# SFWR 4G06B - Draft System Design - Rev 1

Group 9
Gundeep Kanwal 400015267
Ivan Bauer 001418765
Yousaf Shaheen 400026476
Scott Williams 400031554
Lucas Shanks 400029943

March 13, 2020

# Contents

# List of Tables

## List of Figures

## Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 0.0 | 01/06/2020 | GK, IB, YS, SW, LS | Initial revision of the System Design document |
| 1.0 | 03/13/2020 | GK, IB, YS, SW, LS | Modified Controlled/Monitored Variables, Component Trace-ability matrix Diagram, Class Diagram, Normal Operation, Undesired Event Handling, and Likely and Unlikely Change |

## 1 Purpose

This project will involve developing and designing a mobile application that is capable of retrieving and effectively visualizing nutritional information of food to the user, enabling the user to make more health conscious decisions regarding their food selection. This information shall be used to construct a food diary which the user will use in order to track their food intake on a day-to-day basis. The application will provide a high level visualization of a user's fitness and nutritional habits through an interactive timeline. Ultimately, the overall objective is to help the user lead a healthier lifestyle.

## 2 Scope

This project shall primarily be centered on enabling the user to understand the nutritional content of the food that they are eating. The system shall be able to figure out the nutritional information of the food that the user captures with their picture, or scans with a bar-code. It shall be able to summarize a day's nutritional intake and visualize that information to the user. It shall also incorporate

In-scope items of functionality for the system include the following:

- Account system with various settings that can be personalised.
- Daily food diary which tracks nutritional information.
- Camera feature which takes pictures of a single food item.
- Ability to recognize different types of food items of a single sample.
- Ability to display nutritional content of the food from a picture.
- Transference of nutritional information from picture/bar-code to diary.
- Visualization of the fitness progress of a user.
- Visualization of the nutritional progress of a user.
- Game system that enables the user to compete with others.
- Camera feature that detects bar-code of food or drink.

Additionally, the following items are deemed to be out of scope:

- Determining nutritional contents of a drink through a picture of the item.
- Viewing Fitness Scores of users outside their immediate group.
- Determining every nutritional value in a given plate of food.

## 3 Context Diagram

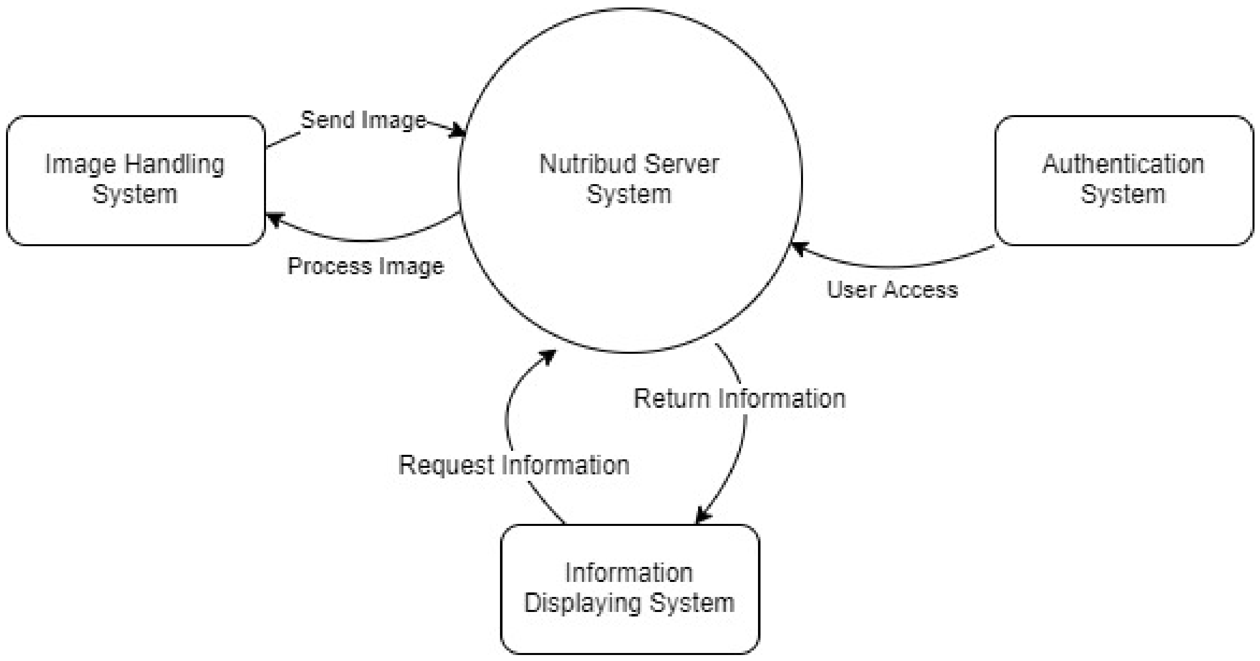


Figure 1: Context Diagram

## 4 Component Diagram

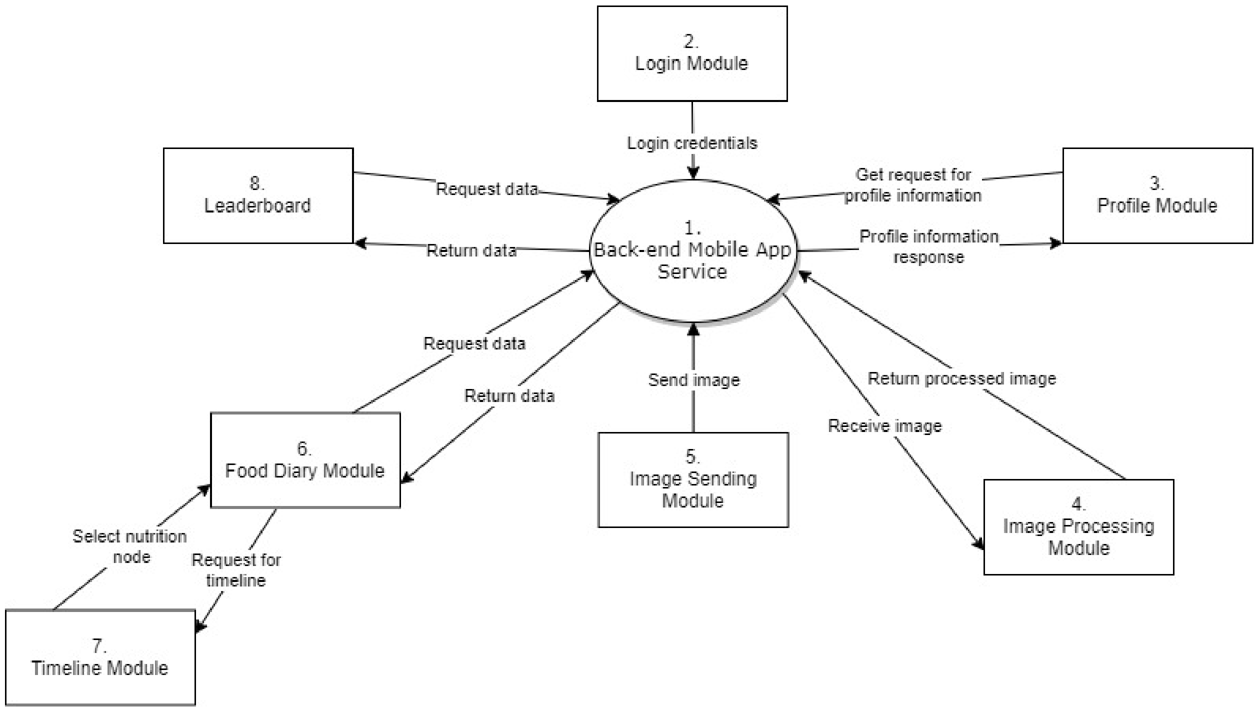


Figure 2: Component Diagram

## 5 Assumptions

| | |
|---|---|
| **A1** | **The user's mobile device will have a camera that is in working condition.** |
| *Rationale* | For the application to function properly, the user must be able to take pictures with their device in order to receive feedback from the system. |
| **A2** | **The user will take pictures under appropriately bright lighting.** |
| *Rationale* | For the application to function properly, Nutribud must be able to identify images, and without proper lighting Nutribud's image processing system will not operate correctly. |
| **A3** | **The user will take pictures in a manner such that the image takes up the majority of the display, but fits all components.** |
| *Rationale* | Having the camera too close to the food, or too far away from the food will affect how Nutribud processes the image, since we are comparing the image to a machine learning algorithm that learned from many images that are all similarly distanced. |
| **A4** | **The user's mobile device will have an active internet connection.** |
| *Rationale* | Without internet the application on the user's phone will be unable to access our cloud servers in order handle requests that are critical to the application. |

