

**CONTEXT-AWARE REMINDER FOR FOOD JOURNAL MOBILE  
APPLICATION**

**BY  
HOE CHIA YONG**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**

**BACHELOR OF COMPUTER SCIENCE (HONS)**

**Faculty of Information and Communication Technology  
(Kampar Campus)**

**JUNE 2020**

UNIVERSITI TUNKU ABDUL RAHMAN

## REPORT STATUS DECLARATION FORM

**Title:** CONTEXT-AWARE REMINDER FOR FOOD JOURNAL MOBILE APPLICATION

**Academic Session:** JUNE 2020

I HOE CHIA YONG

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)



(Supervisor's signature)

**Address:**

2372, MK 15,  
Jalan Kampung Baru,  
14000 Bukit Mertajam, Pulau Pinang

Dr. Ooi Boon Yaik

Supervisor's name

**Date:** 9/9/2020

**Date:** 9/9/2020

**CONTEXT-AWARE REMINDER FOR FOOD JOURNAL MOBILE  
APPLICATION**

**BY  
HOE CHIA YONG**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**


**BACHELOR OF COMPUTER SCIENCE (HONS)**

**Faculty of Information and Communication Technology  
(Kampar Campus)**

**JUNE 2020**

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**CONTEXT-AWARE REMINDER FOR FOOD JOURNAL MOBILE APPLICATION**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : HOE CHIA YONG

Date : 9/9/2020

## **ACKNOWLEDGEMENTS**

First of all, I would like to express my sincere thanks and appreciation to my supervisor, Dr. Ooi Boon Yaik who has given me this opportunity to engage in this project. Aside from guidance on how to write a report, he taught me how to think and do in a scientific way as a Computer Science student. I learnt to be more independent to solve problem, trying not to seek for help easily, as he said we must have the ability to solve problem for customers when we are in career field.

Furthermore, I would like to say thanks to two of my friends for their patience, assistance when I am doing this project. They gave me some suggestion how to improve the report writing. Finally, I love my family for their continuous support and encouragement throughout the course.

## **ABSTRACT**

Most of the existing food journal apps in the market provide reminder functionality to notify users for logging a meal at specific mealtime each day. Yet, all these time-based reminders are not sophisticated enough because the users presently scheduling a reminder may not be able to predict a specific time. Since context-awareness in mobile computing is concerned with gathering information about the user current situation, this project proposes to develop a context-aware reminder for food journal mobile application, that utilises factors of time, device location, and user activity recognition to notify the users in a more efficient way. Provided the user has set the context-aware reminder at certain time interval, for instance 2 - 3 pm for Lunch. At 2 pm, the app will start tracking user device location using Fused Location Provider API. If the user is detected inside a restaurant circular region of specified radius for certain period, the app starts tracking step counter sensors value for activity recognition purposes. Then, if the user current step counter remains the same and still stay inside the restaurant region for certain period, the app assumes user is sitting for a meal in a restaurant within specific mealtime. The app then vibrates the device and shows a notification, which will prompt user for logging the meal he is taking. This project also showed the phone battery consumption when user device is receiving location updates using Fused Location Provider API, and tracking step counter sensor value is such simple yet effective way to detect sitting activity.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENTS</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Motivation	1
1.2 Project Scope	2
1.3 Project Objectives	2
1.4 Impact, Significance and Contribution	3
1.5 Background Study	3
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>5</b>
2.1 Review of Similar Existing Apps to Log Food Journal	5
2.1.1 Bitesnap	5
2.1.2 Lose It!	6
2.2 Google Android APIs	8
2.2.1 Fused Location Provider API	8
2.2.2 Geofencing API	8
2.2.3 Awareness API	9
2.2.4 Fence API	10
2.3 Smartphone Sensors	13
2.3.1 Motion Sensors	13
2.4 Mobile Device Location Detection Technologies	14
2.4.1 AGPS/ GPS	14
2.4.2 Wi-Fi Positioning System	15

2.5	Review of ‘Context Based Reminder System: Supporting Persons using Smartphone Accelerometer Data’	16
<b>CHAPTER 3</b>	<b>SYSTEM DESIGN</b>	<b>19</b>
3.1	Methodologies	19
3.2	System Design/ Overview	20
3.2.1	Use Case Diagram	20
3.2.2	Use Case Descriptions	21
3.2.3	Activity Diagrams	24
3.2.4	Class Diagram	28
<b>CHAPTER 4</b>	<b>IMPLEMENTATION</b>	<b>30</b>
4.1	Tools to Use	30
4.2	User Interface Design	31
4.3	App Function/ Module Implementation	34
4.3.1	Set Reminder	34
4.3.2	Location Tracking	37
4.3.3	Activity Recognition	39
4.3.4	Take Photo for Logging Journal	40
4.3.5	Edit/ Delete Food Journal	41
4.3.6	Navigate and View Food Journal	42
<b>CHAPTER 5</b>	<b>TESTING &amp; RESULTS</b>	<b>43</b>
5.1	Operational Testing of Context-Aware Reminder	43
5.2	Responsiveness Testing of Context-Aware Reminder	44
5.3	Responsiveness Testing Results	46
<b>CHAPTER 6</b>	<b>DISCUSSION</b>	<b>47</b>
6.1	Trigger Time of Alarm (for Time-based Operations)	47
6.2	Location Update Interval and Battery Consumption of Fused Location Provider API (for User Device Location Tracking)	47
6.3	Value of Step Counter (for User Sitting Activity Recognition)	49



<b>CHAPTER 7 CONCLUSION</b>	<b>50</b>
7.1 Project Review	50
7.2 Project Contribution	50
7.3 Future Work	51
 <b>BIBLIOGRAPHY</b>	 <b>53</b>
 <b>APPENDICES</b>	 <b>A1</b>

# LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	Bitesnap UI of homepage, logging a journal, and reminder.	6
Figure 2.2	LoseIT! UI of homepage, logging a journal, and reminder.	7
Figure 2.3	Geofencing API trigger event according to transition type.	8
Figure 2.4	Awareness API context types.	9
Figure 2.5	Flow chart of working theory of TimeFence.	10
Figure 2.6	Implementation theory of A-GPS.	14
Figure 2.7	Comparison between A-GPS and GPS.	15
Figure 2.8	Mathematical formulas to process raw accelerometer data.	16
Figure 2.9	$Sd_{rAcc}$ over 100 seconds from a participant.	17
Figure 2.10	Accuracy of activity recognition.	17
Figure 3.1	Use case diagram of food journal mobile application.	20
Figure 3.2	Activity diagram of 'Set Reminder'.	24
Figure 3.3	Activity diagram of 'Take Photo to Log Food Journal'.	25
Figure 3.4	Activity diagram of 'Edit Food Journal'.	26
Figure 3.5	Activity diagram of 'Delete Food Journal'.	27
Figure 3.6	Activity diagram of 'Navigate and View Food Journal'.	27
Figure 3.7	Class diagram of food journal mobile application.	29
Figure 4.1	App 'Main Screen' UI.	31
Figure 4.2	App 'Reminder' UI.	32
Figure 4.3	App 'Save Journal' UI.	32
Figure 4.4	App 'Update Journal' UI.	33
Figure 4.5	App 'Main Screen' interface.	34
Figure 4.6	User is asked for location accessing permission.	34

Figure 4.7	User can set the reminder time using hour, minute hands of popped-up clock dialog when he clicks button for specific mealtime.	35
Figure 4.8	User device is asked to switch on GPS first if GPS is not enabled.	35
Figure 4.9	Setting a reminder creates alarms.	36
Figure 4.10	Stop all the executing device location and step counter tracking.	36
Figure 4.11	Define longitude and latitude of a target restaurant.	37
Figure 4.12	Define function to track if the user device is near to the restaurant.	37
Figure 4.13	Define parameter settings and action to be triggered for location update.	37
Figure 4.14	Thread to check if user device is near to a restaurant for certain period.	38
Figure 4.15	User device location tracking is running in the background	38
Figure 4.16	Record step counter sensor value if user is detected walking.	39
Figure 4.17	Thread to check if user is sitting in a restaurant for certain period.	39
Figure 4.18	App notification is shown in the notification bar.	40
Figure 4.19	‘Save Journal’ interface to let user enter journal detail after taking a photo.	40
Figure 4.20	‘Main Screen’ interface display logged food journal photo.	41
Figure 4.21	‘Update Journal’ interface to let user update existing journal detail when user clicks a journal photo.	41
Figure 4.22	User can view the logged food journal of certain date using popped-up calendar interface when he clicks button showing current date.	42
Figure 5.1	Flowchart showing how to get expected time taken to trigger the reminder in Responsiveness Testing 1.	45

- Figure 5.2                      Box-and-whisker plot showing time taken to trigger 46 context-aware reminder in Responsiveness Testing 1 and 2.
- Figure 6.1                      Phone battery consumption when user device is 48 receiving location updates at 30 seconds and 120 seconds interval.

## LIST OF TABLES

Table Number	Title	Page
Table 2.1	Comparison Between Fence API and Proposed Method for Building Context-Aware Reminder.	12
Table 2.2	Android Smartphone Sensor Categories.	13
Table 3.1	Use Case Description for ‘Set Reminder’.	21
Table 3.2	Use Case Description for ‘Take Photo to Log Food Journal’.	22
Table 3.3	Use Case Description for ‘Edit Food Journal’.	22
Table 3.4	Use Case Description for ‘Delete Food Journal’.	23
Table 3.5	Use Case Description for ‘Navigate and View Food Journal’.	23
Table 4.1	Smartphone Hardware Specification.	30
Table 4.2	Laptop Hardware Specification.	30
Table 5.1	Operational Testing of Context-Aware Reminder in Different Conditions.	43
Table 5.2	Responsiveness Testing 1 Details.	44
Table 5.3	Responsiveness Testing 2 Details.	45

## LIST OF ABBREVIATIONS

<i>A-GPS</i>	Assisted Global Positioning System
<i>API</i>	Application Programming Interface
<i>BDS</i>	BeiDou Navigation Satellite System
<i>BTS</i>	Base Transceiver Station
<i>CASE</i>	Computer-Aided Software Engineering
<i>CDMA</i>	Code-division Multiple Access
<i>GLONASS</i>	Global Navigation Satellite System
<i>GPS</i>	Global Positioning System
<i>GSM</i>	Global System for Mobile Communications
<i>IDE</i>	Integrated Development Environment
<i>LTE</i>	Long-Term Evolution
<i>MAC</i>	Media Access Control
<i>RSSI</i>	Received Signal Strength Indication
<i>SSID</i>	Service Set Identifier
<i>UI</i>	User Interface
<i>UML</i>	Unified Modelling Language

## CHAPTER 1: INTRODUCTION

### 1.1 Problem Statement and Motivation

A food journal is a great tool to help people in tracking what they ate daily to achieve several purposes such as weight loss goals, maintain healthy eating habits or monitor food allergies. In this digital era, the journal logging process become easier and convenient, as nearly everyone having one smartphone which can use fitness, food diary apps with more advanced features.

However, most users may be busy working and forget to use the apps to log any meal taken daily. While most of those apps provide reminder functionality to notify users in the phone notification bar to log a meal at specific mealtime, these time-based reminders, are not efficient (MyFitnessPal, Inc. 2019; FitNow, Inc. 2019; Azumio, Inc. 2019; Bite AI 2019). A time-based reminder may be set to trigger at some arbitrarily selected time which as it turns out may be inconvenient, because the users presently scheduling a reminder may not be able to predict a specific time.

Consider a case that a user set a reminder for lunch in the app. The user might forget about the reminder easily in situation such as before reminder is shown up, he has taken lunch already, or after reminder is shown up, he does not look at the phone when having the meal and thus forget about it. When part of the users' food journal are missing entries, they feel that the journal becomes less inaccurate and lazy to log food journal persistently. Eventually, their journaling habit will be gradually undermined. (Cordeiro et al. 2015).

Instead, the reminder should only triggers when the user is in a specific location such as a restaurant, cafe or when the user is performing certain activity such as sitting for a meal, within specified time interval. Therefore, there are two problems need to be tackled to encourage users to use a mobile application for logging a food journal, which can be summarised as statements below:

- How to measure activities (sitting or non-sitting) using smartphone sensor technology, and use the measurement results to automate reminder to the users?
- How to design the reminder to be a support for a food journal mobile application so that users will be reminded in a more efficient way to log every meal they are taking?

## **1.2 Project Scope**

The outcome of this project is not just a food journal mobile application, but also an efficient way to remind users to log every meal taken daily, by applying concept of context-awareness.

At certain time interval like lunch time, the app will keep on tracking user device location using Fused Location Provider API at specific interval to determine if the user stays in a target restaurant (defined by longitude and latitude), then use the step counter sensor to recognise user's current activity.

If the app assumes user is sitting for a meal in a restaurant within specific mealtime, the device shall vibrate, and a notification will be shown. Once the user clicks it, he will be prompted to camera screen of the app for logging the meal he is taking. The logged food journal photo and details will be finally saved to app-specific storage and local Room database respectively.

## **1.3 Project Objectives**

This project proposes to develop a context-aware reminder for food journal mobile application, so that users are reminded to use the app when he is sitting for a meal in a restaurant within specific mealtime. The specific objectives of this project:

- To develop a context-aware reminder that can identify users' activity (still or moving) of concern using smartphone step counter sensor.
- To enhance the context-aware reminder with time-based and location-based features to detect if the user is having a meal in a restaurant for specific mealtime.



