figure 3.1, there is a function that categorize food items: `CategorizeItem()`.



(a) Categories Screen



(b) Items Screen

Figure 3.2: Viewing Database From Nutrient-Buddy

3.2 Search Engine

One of the most challenging parts of the project is the design of the search engine. An overview of the search engine is shown in figure 3.3. Nutrient-Buddy is designed so that the user can either input text, or speak naturally to the phone to find the food item. Moreover, Nutrient-Buddy is designed to be integrated with an image classifier (`ImageSearchModel` and the controller`ImageSearchViewController`) to enable food image recognition. There are two ways to search for food items: category filtering search and bag of words model search.

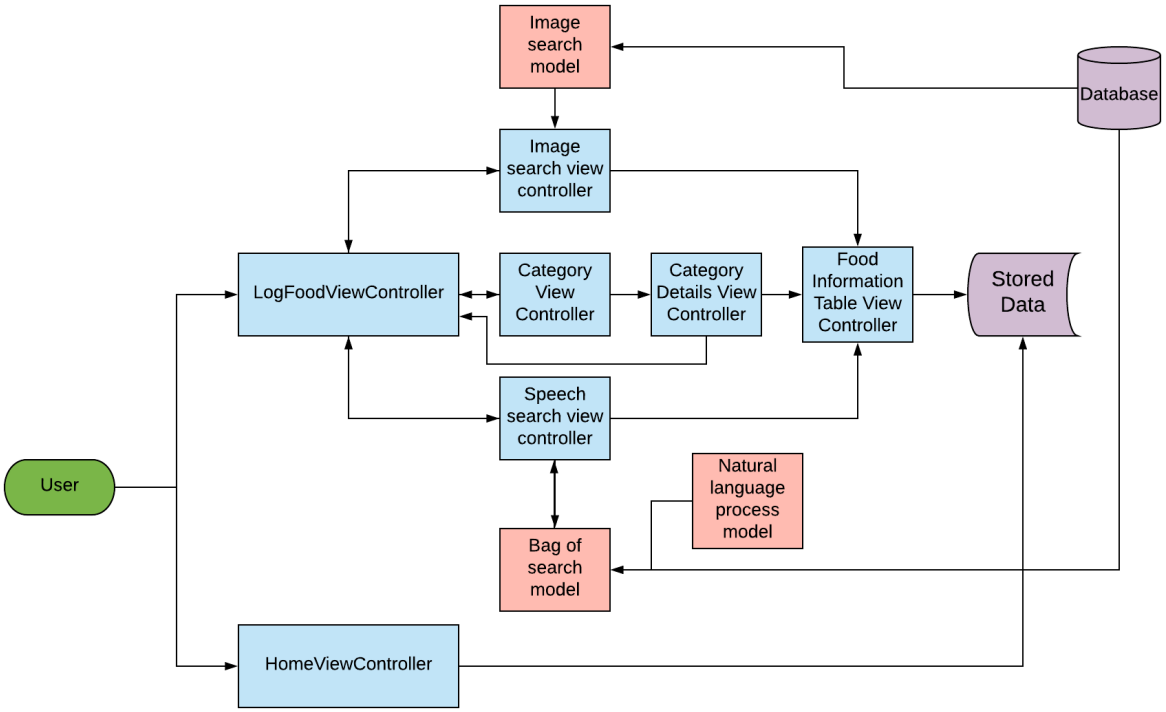
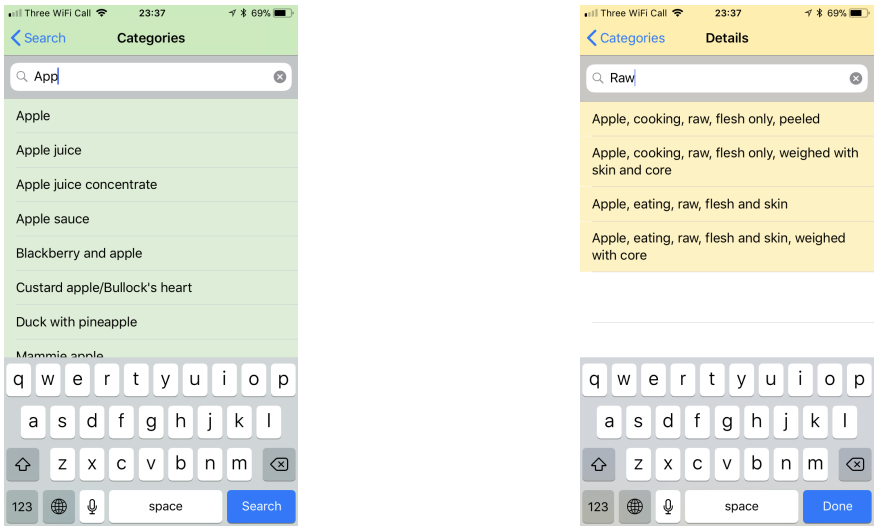


Figure 3.3: Search Engine Class Diagram

3.2.1 Category Filtering Search

Initially, a database filtering search is implemented. There is a search bar in both views, by which unwanted items will be filtered out once the user starts to input text to the search bar. This filtering happens instantly. Searching for a raw apple can be done in two steps as shown below in figure 3.4.



(a) Categories Search (b) Search Item

Figure 3.4: Nutrient-Buddy Database search

3.2.2 Bag-of-Words Model Search

Natural language processing. Users can tell the phone what they just consumed in normal speed. The *Speech Framework*[16] of Swift is used here to recognize the user speech. In order to tag the user's utterance with lexical classes, we make use of the class `NSLinguisticTagger`[17]. In natural language processing, it is necessary to extract the determiner and the nouns of a sentence. For example, if the user says: "I just had an apple", the determiner "an" and searched item (noun) "apple" are pulled out.

Bag-of-words model. The extracted information is then passed on to the Bag-of-word model[18, 19] search engine. This algorithm first stores the collection of all words that occurred in all the food item names within the database. Note that all these words do not repeat. Each word is assigned a unique index. Then all the food item names are encoded with the indexes of their word. The searched item is also encoded in terms of indexes of the searched words. The encoded searched item is then compared to the encoded food item names. The food item names that match most to the search string are returned by the algorithm.

Implementation. The implementation in Swift is slightly different from the ideal description above. All the words that occurred in all the food item names are stored in an array of strings called *allWords*, these words are then encoded with a number. A lookup table of the word and the number is stored, the name of which is *dictionary*. Then

the searched words are encoded in terms of the numbers. Note that the repeated word will be only encoded into one number here. For example, if the user accidentally inputs "apple apple" in the search bar, only one "apple" will be encoded. All the food names in the database are sorted alphabetically and then encoded in numbers according to the dictionary. A second look up table (*allDict*) is created where each line stores a food name with its corresponding number code. These are the preparations before the search begins.

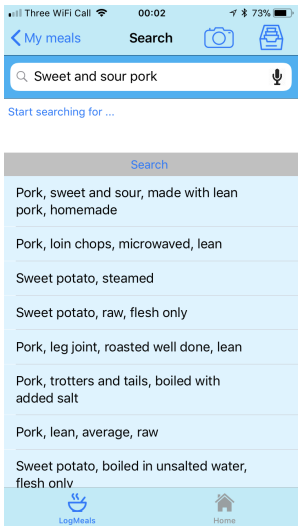
For the searching, the encoded searched word is compared with every encoded food names in `allDict`. Those that have one or more common code will be filtered out. A table of the food names from the database and the number of common code is created (`filteredDict`) where the former are string variables (referred to as key) and the later are doubles (referred to as value). The value is the number of matching words between the food name (key) and the serached words. These values are then altered as

$$n = \frac{n}{index + 1}$$

where n is the initial value in filteredDict and $index$ is the position of the first searched word that appear on the database food item names. The resulting `filteredDict` with the new key value is then sorted with respect to keys. And the values of `filteredDict` are displayed in the searched view.