# 6 System Variables

## 6.1 Monitored Variables

| Requirement | Variable | Unit | Description |
|---|---|---|---|
| FR1 | food_type | N/A | The food type that is being placed on the plate |
| FR5 | barcode_upc | integer | The UPC barcode of the food product. |
| FR14 | manual_entry | word | The worded user input for the food type. |
| FR15 | user_weight | lbs/kgs/st | The recorded weight of the user. |
| FR12 | cal_goal | Calories (kcal) | The recorded number of calories necessary for the user to achieve nutrition projections, as documented by the user. |
| FR12 | protein_goal | g | The recorded number of protein necessary for the user to achieve nutrition projections, as documented by the user. |
| FR12 | carb_goal | g | The recorded number of carbohydrates necessary for the user to achieve nutrition projections, as documented by the user. |
| FR12 | fats_goal | g | The recorded number of fat necessary for the user to achieve nutrition projections, as documented by the user. |
| FR13 | calories_burned | Calories (kcal) | The amount of calories a user burns through their physical activity in a day. |
| FR4 | calories_left | Calories (kcal) | The amount of calories the user has left to eat, or has over consumed relative to their calorie goal. |
| FR10 | leaderboard | List | A leader board which is created from a user's score, which is presented on GainRivals. |

Table 2: NutriBud Monitored Variables Table

## 6.2 Controlled Variables

| Requirement | Variable | Unit | Description |
|---|---|---|---|
| FR4 | tot_cal | Calories (kcal) | Defined as the amount of energy needed to raise the temperature of 1 kilogram of water by 1 degree Celsius. Provided as an energy measurement for food. |
| FR4 | tot_fat | g | The total amount of fat that can be found within the serving size of the dish. |
| FR4 | sat_fat | g | The amount of saturated fats contained within the serving size of food. |
| FR4 | trans_fat | g | The amount of trans fat in the food serving size. |
| FR4 | chol | mg | The amount of cholesterol within the food serving size. |
| FR4 | tot_carbs | g | The total amount of carbohydrates found in grams. |
| FR4 | diet_fib | g | The amount of dietary fiber found in the serving size, in grams. |
| FR4 | sugar | g | The total amount of sugars found in the serving size, in grams. |
| FR4 | sodium | g | The amount of sodium found in the serving size, in grams. |
| FR4 | protien | g | The amount of protein found in the serving size, in grams. |
| FR4 | vit_a | % | The percentage of Daily Value the serving size offers Vitamin A to the user. |
| FR4 | vit_c | % | The Daily Value percentage of Vitamin C to the user according to their restrictions. |
| FR4 | calcium | % | The Daily Value of Calcium held within the serving size to the user. |
| FR4 | iron | % | The Daily Value of Iron according to the user, in percentage. |

Table 3: NutriBud Controlled Variables Table

## 7 Constants

Due to the dynamic nature of the application, it has been determined that there will be no constants for the system.

## 8 Behaviour Overview

1. **User Login Page Module**: Given login credentials, will authenticate the user with the system to track account details. Also contains a registration function
2. **Profile Module**: Contains user profile information (ex. height, weight, email, password, desired weights)
3. **Backend Mobile App Service**: Will route different components of the system to store/retrieve data from the backend
4. **Image Sending Module**: Takes a picture and uploads to the backend service module for processing
5. **Image Processing Module**: Processes the uploaded image and returns the results to the user
6. **IBM Watson Service**: Machine learning IBM product to identify the food items in an image
7. **Food Diary Module**: Shows the nutrition requirements and goals for the selected date
8. **Timeline Module**: Provides a calendar view on nutritional data for past days

## 9 Component Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Login Module | FR8<br>FR15 |

Table 4: Login Page Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Profile Module | FR4<br>FR8<br>FR12<br>FR13<br>FR15 |

Table 5: Profile Page Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Backend Mobile App Service Module | FR1 FR2 FR3 FR5 FR8 FR9 FR11 FR14 |

Table 6: Backend Mobile App Service Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Image Sending Module | FR1 FR2 FR3 FR9 |

Table 7: Image Sending Page Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Image Processing Module | FR2 FR3 FR9 |

Table 8: Image Processing Module Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| IBM Watson Service | FR2 FR3 FR9 FR12 |

Table 9: IBM Watson Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Food Diary Module | FR3 FR4 FR5 FR11 |

Table 10: Food Diary Page Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Timeline Module | FR6<br>FR7<br>FR8<br>FR12 |

Table 11: Timeline Module Trace-ability

| Component Module | Functional And Non-Functional Requirements |
|---|---|
| Leaderboard Module | FR10 |

Table 12: Leaderboard Page Trace-ability

## 10 Component Overview

### 10.1 Login Module

#### 10.1.1 Description

This module will be used to login with an unique user account to record user information over a period of time while using the application. Here also contains a user registration option

#### 10.1.2 Inputs and Outputs

**Inputs**: User input defining

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *username* | String User Input | 10 chars | N/A | User Login Info |
| *password* | String User Input | 20 chars | N/A | User Login Info |

**Outputs**: To be displayed to user

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *failureMessage* | String | 10 chars | N/A | Failure message if incorrect information |
| *gotoPage* | action | N/A | N/A | Sends user to homepage upon authentication success |

#### 10.1.3 Exception Handling

| **Input Name** | **Input Type** | **Exception** | **Exception Handling** |
|---|---|---|---|
| *username* | String User Input | invalid character(s) | Input Regulation |
| *password* | String User Input | invalid character(s) | Input Regulation |

#### 10.1.4 Timing Constraints

Time constraints vary on the server response time to authenticate a user sending a response within t_success time

#### 10.1.5 Initialization

Upon application startup, the user will be prompted to create an account or login to an existing account. User authentication is required to use the application and food diary history

### 10.2 Profile Module

#### 10.2.1 Description

This module will be used to store user profile information while using the application. The user is able to add or update any existing information while logged into the application

#### 10.2.2 Inputs and Outputs

**Inputs**: User input defining

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *height* | Float User Input | [0, 1000] | m | User height |
| *weight* | Float User Input | [0, 1000] | kg | User weight |
| *email* | User String Input | 30 chars | N/A | User email |
| *password* | User String Input | 20 chars | N/A | User password |
| *desiredWeight* | Float User Input | [0, 1000] | m | User desired weight |

**Outputs**: To be displayed to user

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *height* | Float | N/A | m | User height |
| *weight* | Float | N/A | kg | User weight |
| *email* | String | N/A | N/A | User email |
| *password* | String | N/A | N/A | User password |
| *desiredWeight* | Float | N/A | kg | User desired weight |

#### 10.2.3 Exception Handling

| **Input Name** | **Input Type** | **Exception** | **Exception Handling** |
|---|---|---|---|
| *height* | Float User Input | height of type string | Input Regulation |
| *weight* | Float User Input | weight of type string | Input Regulation |
| *desiredWeight* | Float User Input | desiredWeight of type string | Input Regulation |
| *username* | String User Input | invalid character(s) | Input Regulation |
| *password* | String User Input | invalid character(s) | Input Regulation |

#### 10.2.4 Timing Constraints

Time constraints vary on the server response time to update a user profile within t_success time

#### 10.2.5 Initialization

Upon opening up user profile details, the user is able to update or add to existing information stored with the account

## 10.3 Back-end Mobile App Service Module

### 10.3.1 Description

This module will be used to route different components of the system to store/retrieve data from the backend

### 10.3.2 Inputs and Outputs

**Inputs**: User input defining

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *HttpReq* | HTTP Request | N/A | N/A | N/A |
| *username* | String | 10 chars | N/A | N/A |
| *password* | HashCode | N/A | N/A | Check for success result |

**Outputs**: To be displayed to user

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *loginStatus* | String | 10 chars | N/A | Login Success or Fail |
| *Httpresult* | String | 10 chars | N/A | N/A |

### 10.3.3 Exception Handling

| **Input Name** | **Input Type** | **Exception** | **Exception Handling** |
|---|---|---|---|
| *HttpReq* | HTTP Request | HttpReq formatting error | Request Validation |
| *username* | String | N/A | N/A |
| *password* | HashCode | N/A | N/A |

### 10.3.4 Timing Constraints

Time constraints vary on network conditions. In optimal conditions the server may respond within t_success time

### 10.3.5 Initialization

Upon authentication token received, the application will begin to make requests to the back-end component

## 10.4 Image Sending Module

### 10.4.1 Description

This module will be used to capture a picture and upload the image for processing

### 10.4.2 Inputs and Outputs

**Inputs**: User input defining

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *Open Camera* | Button User Input | N/A | N/A | Open the menu for capturing an image |
| *Capture Image* | Button User Input | N/A | N/A | Capture the image |

**Outputs**: To be displayed to user

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *gotoPage* | action | N/A | N/A | Send user to menu to verify food items |

### 10.4.3 Exception Handling

| Input Name | Input Type | Exception | Exception Handling |
|---|---|---|---|
| *Open Camera* | Button User Input | Camera function not found | Hardware regulation |

### 10.4.4 Timing Constraints

Time constraints vary on the upload speed of the user account to upload the desired food image

### 10.4.5 Initialization

Selecting the camera option to capture the image of desired food image

## 10.5 Image Processing Module

### 10.5.1 Description

This module will be used to process an image sent by the user

### 10.5.2 Inputs and Outputs

**Inputs**: User input defining

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *Image* | image file | N/A | N/A | Process the image file |

**Outputs**: Return results to the user

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *gotoPage* | action | N/A | N/A | Return and allow user to select closest food choice and alter nutritional information |

### 10.5.3 Exception Handling

| Input Name | Input Type | Exception | Exception Handling |
|---|---|---|---|
| *Image* | image file | Image could not be processed | Image regulation |