## **1.4 Impact, Significance and Contribution**

Aside from supporting food journal app, the context-aware reminder proposed by this project can be support for many existing apps in the market. For instance, petrol fuelling app reminds user to get the member card points when fuelling petrol at the station, shopping mall app reminds user to buy certain commodities in shopping mall during weekend, or even specific reminder app reminds user to take his personal belongings such as key and wallet when he leaves home.

Furthermore, this project showed the phone battery consumption when user device is receiving location updates in the background using Fused Location Provider API (with highest priority and 30 seconds of update interval settings), which is about 4 mAh per 2 minutes. The same observation is made by changing update interval to 120 seconds for battery saving purposes. The results showed that the difference of battery consumption is very small within short period of time like interval of one hour, and only become bigger and obvious if the user device location tracking is going to run for long period of time.

Lastly, this project showed that in the context of merely recognising user activity whether he is still or moving, using Android built in step counter sensor is a very simple yet effective way by just tracking its value compared to other methods such as using Awareness API, or calculating phone accelerometer value.

## **1.5 Background Study**

### Food Journal

A food journal is a log of your daily food intake, may include nutritional or other dietary information, such as amount of calories intake and body symptoms happened after certain diet. Writing a food diary can help individuals understand their recent eating habits, find out where extra calories come from, and develop a scientific diet plan accordingly. For people with allergic reactions, writing a food diary can be used to determine the relationship between allergies and eating, and to find clues to allergenic food. A food diary also will give your doctor or health care provider a quick way to understand your eating habits and check your progress.

In this modern era, many people are interested in tracking what they ate each day to help them achieve weight loss goals, maintain healthy eating habits or monitor food allergies, using health and fitness or calorie-counter mobile applications like MyFitnessPal, Lose It!, etc. (MyFitnessPal, Inc. 2019; FitNow, Inc. 2019). These apps track user diet and exercise to calculate optimal nutrients and caloric intake for the personal goals. These apps also use gamification elements to motivate users. Besides, users can search the comprehensive food database to find the food item they have taken each day. Then by selecting the serving size and quantity, all nutrition information will be automatically calculated and then stored into their food journal. These apps make the process of logging food journal convenient and easy, especially with more advanced technology of Artificial Intelligence in recent years - some of these existing mobile applications in the market like CalorieMama, Bitesnap have started to apply Artificial Intelligence into the application features to perform image-based food recognition, to assist users easily log their daily meals (Azumio, Inc. 2019; Bite AI 2019).

### Context-Awareness

Context awareness is a term originated from ubiquitous or pervasive computing - a computer science concept where presence, appearing of computing are made anywhere and anytime (Ubiquitous computing 2020). It refers to the property of mobile devices that is able to adapt behaviours according to the information gathered from its environment at any given time.

This term is defined complementarity to location awareness. While context is perceived as a matter of user location initially, as time changes, context is applied more flexibly with mobile users especially with smartphone users. Any data that can be collected and analysed from the surrounding, environment such as temperature, location, object existence, voice, etc. can be regarded as context. Cameras, sensors, microphones, GPS receivers, etc. are all potential sources for context.

In short, context-aware systems emphasises three key phases: the acquisition of context, then understand the context, and finally adaption of application behaviour according to the recognised context (Context awareness 2019).

## **CHAPTER 2: LITERATURE REVIEW**

This chapter firstly reviews the similar existing food journal apps about the UI and functionalities of logging journal and reminder. Then, to know what tools can be used to build a context-aware reminder, several Google Android APIs, smartphone sensors and mobile device location detection technologies are reviewed. Lastly, this chapter also reviews an approach of building context-based reminder system using smartphone accelerometer.

### **2.1 Review of Similar Existing Apps to Log Food Journal**

#### **2.1.1 Bitesnap**

Bitesnap is a food journal application that uses Artificial Intelligence to log a meal by taking a photo and key in a few details such as food name and serving size, to save user time and make it simple to build healthy eating habits. It analyses the image, and automatically calculates the calories and nutrients of the food. As users log meals, this app displays an Instagram-style feed of pictures, each with nutritional facts (Bite AI 2019).

Besides image-based food recognition, user can add entries into his food journal by scanning barcode of a packaged product, or search through Bitesnap food datasets. In this context, user can add new food type with custom calories, nutritional contents to his own food dataset to be searched. Bitesnap also provides a daily nutrient card that shows nutritional breakdown of what user has eaten, in a pie chart format depicting how much carbohydrate, protein, and fat is in the food, and a bar chart depicting adjustable target to guide user on how much of each nutrient to eat. Bitesnap provides reminders by showing app icon in notification bar that remind user to eat at customisable time for four mealtime categories: Breakfast, Lunch, Dinner, End of Day (Bite AI 2019).

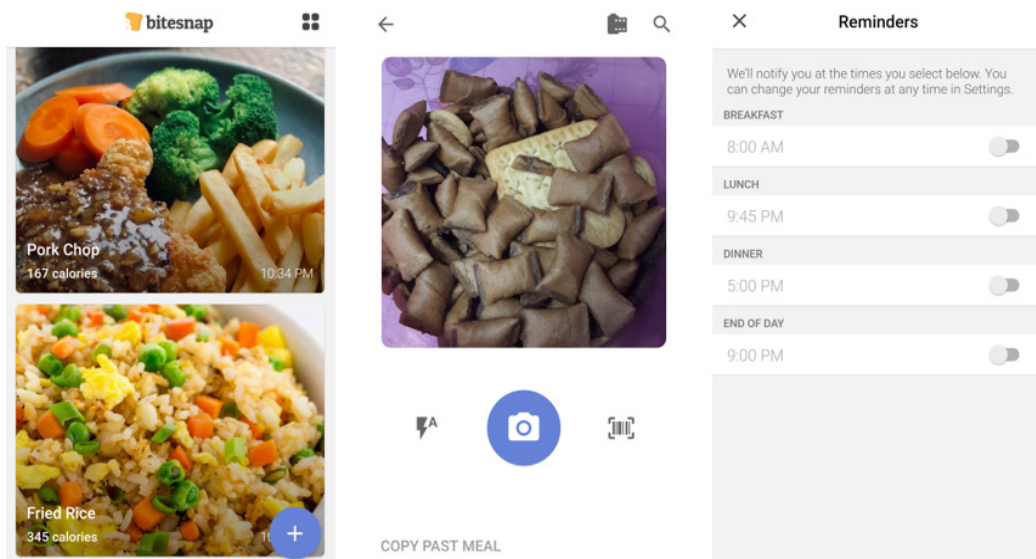


Figure 2.1 - Bitesnap UI of homepage, logging a journal, and reminder (Bite AI 2019).

#### *Strengths:*

- Simple and intuitive user interface design that makes it easy to capture and see what users eat in a day
- Export the log of food journal as CSV or JSON file and share it with professionals

#### *Weaknesses:*

- The food journal is presented in the form of 'Instagram-style feed of photos', which means users have to keep scrolling and scrolling to get to a meal information from a few days ago
- The food logged that presented on the homepage are not categorised into any mealtime category: Breakfast, Lunch and Dinner, which causes users difficult to identify the specific mealtime taken

### **2.1.2 Lose It!**

Lose It! is a calorie-counter app and website that includes an easy-to-use food diary and exercise log. When users first log in to the app, they are asked to fill up personal details and specify target weight to be lost each week, which will affect the calorie totals suggested by the app. Then, the app will display a bar graph showing users' daily calorie

budget on the homepage - how many calories are left to eat throughout the day, deduct any exercise users recorded (FitNow, Inc. 2019).

Users add foods to their log through searching by keyword, or barcode scanner for products. The app has comprehensive food database complete with well-known brand-name foods, grocery stores, restaurants that verified by team of experts, but in case it is lacking something, user can also add the food with custom details created by themselves manually. Lose It! provides reminders by showing app icon in notification bar that remind user to eat at customisable time for four mealtime categories: Breakfast, Lunch, Dinner, End of Day (FitNow, Inc. 2019).

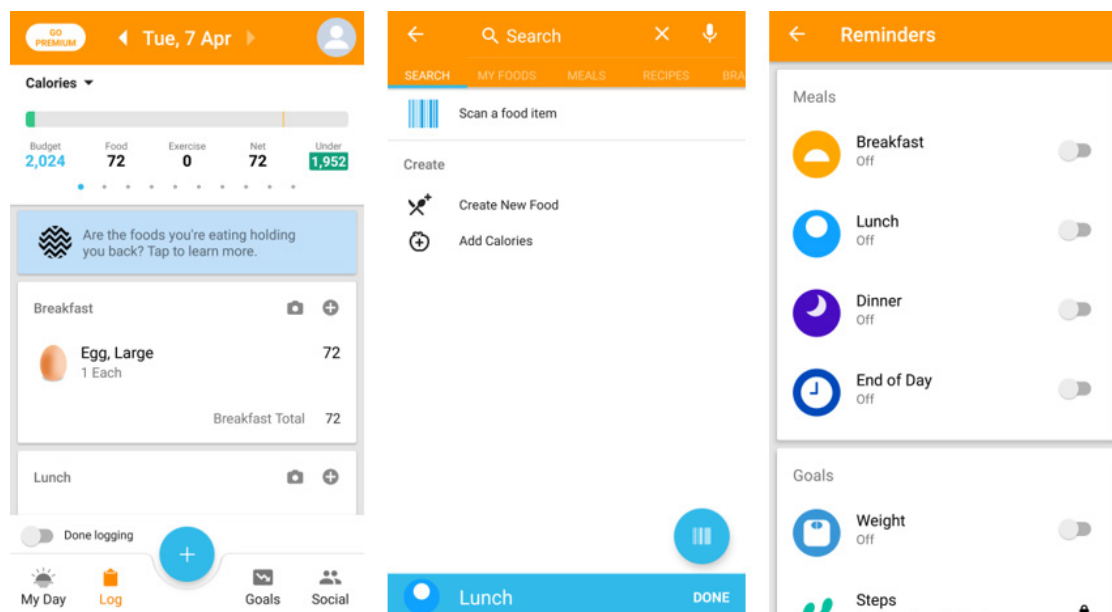


Figure 2.2 - LoseIt! UI of homepage, logging a journal, and reminder (FitNow, Inc. 2019).

#### *Strength:*

- The user can retrieve any daily meal info logged easily using calendar on the homepage to navigate

#### *Weaknesses:*

- The app user interface is somehow complicated, quite tricky to navigate.
- The app does not allow user to take photo of food for when logging, instead can only select specified meal from database or through manual typing only

## 2.2 Google Android APIs

### 2.2.1 Fused Location Provider API

This API combines multiple signals to deliver the location information to an app by managing the underlying location technologies like GPS, and provides a simple interface to allow user to select the required quality of service. The API is commonly used for requesting the last known location of the user's device, and also delivering location updates to an app at specific intervals. In this context, the API also allows user to change location settings, such as specifying the desired update interval, required level of accuracy and power consumption (Fused Location Provider API 2020).

### 2.2.2 Geofencing API

This API allows user to define a circular region - geofences, that surround the areas of interest by specifying its latitude, longitude, radius and transition type. The transition types indicate the events that trigger the geofence: when user enters or exit a geofence, stay in the geofence for certain interval. Then, the app shall handle the event by doing certain action (Geofencing API 2020).

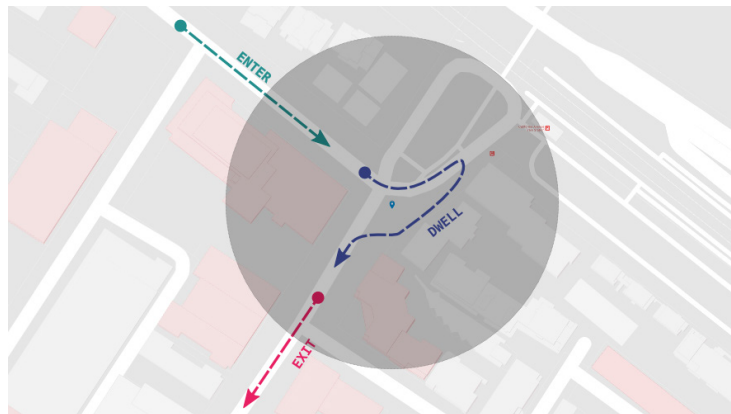


Figure 2.3 - Geofencing API trigger event according to transition type (Geofencing API 2020).

### 2.2.3 Awareness API

Awareness API unifies five location and context signals in a single API, enable user to create an app which intelligently react to the user's current situation, with minimal impact on system resources (What's the Awareness API 2020).

The five different context types exposed:

Context type	Example
Time	Current local time
Location	Latitude and longitude
Activity	Detected user activity, like walking, running, or biking
Beacons	Nearby beacons that match the specified namespace
Headphones	Status of whether headphones are plugged in, or not

Figure 2.4 - Awareness API context types (What's the Awareness API 2020).

The Awareness API provides some benefits:

- Ease of implementation - The user only needs to add a single API, which utilise a group of location and context signals, for creating a context-aware app. This greatly simplifies integration and improves the productivity (What's the Awareness API 2020).
- Better context data - Intelligently process and combine raw signals from multiples sources for maximum accuracy and efficiency. Advanced algorithms are used to recognise the user's activity accurately (What's the Awareness API 2020).
- Optimal system health - Automatically optimise memory usage and power consumption to maximise memory capacity and battery life on the users' device (What's the Awareness API 2020).

The Awareness API consists of two distinct APIs:

- Snapshot API - Get instant details about the user's current situation, by accessing those five signals from single API surface (What's the Awareness API 2020).
- Fence API – Building on top of the concept of Geofencing API to include other four context signals besides only using 'location' context type, to create a geofence that reacts to the user's environment changes (What's the Awareness API 2020).