3.3 Search Results

A list of possible items is shown in the table once they are narrowed down. Here are some Search Results for the model when the user attempts to search for "sweet and sour pork" as well as "curry chicken" (figure 3.5). For these two items, the most desired item appears on top of the displayed list.
Once the table cell of the item the user desires is clicked, the food information is displayed. The display contains the category name, followed by a more detailed description, an image, and a list of nutrient information(figure 2.3g). This view is controlled by `FoodInformationTableViewController` (figure 3.3).



(a) Search for Sweet and Sour Pork

(b) Search for Chicken Curry

Figure 3.5: Bag of words Search Results

3.4 On-Device Storage

Core Data is a framework developed and maintained by Apple Inc. It is used to managed the model layer objects in the application[20]. One important feature of this framework related to Nutrient-Buddy is that it provides solutions associated with object life cycle and object saved onto the hard disk of the phone. The Core Data model is a collection of *Entity* object. Each Entity has a name and some properties (attributes and relationships) defined by the developers. The name and type (string, double, etc) of attributes are also defined by the developers.

In Nutrient-Buddy, the Core Data framework is used mainly due to the fact that it can save data on device. Data that relates to user related features: diary information, user goals/limits and nutrient type selection can all be stored on the mobile phone using Core Data. The details of these entities will be explained in the next few sections.

3.5 Diary

3.5.1 Display of Diary

According to *myPlate*, the current nutrition guide published by the USDA Center for Nutrition Policy and Promotion[21], a balance plate of protein, carbohydrate and vitamins should be achieved daily. Therefore the design of nutrient visualization should include

these three nutrients.

A view of the Nutrient diary of the day can be seen in figure 2.3c. In here four ring progress bars can be seen. These rings represent the percentage of energy (in red), carbohydrate (in yellow), protein (in blue) and fat (in purple) consumed during the day compared to the goals/limits user has set (details in section 3.6). Ring shaped progression bar is used in the final design due to the fact that these bars allows over consumption of a certain type of nutrients. This excess amount of intake is hard to visualizes in pie chart suggested in myPlate nutrition guide. Below the ring charts, three progress bar are shown for water, vitamin C and sugar consumption. Further down the table, a list of nutrient intake of different types of nutrient are shown.

3.5.2 Implementation

There are two Core Data Entries that are used here to store user nutrient diary: *Diary* and *Summary*.

Diary Entry. Once a food item is saved by the user, an entry of Core Data *Diary* is saved at the same time. One Diary Entry represents one food item consumed which contains information for *Food Name*, *Date of Saving*, *Amount Consumed*, and the nutrient information about the corresponding food item, which is similar to `foodInfo` structure mentioned in section 3.1 but scaled by the amount logged by the user. Each time the user adds a food item in the Diary, a line of Diary data will be created to the database.

Summary Entry. After adding the food item, the user should go to the Home Screen (figure 2.3c) to view the nutrition diary of the day. The algorithm for calculating the diary for the day is performed every time the Home Screen appears on the screen. Here, a second Core Data Entry is used called *Summary* is used to store the sum of Diary Entries. The Summary therefore contains the date, the amount of water consumed, the amount of energy consumed as well as the amount of nutrient consumed. Figure 3.6 shows the relation between Diary and Summary. Also it shows the `HomeViewController` which has the functions to fetch Diary and calculate Summary.

The logic flow diagram of Summary calculation are shown in the figure 3.7. First of all, the date is found by the `HomeViewController`. Later on, the Diary Entries that contain the corresponding date value are fetched from the database. By summing up nutrient values from every fetched Diary Entries, a new Summary Entry is created. There

is a date attribute on Summary, and also all nutrient information that is found by summing up those in fetch Diary Entries. The program then check if the Summary of the given date exists. If it does, the Entry stored from an earlier time will be replaced by the new one. The blocks are colored in purple to show the interaction with on-device data storage and the yellow ones indicates logic related blocks.

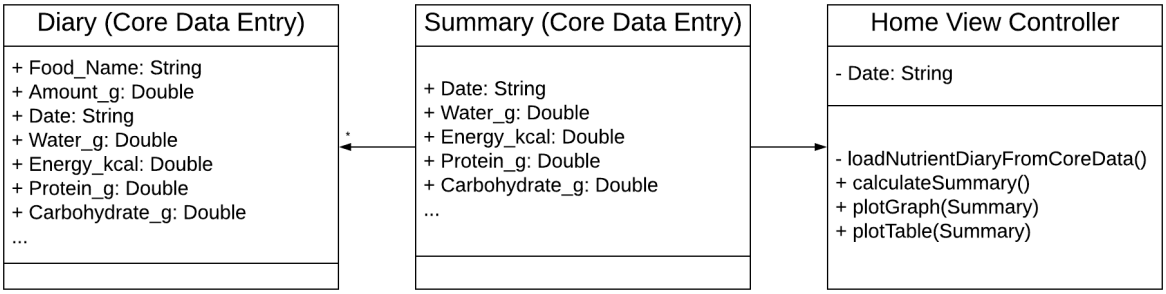


Figure 3.6: Relation Between Diary and Summary

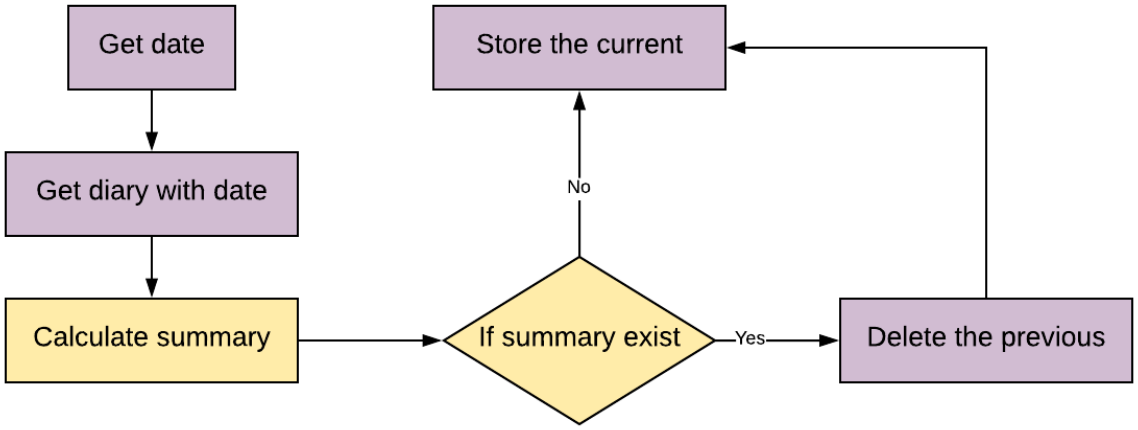


Figure 3.7: Flow Diagram for Nutrient Diary Summary Of The Day

Graph plotting. The design of the ring graphs was taken from the third party [22] where as bar charts are designed as customized UIViews [1] with two filling colors. The percentage display are calculated as shown in the table 3.2. In the calculation, the conversion of carbohydrate to energy is 1:4. The conversion of protein to energy is 1:4. And conversion of fat to energy is 1:9.

[1] An object that manages the content for a rectangular area on the screen[23].

Table 3.2: Consumption Percentage Calculation Details

Calculation Details	
Energy percentage	Consumed energy from the summary over energy goal set by the user
Carbohydrate goal	Energy goal divided by 4 and then multiplied by the fraction of carbohydrate out of three combined[2]
Carbohydrate percentage	Consumed carbohydrate from the summary over carbohydrate goal
Protein goal	Energy goal divided by 4 and then multiplied by the fraction of protein out of three combined
Protein percentage	Consumed protein from the summary over carbohydrate goal
Fat goal	Energy goal divided by 9 and then multiplied by the fraction of fat out of three combined
Fat percentage	Consumed fat from the summary over fat goal

3.6 Settings

3.6.1 Setting Design

User goals/limits. The goals and limits that the user can set are listed below:

- Limit of energy consumption.
- Ratio of carbohydrate, protein and fat consumed.
- Goal of water consumption.
- Goal of vitamin C consumption.
- Limit of sugar consumption.

Initially, all these goals are given by default values. The users can change the goal according to their own needs.