### 10.5.4 Timing Constraints

Time constraints should be within t_success time to process and return the results for an image

### 10.5.5 Initialization

Operation is initialized when a photo has been uploaded to the server for processing

## 10.6 Food Diary Module

### 10.6.1 Description

This module will be used to show the nutritional requirements and goals for current date updated as more meals are added

### 10.6.2 Inputs and Outputs

**Inputs**: User input defining

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *foodItemSelected* | action | N/A | N/A | Select individual meal to edit/view nutritional information |

**Outputs**: Return results to the user

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *gotoPage* | action | N/A | N/A | Bring to front selected meal item |

### 10.6.3 Exception Handling

| Input Name | Input Type | Exception | Exception Handling |
|---|---|---|---|
| *select date* | action | date does not exist | Image regulation |

### 10.6.4 Timing Constraints

Past date on calendar should be returned between 0.1-1s for a responsive mobile application

### 10.6.5 Initialization

Operation is initialized when a user selects a date in the calendar

## 10.7 Timeline Module

### 10.7.1 Description

This module will be used to show the nutritional requirements and goals for future and past dates

### 10.7.2 Inputs and Outputs

**Inputs**: User input defining

| Input Name | Input Type | Range | Units | Comment(s) |
|---|---|---|---|---|
| *Select date* | date selection | N/A | N/A | Select individual dates to view past results |

**Outputs**: Return results to the user

| **Input Name** | **Input Type** | **Range** | **Units** | **Comment(s)** |
|---|---|---|---|---|
| *gotoPage* | action | N/A | N/A | Allow the user to see the selected date |

| **Input Name** | **Input Type** | **Exception** | **Exception Handling** |
|---|---|---|---|
| *select date* | action | date does not exist | Image regulation |

#### 10.7.3 Exception Handling

#### 10.7.4 Timing Constraints

Past date on calendar should be returned between 0.1-1s for a responsive mobile application

#### 10.7.5 Initialization

Operation is initialized when a user selects a date in the calendar

## 11 FSM Diagram with Transition Descriptions

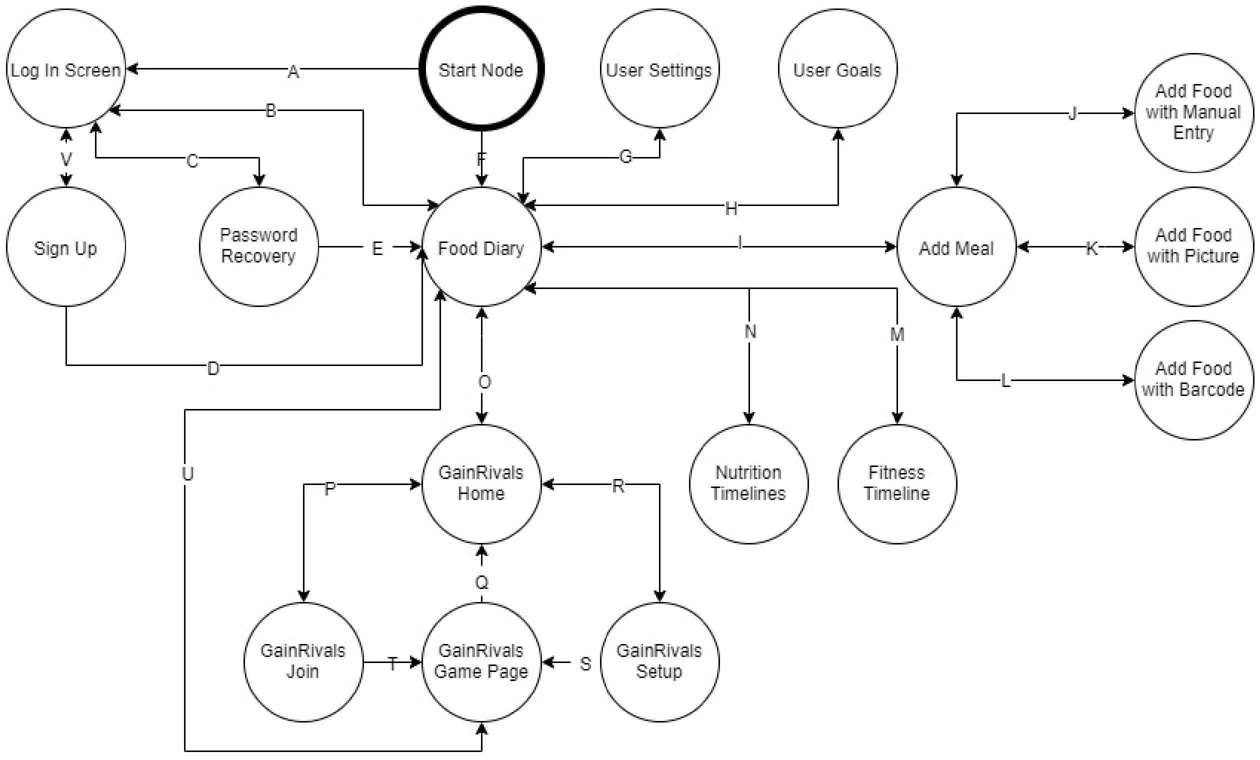


Figure 3: FSM Diagram

### 11.1 Description of Transitions in the FSM Diagram

(A) The user will open the application on their phones, and if they had not selected to remember their password in their previous session, they will be directed to the ”Log In Screen”.

(B) The user will be directed to the ”Food Diary” if they successfully fill out their username, password, and then tap on the sign in button.

The user will be directed back to the ”Log In Screen” if they decide to sign out by signing out from their ”Food Diary”.

(C) The user will user will be directed to ”Password Recovery” if they tap on the prompt which asks them if they had forgotten their password.

The user will be directed back to the ”Log In Screen”, by tapping on a back button, if they had went to this state by accident.

(D) The user will be directed to the ”Food Diary”, if they complete their sign up and tap on the complete sign up button.

(E) The user will be directed to the ”Food Diary”, if they successfully complete their password recovery and tap on the complete verification button.

(F) The user will be directed to the ”Food Diary” directly if they access the application and if they had decided to remember their password on the application.

(G) The user will be directed to the ”User Settings” if they tap on the user setup button on the ”Food Diary”.

The user will be directed to the ”Food Diary” if they tap on the back button in the ”User Settings”.

(H) The user will be directed to the ”User Goals” if they tap on the user setup button on the ”Food Diary”.

The user will be directed to the ”Food Diary” if they tap on the back button in the ”User Goals”.

(I) The user will be directed to ”Add Meal” if they tap on the add meal button on the ”Food Diary”.

The user will be directed to the ”Food Diary” if they tap on the back button in ”Add Meal”.

(J) The user will be directed to ”Add Food with Manual Entry” if they tap on the add food with manual entry button in ”Add Meal”.

The user will be directed to ”Add Meal” if they tap on the back button in ”Add Food with Manual Entry”.

(K) The user will be directed to ”Add Food with Picture” if they tap on the add food with manual entry button in ”Add Meal”.

The user will be directed to ”Add Meal” if they tap on the back button in ”Add Food with Picture”.

(L) The user will be directed to ”Add Food with Barcode” if they tap on the add food with manual entry button in ”Add Meal”.

The user will be directed to ”Add Meal” if they tap on the back button in ”Add Food with Barcode”.

(M) The user will be directed to the ”Fitness Timeline” if they tap on the fitness timeline button in the ”Food Diary”.

The user will be directed to the ”Food Diary” if they tap on the back button in the ”Fitness Timeline”.

(N) The user will be directed to ”Nutrition Timelines” if they tap on the nutrition timelines button in the ”Food Diary”.

The user will be directed to the ”Food Diary” if they tap on the back button on ”Nutrition Timelines”.

(O) The user will be directed to the ”GainRivals Home” if they tap on the GainRivals button in the ”Food Diary”, and if they don’t have an ongoing game at the moment.

The user will be directed to the ”Food Diary” if they tap on the back button on the ”GainRivals Home”.

(P) The user will be directed to ”GainRivals Join” if they wish to join a group, and tap on the join a group button in the ”GainRivals Home”.

The user will be directed to the ”GainRivals Home” if they tap on the back button on ”GainRivals Join”.

(Q) The user will be directed to the ”GainRivals Home” if they decide to leave a game session and tap on the leave group button on the ”GainRivals Game Page”.

(R) The user will be directed to the ”GainRivals Setup” if they decide to create their own game and tap on the setup a game button on the ”GainRivals Home”.

The user will be directed to the ”GainRivals Home” if they decide to tap the back button on the ”GainRivals Setup”.

(S) The user will be directed to the ”GainRivals Game Page” if they decide to create their game and tap on the create game button on the ”GainRivals Setup”.

(T) The user will be directed to the ”GainRivals Game page” if they fill out a valid game id in the appropriate prompt and tap on the join the game button in ”GainRivals Join”.