### 2.2.4 Fence API

Fence API can be used to create five types of AwarenessFence, three of which are TimeFence, LocationFence, and ActivityFence which are very convenient in building a context-aware app that is able to track user device location and recognise user activity within certain time interval (Fence API overview 2019).

#### TimeFence

Unlike Alarm which trigger time-based operation at specific time set by user, the TimeFence does not trigger anything, but it functions more like time interval condition checking (Fence API overview 2019).

For instance, the user creates a combined AwarenessFence from TimeFence of 2 - 3 pm and ActivityFence of detecting user walking. When the device system time reaches 2 pm, nothing will be triggered. Instead, if that ActivityFence is triggered, then Fence API check if current system time is within 2 - 3 pm, if yes then only trigger the AwarenessFence event. Figure 2.5 shows how the TimeFence works (Fence API overview 2019).

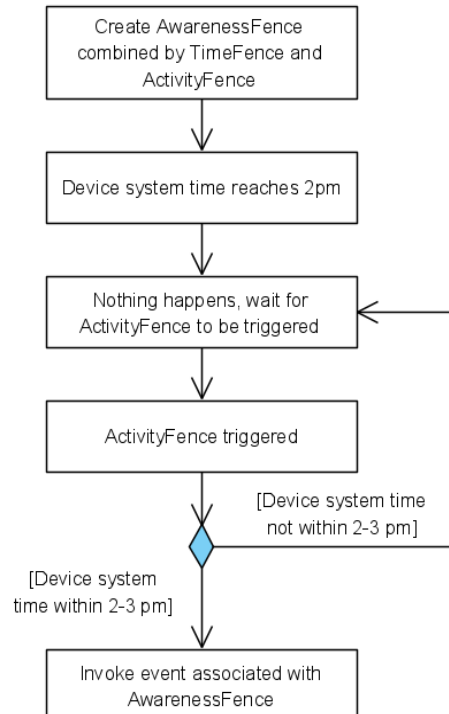


Figure 2.5 - Flow chart of working theory of TimeFence (Fence API overview 2019).



Since TimeFence does not trigger anything, it has no meaning to be used alone but can only cooperate with other types of AwarenessFence for time interval condition checking purpose.

### **LocationFence**

LocationFence acts just like Geofencing API. It uses only Wi-Fi and cell tower positioning for tracking user device location. LocationFence delivers location update once every 2-3 minutes on average. The location update interval may decrease to less than 2 minutes if user device is detected moving, and increases up to 6 minutes if user device is detected stationary for long period of time (Create and monitor geofences 2020).

However, LocationFence cannot change the priority of the location update request to use GPS sensor, which may cause poor performance of user device location tracking in indoor position without Wi-Fi available, and in outdoor position without strong signal from cell tower. Besides, LocationFence cannot change the update interval of the location update request (Create and monitor geofences 2020).

### **ActivityFence**

ActivityFence acts just like another Android API - Activity Recognition API, which automatically detects user activities by utilising device built-in sensors. It periodically applies machine learning models to process short bursts of sensor data read. If the device has been still for a while, the API may stop activity reporting and only resumes reporting using low power sensors when it detects movement, for the sake of resource optimisation (Activity Recognition API 2020).

However, since the exact period of ActivityFence to process the sensor data read is not known, the activity recognition result tends to be not accurate (Activity Recognition API 2020). For instance, an ActivityFence of detecting if user is walking is created. In this case, two situations might happen:

- i. The ActivityFence is not processing the sensors data when the user is walking, but only after the user stopped walking (Activity Recognition API 2020).
- ii. Once the ActivityFence is triggered, it might not be triggered again next time when user stops and walks again (Activity Recognition API 2020).

### Critical Remarks

<b>ActivityFence</b>	<b>Proposed method using step counter sensor</b>
Once the ActivityFence of detecting user is being still is triggered, it might not be triggered again, even user walks for few minutes then sits again.	When the user walks for certain seconds, the step counter increases, and reminder can confirm user is moving. In contrast, when user is not moving, step counter remains, and reminder can confirm user is not moving.  According to developer, he decides the duration of unchanged step counter value that will assume user is sitting.
<b>TimeFence</b>	<b>Proposed method using Alarm</b>
TimeFence does not trigger anything. It can be only combined with another type of AwarenessFence created by Fence API to perform condition checking.	Alarm is used to trigger time-based operation of user device location tracking.
<b>LocationFence</b>	<b>Proposed method using Fused Location Provider API</b>
Cannot change priority and update interval of location update request.  LocationFence uses only Wi-Fi and cell tower positioning for tracking user device location - performs poorly in indoor position without Wi-Fi available, and outdoor position without good signal from cell tower.	Can change the priority and update interval of location update request.

Table 2.1 - Comparison Between Fence API and Proposed Method for Building Context-Aware Reminder.

The developer has no control of the implementation details using Fence API (of Awareness API) - unable to set the exact time of alarms to trigger an operation, unable to configure the location services accuracy or location update interval, unable to determine and make any configuration to the period of Fence API applying machine learning models to process short bursts of sensor data. These situations cause unreliability on the accuracy aspect of a context-aware system to be built.

## 2.3 Smartphone Sensors

Most Android smartphones have sensors that are able to measure motion, orientation, and various environmental conditions (Sensors 2019). Android provides API from which the user can derive precise and accurate raw data. There are three broad categories of sensors:

Sensors Type	Measurement Value	Examples
Motion	Acceleration and rotational forces along three axes	Accelerometers, gyroscopes
Environmental	Ambient air temperature and pressure, humidity, illumination	Barometers, thermometers
Position	Physical position of device	Orientation sensors, magnetometers

Table 2.2 - Android Smartphones Sensor Categories (Sensors 2019).

### 2.3.1 Motion sensors

Motion sensors can be divided into:

- Hardware-based sensors, such as accelerometer and gyroscope sensors.
- Software-based sensors, such as linear accelerator, rotation vector, step counter and gravity sensors.

An accelerometer sensor measures the tilting motion and orientation of a device by measuring acceleration values along the x, y and z axis, while a gyroscope sensor is used for measuring, maintaining angular velocity and orientation (Sensors 2019).

Software-based sensors are sometimes known as virtual sensors. Unlike hardware-based sensors which report results directly, this type of sensors derives the data of one or more hardware-based sensors by specific computation formula. For example, linear accelerator data is derived from accelerometer. Step counter is one of the software-based motion sensors that used by this project for activity recognition purposes. It provides the number of user steps taken since the last reboot of device. The step counter has latency up to 10 seconds (Sensors 2019).

## 2.4 Mobile Device Location Detection Technologies

### 2.4.1 A-GPS/ GPS

A-GPS (Assisted Global Positioning System) is a system extensively employed in GPS-capable cellular phones initially, to significantly improve the start-up performance of GPS, so that emergency call dispatchers can get cell phone location data easily (Assisted GPS 2020).

GPS works by communicating with satellites through trilateration - a process that obtains the intersection points from three (usually four) or above of the radius coverage of satellites to determine device location information. On the other hand, A-GPS has done the improvement on GPS which reduces the response delay, by determining location information from base transceiver station (BTS) using technology of CDMA, GSM or LTE employed in the mobile terminal. The BTS in turn communicate with the satellites (GPS vs A-GPS 2012).

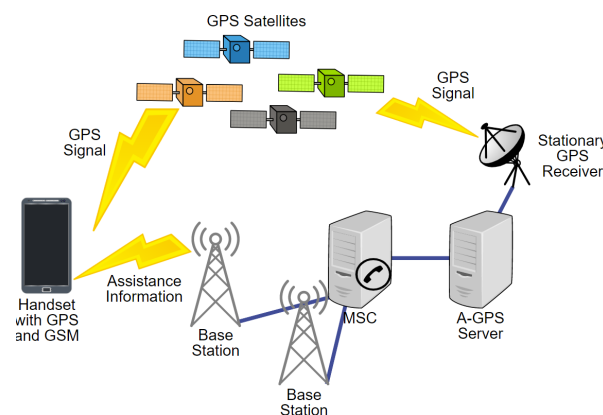


Figure 2.6 - Implementation theory of A-GPS (Assisted GPS 2020).

Since GPS devices are communicating with satellite, it can get information only when satellite is reachable without any interference, for example under clear sky conditions or outdoor area. Also, it is slower in requesting information from satellite. While A-GPS devices are communicating with BTS instead of satellites, it gets information even when the sky condition is cloudy, but not during network is unreachable. In this context, A-GPS devices will fall back and work as regular GPS. Also, A-GPS devices are generally faster in getting response (GPS vs A-GPS 2012).

	<b>A-GPS</b>	<b>GPS</b>
<b>Stands for</b>	Assisted Global Positioning System	Global Positioning System
<b>Source of triangulation information</b>	Radio signals from satellites and assistance servers e.g. mobile network cell sites	Radio signals from GPS satellites
<b>Speed</b>	A-GPS devices determine location coordinates faster because they have better connectivity with cell sites than directly with satellites.	GPS devices may take several minutes to determine their location because it takes longer to establish connectivity with 4 satellites.
<b>Reliability</b>	Location determined via A-GPS are slightly less accurate than GPS	GPS devices can determine location coordinates to within 1 meter accuracy
<b>Cost</b>	It costs money to use A-GPS devices on an ongoing basis because they use mobile network resources.	GPS devices communicate directly with satellites for free. There is no cost of operation once the device is paid for.
<b>Usage</b>	Mobile phones	Cars, planes, ships/boats

Figure 2.7 - Comparison between A-GPS and GPS (A-GPS vs GPS 2011).

#### 2.4.2 Wi-Fi Positioning System

Wi-Fi positioning system is a geolocation system that measures the RSSI of nearby Wi-Fi hotspots and also other wireless access points to determine the device location. It is mainly used where satellite navigation like GPS is not performing well due to various conditions such as signal blockage indoors, or taking too long time to acquire a satellite response (Wi-Fi positioning system 2020).

The accuracy of this system depends on how many nearby access points whose positions that are identified by a device SSID and MAC address, have been saved into the database. In this context, using these known points as reference, trilateration algorithms can be applied to further improve the accuracy (Wi-Fi positioning system 2020).

While Wi-Fi positioning system performs much better for tracking position in indoor compared to GPS, its accuracy is much lower than GPS.

## 2.5 Review of ‘Context Based Reminder System: Supporting Persons using Smartphone Accelerometer Data’

This thesis proposed an alarm system that based on activity recognition, which can be recognised as a context-aware reminder system, that can be identified from built-in smartphone accelerometer. An experiment using an activity recognition algorithm was devised to identify a particular activity. In this study the activities were walking and sitting. These two activities were recognised by calculating standard deviation of raw accelerometer data (Khan & Khan 2013).

### Data Collection

A simple data recording application was developed to collect users’ walking activity data using smartphone sensor: accelerometer which returns the acceleration values along x, y, z axis in the units of  $m/s^2$  and stored in a data file. Data acquisition rate was 10 samples per second. After that, this project invited 10 participants, then installed the data recording application in the smartphone and asked them to use that application during walking, by putting the smartphone inside their pocket for 100 seconds. Eventually, for each participant, total of  $10 \times 100 = 1000$  samples of raw accelerometer data were collected (Khan & Khan 2013).

### Data Processing

Then those data files collected from 10 participants were processed using mathematical formulas defined in Figure 2.2 to get  $Sd_{rAcc}$ , the standard deviation of resultant accelerometer values (Khan & Khan 2013).

Features within one second (10 samples)	Description
Mean of Accelerometer x, y, z axis	$\mu_x = \frac{1}{10} \sum_{i=1}^{10} X_i$ $\mu_y = \frac{1}{10} \sum_{i=1}^{10} Y_i$ $\mu_z = \frac{1}{10} \sum_{i=1}^{10} Z_i$
Resultant Accelerometer values	$rAcc_i = \sqrt{X^2 + Y^2 + Z^2}$
Mean of resultant Accelerometer value	$\mu_{rAcc} = \frac{1}{10} \sum_{i=1}^{10} rAcc_i$
Standard deviation of resultant accelerometer values	$Sd_{rAcc} = \sqrt{\frac{\sum_{i=1}^{10} (rAcc_i - \mu_{rAcc})^2}{9}}$

Figure 2.8 - Mathematical formulas to process raw accelerometer data (Khan & Khan 2013).

### Participant Walking Activity Plots

For each participant, a line graph with y-axis:  $Sd_{rAcc}$  was plotted against x-axis: 100 seconds, as the experiment is conducted by 100 seconds using MATLAB (Khan & Khan 2013).

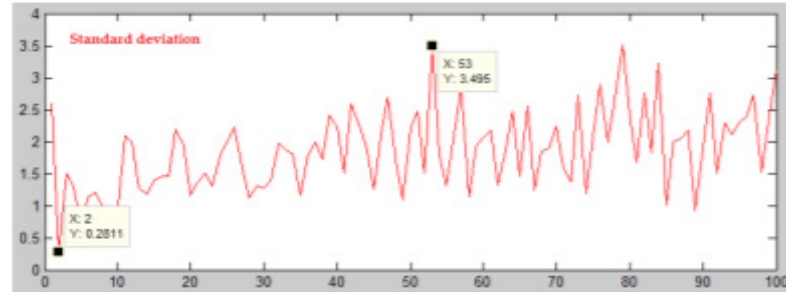


Figure 2.9 -  $Sd_{rAcc}$  over 100 seconds from a participant (Khan & Khan 2013).

### Experiment Result

The line graph plotted was used to determine the lower bound and upper bound of  $Sd_{rAcc}$  over 100 seconds for each participant. As there were 10 participants, the minimum lower bound and maximum upper bound were identified among these participants. Lastly, this minimum lower bound of  $Sd_{rAcc}$  for walking activity was identified to be 0.1316, which represents the threshold value between sitting and walking activity (Khan & Khan 2013).