Nutrient selections. Moreover in settings, the users can select what type of nutrient they want to view from the Nutrient Selection Setting Screen previously shown in figure 2.3h. The selected nutrient type plus consumed amount information will be shown in Home Screen (figure 2.3c) and Food Information Screen (figure 2.3g).

3.6.2 Implementation

There are two Core Data Entries that are used here to store user goals/limits and user nutrient type selection: *NutrienToView* and *Goals*.

Goal Entry. Goals contains seven attributes which contains the goal/limit of water, energy, sugar, vitamin C, carbohydrate, protein and fat. On loading of Home Screen (figure 2.3c), the program checks out if an Entry of goals exists in the Core Data and fetch what is stored. If the Core Data for this Entry is empty, default values will be set. The default values are 8 glasses of water goal, 3000kcal of energy limit, 200mg of vitamin C goal and 100g of sugar limit. The ratio goal of carbohydrate, protein and fat is 2:3:5.

NutrientToView Entry. NutrientToView contains only four attributes: type, select, amount and unit. Type are all 78 nutrient types from the database such as energy and water. Select is a binary value that indicates if the user wants to see this nutrient type in the Home Screen. 1 for selected to view and 0 for not selected. Amount and unit temporarily stores the corresponding nutrient type value with its measuring unit for a given item (in Food Information Screen shown in figure 2.3g) or summary (in Home Screen shown in figure 2.3c).

Chapter 4

Testing and Feedback Analysis

4.1 Testing

Various forms of testing were carried out to evaluate Nutrient-Buddy. The details and analysis of which are explained in this section.

4.1.1 Unit and Functional Testing

Unit test and functional testing were used throughout the development of the program to check if certain functions produced desirable results. There is a file in the source code called *debug.swift* which contains a number of global boolean variables. Each of the boolean variables controls testing of one particular functionality in the code, varying from Home Screen display to storing nutrient selection settings. Setting these variables to *true* will result in printing certain information from which the developers can check if the performance of corresponding functions is as expected. Table 4.1 shows information of all these boolean variables with the functionality it controls to test.

Table 4.1: Debug Variables Details

Debug Variables and Functionality Testing	
Debug Variables	Functionality to test
debugHomeView	test if all the functions related to calculation and logging of the nutrient summary are working well
debugLogMeal	test if the log food function on `foodInformationViewController` is working well, if the food item is saved with the selected amount
debugSearch	test on `SearchScreenController`, check if the correct food item name is pulled out given natural language input
debugViewLoading	prints out the time of loading the view
debugPersonalSetting	test on setting view controller, print out the saved personal settings of goals/limits if true
debugNutrientSetting	test on `NutrientTypeSelectionViewController`, print out NutrientToView explained in section 3.6.2 if true

4.1.2 User Testing

Exploratory testing. Exploratory testing was conducted on 5 fourth year students from the Department of Engineering in Cambridge. Exploratory testing is a hands-on approach in which testers who do not know much about the functionality of the software are involved. This testing allows full power of human brain to be brought to bear on finding bugs and verifying functionality without preconceived restrictions[24]. All the test users were able to ascertain the purpose of Nutrient-Buddy within five minutes of use. Moreover, some unexpected bugs were found, due to the confusing display, and fixed at this stage.

Alpha testing. Alpha test was conducted on a group of 14 testers with the intention of finding bugs to refine the software product.

Once the developer enrolls in the *iOS Developer Program*[1], the testing can be carried out on *Itunes Connect*. To start the test, Nutrient-Buddy was first submitted to *App Store*[2]. ITunes Connect is a collection of web-based tools for managing apps sold on the App Store for iPhone. It can be used to submit and manage apps as well as invite users to test with

[1] A Membership with Apple Inc. which includes the right to release, test and sell iOS apps.

[2] A digital distribution platform, developed and maintained by Apple Inc., for mobile apps on its iOS operating system. The store allows users to browse and download apps developed with Apple's iOS software development kit.

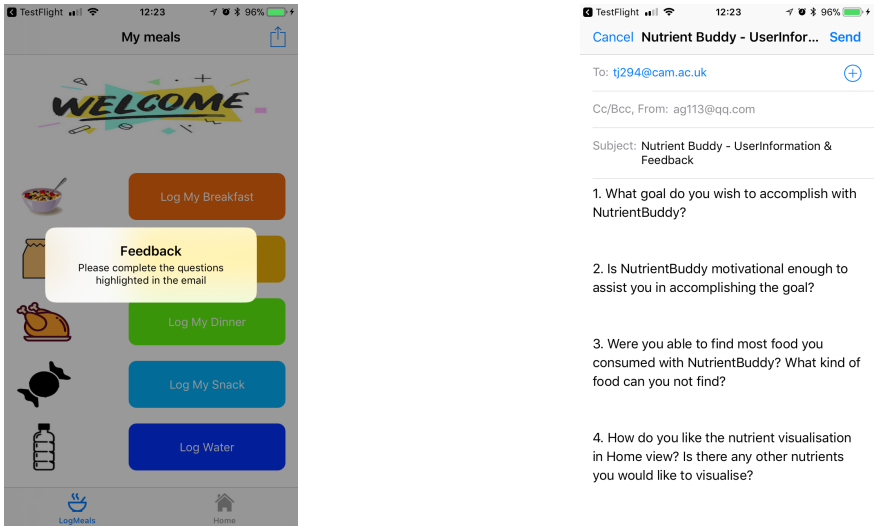
TestFlight [3]. Developers are able to easily create and manage certificates, provisioning profiles, app IDs, and test devices with the use of Itunes Connect.

The alpha testers includes eight engineers, two medicine students, one chemists and two physicists from University of Cambridge, as well as one trader from an investment bank in London. They were instructed to continuously use Nutrient-Buddy for seven days before complete the feedback questionnaire. The feedback questionnaire was designed in order to collect both quantitative and qualitative comments from the testers. The questions in the feedback form includes:

1. What goal do you wish to accomplish with Nutrient-Buddy?
2. Is Nutrient-Buddy motivational enough to assist you in accomplishing the goal?
3. Were you able to find most food you consumed with Nutrient-Buddy? What kind of food can you not find?
4. How do you like the nutrient visualization in Home Screen? Is there any other nutrients you would like to visualize?
5. How was your experience to find food? Is that easy enough to figure out where to go? And did you know that you can set your own nutrient goal?
6. Did you know that you can view more nutrients types?
7. Did you know that you can browse through our whole database?
8. What are the features of Nutrient-Buddy that you do not like?
9. What are the features of Nutrient-Buddy that you like?
10. Please rate from 1-10, your experience with logging meals. Any comments?
11. Please rate from 1-10, the visual of home page. Are the graphics understandable? Any comments?
12. Please rate from 1-10, your experience with Nutrient-Buddy? Any comments?

[3]TestFlight is a tool, developed and maintained by Apple Inc., that helps developers invite users to test apps and collect valuable feedback before release them on the App Store

Questionnaire collection. The completed questionnaires are sent to the developer through emails. At the end of the alpha testing period, testers were asked to press the share button on the top right of Log Food Screen (figure 2.3b). A pop up box will be presented to inform testers to complete the questionnaire (figure 4.1a). Not long after, the email view with questionnaire addressed to the developer will be loaded as shown in figure 4.1b. The email also contains a CSV attachment which comprises the Core Data information stored in the tester's phone by Nutrient-Buddy. Testers are required to answer all the questions in the feedback questionnaire before sending the email to the developer. The author has received feedback emails from ten out of fourteen testers. Out of these ten feedback, five have used Nutrient-Buddy for sufficient times according to the Core Data information in the attachment.



(a) Inform Pop Up Box (b) Questionnaire Email View

Figure 4.1: Send Feedback On Email

4.2 Feedback Analysis

Overall. The user interface was favored by most testers, as it simplifies the user action of Log Food and View Diary mentioned in chapter 2.3. The ring graph on the Home Screen and the fact that users are able to keep a personal record has received a significant amount of positive feedback.