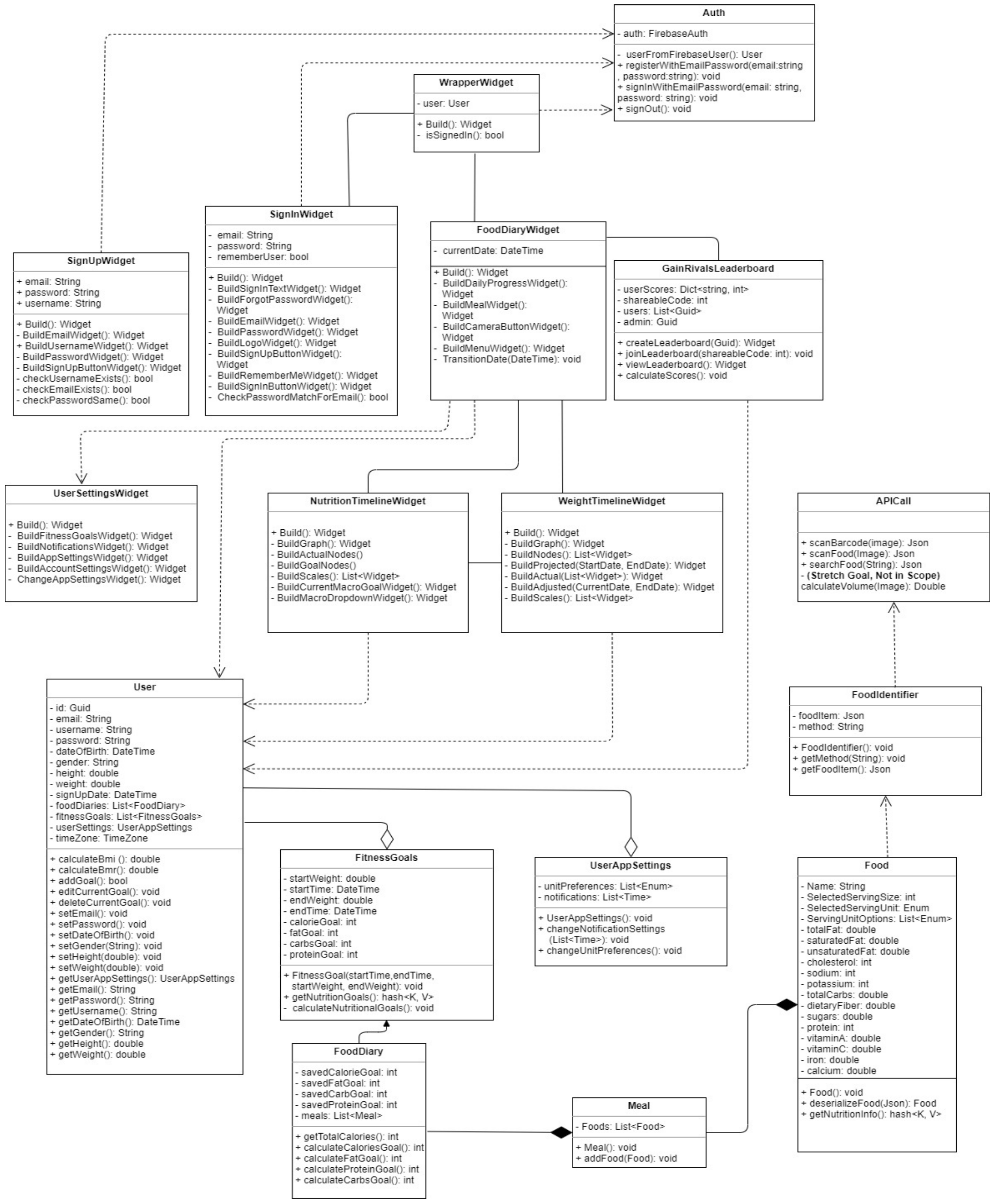
(U) The user will be directed to the "GainRivals Game Page" if the user has joined a game already, and they tap on the GainRivals button in the "Food Diary".

The user will be directed to the "Food Diary" if the user taps the exit game button in the "GainRivals Game Page".

(V) The user will be directed to the "Sign Up" if they don't have an account and tap on the sign up button in the "Log in Screen".

The user will be directed to the "Log in Screen" if they made a istake and tap on the back button in the "Sign Up".

## 12 Class Diagram



Auth
- auth: FirebaseAuth
- userFromFirebaseUser(): User
+ registerWithEmailPassword(email:string , password:string): void
+ signInWithEmailPassword(email: string, password: string): void
+ signOut(): void
WrapperWidget
- user: User
+ Build(): Widget
- isSignedIn(): bool
SignInWidget
- email: String
- password: String
- rememberUser: bool
+ Build(): Widget
- BuildSignInTextWidget(): Widget
- BuildForgotPasswordWidget(): Widget
- BuildEmailWidget(): Widget
- BuildPasswordWidget(): Widget
- BuildLogoWidget(): Widget
- BuildSignUpButtonWidget(): Widget
- BuildRememberMeWidget(): Widget
- BuildSignInButtonWidget(): Widget
- CheckPasswordMatchForEmail(): bool
SignUpWidget
+ email: String
+ password: String
+ username: String
+ Build(): Widget
- BuildEmailWidget(): Widget
+ BuildUsernameWidget(): Widget
- BuildPasswordWidget(): Widget
- BuildSignUpButtonWidget(): Widget
- checkUsernameExists(): bool
- checkEmailExists(): bool
- checkPasswordSame(): bool
FoodDiaryWidget
- currentDate: DateTime
+ Build(): Widget
- BuildDailyProgressWidget(): Widget
- BuildMealWidget(): Widget
- BuildCameraButtonWidget(): Widget
- BuildMenuWidget(): Widget
- TransitionDate(DateTime): void
GainRivalsLeaderboard
- userScores: Dict&lt;string, int&gt;
- shareableCode: int
- users: List&lt;Guid&gt;
- admin: Guid
+ createLeaderboard(Guid): Widget
+ joinLeaderboard(shareableCode: int): void
+ viewLeaderboard(): Widget
+ calculateScores(): void
UserSettingsWidget
+ Build(): Widget
- BuildFitnessGoalsWidget(): Widget
- BuildNotificationsWidget(): Widget
- BuildAppSettingsWidget(): Widget
- BuildAccountSettingsWidget(): Widget
- ChangeAppSettingsWidget(): Widget
NutritionTimelineWidget
+ Build(): Widget
- BuildGraph(): Widget
- BuildActualNodes()
- BuildGoalNodes()
- BuildScales(): List&lt;Widget&gt;
- BuildCurrentMacroGoalWidget(): Widget
- BuildMacroDropdownWidget(): Widget
WeightTimelineWidget
+ Build(): Widget
- BuildGraph(): Widget
- BuildNodes(): List&lt;Widget&gt;
- BuildProjected(StartDate, EndDate): Widget
- BuildActual(List&lt;Widget&gt;): Widget
- BuildAdjusted(CurrentDate, EndDate): Widget
- BuildScales(): List&lt;Widget&gt;
APICall
+ scanBarcode(image): Json
+ scanFood(Image): Json
+ searchFood(String): Json
- (Stretch Goal, Not in Scope) calculateVolume(Image): Double
User
- id: Guid
- email: String
- username: String
- password: String
- dateOfBirth: DateTime
- gender: String
- height: double
- weight: double
- signUpDate: DateTime
- foodDiaries: List&lt;FoodDiary&gt;
- fitnessGoals: List&lt;FitnessGoals&gt;
- userSettings: UserAppSettings
- timeZone: TimeZone
+ calculateBmi (): double
+ calculateBmr(): double
+ addGoal(): bool
+ editCurrentGoal(): void
+ deleteCurrentGoal(): void
+ setEmail(): void
+ setPassword(): void
+ setDateOfBirth(): void
+ setGender(String): void
+ setHeight(double): void
+ setWeight(double): void
+ getUserAppSettings(): UserAppSettings
+ getEmail(): String
+ getPassword(): String
+ getUsername(): String
+ getDateOfBirth(): DateTime
+ getGender(): String
+ getHeight(): double
+ getWeight(): double
FoodIdentifier
- foodItem: Json
- method: String
+ FoodIdentifier(): void
+ getMethod(String): void
+ getFoodItem(): Json
FitnessGoals
- startWeight: double
- startTime: DateTime
- endWeight: double
- endTime: DateTime
- calorieGoal: int
- fatGoal: int
- carbsGoal: int
- proteinGoal: int
+ FitnessGoal(startTime,endTime, startWeight, endWeight): void
+ getNutritionGoals(): hash&lt;K, V&gt;
- calculateNutritionalGoals(): void
UserAppSettings
- unitPreferences: List&lt;Enum&gt;
- notifications: List&lt;Time&gt;
+ UserAppSettings(): void
+ changeNotificationSettings (List&lt;Time&gt;): void
+ changeUnitPreferences(): void
Food
- Name: String
- SelectedServingSize: int
- SelectedServingUnit: Enum
- ServingUnitOptions: List&lt;Enum&gt;
- totalFat: double
- saturatedFat: double
- unsaturatedFat: double
- cholesterol: int
- sodium: int
- potassium: int
- totalCarbs: double
- dietaryFiber: double
- sugars: double
- protein: int
- vitaminA: double
- vitaminC: double
- iron: double
- calcium: double
+ Food(): void
+ deserializeFood(Json): Food
+ getNutritionInfo(): hash&lt;K, V&gt;
FoodDiary
- savedCalorieGoal: int
- savedFatGoal: int
- savedCarbGoal: int
- savedProteinGoal: int
- meals: List&lt;Meal&gt;
+ getTotalCalories(): int
+ calculateCaloriesGoal(): int
+ calculateFatGoal(): int
+ calculateProteinGoal(): int
+ calculateCarbsGoal(): int
Meal
- Foods: List&lt;Food&gt;
+ Meal(): void
+ addFood(Food): void

Figure 4: Class Diagram

### 12.1 Description of Class Diagram Terminology

As mentioned in the Class Diagram, Widgets are components of the Flutter framework and are part of the Dart language that makes up the software development kit. Widgets describe UI elements of different screens, and the Widget Classes are described according to single screens for the mobile application. The subsequent private methods for each Widget Class describe smaller portions of the overall widget. For example, the method BuildRememberMeWidget in the SignInWidget Class only consists of a Checkbox Widget in order to better modularize the Widgets on a component level.