### Activity Recognition

The threshold value, produced by the activity recognition algorithm, is used by the proposed alarm system to identify activity performed by a user, and then to decide for alarm prompting. If  $Sd_{rAcc}$  calculated from user smartphone accelerometer sensor was less than threshold value (sitting), alarm will be activated otherwise (walking), alarm will be turned off (Khan & Khan 2013).

### Analysis and Performance of Activity Recognition Algorithm

3 algorithms: Naïve bayes, Decision Trees (C4.5) and Multilayer perceptron were used to test the activity recognition algorithm accuracy (Khan & Khan 2013).

	% Records Correctly Predicted		
	Naïve bayes	C4.5	Multilayer perceptron
Walking	95.91	100	93.87
Sitting	100	98.03	100

Figure 2.10 - Accuracy of activity recognition (Khan & Khan 2013).

### Critical Remarks

The accuracy of activity recognition algorithm is very high, and can be used for reminder of other system/ apps to identify human sitting or walking other than the proposed alarm system. However, this alarm system proposed does not have location-based feature which decrease the practicability because it might trigger at any location.



## CHAPTER 3: SYSTEM DESIGN

This chapter defines the methodology for realising this project, and the design of building the food journal app with context-aware reminder from the perspective of use case diagram, use case descriptions, activity diagrams and class diagram.

### 3.1 Methodologies

The methodology for realising this project will be the Throwaway Prototyping, as development of context-aware reminder are not well understood. Therefore, the app development is examined by analysing, designing then building a design prototype which allow user to evaluate the proposal by trying the reminder and give feedback.

- Planning: Discuss title with FYP supervisor, study the related background information, define problem statement, project scope and objectives to achieve.
- Analysis: Do literature review on similar existing apps in the market, mobile device location detection technologies, Android APIs, similar solution proposed by others.
- Design: Define the use case diagram and class diagram for the app, use case descriptions and activity diagrams.
- Implementation: Realising system design by start coding, sketching UI, testing and debugging.

Then, this methodology repeatedly performs the three phases: analysis, design, implementation concurrently in a cycle until the design prototypes are thrown away – context-aware reminder is built. Finally, the project progresses into design and implementation phases for developing the food journal mobile application.

## 3.2 System Design/ Overview

### 3.2.1 Use Case Diagram

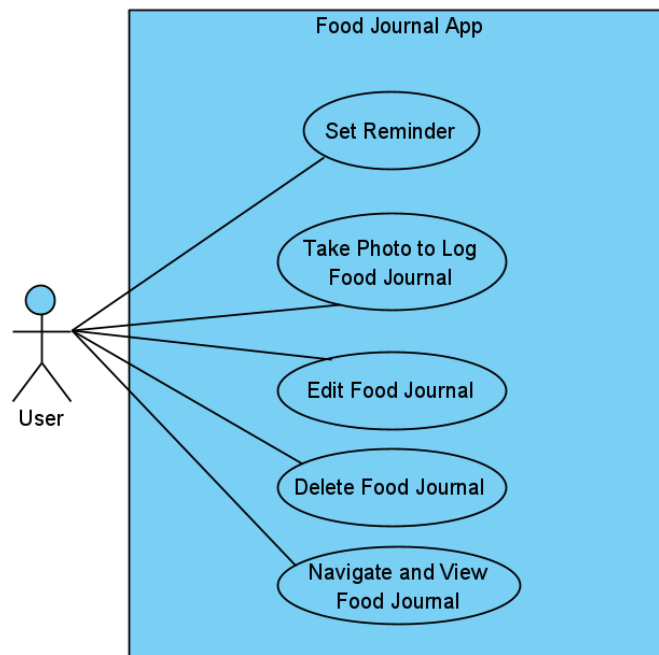


Figure 3.1 - Use case diagram of food journal mobile application.

This food journal mobile application contains five use cases:

- i. **Set Reminder:** User can set the context-aware reminder for specific mealtime: Breakfast, Lunch, Dinner. The reminder creates time-based alarm, which will trigger the app to start tracking user device location and step counter. When the user is assumed to be sitting in a restaurant within specific mealtime, the app vibrates the phone and shows a notification in notification bar. By clicking the notification, the user will be prompted to camera screen to take photo and enter details for logging a food journal.
- ii. **Take Photo to Log Food Journal:** Besides using the reminder to log a food journal, user can do the same thing on the app main screen. The food journal photo and details will be saved into app-specific storage, local Room database respectively.
- iii. **Edit Food Journal:** User can update the food journal details stored in local Room database.
- iv. **Delete Food Journal:** User can delete the food journal details stored in local Room database.

- v. Navigate and View Food Journal: User can view the food journal logged on certain date on the app main screen with the aid of a calendar interface.

### 3.2.2 Use Case Descriptions

<b>Use Case</b>	Set Reminder	<b>Use Case ID</b>	1
<b>Actor</b>	User		
<b>Description</b>	To remind user by vibrating device and showing notification when user is sitting in a restaurant during specific mealtime.		
<b>Trigger</b>	User toggles on the switch for specific mealtime.		
<b>Precondition</b>	User device must have location service including GPS enabled.		
<b>Normal flow of events</b>	<ol style="list-style-type: none"> <li>1. User selects certain time interval for specific mealtime: Breakfast, Lunch or Dinner, for instance 2 – 3 pm for Breakfast.</li> <li>2. User switches on reminder.</li> <li>3. System creates two alarms: one to trigger <i>step 4</i> at the start of mealtime, another one will stop the services of tracking user device location and step counter at the end of mealtime.</li> <li>4. System is delivering the device location updates at specific interval.</li> <li>5. System checks if the user device stays in a restaurant location circular region with certain radius within certain period. If so, proceed to <i>step 6</i>.</li> <li>6. System is delivering the latest device step counter sensor value.</li> <li>7. System checks if the user current step counter remains same and user device remains staying in the restaurant location region within certain period. If so, proceed to <i>step 8</i>.</li> <li>8. System stops the executing user device location and step counter tracking.</li> <li>9. System vibrates the device and shows notification.</li> <li>10. User taps the notification shown and will be prompted to camera screen of the app.</li> <li>11. User takes photo of the food and enter detail.</li> <li>12. User confirms the logging of journal by clicking ‘SAVE’ button.</li> <li>13. System saves the logged food journal photo and detail into app-specific storage and local Room database respectively.</li> <li>14. System displays the food journal photo on the app main screen.</li> </ol>		

<b>Subflows</b>	5a. If the user device stays outside restaurant location circular region with certain radius within certain period, repeat <i>step 5</i> .  7a. If the user current step counter changes within certain period, repeat <i>step 7</i> .
<b>Alternate flow</b>	12a. The user does not save the logging of journal, by either clicking 'return', or exit the app.

Table 3.1 - Use Case Description for 'Set Reminder'.

<b>Use Case</b>	Take Photo to Log Food Journal	<b>Use Case ID</b>	2
<b>Actor</b>	User		
<b>Description</b>	To allow user take photo of the food and enter detail, for logging food journal.		
<b>Trigger</b>	Click the '+' button on the top right corner of app main screen.		
<b>Normal flow of events</b>	<ol style="list-style-type: none"> <li>1. User takes photo of the food and enter detail.</li> <li>2. User confirms the logging of journal by clicking 'SAVE' button.</li> <li>3. System saves the logged food journal photo and detail into app-specific storage and local Room database respectively.</li> <li>4. System displays the food journal photo on the app main screen</li> </ol>		
<b>Alternate flow</b>	2a. The user does not save the logging of journal, by either clicking 'return', or exit the app.		

Table 3.2 - Use Case Description for 'Take Photo to Log Food Journal'.

<b>Use Case</b>	Edit Food Journal	<b>Use Case ID</b>	3
<b>Actor</b>	User		
<b>Description</b>	To allow user to edit the food journal detail.		
<b>Trigger</b>	Click the desired food journal photo displayed on the app main screen.		
<b>Normal flow of events</b>	<ol style="list-style-type: none"> <li>1. User edits the food journal detail.</li> <li>2. User confirms the change by clicking 'UPDATE' button.</li> <li>3. System updates the modified journal detail in local Room database.</li> <li>4. System displays the food journal photo on the app main screen.</li> </ol>		
<b>Alternate flow</b>	2a. The user does not save the changes of journal, by either clicking 'return', or exit the app.		

Table 3.3 - Use Case Description for 'Edit Food Journal'.

<b>Use Case</b>	Delete Food Journal	<b>Use Case ID</b>	4
<b>Actor</b>	User		
<b>Description</b>	To allow user to delete the food journal logged.		
<b>Trigger</b>	Click the desired food journal photo displayed on the app main screen.		
<b>Normal flow of events</b>	<ol style="list-style-type: none"> <li>1. User clicks 'DELETE' button.</li> <li>2. System deletes the food journal detail in local Room database</li> <li>3. System displays the food journal photo on the app main screen.</li> </ol>		
<b>Alternate flow</b>	1a. The user does not delete the journal, by either clicking 'return', or exit the app.		

Table 3.4 - Use Case Description for 'Delete Food Journal'.

<b>Use Case</b>	Navigate and View Food Journal	<b>Use Case ID</b>	5
<b>Actor</b>	User		
<b>Description</b>	To allow user view daily food journal logged by selecting certain date on calendar interface.		
<b>Trigger</b>	Click the 'current date' on the top middle of app main screen, which will display a calendar interface, then select desired date.		
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. User selects certain date on calendar interface.</li> <li>2. System displays the food journal photo of that date on the app main screen.</li> </ol>		

Table 3.5 - Use Case Description for 'Navigate and View Food Journal'.

### 3.2.3 Activity Diagrams

Figure 3.2 shows activity diagrams of use case ‘Set Reminder’: Given a user set the reminder for specific mealtime, the reminder creates time-based alarm, which will trigger the app to start tracking user device location and step counter. If the location is near to a target restaurant and step counter value remains unchanged within certain period, the user is assumed to be sitting in a restaurant within specific mealtime, the app vibrates the phone and shows a notification in notification bar.

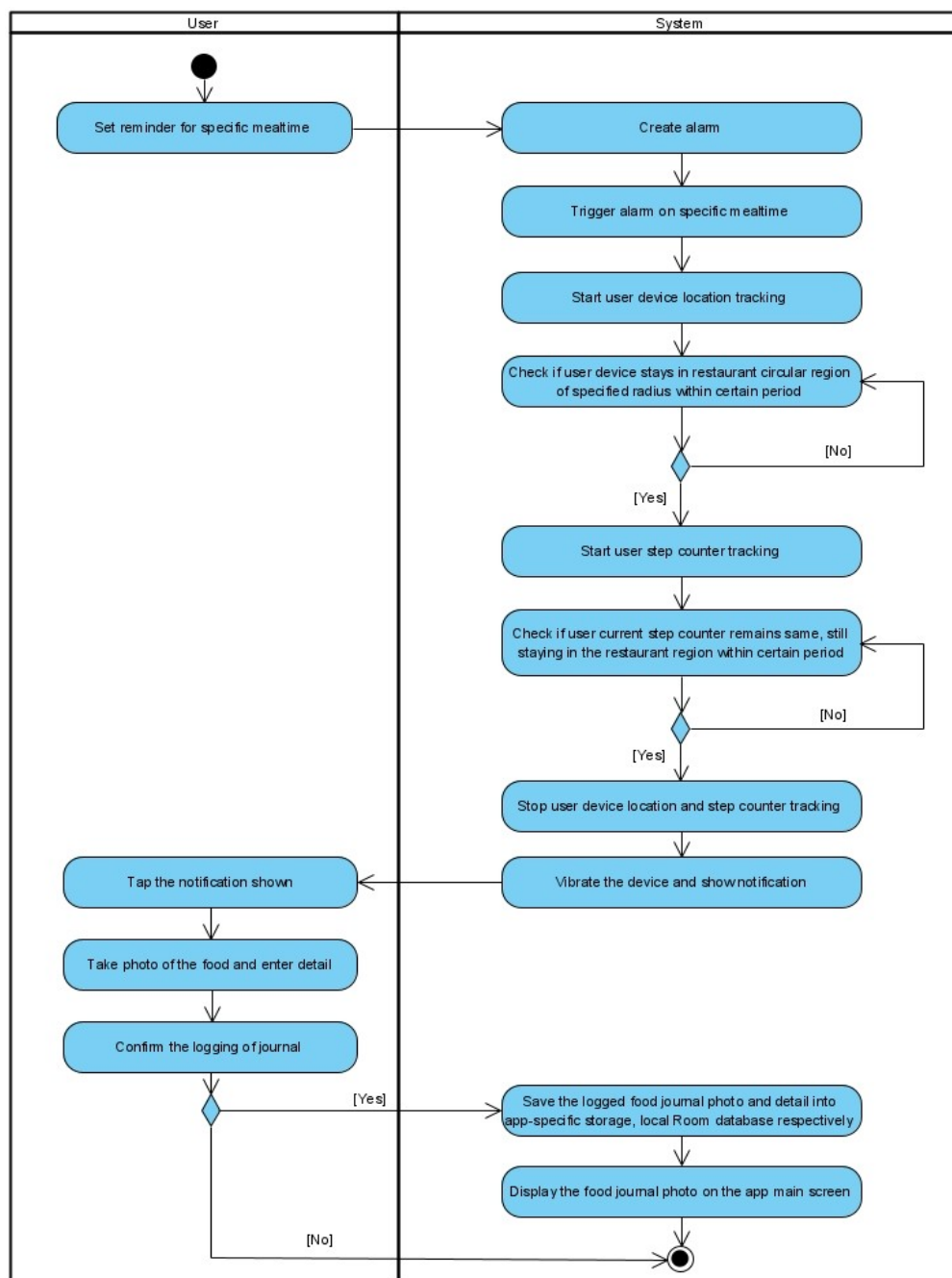


Figure 3.2 – Activity diagram of ‘Set Reminder’.