Some testers suggested that aesthetics could be enhanced so that the app looks more appealing. Moreover, the loading and the searching being very slow is a major problem

according to the feedback. Some user suggested to link the app to a number of health advice websites since a few of them were confused about what calorie limit and sugar limit to set.

Launch screen. All test users finds the launch time of Nutrient-Buddy long, however the animated splash screen made the waiting a more tolerable. A number of testers found loading page design one of their favorite part of Nutrient-Buddy.

Log food and Home Screen. There is not much comment on Log Food Screen, but a lot on Home Screen. All alpha testers give positive feedback for the over all design of the graphics on Home Screen and find the number of nutrients displayed sufficient for their needs.

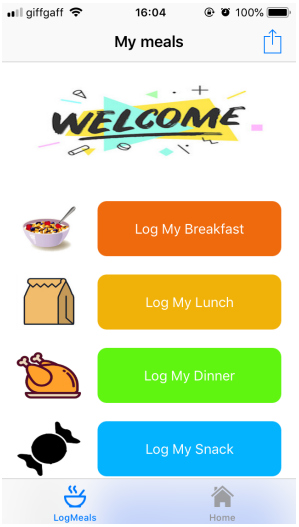
Users observed from exploratory testing are mostly confused with percentage labels that are displayed in color. It makes the whole screen less tidy and also presents repeated information since the all percentages of energy, carbohydrate, protein and fat are drawn on the ring graph. Users found it would be more useful to observe the exact amount of nutrients intake out of their set goals/limits.

One alpha tester who was using iPhone 5 with a 4-inch screen could not view part of the page since this page was designed for iphone 6 with 4.7-inch screen or above. The display on iPhone 5 is shown in figure4.2. There should be five buttons in total in Log Food Screen whereas only four was shown in iPhone 5 screens (figure 4.2a). Moreover, the color labels are squashed in Home Screen as shown figure 4.2b.

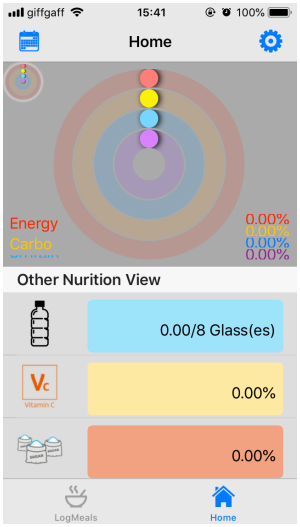
Moreover, a small number of testers found the order of nutrient display on tabulated nutrient information section confusing due to different ordering.

Search experience. All exploratory testers thought Nutrient-Buddy crashed when they press search on the Search Screen because it takes noticeable time to search food item using the Bag-of-words model. The alpha testers felt frustrated and wanted to quit the app because the search engine was very slow.

One user suggested it would be nice to have Google like associated searching function. Since mistyping of words could lead to no results at all even though the desired food item exists in the database. Another tester also suggested to display Search Results ordered



(a) Incomplete Log Food Screen



(b) Incomplete Home Screen

Figure 4.2: Incomplete Display

in terms of popularity. One good suggestions coming from a few testers is to enable the users to add their own food items if it does not exist in the database.

In addition, there are also some comments on the database category search shown in figure 3.2. Very few users used this search method and one user found this function quite pointless to have.

Logging food. Most of the users found logging food in terms of grams hard and annoying since sometimes people are unaware of the weight but quantity of the items. For example, if the user eats fruits like apples and oranges, its better to input in terms of quantities consumed, not in grams because that is hard to find out. Also coffee is usually measured in terms of cups and juice is conventionally measured in milliliters. So giving each food relevant units for more convenient inputs could enhance the user experience. Also the weight scale has a maximum of 200 gram, limiting the user to input larger amounts. Since the amount slider has a accuracy to two significant figures, it is difficult for the user to scroll to the exact amount they want. Moreover, some users preferred adding a rough estimate of the food consumed.

User feedback rating. At the end of the alpha testing questionnaire, the testers are asked to rate the experience they had with finding and logging the food, the UI design as well as their overall experience with Nutrient-Buddy. They were all asked to give a rating out of ten how much they enjoyed using the app. Table 4.2 shows the mean and standard

deviation of all the user ratings on these three aspects. Figure 4.3 shows the comparison of the three mean values with error bars where error bars are determined by standard deviation. It can be seen that the means vary similarly for all three areas.

Overall, testers in general found the app easy to navigate but the experience of searching for food negatively affected their overall experience. An improvement of database, searching function as well as amount logging options is essential.

Table 4.2: User Feedback Rating

User Feedback Rating		
	Mean	Standard Deviation
Logging Meals	5.5	0.95
UI	7.6	1.07
Overall Experience	6.85	0.94