## 13 MIS

### 13.1 SignUpWidget

**Uses**: Auth
**Public Functions**
**SignUpWidget(String email, String password, String username): widget** Constructor for SignUp-Widget. Takes email, password, and username of the user
**Build():Widget** Opens Parent Widget for the signup page
**BuildEmailWidget():Widget** Returns the email input component of the page
**BuildUsernameWidget():** Returns the username component of the page
**BuildPasswordWidget():** Returns the password component of the page
**BuildSignUpButtonWidget():** Returns the signup button component of the page
**checkUsernameExists():bool** Returns True if a username exists
**checkEmailExits():bool** Returns True if an email exists
**checkPasswordSame():bool** Returns True if the password and confirmed password as identical

### 13.2 SignInWidget

**Uses**: Auth
**Public Functions**
**BuildSignInTextWidget(String email, String password, rememberUser bool): widget** Constructor for SignInWidget. Takes email, password and bool if previous user was remembered
**Build():Widget** Opens Parent Widget for the signin widget
**BuildSignInTextWidget():Widget** Returns the sign in text component
**BuildForgotPassword():Widget** Returns the forgot password widget
**BuildEmailWidget():Widget** Returns the email component of the page
**BuildPasswordWidget():Widget** Returns the password component of the widget
**BuildLogoWidget():Widget** Returns the logo state
**BuildSignUpButtonWidget():Widget** Returns the signup button state of the widget
**BuildRememberMeWidget():Widget** Returns remember me state of in the signup page
**BuildSignInButtonWidget():Widget** Returns the sign in button state of the widget
**CheckPasswordMatchForEmail():bool** Returns True if the password matches the email

### 13.3 UserSettingsWidget

**Uses**: None
**Public Functions**:
**UserSettingsWidget(): widget** Opens parent widget for WrapperWidget
**Private Functions**:
**BuildFitnessGoalsWidget(): widget** Opens child widget for fitness goal
**BuildNotificationsWidget(): widget** Opens child widget for user notifications
**BuildAppSettingsWidget(): widget** Opens child widget for application settings

**BuildAccountSettingsWidget(): widget** Opens child widget for user account settings
**ChangeAppSettingsWidget(): widget** Opens child widget for changing the application settings

## 13.4 Auth

**Uses**: None
**Public Functions**
**Auth(FirebaseAuth auth): widget**
Consutructor for Auth. Takes a FirebaseAuth **userFromFirebaseUser():User** Constructs a User variable from Firebase User
**registerWithEmailPassword(String email, String password):void** Register account
**signInWithEmailPassword(String email, String password):void** Sign in state
**signOut(): void** Sign out of user account

## 13.5 FoodDiaryWidget

**Uses**: User, UserSettingsWidget
**Public Functions**
**BuildFoodDiaryWidget(currentDate: DateTime): widget** constructor
**Build():Widget** Opens Parent Widget for Food Diary Widget
**BuildDailyProgressWidget():Widget** Returns the Daily Progress page
**BuildMealWidget():Widget** Returns the selected meal
**BuildCameraButtonWidget():Widget** Opens the camera widget
**BuildMenuWidget():Widget** Opens the menu widget
**TransistionDate(DateTime):void** Change the transition date

## 13.6 WrapperWidget

**Uses**: Auth
**Public Functions**
**WrapperWidget(user: User): widget** constructor for WrapperWidget
**Build():Widget** Builds widget for WrapperWidget
**isSignedIn():bool** Returns True if user is signed in

## 13.7 GainRivalsLeaderboard

**Uses**: User
**Public Functions**
**GainRivals(Dict<string, int> userScores, int shareableCode, list<Guid> users, Guid admin): widget** Constructor for GainRivals
**createLeaderboard(Guid):Widget** Creates the leaderboard widget
**joinLeaderboard(int shareableCode):void** Joins the leaderboard with an int shareable code
**viewLeaderboard():Widget** Returns the widget to view the leaderboard
**calculateScores():void** Compute the scores of users

## 13.8 NutritionTimelineWidget

**Uses**: User
**Public Functions**
**Build():Widget)** Constructor for nutrtition timeline

**BuildGraph():Widget** Creates a graph widget
**BuildActualNodes()** Creates nodes for the actual amount of a specific macro consumed
**BuildGoalNodes()** Creates nodes for the target amount of a macro needed
**BuildScales():List<Widget>** Opens widgets for the scales for x and y axis
**BuildCurrentMacroGoalWidget():Widget** Opens widget for label denoting current macro amount consumed
**BuildMacroDropdownWidget():Widget** Opens widget for a dropdown menu to switch between timelines

## 13.9 WeightTimelineWidget

**Uses**: User
**Public Functions**
**Build():Widget)** Opens the parent widget for weight Timeline screen
**BuildGraph():Widget** Opens widget for graph to display data
**BuildNodes():List⟨Widget⟩** Opens nodes to denote the target and actual weight of user
**BuildProjected(StartDate, EndDate):Widget** Opens widget for a line graph to show the projected progress from start date to end date
**BuildActual(List⟨Widget⟩):Widget** Opens widget for a line graph to show the actual weight of the user from start date to current date
**BuildAdjusted(CurrentDate, EndDate):Widget** Opens widget for a line graph to show the projected progress from current date to end date
**BuildScales(): List <Widget>** Opens widgets to create x and y scales

## 13.10 User

**Uses**: None
**Public Functions**:
**User(Guid id, String email, String username, String password, DateTime dateOfBirth, char gender, double height, double weight, DateTime signUpDate, List ⟨FoodDiary⟩foodDiaries, List ⟨FitnessGoals⟩fitnessGoals, UserAppSettings userSettings, TimeZone timeZone)**
**calculateBmi():double** Calculates BMI of user
**calculateBmr():double** Calculates BMR of user
**addGoal(FitnessGoal): bool** - Adds a FitnessGoal to fitnessGoals internal variable. Returns true or false depending on the user setting a healthy goal according to their weight and height (for example, if the goal is to lose 1 pound every two days).
**editCurrentGoal(): void** - Edits FitnessGoal at the last element of the fitnessGoals internal variable.
**deleteCurrentGoal(): void** - Deletes FitnessGoal at the last element of the fitnessGoals internal variable. State changes such that there will be one less element in the List.
**setEmail(String email): void** - Sets email to new value, as passed into this method.
**setPassword(String password): void** - Sets the password for the User.
**setDateOfBirth(DateTime dateOfBirth): void** - Sets the Date of Birth for the user, if the user chooses to switch their Date of Birth.
**setGender(String gender): void** - Sets the Gender for the User.
**setHeight(Double height): void** - Sets the Height for the user in centimetres.
**setWeight(Double weight): void** - Sets the Weight for the user in kilograms.
**getUserAppSettings(): UserAppSettings** - Gets the UserAppSettings for the User.
**getEmail(): String** - Gets the email address for the given user.
**getPassword(): String** - Gets the password for the given user.
**getUsername(): String** - Retrieves the username for the given user.
**getDateOfBirth(): DateTime** - Retrieves the Date of Birth for the given user.
**getGender(): String** - Retrieves the Gender for the user.