Figure 3.3 shows activity diagram of use case ‘Take Photo to Log Food Journal’: By clicking ‘+’ button on app main screen, user is prompted to take a food photo and enter detail. Once user confirms the logging, the logged food journal photo and details will be saved into app-specific storage, local Room database respectively.

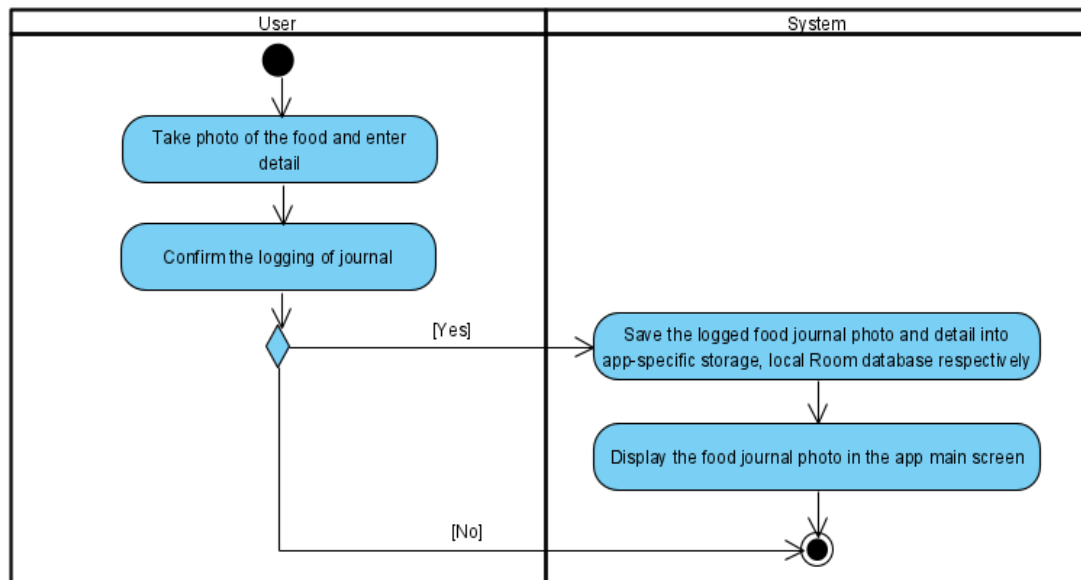


Figure 3.3 - Activity diagram of ‘Take Photo to Log Food Journal’.

Figure 3.4 shows activity diagram of use case ‘Edit Food Journal’: By selecting the desired food journal photo on the app main screen, user is provided option to edit the journal details. Once user confirm the changes, the edited journal detail is updated in local Room database.

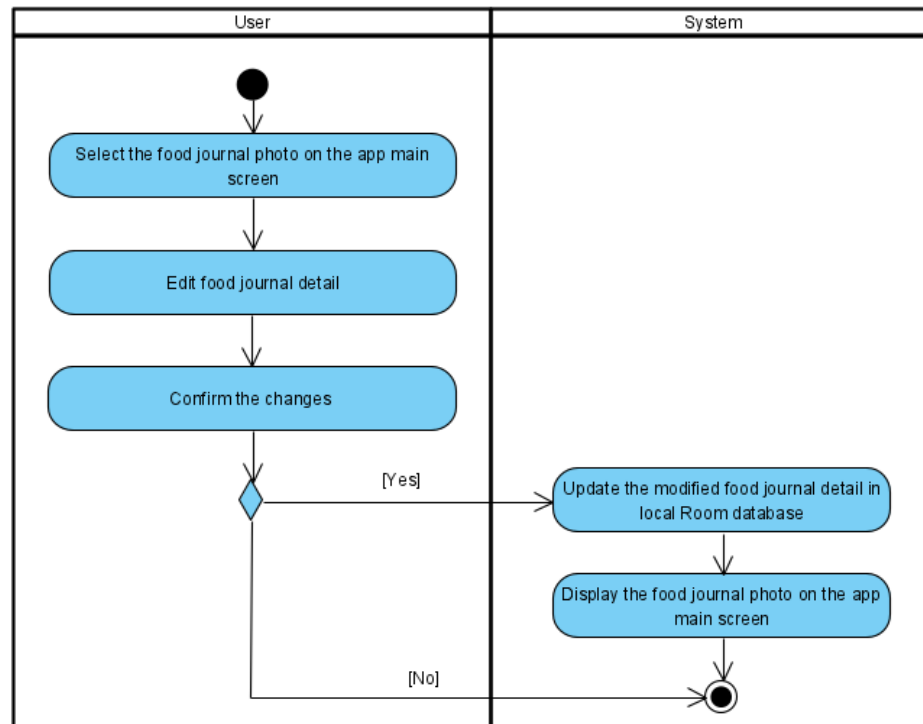


Figure 3.4 - Activity diagram of ‘Edit Food Journal’.



Figure 3.5 shows activity diagram of use case ‘Delete Food Journal’: By selecting the desired food journal photo on the app main screen, user is provided option to delete the journal details stored in local Room database.

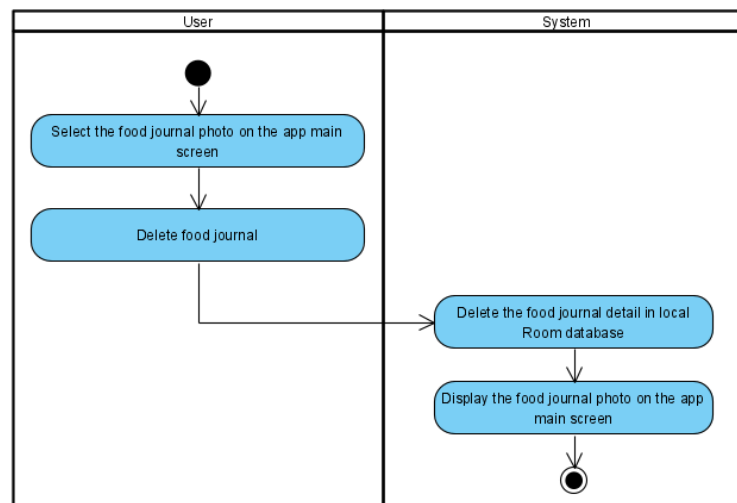


Figure 3.5 - Activity diagram of ‘Delete Food Journal’.

Figure 3.6 shows activity diagram of use case ‘Navigate and View Food Journal’: By selecting desired date using calendar interface, user can view the food journal logged on that date on the app main screen.

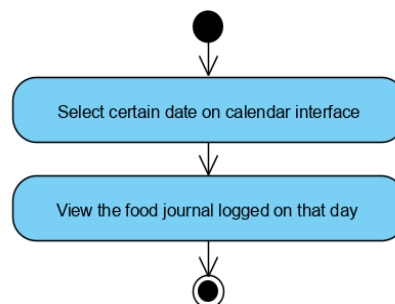


Figure 3.6 - Activity diagram of ‘Navigate and View Food Journal’.

### 3.2.4 Class Diagram

The Java classes used in this food journal mobile application is divided into three categories based on the functionality:

- User interface to create, view, update and delete food journal: MainActivity, UpdateJournal, SaveJournal and Camera.
- Context-awareness of reminder: Reminder, LocationTracking, ActivityRecognition, StopService
- Data access: FoodJournalDatabase, FoodJournalDao, FoodJournal

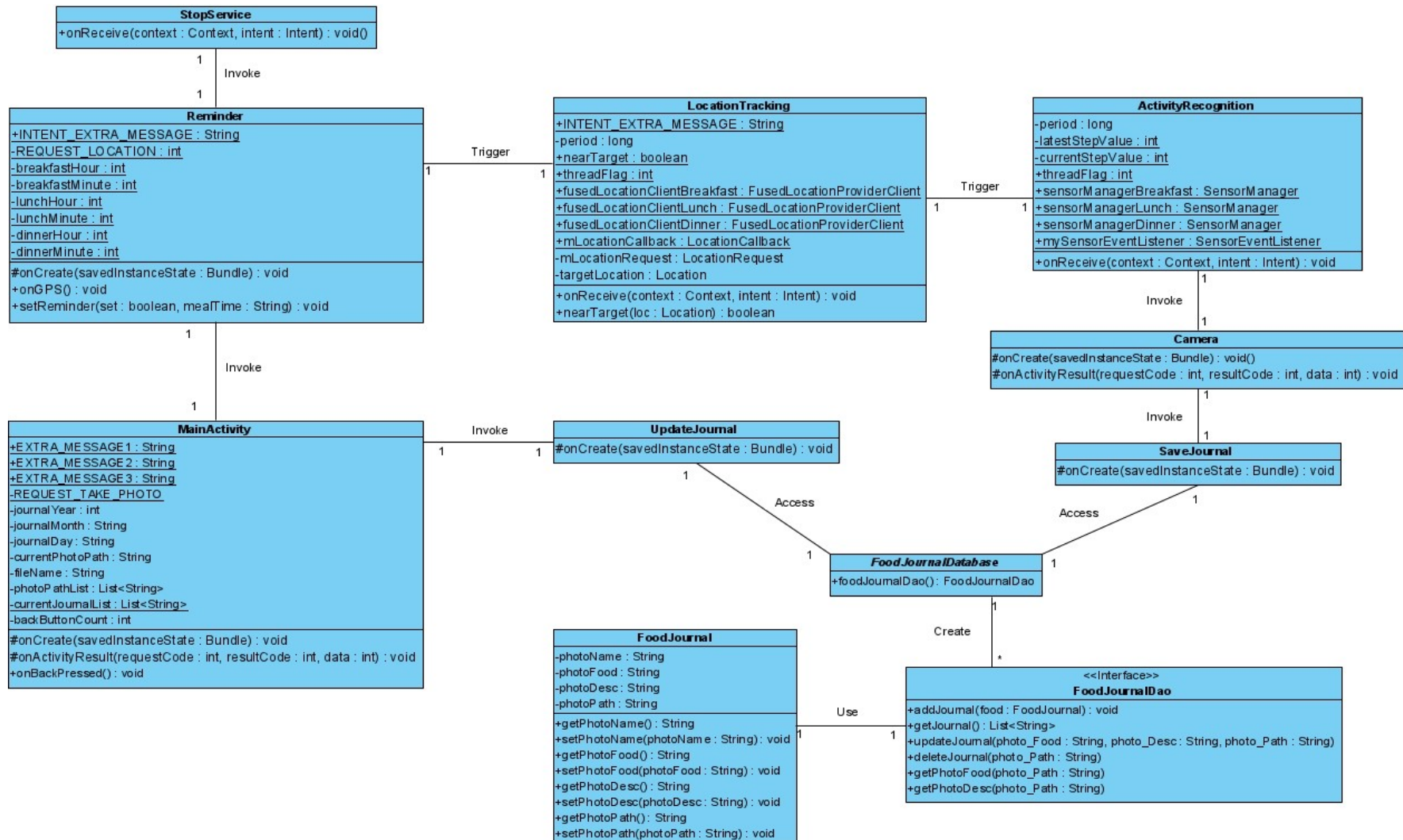


Figure 3.7 - Class diagram of food journal mobile application.

## CHAPTER 4: IMPLEMENTATION

### 4.1 Tools to Use

#### Hardware

1. Smartphone – used for testing the food journal mobile application.

<b>Operating System</b>	Android 7.0 Nougat
<b>CPU</b>	Octa-core (4 × 2.1 GHz Cortex-A57 and 4 × 1.5 GHz Cortex-A53)
<b>GPU</b>	Mali-T760MP8
<b>RAM</b>	3GB
<b>WLAN</b>	Wi-Fi 802.11 a/b/g/n/ac (dual-band), Wi-Fi hotspot
<b>GPS</b>	Yes, with A-GPS, BDS, GLONASS
<b>Sensors</b>	Accelerometer, barometer, compass, fingerprint, gyroscope, heart rate, proximity

Table 4.1 - Smartphone Hardware Specification.

2. Laptop – use Android Studio to develop the food journal mobile application.

<b>Operating System</b>	Windows 10 64-bit
<b>Processor</b>	Intel® Core™ i7-10510U CPU @ 1.80 GHz, 2.30 GHz
<b>RAM</b>	16GB
<b>Graphic Card</b>	Nvidia GeForce MX250

Table 4.2 - Laptop Hardware Specification.

#### Software

1. Android Studio - IDE for Android operating system.
2. Visual Paradigm - UML CASE Tool.

## 4.2 User Interface Design

This app contains four user interfaces:

- Main Screen
- Reminder
- Save Journal
- Update Journal

### ‘Main Screen’ interface

This interface displays the food journal photos logged by current date. If user wants to view the journal of certain date, he can click the button showing current date on the top middle of the interface, which will display a calendar interface, then select desired date.

User can click those photos, which navigate the app to ‘Update Journal’ interface for editing or deleting purposes. Besides, there is a ring bell button to navigate the app to ‘Reminder’ interface. The ‘+’ button will prompt user to camera screen to take a photo, which will then navigate to ‘Save Journal’ interface.