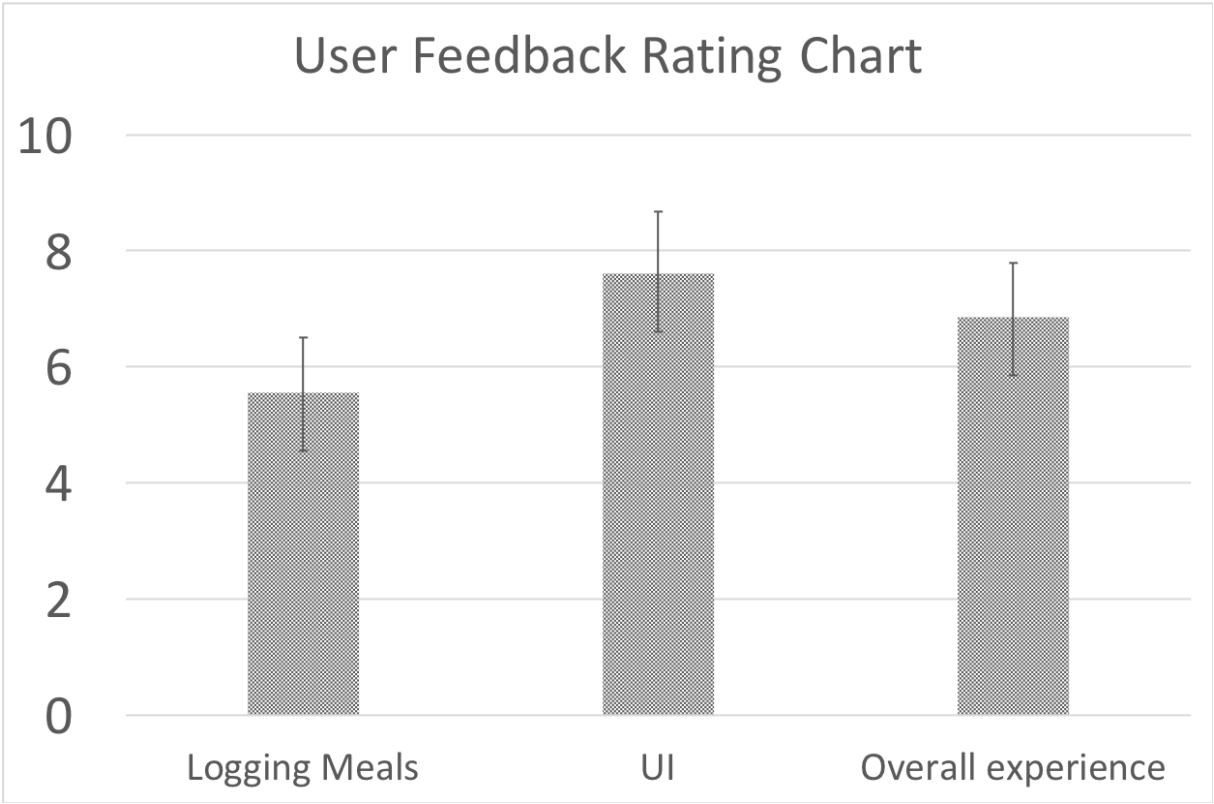


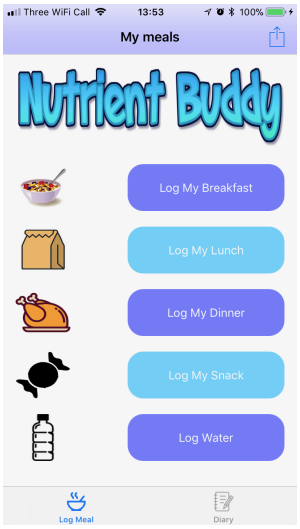
Figure 4.3: User Feedback Rating

Chapter 5

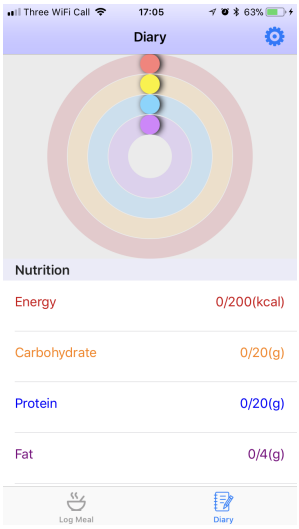
Improvement Phase

5.1 Color Scheme

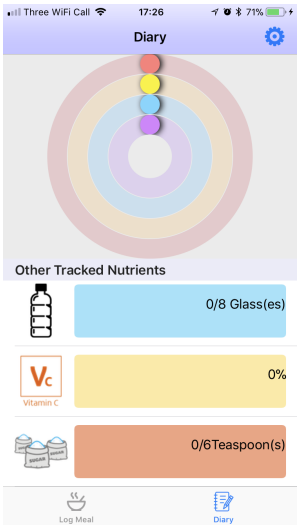
In accordance with the Human Interface Guidelines on the Apple developer website, [25] the color scheme of Nutrient-Buddy was improved to two major colors: sky blue and blue purple, and four minor colors: dark red, orange, blue and purple in the diary ring graph. Some of the major user interfaces are shown in figure 5.6b. The menu button colors in Log Food Screen (figure 2.3b) were changed from five rainbow colors (red, orange, green, blue, purple) to two alternating colors (light blue and blue purple) in figure 5.1a. The altered color scheme sets a healthy mood for the professional looking app; the initial grey color for Home Screen was associated with negative mood for the users (figure 2.3c). Changing the overall background color to white and adding a blue navigation bar makes the screen appear orderly. In addition, the title of the Home Screen was later changed to "Diary" to clarify that the screen visualizes the nutrient diary. A similarly color and layout theme was applied to the database view as shown in figure 5.1d and figure 5.1e.



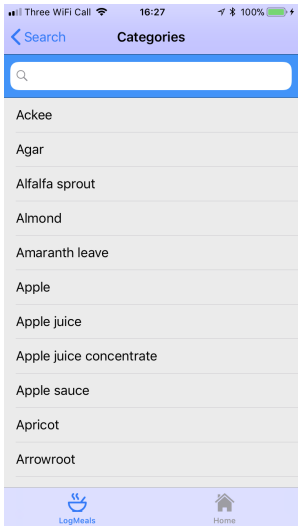
(a) Log Food Screen



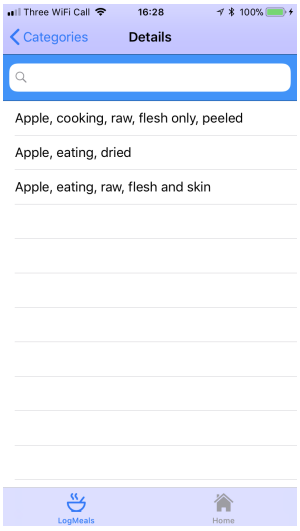
(b) Home Screen



(c) Home Screen On Scroll



(d) Improved Categories Database

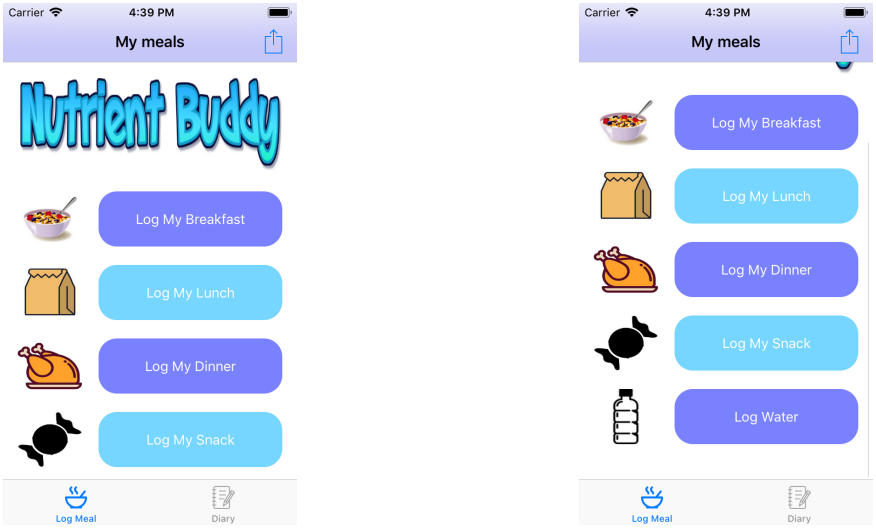


(e) Improved Category Item Database

Figure 5.1: Improved Color Scheme for Nutrient-Buddy

5.2 Log Food Screen

To address the trouble that iPhone 5 users had on displaying all the buttons on log food screen, a *UIScrollView*[1] is added. The buttons are all placed on top of the scroll view. With the new change, the users are able to scroll down the page to select buttons. Figure 5.2a shows the initial log food page once Nutrient-Buddy is loaded after improvement. And figure 5.2b shows how the user with can scroll down to fully view and interact with the page.



(a) iPhone 5 Log Food Screen (b) iPhone 5 Log Food Screen on Scroll

Figure 5.2: Improved Log Food Screens For iPhone 5

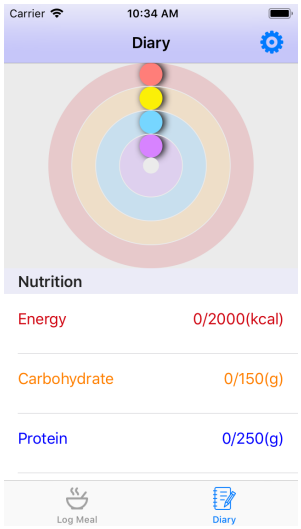
5.3 Home Screen

An improved view of Home Screen is shown in figure 5.1b and figure 5.1c compared to figure 2.3c. The colored labels on the ring graph were dismissed and the labels on the table cell that display nutrient diary details were color coded. The labels for percentages of consumption versus goals/limits were also deleted. Instead, the table cells below shows the ring graph show the fraction of amount consumed with the user set goal/limit. Moreover, the order of nutrients in the display has been fixed to energy, carbohydrate, protein, fat, water, vitamin C and sugar. The ring graph and bar chart follow the same order of nutrients (figure 5.1b and figure 5.1c).

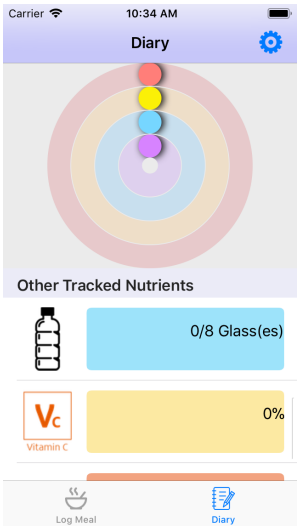
[1]A view that allows the users to scroll and zoom the conponent on the screen[26].

The improved view presented in figure 5.1b is easily operable and understandable. The following nutrient consumption are represented in different colors: red for energy consumption, orange for carbohydrate, blue for protein, and purple for fat. The consumption of protein is represented by blue color and fat is in purple. The table below the ring graphs shows the current consumption of different nutrients and the goal/limits for consumption set by the user.

The improved display also corrected the occlusion problem for iPhone 5 users as shown below in figure 5.3a and 5.3b (scrolled view).