**getHeight(): Double -** Retrieves the height of the user.
**getWeight(): Double -** Retrieves the weight of the user.

## 13.11 FitnessGoals

**Uses**: None
**Public Functions**
**FitnessGoal(DateTime startTime, DateTime endTime, Double startWeight, Double endWeight):void** Fitness goals constructor
**getNutritionGoals():hash ⟨K,V⟩** get goals
**calculateNutritionalGoals():void** Compute how well the user is reaching nutritional goals

## 13.12 FoodDiary

**Uses**: None
**Public Functions**
**FoodDiary(int savedCalorieGoal, int savedFatGoal, int savedCarbGoal, int savedProteinGoal, List ⟨Meal⟩meals)** Cosntructor for FoodDiary
**getTotalCalories():int** Returns the total calorie count
**calculateCaloriesGoal():int** Returns the calories remaining in preset goal
**calculateFatGoal():int** Returns fat remaining in preset goal
**caluclateProteinGoal():int** Returns protein remaining in preset goal
**calculateCarbsGoal():int** Returns carbs remaining in reset carbs goal

## 13.13 UserAppSettings

**Uses**: None
**Public Functions**
**UserAppSettings(List⟨Enum⟩, List ⟨Time⟩notifications)** User App Settings constructor
**changeNotificationSettings(List ⟨Time⟩):void**
**changeUnitPreferences():void**

## 13.14 Meal

**Uses**: None
**Public Functions**
**Meal(List<Food> Foods)** constructor
**addFood(Food):void** Add a food to meal list

## 13.15 APICall

**Uses**: None
**Public Functions**
**scanBarcode(image)**: Scans barcode and returns a Json
**scanFood(image)**: Scans image and returns a Json
**searchFood(String)**: Searches for given food name and returns Json
***(Stretch goal not in scope)*** **calculateVolume(Image):Double** Calculates the total volume of the given food

### 13.16 FoodIdentifier

**Uses**: APICall
**Public Functions**
**FoodIdentifier(Json foodItem, String method)** Constructor
**getMethod(String):void** Gets the method of identification (picture, barcode, food name)
**getFoodItem():json** Gets the Json file for the food item

### 13.17 Food

**Uses**: FoodIdentifier
**Public Functions**
**food(): void** constructor
**deserializedFood(Json): Food** Converts the Json file and converts it to a food object
**getNutritionInfo(): hash<K,V>** Returns all the nutritional information of the food in a hash

## 14 MID

### 14.1 User

**Uses**: None
**Internal Values**:
**Guid id**: A global unique identifier that is the key for User table in Firebase database.
**String email**: The email address that is associated with the particular user account.
**String username**: The username for the given user account. This is displayed in the GainRivals leaderboard and in the Menu child Widget of the FoodDiaryWidget class.
**String password**: The password for the given user account.
**DateTime dateOfBirth**: The user's given Date of Birth, to determine age that is needed for BMR.
**String gender**: The gender that the user identifies as, stored as a String.
**Double height**: The height of the user in centimeters, needed for BMR and BMI calculations.
**Double weight**: The weight of the user in kilograms, needed for BMR and BMI calculations.
**DateTime signUpDate:** The date that the User's account was started. Used to log the earliest possible time for Timeline Widgets (Nutrition and Fitness).
**List<FoodDiary> foodDiaries:** A list of associated food diary objects with corresponding food items. Used for total calories and macro calculations to determine if user meets fitness/nutrition goals.
**List<FitnessGoal> fitnessGoals** A list of fitness goals that are used for data trending on the FitnessTimelineWidget for the User.
**UserAppSettings userSettings** The NutriBud App Settings for the user, storing unit preferences and notification alert times.
**TimeZone timeZone**: The TimeZone of the User's location.
**Functions:**
**calculateBmr(): void** - Calculated BMR of a User, Equation we use is the Mifflin - St Jeor Equation : Male BMR = 10 * weight + 6.25 * height - 5 * age + 5,Female BMR = 10 * weight + 6.25 * height - 5 * age - 161
**calculateBmi():void** - Calculated BMI of a User, Equation we use is BMI = weight / $(\text{height}/100)^2$
**addGoal(FitnessGoal): bool** - Adds a FitnessGoal to fitnessGoals internal variable. Returns true or false depending on the user setting a healthy goal according to their weight and height (for example, if the goal is to lose 1 pound every two days).
**editCurrentGoal(): void** - Edits FitnessGoal at the last element of the fitnessGoals internal variable.
**deleteCurrentGoal(): void** - Deletes FitnessGoal at the last element of the fitnessGoals internal variable. State changes such that there will be one less element in the List.
**setEmail(String email): void** - Sets private email internal variable to be email passed into the method.

**setPassword(String password): void** - Sets the password for the User.
**setDateOfBirth(DateTime dateOfBirth): void** - Sets the Date of Birth for the user, if the user chooses to switch their Date of Birth.
**setGender(String gender): void** - Sets the Gender for the User.
**setHeight(Double height): void** - Sets the Height for the user in centimetres.
**setWeight(Double weight): void** - Sets the Weight for the user in kilograms.
**getUserAppSettings(): UserAppSettings** - Gets the UserAppSettings for the User.
**getEmail(): String** - Gets the email address for the given user.
**getPassword(): String** - Gets the password for the given user.
**getUsername(): String** - Retrieves the username for the given user.
**getDateOfBirth(): DateTime** - Retrieves the Date of Birth for the given user.
**getGender(): String** - Retrieves the Gender for the user.
**getHeight(): Double** - Retrieves the height of the user.
**getWeight(): Double** - Retrieves the weight of the user.

## 14.2 FitnessGoals

**Uses:** None

**Internal Values:**
**Double startWeight:** Initial starting weight of the user, used in calculating speed of progress nedded to achieve goal.
**DateTime startTime:** Initial date that the user begun working for their goal.
**Double endWeight:** Final weight goal of the user.
**DateTime endTime:** The date at which the user intends to achieve their goal by.
**int calorieGoal:** The target daily calorie that the user needs to consume in order to achieve the goal by the give date. This is calculated with the following equation. calorieGoal = BMR + ((endWeight - startWight) * 3500)/(endTime - startTime)
**int fatGoal:** The target daily fat that the user needs to consume in order to achieve the goal by the give date. This is calculated with the following equation. fatGoal = (calorieGoal * 0.3)/ 9. calorieGoal = fatGoal * 9 + 4 * (carbsGoal + proteinGoal)
**int carbsGoal:** The target daily carbohydrates that the user needs to consume in order to achieve the goal by the give date. This is calculated with the following equation. carbsGoal = (calorieGoal * 0.5) / 4.
**int proteinGoal:** The target daily protein that the user needs to consume in order to achieve the goal by the give date. This is calculated with the following equation. proteinGoal = (calorieGoal * 0.2) / 4.

**Functions:**
**Public FitnessGoals(DateTime startTime, DateTime endTime, Double startWeight, Double endWeight): void** - Gets the initial and end values from the user and initializes the values for the goal.
**Public getNutritionGoals(): hash<K,V>** - Formats all the info in properties into key value pairs and returns it for use by other classes.
**Private calculateNutritionGoals(): void** - Uses formulas and equations to calculate the optimal amount of nutrition needed per day to achieve goal.

## 14.3 FoodDiary

**Uses**: None
**Internal Values:**
**int savedCalorieGoal:** The target daily calorie that the user needs to consume in order to achieve the goal by the give date.
**int savedFatGoal:** The target daily fat that the user needs to consume in order to achieve the goal by the give date.
**int savedCarbsGoal:** The target daily carbohydates that the user needs to consume in order to achieve

the goal by the give date.
**int savedProteinGoal:** The target daily protein that the user needs to consume in order to achieve the goal by the give date.

**Functions:**
**Public getTotalCalories(): int** - Calculates the amount of calories for the current day that the user has currently achieved.
**Public calculateCaloriesGoal(): int** - Calculates the amount of calories for the current day that the user must achieve.
**Public calculateFatGoal(): int** - Calculates the amount of fat for the current day that the user must achieve.
**Public calculateProteinGoal(): int** - Calculates the amount of protein for the current day that the user must achieve.
**Public calculateCarbsGoal(): int** - Calculates the amount of carbohydrates for the current day that the user must achieve.

## 14.4 Meal

**Uses:** None
**Internal Values:**
**List<Food> Foods:** Stores the list of food items that make up the meal.

**Functions:**
**Public Meal(): void** - Initializes internal values.
**Public addFood(Food food): void** - Appends the food item to the list of foods.

## 14.5 UserAppSettings

**Uses**: None
**Internal Values:**
**List¡Enum¿ unitPreferences:** Key value store that holds conversion ratios (ex. kilograms to pounds, inches to centimetres).
**List¡Time¿ notifications:** List that contains all of the notifications that a user sets up for themselves.

**Functions:**
**Public UserAppSettings(): void** - initializes the variables for the object **Public changeNotificationSetting(List¡Time¿): void** - Changes the notifications that the user sets up for themselves. **Public changeUnitPreferences(): void** - Changes measurements from metric to imperial and vice versa.

## 14.6 Food

**Uses:** FoodIdentifier
**Internal Values:**
**String Name:** The identified food.
**Int SelectedServingSize:** The serving size of the identified food.
**Enum SelectedServingUnit:** The unit of measurement used to measure the serving size.
**List<Enum> ServingUnitOptions:** A list of common units of measurements for the user to use.
**Double totalFat:** The calculated total fat in the food.
**Double saturatedFat:** The calculated total saturated fat in the food.
**Double unsaturatedFat:** The calculated total unsaturated fat in the food.
**Int cholesterol:** The calculated total cholesterol in the food.
**Int sodium:** The calculated total sodium in the food.
**Int potassium:** The calculated total potassium in the food.
**Double totalCarbs:** The calculated total carbohydrates in the food.

**Double dietaryFiber:** The calculated total fiber in the food.
**Double sugars:** The calculated total sugar in the food.
**Double vitaminC:** The calculated total vitamin C in the food.
**Double calcium:** The calculated total calcium in the food.

**Functions:**
**Public Food(): void** - Initializes variables.
**Public deserializeFood(Json): Food** - Converts the Json file and converts it to a food object.
**Public getNutrtionInfo(): hash<K,V>** - Returns all the nutritional information of the food in a hash.