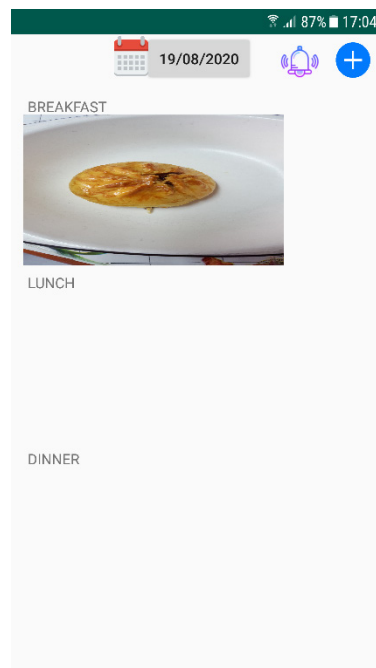


Figure 4.1 - App ‘Main Screen’ UI.

### **‘Reminder’ interface**

This interface consists of three buttons and switches that correspond to breakfast, lunch and dinner, to allow user set reminder for certain time interval according to specific mealtime.

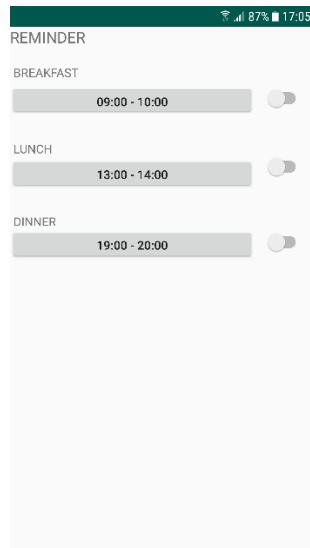


Figure 4.2 - App ‘Reminder’ UI.

### **‘Save Journal’ interface**

This interface shows the photo after the user takes a food photo using the app. User can enter the journal detail: food name and description, then confirms the logging of journal by clicking ‘SAVE’ button.

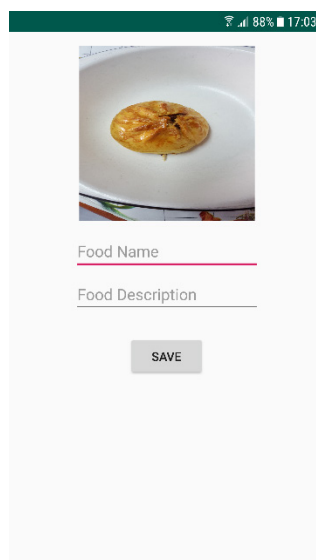


Figure 4.3 - App ‘Save Journal’ UI.

### **‘Update Journal’ interface**

This interface shows the photo after the user clicks a food journal photo on the app main screen. User can edit the journal detail: food name and description, then confirm the changes by clicking ‘UPDATE’ button. User can also delete the journal detail by clicking ‘DELETE’ button.

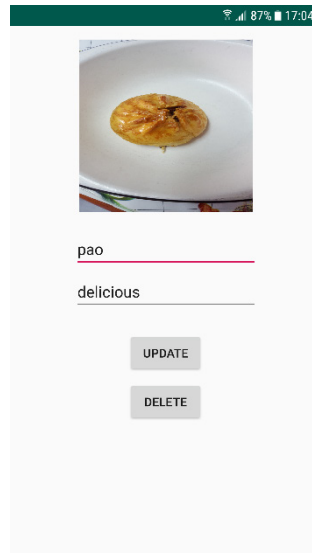


Figure 4.4 - App ‘Update Journal’ UI.

### 4.3 App Function/ Module Implementation

This section details the implementation of function or module required to fulfil all the app use cases.

#### 4.3.1 Set Reminder

1. User clicks the bell ring button on the app main screen, the app navigates to 'Reminder' interface.

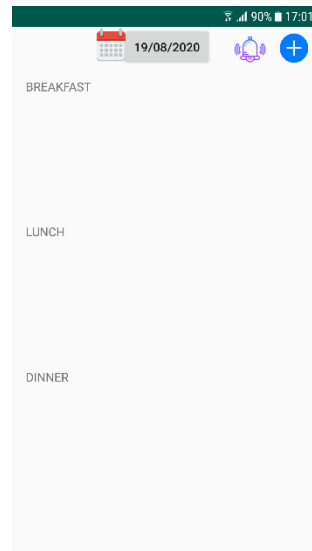


Figure 4.5 - App 'Main Screen' interface.

2. First time launching the app, user has to grant permission for location accessing for the reminder to function.

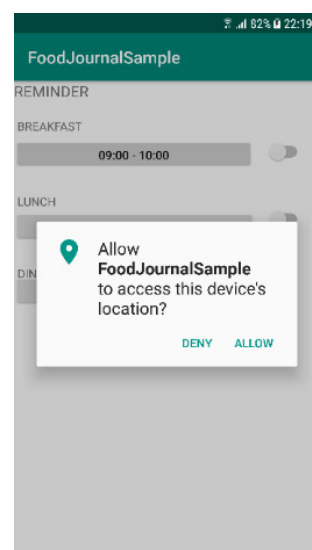


Figure 4.6 - User is asked for location accessing permission.



- When the button for specific mealtime is clicked, it pops up a clock dialog, to let user select when to trigger the reminder. *(If the reminder is enabled already, then it will be removed and set again to reflect the time changes)*

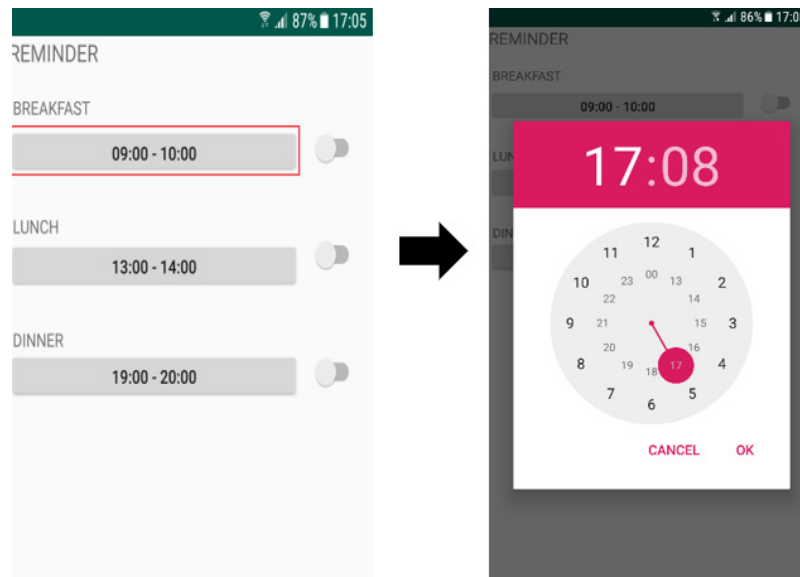


Figure 4.7 - User can set the reminder time using hour, minute hands of popped-up clock dialog when he clicks button for specific mealtime.

- When the switch is toggled on, user device is asked to switch on GPS first if GPS is not enabled, then a reminder for the specific mealtime is set.

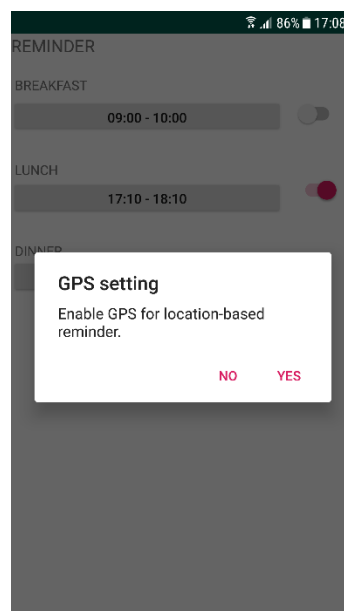


Figure 4.8 - User device is asked to switch on GPS first if GPS is not enabled.

5. The reminder creates two daily repeating alarms: one to trigger **Location Tracking** at the start of mealtime, another one to trigger **Stop Service** at the end of mealtime.

```
public void setReminder(boolean set, String mealTime) {
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);

    if (mealTime.compareTo("breakfast") == 0) {
        Intent startIntentBreakfast = new Intent(this, LocationTracking.class);
        Intent endIntentBreakfast = new Intent(this, StopService.class);
        startIntentBreakfast.putExtra(INTENT_EXTRA_MESSAGE, "breakfast");
        endIntentBreakfast.putExtra(INTENT_EXTRA_MESSAGE, "breakfast");
        PendingIntent startPendingIntentBreakfast = PendingIntent.getBroadcast(this, 0, startIntentBreakfast, PendingIntent.FLAG_UPDATE_CURRENT);
        PendingIntent endPendingIntentBreakfast = PendingIntent.getBroadcast(this, 1, endIntentBreakfast, PendingIntent.FLAG_UPDATE_CURRENT);

        if (set) {
            Calendar startCalendarBreakfast = Calendar.getInstance();
            startCalendarBreakfast.setTimeInMillis(System.currentTimeMillis());
            startCalendarBreakfast.set(Calendar.HOUR_OF_DAY, breakfastHour);
            startCalendarBreakfast.set(Calendar.MINUTE, breakfastMinute);
            Calendar endCalendarBreakfast = Calendar.getInstance();
            endCalendarBreakfast.setTimeInMillis(System.currentTimeMillis());
            endCalendarBreakfast.set(Calendar.HOUR_OF_DAY, breakfastHour + 1);
            endCalendarBreakfast.set(Calendar.MINUTE, breakfastMinute);

            alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, startCalendarBreakfast.getTimeInMillis(), 24 * 60 * 60 * 1000, startPendingIntentBreakfast);
            alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, endCalendarBreakfast.getTimeInMillis(), 24 * 60 * 60 * 1000, endPendingIntentBreakfast);
        }
    }
}
```

Figure 4.9 - Setting a reminder creates alarms.

6. If the switch is toggled off, the reminder for the specific mealtime is removed, and triggers **Stop Service** - Threads and services of executing user device location tracking and step counter tracking are stopped.

```
public void onReceive(Context context, Intent intent) {
    String mealTime = intent.getStringExtra(MainActivity.INTENT_EXTRA_MESSAGE);

    if (mealTime.compareTo("breakfast") == 0) {
        if (fusedLocationClientBreakfast != null) {
            LocationTracking.threadFlag = 0; //to stop thread
            fusedLocationClientBreakfast.removeLocationUpdates(LocationTracking.mLocationCallback);
            fusedLocationClientBreakfast = null;
        }
        if (sensorManagerBreakfast != null) {
            ActivityRecognition.threadFlag = 0;
            sensorManagerBreakfast.unregisterListener(ActivityRecognition.mySensorEventListener);
            sensorManagerBreakfast = null;
        }
    }
}
```

Figure 4.10 - Stop all the executing device location and step counter tracking.

### 4.3.2 Location Tracking

1. The longitude and latitude of a target restaurant is defined.

```
targetLocation = new Location("restaurant location");
targetLocation.setLatitude(5.340359);
targetLocation.setLongitude(100.479287);
```

Figure 4.11 - Define longitude and latitude of a target restaurant.

2. A function to track if the user device stays in the target restaurant region (distance between lower than 60 metre) is defined.

```
public boolean nearTarget(Location loc) {
    float rad = 60.00f;
    float distance = loc.distanceTo(fixedLocation);

    if( distance < rad ){
        return true;
    }
    else {
        return false;
    }
}
```

Figure 4.12 - Define function to track if the user device is near to the restaurant.

3. By applying Fused Location Provider API, the app starts receiving location updates at the interval of 30 seconds, and with priority of high accuracy. Whenever the user device location receives update, the current location is checked if it is near to the target restaurant.

```
public static LocationCallback mLocationCallback;
private LocationRequest mLocationRequest = new LocationRequest()
    .setInterval(30*1000)
    .setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);

mLocationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        Location location = locationResult.getLastLocation();
        if (location != null) {
            nearTarget = nearTarget(location, ctxt);
        }
    }
};

if (mealTime.compareTo("breakfast") == 0) {
    fusedLocationClientBreakfast = LocationServices.getFusedLocationProviderClient(context);
    fusedLocationClientBreakfast.requestLocationUpdates(mLocationRequest, mLocationCallback, null);
}
```

Figure 4.13 - Define parameter settings and action to be triggered for location update.

4. A thread is spawned: within certain period of 90 seconds, if the device distance to that specific location remains lower than defined radius of 60 metre, then an intent is sent to **Activity Recognition**. If not, interval of that period is refreshed to 90 seconds again, and the device location will be kept on tracking until the specific mealtime finished.

```
new Thread() {
    @Override
    public void run() {
        threadFlag = 1;
        period = System.currentTimeMillis() + (90*1000);
        while(System.currentTimeMillis() < period){
            if(threadFlag == 0) {
                break;
            }
            if ( nearTarget ) {
                continue;
            }
            else {
                period = System.currentTimeMillis() + (90*1000);
            }
        }
        if(threadFlag == 1){
            if(mealTime.compareTo("breakfast") == 0){
                Intent locationIntentBreakfast = new Intent(ctxt, ActivityRecognition.class);
                locationIntentBreakfast.putExtra(INTENT_EXTRA_MESSAGE, "breakfast");
                ctxt.sendBroadcast(locationIntentBreakfast);
            }
        }
    }
}
```

Figure 4.14 - Thread to check if user device is near to a restaurant for certain period.

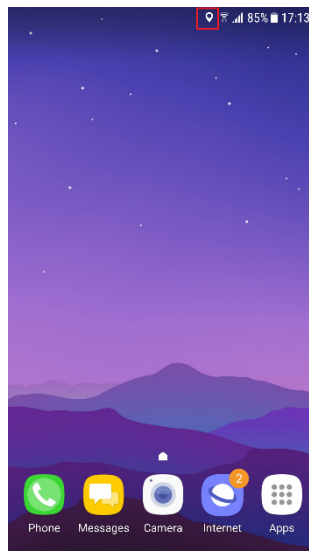


Figure 4.15 - User device location tracking is running in the background.