(a) iPhone 5 Home Screen



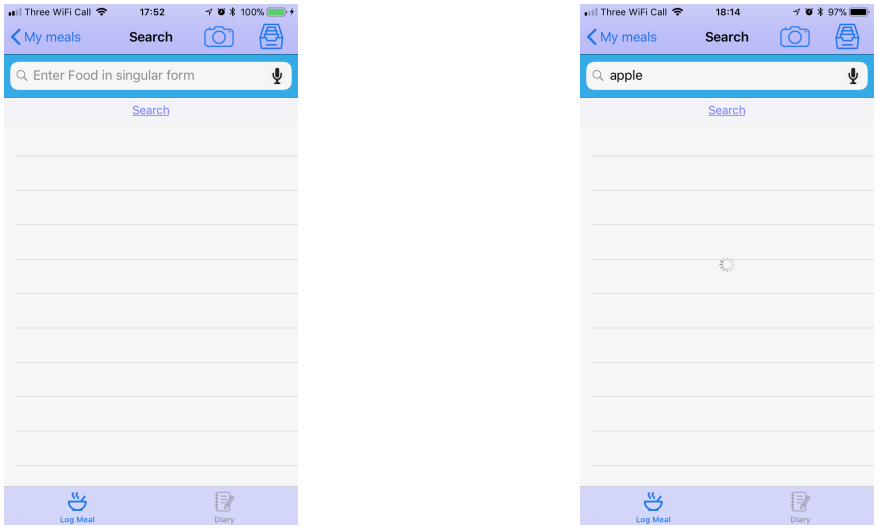
(b) iPhone 5 Home Screen on Scroll

5.4 Search Screen

The Search Screen (seen in figure 5.4a) for Nutrient-Buddy has been simplified from the old view in figure 2.3e. The text field where instructions are displayed are deleted due to the fact that users hardly ever found them useful and they are more familiar with a screen with just a search bar.

To address the problem of long loading times, an activity indicator - to symbolize that a task is in progress - was added to the view during Bag-of-words search[27]. The indicator screen is shown in figure5.4b during the wait, user interactions are ignored until search is complete. Multi-threading technique is used here to implement the waiting screen. The indicator is controlled by the main thread whereas the Bag-of-Words model search function runs on a background thread.

The waiting screen informs the user to wait for the search and also disguise the length of time. It significantly improved user experience, since the waiting is made more enjoyable and understandable.



(a) Search Screen (b) Search Indicator Screen

Figure 5.4: Improved Search Screens

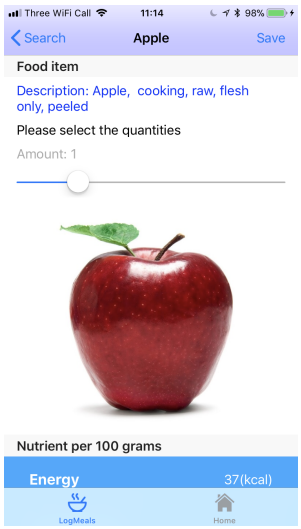
5.5 Food Information Screen

Another issue arose from Nutrient-Buddy's Amount Selector. Previously, the amount of food consumed had to be specified by the user in grams. But a number of testers were unhappy with the constrains applied due to the use of single unit. The improved version of the app allows the user to specify the amount not only in grams, but also in terms of quantity, cup, tablespoon and volume.

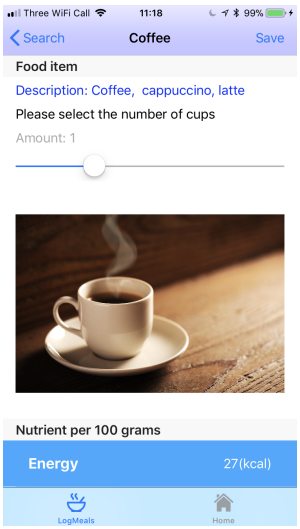
To resolve this problem, the database has been modified manually. Four more columns in the CSV file were added, meaning that four more entries for the sturct `FoodInfo` mentioned in section 3.1 were added. The added entries are `Quantity Weight`, `Cup Weight`, `Tablespoon Weight` and `Liquid Volume` and they are all of data structure type Double. These values represent the weight in grams there are in one quantity, cup, tablespoon, or volume of the corresponding food item.

In the improved version, if the user is searching for countable food items such as apple and orange, the display of the Food Information Screen will be shown as figure 5.5a.

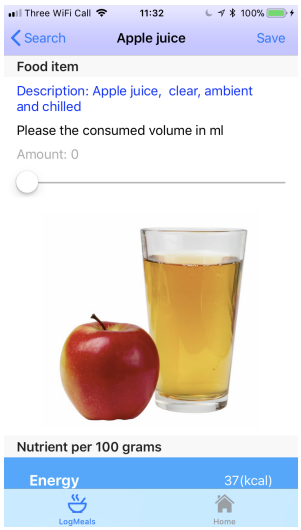
The user can select quantity from the slider with a range of 0 to 5. If the user wants to log food such as coffee, they will be guided to select the number of cups (from zero to five) consumed as shown in figure 5.5b. Figure 5.5c depicts how the user can log amount of juice consumed, measured in milliliters, from 0 to 300 ml. Other food items such as BBQ sauce and sugar can be quantified using the number of tablespoons from 0 to 5 (figure 5.5d). All other types of food items will still be logged in grams.



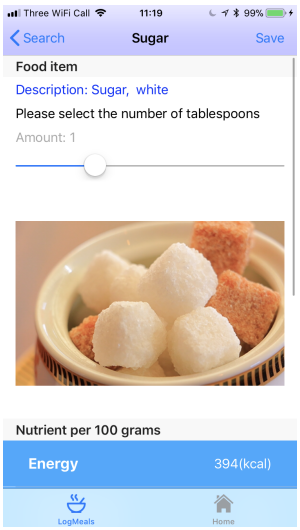
(a) Log On Food With Quantity



(b) Log On Food With Cups



(c) Log On Food With Volume



(d) Log On Food With Tablespoon

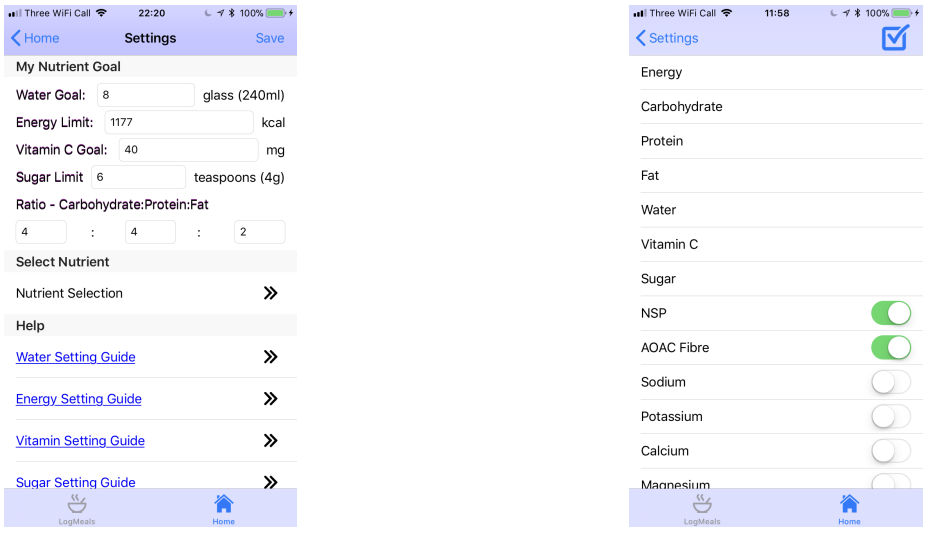
Figure 5.5: Improved Food Information View

5.6 Setting Screen

Goal setting. Since a number of users complained that there is no link or guidance for setting a healthy goals/limits, a new adaption of the setting view is developed as shown in figure 5.6a. Under the "help" section, five links are attached where the users can get guidance to set daily water goal, energy limit, vitamin C goal, sugar limit and ratio consumed for carbohydrate, protein and fat.