## 14.7 FoodIdentifier

**Uses**: APICall
**Internal Values:**
**Json foodItem:** Json object that contains the food information fields
**String method:** The method used by the user to send the image parameters to the system

**Functions:**
**Public FoodIdentifier(): void** - Initializes the variables for the object **Public getMethod(String): void** - Returns the method of identifiation used by the user (ex. picture taken, written in food name and information, barcode). **Public getFoodItem(): Json** - Gets the Json object of the food item that contains all important fields of the object.

## 14.8 APICall

**Uses**: None
**Internal Values:** None

**Functions:**
**Public scanBarcode(Image): Json** - Image of barcode is used to calculate nutrition information of food items.
**Public scanFood(Image): Json** - Image of food is used to calculate nutrition information of food items.
**Public searchFood(String): Json** - Searches for food nutrition information based in search input entered by the user.
**Private calculateVolume(image): Double** - Gets the volume of the food items

## 14.9 SignInWidget

**Uses**: Auth

**Internal Values:**
**String email:** The user's email address that was used to sign in.
**String password:** The user's password that was used to sign in.
**bool rememberUser:** The status of whether a user wants their email address remembered by the system.

**Functions:**
**Public BuildSignInTextWidget(String email, String password, rememberUser bool)** Constructor for SignInWidget. Takes email, password and bool if previous user was remembered
**Private Build():Widget** Returns the signin widget
**Private BuildSignInTextWidget():Widget** Returns the sign in text component
**Private BuildForgotPassword():Widget** Returns the forgot password widget
**Private BuildEmailWidget():Widget** Returns the email component of the page
**Private BuildPasswordWidget():Widget** Returns the password component of the widget
**Private BuildLogoWidget():Widget** Returns the logo state

**Private BuildSignUpButtonWidget():Widget** Returns the signup button state of the widget
**Private BuildRememberMeWidget():Widget** Returns remember me state of in the signup page
**Private BuildSignInButtonWidget():Widget** Returns the sign in button state of the widget
**Private CheckPasswordMatchForEmail():bool** Returns True if the password matches the email

### 14.10 SignUpWidget

**Uses**: Auth

**Internal Values:**
**String email:** The user's email address that was used to sign up.
**String password:** The user's password that was used to sign up.
**String username:** The user's username that was used to sign up.

**Functions:**
**Public SignUpWidget(String email, String password, String username)** Constructor for SignUp-Widget. Takes email, password, and username of the user
**Private Build():Widget** Returns the signup page
**Private BuildEmailWidget():Widget** Returns the email input component of the page
**Private BuildUsernameWidget():** Returns the username component of the page
**Private BuildPasswordWidget():** Returns the password component of the page
**Private BuildSignUpButtonWidget():** Returns the signup button component of the page
**Private checkUsernameExists():bool** Returns True if a username exists
**Private checkEmailExits():bool** Returns True if an email exists
**Private checkPasswordSame():bool** Returns True if the password and confirmed password as identical

### 14.11 NutritionTimelineWidget

**Uses**: User

**Functions:**
**Public Build():Widget)** Constructor for nutrtition timeline
**Private BuildGraph():Widget** Creates a graph widget
**Private BuildActualNodes()** Creates nodes for the actual amount of a specific macro consumed
**Private BuildGoalNodes()** Creates nodes for the target amount of a macro needed
**Private BuildScales():List<Widget>** Opens widgets for the scales for x and y axis
**Private BuildCurrentMacroGoalWidget():Widget** Opens widget for label denoting current macro amount consumed
**Private BuildMacroDropdownWidget():Widget** Opens widget for a dropdown menu to switch between timelines

### 14.12 WeightTimelineWidget

**Uses**: User

**Internal Values:**
**String email:** The user's email address that was used to sign up.
**String password:** The user's password that was used to sign up.
**String username:** The user's username that was used to sign up.

**Functions:**
**Public Build():Widget)** Opens the parent widget for weight Timeline screen
**Private BuildGraph():Widget** Opens widget for graph to display data
**Private BuildNodes():List⟨Widget⟩** Opens nodes to denote the target and actual weight of user
**Private BuildProjected(StartDate, EndDate):Widget** Opens widget for a line graph to show the projected progress from start date to end date
**Private BuildActual(List⟨Widget⟩):Widget** Opens widget for a line graph to show the actual weight of the user from start date to current date
**Private BuildAdjusted(CurrentDate, EndDate):Widget** Opens widget for a line graph to show the projected progress from current date to end date
**Private BuildScales(): List <Widget>** Opens widgets to create x and y scales

## 14.13 FoodDiaryWidget

**Uses**: User, UserSettingsWidget

**Internal Values:**
**DateTime currentDate:** The current date for the food diary.

**Functions:**
**Public BuildFoodDiaryWidget(currentDate: DateTime): widget** constructor
**Private Build():Widget** Opens Parent Widget for Food Diary Widget
**Private BuildDailyProgressWidget():Widget** Returns the Daily Progress page
**Private BuildMealWidget():Widget** Returns the selected meal
**Private BuildCameraButtonWidget():Widget** Opens the camera widget
**Private BuildMenuWidget():Widget** Opens the menu widget
**Private TransistionDate(DateTime):void** Change the transition date

## 14.14 WrapperWidget

**Uses**: Auth

**Internal Values:**
**User user:** The currently logged in user.

**Functions:**
**Public WrapperWidget(user: User): widget** constructor for WrapperWidget
**Private Build():Widget** Builds widget for WrapperWidget
**Private isSignedIn():bool** Returns True if user is signed in

## 14.15 GainRivalsLeaderboard

**Uses**: User

**Internal Values:**
**Dict¡string, int¿ userScores:** Key value store connecting names with scores
**int shareableCode:** Entry code in order to enter the leaderboard group.
**List¡Guid¿ users:** The members of the specified leaderboard group.
**Guid admin:** The admin of the leaderboard group.

**Functions:**
**Public GainRivals(Dict¡string, int¿ userScores, int shareableCode, list¡Guid¿ users, Guid admin): widget** Constructor for GainRivals
**Public createLeaderboard(Guid):Widget** Creates the leaderboard widget
**Public joinLeaderboard(int shareableCode):void** Joins the leaderboard with an int shareable code
**Public viewLeaderboard():Widget** Returns the widget to view the leaderboard
**Public calculateScores():void** Compute the scores of users

# 15 Normal Operation

## 15.1 Description/Behaviour

The system must be able to, upon taking a picture of a food item on a plate, predict the particular type of food that is on the plate. Afterwards, the system must use this data to accurately deliver the nutritional contents of the food item to the user on an haptic input screen that allows for the user to input the serving size. These nutritional contents are outlined through the context diagram on Figure **??**.

Alternatively, the system must either allow the user to scan a UPC barcode through the phone's camera hardware or allow the user to interact with the phone's haptic touch screen to manually search for food products.

Additionally, the system will allow the user to track and visualize their fitness and nutritional progress. They shall be able to plot their progress over a timeline, which they can modify the size of in order to fit the view they wish to look at it with.

Finally, the system must offer users the ability to create an account through their email address. The system will allow users to create private Friend Groups to play a game called GainRivals, a customizable leaderboard setup. This leaderboard will be composed of user's scores that will be ranked to create a competitive atmosphere.

As a whole, the system must perform all of these behaviours accordingly.

## 15.2 Notation

GainRivals = The game feature of the application, consisting of a leaderboard with a custom score options.
Macros = Macro nutritional content which the application will be tracking, including fats, proteins and carbohydrates.

## 15.3 Normal Use Cases/Scenarios

### 15.3.1 User Logs into NutriBud Account

The application is logged onto by the user, and will subsequently display the current food diary for the given day they log on.

### 15.3.2 Scan Food Barcode

The system will detect if a barcode in the camera view appears within a centered box boundary. Afterwards, the system will transfer the user to the logging screen for the daily food diary after retrieving the nutritional information according to the food.