### 4.3.3 Activity Recognition

1. SensorEventListener is registered with SensorManager which has access to the step counter sensor. The app starts step counter tracking. Whenever the device (SensorEventListener) detects if user is walking, the value is recorded.

```
public static SensorEventListener mySensorEventListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        latestStepValue = (int) sensorEvent.values[0];
    }
    public void onAccuracyChanged(Sensor sensor, int accuracy) { }
};

if (mealTime.compareTo("breakfast") == 0) {
    sensorManagerBreakfast = (SensorManager) context.getSystemService(SENSOR_SERVICE);
    Sensor countSensor = sensorManagerBreakfast.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    sensorManagerBreakfast.registerListener(mySensorEventListener, countSensor, sensorManagerBreakfast.SENSOR_DELAY_NORMAL);
}
```

Figure 4.16 - Record step counter sensor value if user is detected walking.

2. A thread is spawned: within certain period of 20 seconds, if the user current step counter remains the same and distance to the target restaurant still remains lower than 60 metre (assumed to be seated in a restaurant), then the user device location and step counter tracking are stopped, proceed to *step 3*. If not, interval of that period is refreshed to 20 seconds again and the user device and current step counter will be kept on tracking until the specific mealtime finished.

```
(Thread) run() -> {
    threadFlag = 1;
    period = System.currentTimeMillis() + (20 * 1000);
    while(System.currentTimeMillis() < period) {
        if(threadFlag == 0) {
            break;
        }
        if (currentStepValue == latestStepValue & nearTarget) {
            continue;
        }
        else {
            period = System.currentTimeMillis() + (20 * 1000);
            currentStepValue = latestStepValue;
        }
    }
    if(threadFlag == 1) {
        if (mealTime.compareTo("breakfast") == 0) {
            fusedLocationClientBreakfast.removeLocationUpdates(LocationTracking.mLocationCallback);
            fusedLocationClientBreakfast=null;
            sensorManagerBreakfast.unregisterListener(mySensorEventListener);
            sensorManagerBreakfast=null;
        }
    }
}
```

Figure 4.17 - Thread to check if user is sitting in a restaurant for certain period.

3. The device vibrates and a notification is shown in phone notification bar.

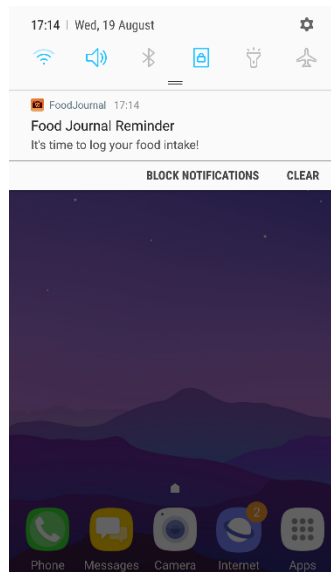


Figure 4.18 - App notification is shown in the notification bar.

#### 4.3.4 Take Photo for Logging Journal

1. When the user taps the notification shown, he is prompted to the camera screen.  
(The user can also click the '+' button on top right corner of app main screen, to be prompted to the camera screen)
2. Once the user takes a photo, he is prompted to 'Save Journal' interface. He can enter the journal detail: food name and description. Then, he confirms the logging of journal by clicking 'SAVE' button.

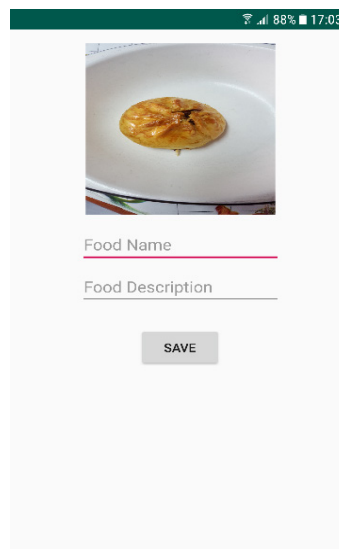


Figure 4.19 - 'Save Journal' interface to let user enter journal detail after taking a photo.

3. The app will navigate to main screen, displaying the logged food journal photos.

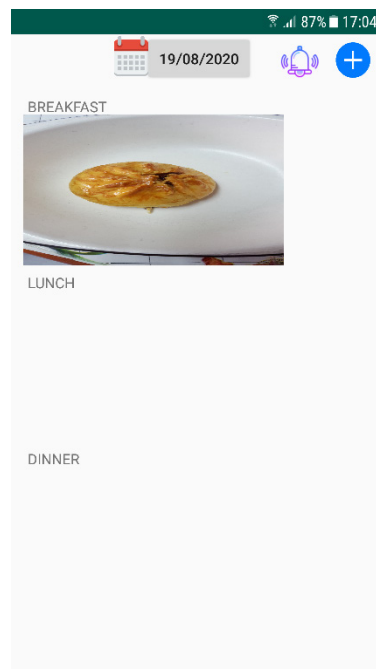


Figure 4.20 - 'Main Screen' interface display logged food journal photo.

#### 4.3.5 Edit/ Delete Food Journal

1. User clicks the desired food journal photo on the app main screen.
2. The app navigates to 'Update Journal' interface.

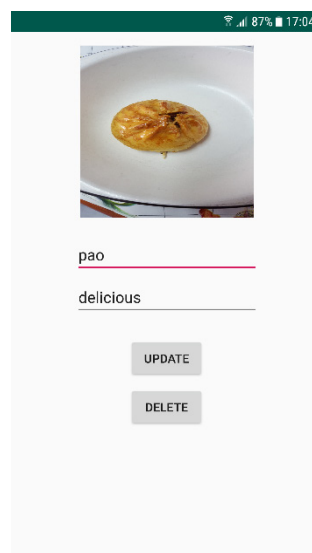


Figure 4.21 - 'Update Journal' interface to let user update existing journal detail when user clicks a journal photo.

3. User edits the journal detail, then confirm the changes by clicking 'UPDATE' button. User can also delete the food journal detail by clicking 'DELETE' button.
4. App updates the modified journal detail or delete the journal detail in local Room database.

#### 4.3.6 Navigate and View Food Journal

1. User clicks the current date on the top middle of app main screen, which will pop up a calendar interface, then select desired date.

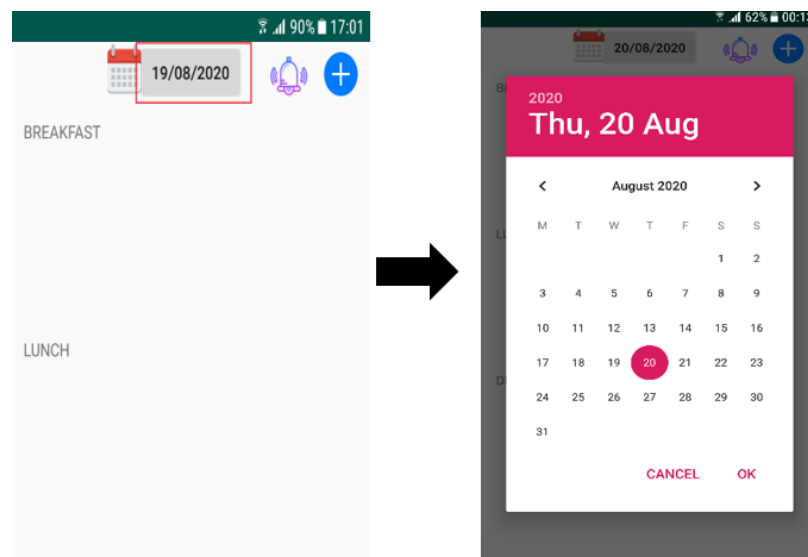


Figure 4.22 - User can view the logged food journal of certain date using popped-up calendar interface when he clicks button showing current date.

2. The app displays the food journal photos of that date on the app main screen.



## CHAPTER 5: TESTING & RESULTS

### 5.1 Operational Testing of Context-Aware Reminder

After the implementation was done, operational testing is done to make sure when a reminder is set by user, it is working as the expected outcome in different conditions as shown in Table 5.1, without disrupting its functionality. According to testing results, all the expectations are met.

Test Cases	Tested Conditions	Expected Outcome
Reminder: ON	Change the reminder time	The reminder updates the time change
	Switch off, then switch on the reminder	No effect on the reminder, working well
	Turn off the phone screen, exit the app (onDestroy() function called)	No effect on the reminder, working well
	The device system time reaches time set by reminder	User device location tracking starts within 2 minutes
Reminder: ON Device location /step counter tracking service: ON	Switch off the reminder	The executing user device location or step counter tracking service stop
	During user device location tracking, user is not near to a target restaurant	The executing user device location tracking service continues until user device is near to a target restaurant for 90 seconds /mealtime ends
	During user step counter tracking, user is walking	The executing user step counter tracking service continues until user stops walking for 20 seconds /mealtime ends
	Switch off another reminder (other than current switched on reminder)	No effect on current reminder, the executing user device location or step counter tracking service continue. (Reminders of breakfast, lunch, dinner are independent of each other)
	Turn off the phone screen, exit the app (onDestroy() function called)	No effect on the reminder, the executing user device location or step counter tracking service continue in the background

Table 5.1 - Operational Testing of Context-Aware Reminder in Different Conditions.

## 5.2 Responsiveness Testing of the Context-Aware Reminder

The responsiveness testing of context-aware reminder is conducted to measure the time taken to trigger context-aware reminder in a given situation. This testing is conducted 5 times to observe if there is outlier result of expected time taken interval. Table 5.2 shows the testing details.

Testing 1	
Prerequisite	An observer is staying around 10 - 15 meters outside the 60-metre radius circular region of target restaurant location, according to the implementation done in section 4.2.  (The circular region border is estimated using Fused Location Provider API to check if observer stays in the area or not)
Steps	<ol style="list-style-type: none"><li>1. Starts the stopwatch when alarm triggers.</li><li>2. Walks to the target restaurant at speed as constant as possible.</li><li>3. Stop walking after reach inside the restaurant.</li><li>4. Stop the stopwatch once the phone vibrates (reminder triggered) and record the time taken.</li></ol>
Expected Time Taken Interval to Trigger Reminder	In between 2 minutes 20 seconds and 2 minutes 30 seconds.  When alarm triggers, user device location is updated but observer is outside the circular region. After 30 seconds, location is updated, and observer has already reached inside the region. Then, the context-aware reminder takes 90 seconds to detect if observer is staying in a restaurant, and another 20 seconds for additional step counter tracking. These situations sum up to 2 minutes 20 seconds. Figure 5.1 shows brief explanation using flow chart.  The expected time taken is increased 10 more seconds due to several issues: <ul style="list-style-type: none"><li>• 30 seconds of location update interval may delay few seconds.</li><li>• Observer manually record the time taken using stopwatch on hand.</li></ul>

Table 5.2 - Responsiveness Testing 1 Details.

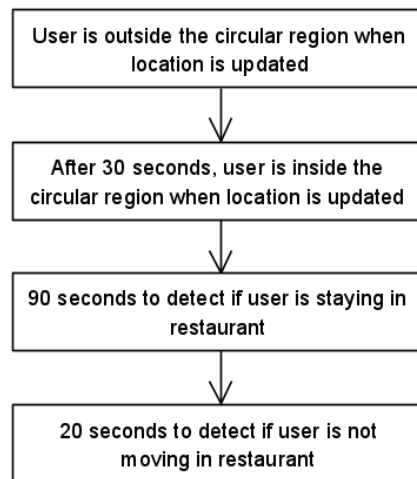


Figure 5.1 - Flowchart showing how to get expected time taken to trigger the reminder in Responsiveness Testing 1.

Since the 60-metre radius is not big, the observer will reach inside the restaurant and stop walking before step counter tracking starts. Therefore, Testing 2 (with minor step detail changes to Testing 1) is conducted to make sure step counter is detecting if observer is moving inside restaurant.

Testing 2	
Prerequisite	An observer is staying around 10 - 15 meters outside the 60-metre radius circular area of target restaurant location, according to the implementation done in section 4.2.  (The circular area border is estimated using Fused Location Provider API to check if observer stays in the area or not)
Step	<ol style="list-style-type: none"> <li>1. Starts the stopwatch when alarm triggers</li> <li>2. Walks to the target restaurant at speed as constant as possible.</li> <li>3. <b>Continue walking after reach inside the restaurant.</b></li> <li>4. <b>Once the step counter tracking starts, continue walking for around 10 seconds then stop.</b></li> <li>5. Stop the stopwatch once the phone vibrates and record the time taken.</li> </ol>
Expected Time Taken Interval to Trigger Reminder	<p>In between 2 minutes 30 seconds and 2 minutes 40 seconds.</p> <p>For the expected time taken interval in Testing 2, just add 10 more seconds to the expected time taken interval in Testing 1 since observer continues walking for around 10 seconds when he reaches restaurant.</p>

Table 5.3 - Responsiveness Testing 2 Details.

### 5.3 Responsiveness Testing Results

Figure 5.2 shows the results of Responsiveness Testing 1 and 2.