The default values for goals/limits were set. By default, the user has a daily goal of eight glasses of water, 40 milligrams of vitamin C, and a carbohydrate, protein, fat ratio of 4:4:2. There is also a limit of energy intake of 2000 kcal and six teaspoon of sugar. The unit for measuring sugar has been changed from gram to teaspoon to improve user experience. The display of the text boxes are change from grey in the previous version to black, indicating the users that these values can be changed.

Nutrient selection setting. Since there are six types of nutrient information that are displayed graphs in Home Screen, these six nutrient should be shown in the view all the time. The selection of these nutrients is disabled

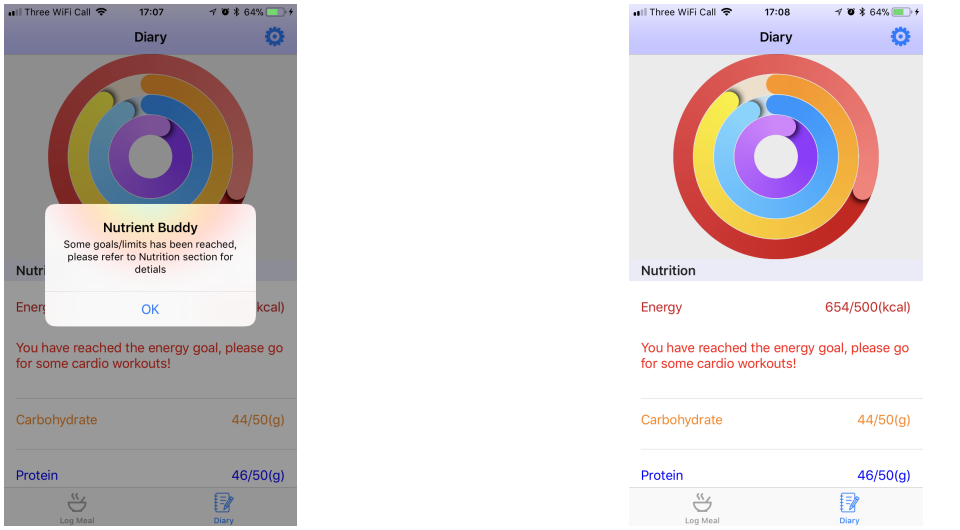


(a) Setting Screen (b) Nutrient Selection Screen

Figure 5.6: Improved Nutrient-Buddy Setting Screen

5.7 Notification

A notification scheme is created to motivate user and give them suggestions on nutrient intake. When one or more goals/limits are reached, the user will see an alert box on the Home Screen as shown in figure 5.7a. This notification box will tell the user to scroll down to check what limit or goal they have reached. Figure 5.7b shows the suggestions when the users reaches certain goals/limits. The are listed below in table:



(a) Notification (b) Notification Suggestions

Figure 5.7: Nutrient-Buddy Notification Screen

Table 5.1: Suggestion List

Suggestion List	
Reached goal/limit	Suggestions
Energy	You have reached the energy goal, please go for some cardio exercise.
Carbohydrate	You have reached the carbohydrate limit, you can do some cardio or increase protein consumption.
Protein	You have reached the protein limit, well done! Now you can consume more carbohydrate or fat to achieve the goal.
Fat	You have reached the fat limit, you can do some cardio or increase protein consumption.
Water	Well done, you have had enough water today.
Sugar	You have reached the sugar limit, you have to be careful with carbohydate consumption today.
Vitamin C	You have reached the vitamin C goal, well done.

Chapter 6

Discussion and Future Work

6.1 Discussion on Current Version

Nutrient-Buddy has been improved noticeably after testing. The pastel color scheme makes it more professional and refreshing. Also the user interface has been simplified significantly from the original design to improve user experience. There are some major problems for Nutrient-Buddy such as the loading time and searching time being very long. An effective fix has been implemented to negate this problem: the animated loading screen and activity indicator respectively. The focus of the app was on user interface and the exploration of other forms of food logging (text search, speech search, image search). Once the backend search and database had been implemented in a user friendly manner, additional extension to the app are non trivial. In the following section, some possible extensions are explained in details.

6.2 Calendar Diary View

In the current version of Nutrient-Buddy, the user is only able to view nutrient diary of the current date. An advanced development would be enabling the user to access the nutrient diary from a calendar as shown in figure 6.1. The user will have access to the nutrient information from the previous days in their diary. Since the current version of Nutrient-Buddy saves the diary everyday on the disk, only the user interface needs to be developed to display the calendar as well as the diary of the day.

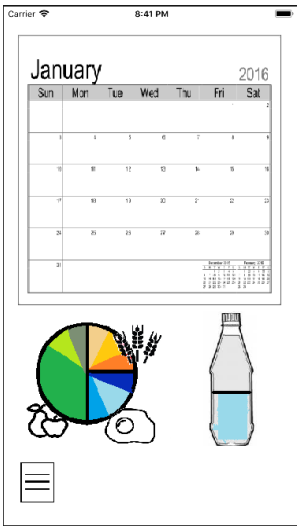


Figure 6.1: Diary View Screen

6.3 Refined Database

All ten testers in the alpha test found the database difficult to use considering there are many confounding options available. For example, if "apple" is searched, most testers expected to find information on the fruit apple. However, the database presented them a list of items such as "*apple, cooking, baked with sugar, flesh only*" and "*apple, cooking, baked with sugar, flesh only, weighted with skin*". Most of these were irrelevant to what user wanted. Only the fifth and the sixth entries: "*apple, cooking, raw, flesh only, peeled*" and "*apple, cooking, raw, flesh only, weighted with skin and core*" were viable and popular options among the testers. And even these two Search Results were difficult to distinguish from each other.

Furthermore, the variety of food items in the database is very limited since only British household cuisine was involved in the database. Despite the fact that all testers are living in the UK, those who come from countries like China, Czech Republic and Vietnam found it hard to log what they have eaten since the food item does not exist in the database. Last but not least, a number of users suggests that soft drinks such as Fanta and Coca Cola etc should be added to the database since they could make up significant amount of certain nutrient intake such as sugar. Overall, a new online database should be used to replace the current one just so Nutrient-Buddy becomes a more useful tool for people from all around the world. Having an online database can also significantly reduce the loading time.

One possible solution for the database would be to make use of the food nutrient database

that Google is using - USDA Food Composition Databases published by United States Department of Agriculture Agricultural Research Service. This database is much larger and contains a much wider range of food items. This online database also provides an application programming interface (API) that is designed to assist application developers who are trying to utilize food nutrient information in their applications. This API provides food reports that contains lists of nutrient values for specified food item.

6.4 Refined Searching Engine

One user suggested that when he mistyped a word of the food item to search, he was unable to find anything from the database. He suggested to introduce associative search. Also, some testers raised that adding ranking for popular items would improve the experience significantly. Therefore, looking for a way to utilize Google to reach the USDA database would be a good direction. This would allow reading nutrient information from website source-code. Moreover, Nutrient-Buddy is designed to be integrated with a image search engine that can detect the type and amount of food consumed from images in the future. The image search will significantly save time and effort for the users.

6.5 Amount Converter

Despite the Amount Converter implemented in section 5.5, the app is still non user friendly because some users might prefer logging and estimate of quantities consumed, whereas others might prefer more accurate measurement. To resolve the disparity between the users, a button or entrance "specify amount" could be implemented to enable the user to enter accurate or estimated amount. Once this button is clicked, an extra dialog box would allow the users to specify the amount in grams, quantity or cups etc. And then the user who favor accurate measurements could input the value either in a text box or on a picker view[1].

6.6 Goal Setting

One inconvenient feature of Nutrient-Buddy is the fact that the user has to set their energy, sugar limit, water, vitamin C and ratio goal in text boxes. Due to the limitation of Swift, the text box will only read the user's input once the user has finished editing (pressed

[1] A view that uses a spinning-wheel or slot-machine metaphor to show one or more sets of values.

"return" on keyboard) or has moves on to the next. Therefore after altering all the goal values, the user has to press return on the keyboard to indicate the end of editing event before pressing "save" button at the right hand top corner.