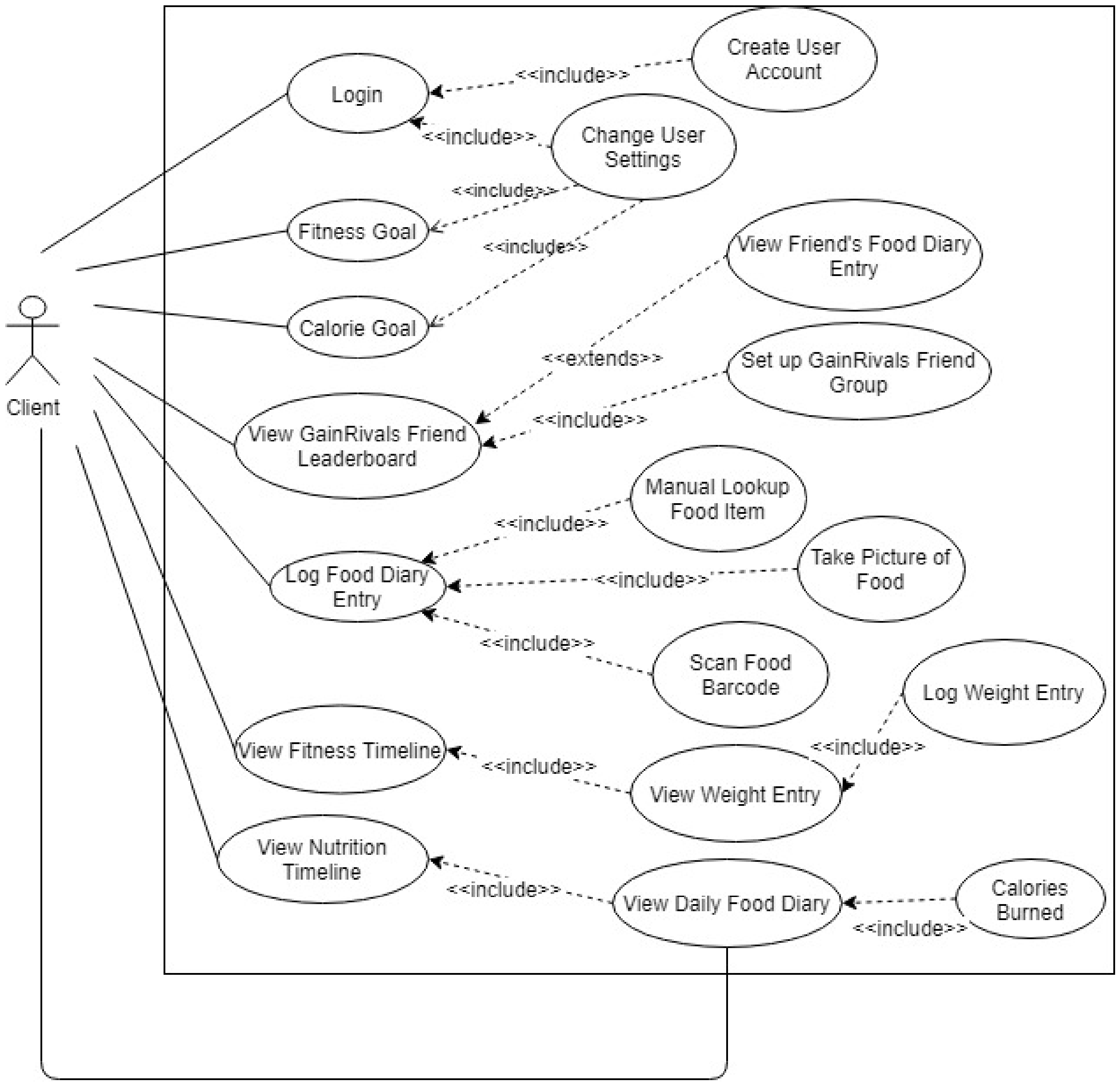


Figure 5: NutriBud Use Case Diagram

#### 15.3.3 Set Up GainRivals Friend Group

The system will have the ability to let users add friends to their account. Upon doing this, the system shall also allow users to create custom lobbies for the GameRivals game, which promotes competitive play by displaying a leaderboard for how closely they meet a user predefined scoring regimen.

#### 15.3.4 View GainRivals Friend Leaderboard

The system will allow the user to proceed to GainRivals by pressing a button from the main menu. The system will allow users to select which friend group they want to view for leaderboard statistics.

### 15.3.5 Manual Lookup Food Item

The system will be able to, upon the user selecting a search button, to produce a list of food items according to a filter that is provided by the user and described by the control variable manual_entry.

### 15.3.6 Take Picture of Food

The system must be able to open the phone's camera and allow the user to take a picture of the food for the system to recognize and process for nutritional content display.

### 15.3.7 Create User Account

The system must allow for the user to create a NutriBud account using an email address and password combination.

### 15.3.8 Log Weight Entry

The system must allow for the user to log their weight on any day and store this data.

### 15.3.9 Log Food Diary Entry

The system must allow for the user to log their nutritional consumption on any day and store this data.

### 15.3.10 View Weight Entry

The system must allow the user to view a logged weight entry from the fitness timeline by selecting a day.

### 15.3.11 View Daily Food Diary

The system must allow the user to view a logged food diary on any current or past date showcasing a breakdown of nutrient consumption from meals.

### 15.3.12 View Fitness Timeline

The system must create a fitness timeline to be shown for the user to view a history of all logged weight entries. This can be viewed on a custom time scale to be selected by the user.

### 15.3.13 View Nutrition Timeline

The system must create a nutrition timeline that showcases all logged food diary entries by the user. Upon touch input from the user, the system will then display the food diary entry for that day

### 15.3.14 Change User Settings

The system must provide the user with the ability to change and update their relative user settings. Upon tapping on the change setting icon for each user setting, the system will provide a dialog box for the user to change the relative setting.

### 15.3.15 Fitness Goals

The system will provide a functionality that allows the user to create and update a fitness goal. The fitness goal will allow the user to set a target weight to aim for, they will also be able to view the timeline that is based on their fitness goal.

#### 15.3.16 Calorie Goals

The system will provide a functionality that will allow the user to create and update a calorie goal. The calorie goal will allow the user to set a target calorie intake for a day according to macronutrients. There will also be a calorie intake timeline for the user to view.

#### 15.3.17 Calories Burned

The system will allow the user to input the amount of calories they burned in a certain day. The Food Diary home page will provide the user with a button that will create a dialog box to input the amount of calories burned that will change the daily calorie goal respectively

# 16 Undesired Event Handling

The application should fail elegantly under critical software failure and handle errors or exceptions encountered during application use.

## 16.1 Barcode not Detected in Camera's Boundary Box

- Camera is too close to the barcode to detect the full barcode
- Camera is too far from the barcode to recognize the barcode
- Visual conditions too poor for the barcode to be recognized by the camera.

If the barcode cannot be detected in the camera's boundary box, the system will instruct the user to reposition the camera. In the case that it cannot be detected at all, they will be redirected to the manual search.

## 16.2 Barcode not Recognized

- Barcode is damaged
- Picture quality of barcode is too low
- Barcode is not valid
- Barcode is not able to be resolved
- Barcode cannot be found in database

If the barcode cannot be determined by the system, the system will create a dialog box that is displayed to the user. This will ask the user whether they wish to continue using the camera functionality or provide an alternate solution of manually looking up the food item they wish to detect.

## 16.3 Failure to Detect Food Item

- Food items are too close or too far from the camera
- Lighting conditions are too poor for the camera to detect the food item

If the application cannot detect the food item upon a user's request, the application shall prompt the user to retake their picture with the food at a further distance of 3 feet away. In the case that it cannot be detected at all, they will be redirected to the manual search.

## 16.4 Failure to Identify Food Item

- Distorted food items are present and cause application to misidentify
- Image recognition model fails to accurately identify the food item type

If the NutriBud system misidentifies a food item, then the percentage in confidence should be given to the user in order to highlight the inaccurate nature of the prediction. The system should advise the user to manually enter the name of the food item to ensure the item is properly logged. In the case the food item may not be detected at all, they will be redirected to the manual entry.

### 16.5 System Failure

- A bug/glitch causes the NutriBud application to crash and shut down
- A bug/glitch that distorts the computed nutritional values
- The application does not complete a task correctly

The application will save any completed tasks and/or data it can. During the next time the application is launched, it will resume the same task it was performing before the crash if possible.

## 17 Likely and Unlikely Changes

### 17.1 Likely Changes

1. Creation of new variables that are declared as our implementation is developed. Deleting variables that we decide aren't useful while coding our product.
2. Function Modifications to allow better modularity of calculations, and breaking down series of steps. Additionally, functions could change in how many parameters are passed as a result of better modularity. We wish to follow common object-oriented programming principles in our design, including the single responsibility principle and others that make up the S.O.L.I.D. programming acyonym's principles.
3. Classes could be modified in order to allow for a greater level of abstractions (i.e. possible abstract class for Timelines in the future)

### 17.2 Unlikely Changes

1. API Class that talks with IBM Watson Model Service will be write in the Python language.
2. IBM Watson Model will be used (might stay the same or change to accommodate new types of foods unfamiliar with the model)
3. Widgets and Data Models (i.e. Food, Meal, etc.) will be written in the Dart language as part of the Flutter software development kit.
4. The general architecture, like our states, modules and components of our systems design.

## 18 References

### 18.1 Reference Used to Figure out caloric composition of Macronutrients

`https://www.nal.usda.gov/fnic/how-many-calories-are-one-gram-fat-carbohydrate-or-protein`

### 18.2 Reference used to Figure out the BMI Formula

`https://www.calculator.net/bmi-calculator.html`

### 18.3 Reference used to Figure out the BMR Formula

`https://www.omnicalculator.com/health/bmr`

### 18.4 Reference used to figure out formula to help the user loose or gain list

`https://www.mayoclinic.org/healthy-lifestyle/weight-loss/in-depth/calories/art-20048065`

### 18.5 Reference used to figure out the default values for the macro nutrients.

https://wa.kaiserpermanente.org/healthAndWellness?item=%2Fcommon%2FhealthAndWellness%2Fconditions%2Fdiabetes%2FfoodBalancing.html

### 18.6 References used for Purpose, Scope, System Variables, Controlled Variables, Constants, and for the Function and Non-Functional Requirements

Refer to our System Requirements Deliverable.