This is not a easily operable feature. An quick improvement would be to delete the "save" button and move the code relating to saving function to the delegate that detects the end of editing of a text box. With this change, it becomes more intuitive for the users to press "return" button on the keyboard at the end of editing text box values.

A different way of taking user input should be considered. For future development, a pop up picker box could be included. An ideal user interaction would be performed with the aid of picker view for customized input. A good example is the goal setting on iOS application CalorieMama developed by Azumio Inc. As shown in figure 6.2. This app guides the user to enter gender, weight and height information using picker view and generate user energy goal in accordance with the input. This kinds of set up is highly adaptable for users since the they do not have to worry about checking the calorie limits themselves. The above description could also be applied for goals/ limits settings for water, sugar, vitamin C and ratio.

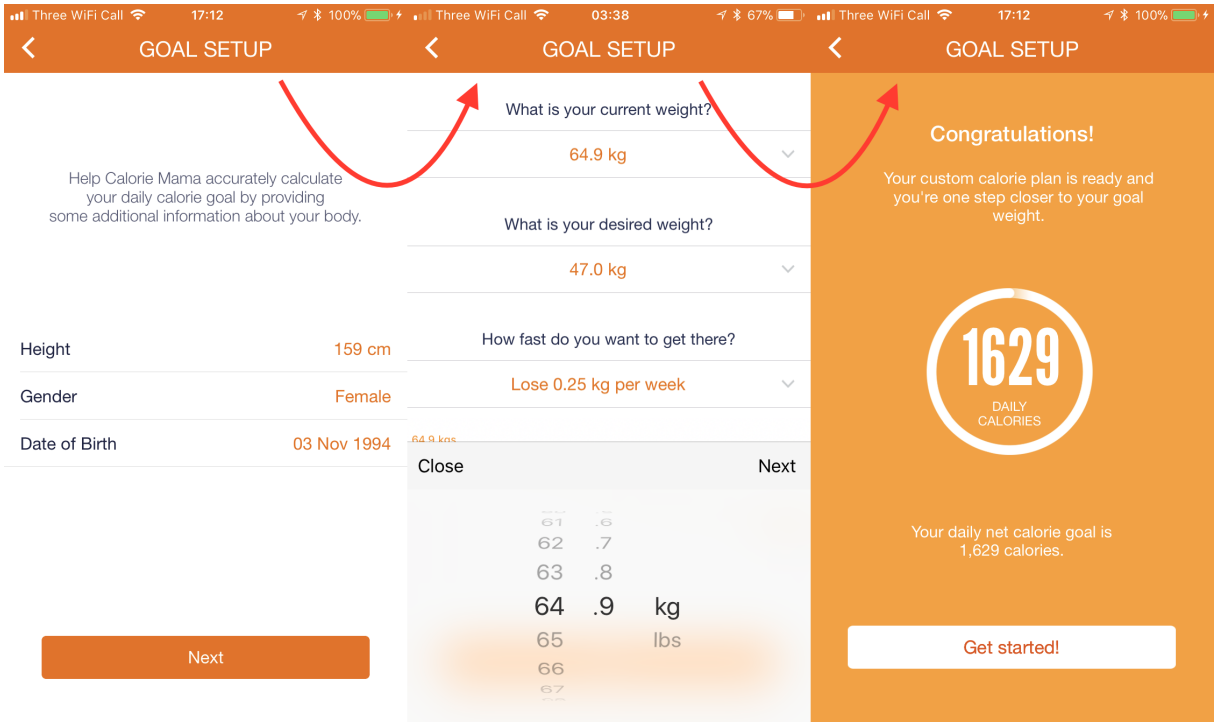


Figure 6.2: Setting goals on CalorieMama

6.7 Motivation Scheme

To motivate users to keep a good eating habit, a more user friendly motivation scheme should be invented to encourage user to reach the goal as well as not go past the limit everyday. Nutrient-Buddy should be able to send notifications to the user once certain goal or limit is reached. The user should be awarded for reaching goals and should be penalized for surpassing limits. And a scoring system should be implemented to calculate the score they get in one week time. The user should be able to log in to the app via social media such as Facebook so that they can compete with their own friend on how healthy they eat everyday. This app could be extended to allow user to add friends to compete with and view their statistics.

Bibliography

[1] Marion Nestle. *Food politics: How the food industry influences nutrition and health*, volume 3. Univ of California Press, 2013.

[2] The risks of poor nutrition. `http://www.sahealth.sa.gov.au/wps/wcm/connect/public+content/sa+health+internet`. Accessed: 21-05-2018.

[3] Michael S Donaldson. Nutrition and cancer: a review of the evidence for an anti-cancer diet. *Nutrition journal*, 3(1):19, 2004.

[4] Marion Eugene Ensminger and Audrey H Ensminger. *Foods & Nutrition Encyclopedia, Two Volume Set*. CRC press, 1993.

[5] Kerry Shih-Ping Chang, Catalina M Danis, and Robert G Farrell. Lunch line: using public displays and mobile devices to encourage healthy eating in an organization. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 823–834. ACM, 2014.

[6] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy. Im2calories: towards an automated mobile vision food diary. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1233–1241, 2015.

[7] MyFitnessPal: Free Calorie Counter, Diet & Exercise Journal. `https://www.myfitnesspal.com`. Accessed: 18-05-2018.

[8] Calorie Mama AI - Food Image Recognition and Calorie Counter using Deep Learning. `http://www.caloriemama.ai`. Accessed: 18-05-2018.

[9] Longqi Yang, Cheng-Kang Hsieh, Hongjian Yang, John P Pollak, Nicola Dell, Serge Belongie, Curtis Cole, and Deborah Estrin. Yum-me: a personalized nutrient-based

meal recommender system. *ACM Transactions on Information Systems (TOIS)*, 36(1):7, 2017.

[10] Swift 4. `https://developer.apple.com/swift/`. Accessed: 11-05-2018.

[11] Xcode. `https://developer.apple.com/xcode/`. Accessed: 24-05-2018.

[12] Tabacu Iulia-Maria and Horia Ciocarlie. Best practices in iphone programming: Model-view-controller architecturecarousel component development. In *EUROCON-International Conference on Computer as a Tool (EUROCON), 2011 IEEE*, pages 1–4. IEEE, 2011.

[13] Trygve Reenskaug and James O Coplien. The dci architecture: A new vision of object-oriented programming. *An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html*, 2009.

[14] Rui Peres. Model-view-controller (MVC) in iOS: A modern approach. `https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach`. Accessed: 10-05-2018.

[15] McCance and Widdowsons The Composition of Foods Integrated Dataset 2015 User guide, March 2015.

[16] Speech - Apple Developer Documentation. `https://developer.apple.com/documentation/speech`. Accessed: 12-05-2018.

[17] NSLinguisticTagger - Apple Developer Documentation. `https://developer.apple.com/documentation/foundation/nslinguistictagger`. Accessed: 12-05-2018.

[18] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[19] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606, 2009.

[20] Core Data - Apple Developer Documentation. `https://developer.apple.com/documentation/coredata`. Accessed: 24-05-2018.

[21] Choose myplate. `https://www.choosemyplate.gov/MyPlate`. Accessed: 13-05-2018.

[22] Ring Progress View. `https://github.com/maxkonovalov/MKRingProgressView`. Accessed: 13-05-2018.

[23] UIView - Apple Developer Documentation. https://developer.apple.com/documentation/uikit/uiview?changes=_4. Accessed: 25-05-2018.

[24] James A Whittaker. *Exploratory software testing: tips, tricks, tours, and techniques to guide test design*. Pearson Education, 2009.

[25] Human Interface Guidelines - Apple Developer Documentation. https://developer.apple.com/ios/human-interface-guidelines/visual-design/color/. Accessed: 18-05-2018.

[26] UIScrollView - Apple Developer Documentation. https://developer.apple.com/documentation/uikit/uiscrollview. Accessed: 12-05-2018.

[27] UIActivityIndicatorView - Apple Developer Documentation. https://developer.apple.com/documentation/uikit/uiactivityindicatorview?changes=_5. Accessed: 20-05-2018.