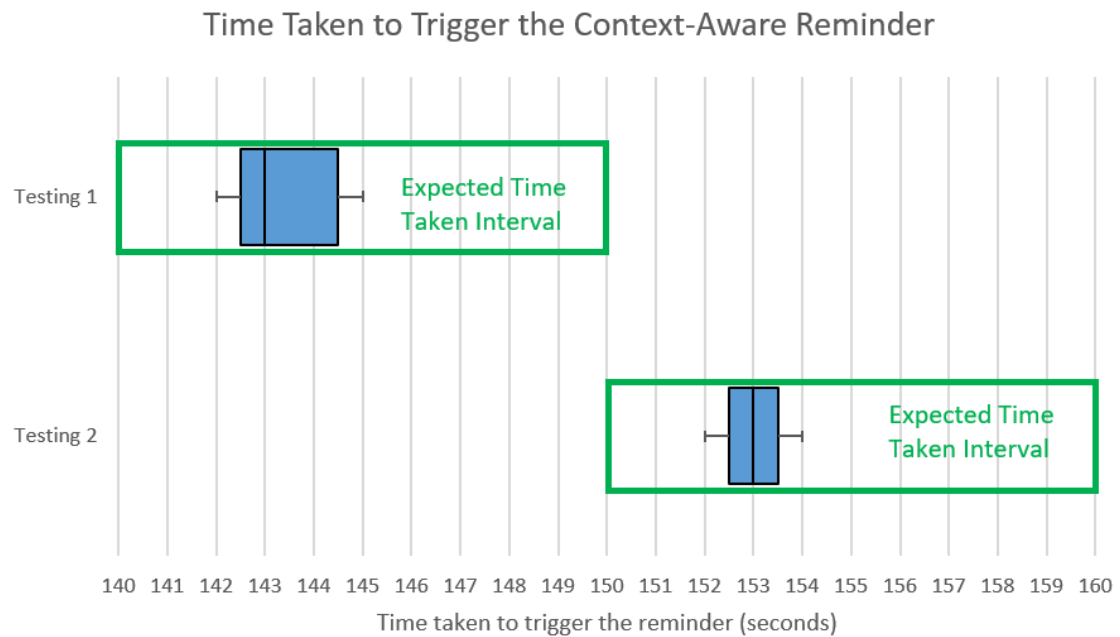


Figure 5.2 - Box-and-whisker plot showing time taken to trigger context-aware reminder in Testing 1 and 2.

The result showing time taken to trigger the context-aware reminder in Testing 1 and 2 are within expected interval. In each testing, the five results only differ by few seconds. The responsiveness of the reminder is satisfied and desired.

## CHAPTER 6: DISCUSSION

This chapter discusses some experiences or observations when using Alarm, Fused Location Provider API and step counter sensor to build the context-aware reminder in this project.

### 6.1 Trigger Time of Alarm (for Time-based Operations)

The context-aware reminder uses ‘setRepeating’ Alarm to trigger time-based operation of user device location tracking. However, the Alarm may delay up to 2 minutes to trigger. According to Android Developer official documentation, all repeating alarms are inexact as of Android 4.4. Therefore, if the app needs very precise delivery times, one-time exact alarms must be used instead (AlarmManager 2020).

The minor time delay does not really affect the responsiveness of context-aware reminder, unless the user is having a meal instantly when his mealtime starts. If that is the case, the user can just set reminder few minutes earlier.

### 6.2 Location Update Interval and Battery Consumption of Fused Location Provider API (for User Device Location Tracking)

The context-aware reminder in this project implements Fused Location Provider API to request location updates for the purpose of user device location tracking. The request is set to high priority and 30 seconds of update interval, to ensure reminder is accurate in detecting if user stays in a restaurant. The issue of concern arises in this context is whether this setting consumes a lot of battery, and how much battery can be saved if it is changed to medium priority or longer update interval.

Changing the request setting to medium priority for battery saving is not recommended. Without using GPS sensor, this will decrease the accuracy of reminder in tracking user device location, especially when Wi-Fi is not available when user stays in indoor location. Therefore, if developer wants to use medium priority, he must ensure the circular region of restaurant location defined is large enough to compensate the loss of location tracking accuracy. However, in this case, the reminder will wrongly assume the user is staying in a restaurant even he is located far from the restaurant. Therefore, only the option of changing location update interval is considered.

An observation for phone battery consumption when the user device is receiving location updates in the background once the alarm triggered, is made. Meanwhile, the same observation but changing location update interval to 120 seconds is made to compare with that 30 seconds.

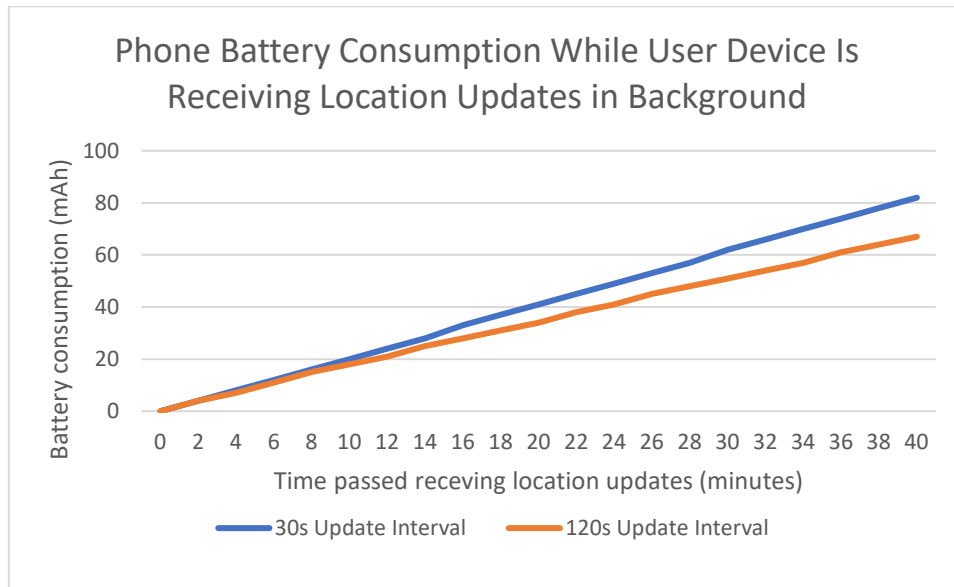


Figure 6.1 - Phone battery consumption when user device is receiving location updates at 30 seconds and 120 seconds interval.

The phone battery consumes at the rate of about 4 mAh per 2 minutes and slightly less while requesting location updates at 30 seconds and 120 seconds of interval respectively. The battery consumption difference will become bigger and obvious if the user device location tracking is going to run for long period of time. However, usually mealtime is only 1-hour interval. It is not worth by setting the location update request to longer interval for this little amount of battery saving, because by doing so, then the time taken for context-aware reminder to detect user is staying in restaurant also needs to be set much longer. This will decrease responsiveness of the reminder, such as the user may already start eating when reminder triggers.

Besides battery consumption, there is one observation regarding the interval of location update request using Fused Location Provider API. Usually, the interval is not exact and may delay up to few seconds. This minor time delay is not significant, since it only causes the context-aware reminder to be late few seconds to trigger when user

is sitting in restaurant within specific mealtime, as shown in result of responsiveness testing conducted in section 5.2.

### **6.3 Value of Step Counter (for User Sitting Activity Recognition)**

The context-aware reminder in this project tracks user step to detect if user is sitting. When a stationary user starts walking or running, step counter takes around 5 seconds to increase its value. When a walking or running user stops, step counter takes less than 1 second to stop increasing value and remains same.

While step counter value is not accurate when user is not walking in constant rate, for example, suddenly walks faster or running, it does not matter. As long as the value is increasing, the user is confirmed to be moving. Therefore, even step counter may not be accurate in differentiating whether user is walking, jogging or running, but it is very suitable for merely recognising whether the user is moving or not. In this context, it is more accurate and response faster than Awareness API, and simpler to implement than the method measuring phone accelerometer value reviewed in Chapter 2.

Since step counter remains same when user is being still, the context-aware reminder built in this project decides if a user is being still for 20 seconds, then he is assumed to be sitting. Of course, developer can decide himself what is the duration of user being still that will be considered as sitting. Therefore, tracking step counter sensor value is simpler yet effective way to detect sitting activity.

## **CHAPTER 7: CONCLUSION**

### **7.1 Project Review**

A food journal app is a great tool to help people in tracking what they eat daily to achieve several purposes such as weight loss goals, maintain healthy eating habits or monitor food allergies. However, busy schedules in daily life lead people to forget to use the apps to log any meal taken daily. While most of those apps provide reminder functionality to notify users in the phone notification bar to log a meal at specific mealtime, these time-based reminders, are not efficient. A time-based reminder may be set to trigger at some arbitrarily selected time, because the users presently scheduling a reminder may not be able to predict a specific time.

Therefore, this project proposes to develop a context-aware reminder for food journal mobile application, that utilises factors of time, device location, and user activity recognition to notify the users in a more efficient way to log every meal they are taking.

Provided the user has set the context-aware reminder at certain time interval, for instance 2 - 3 pm for Lunch. At 2 pm, the app will start tracking user device location using Fused Location Provider API. If the user is detected inside a restaurant circular region of specified radius for 90 seconds, the app starts tracking step counter sensors value for activity recognition purposes. Then, if the user current step counter remains the same and still stay inside the restaurant region for 20 seconds, the app assumes user is sitting for a meal in a restaurant within specific mealtime. The app then vibrates the device and shows a notification. Once the user clicked it, he will be prompted to camera screen of the app for logging the meal he is taking. The logged food journal photo and detail will be finally saved to app-specific storage, local Room database respectively.

### **7.2 Project Contribution**

Aside from supporting food journal app, the context-aware reminder proposed by this project can be support for many existing apps in the market. For instance, petrol fuelling app reminds user to get the member card points when fuelling petrol at the station, shopping mall app reminds user to buy certain commodities in shopping mall during



weekend, specific reminder app reminds user to take his personal belongings such as key and wallet when he leaves home.

Furthermore, this project showed the phone battery consumption when user device is receiving location updates in the background using Fused Location Provider API (with highest priority and 30 seconds of update interval settings), which is about 4 mAh per 2 minutes. The same observation is made by changing update interval to 120 seconds for battery saving purposes. The results showed that the difference of battery consumption is very small within short period of time like interval of one hour, and only become bigger and obvious if the user device location tracking is going to run for long period of time.

Lastly, this project showed that in the context of merely recognising user activity whether he is still or moving, using Android built in step counter sensor is a very simple yet effective way by just tracking its value compared to other methods such as using Awareness API, or calculating phone accelerometer value.

### **7.3 Future Work**

The context-aware reminder will become an independent app to remind user of important things to do by showing notification message. The reminder can let user choose flexible and different combination of context signals to trigger the message set by himself. User himself can specify the time (hour and minute interval, specific day, repeating or trigger once only), user activity (walking, running, driving, still and others), location (latitude & longitude with the aid of Google Maps, radius of region) will trigger what the notification message to be shown.

For instance, user can set the reminder: During Saturday 2 - 3 pm, when user is inside a food court location with 100-metre of radius, then the reminder app will notify him to go buy out-of-stock dog food in nearby pet shop by showing a notification message “Remember to buy dog food!”. By doing so, this context-aware reminder app not only remind user of daily or personal matters, but more significantly it serves as the universal reminder for all other apps, so that those apps no need to enhance the built-in reminder itself or purposely add reminder feature. For instance, this context-aware

reminder app will assist petrol fuelling app to notify user to get the member card points when fuelling petrol at the station.

## BIBLIOGRAPHY

*Access Raw Sensor Data | Google Fit | Google Developers*, 2019. Available from: <<https://developers.google.com/fit/android/sensors>>. [8 April 2020].

*Activity Recognition API | Google Developers*, 2020. Available from: <<https://developers.google.com/location-context/activity-recognition>>. [9 April 2020].

*A-GPS Vs GPS – Difference and Comparison | Diffen*, 2011. Available from: <[https://www.diffen.com/difference/A-GPS\\_vs\\_GPS](https://www.diffen.com/difference/A-GPS_vs_GPS)>. [8 April 2020].

*AlarmManager | Android Developers*, 2019. Available from: <<https://developer.android.com/reference/android/app/AlarmManager>>. [30 Aug 2020].

*Assisted GPS*, 2020. Available from: <[https://en.wikipedia.org/wiki/Assisted\\_GPS](https://en.wikipedia.org/wiki/Assisted_GPS)>. [8 April 2020].

Azumio, Inc, 2019. *Calorie Mama AI: Meal Planner & Food Macro Counter*. Mobile app. Version 5.36.4302. Available from: <https://play.google.com/store/apps/details?id=com.azumio.android.caloriesbuddy&hl=en>. Accessed 15 August 2019.

Bite AI, 2019. *Bitesnap: Photo Food Tracker and Calorie Counter*. Mobile app. Version 1.6.3. Available from: <https://play.google.com/store/apps/details?id=ai.bite.biteapp.beta&hl=en>. Accessed 15 August 2019.

*Build Location-Aware Apps | Android Developers*, 2020. Available from: <<https://developer.android.com/training/location>>. [9 April 2020].

*Context Awareness*, 2019. Available from: <[https://en.wikipedia.org/wiki/Context\\_awareness](https://en.wikipedia.org/wiki/Context_awareness)>. [8 April 2020].

Cordeiro, F., Epstein, D. A., Thomaz, E., Bales, E., Jagannathan, A. K., Abowd, G. D. & Fogarty, J. 2015, 'Barriers and Negative Nudges: Exploring Challenges in Food Journaling', *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI 15*, pp.1159-1162. Available from: ACM Digital Library [30 Aug 2020].

*Create And Monitor Geofences* | *Android Developers*, 2020. Available from: <<https://developer.android.com/training/location/geofencing>>. [9 April 2020].

*Fence API Overview* | *Google Awareness API* | *Google Developers*, 2020. Available from: <<https://developers.google.com/awareness/android-api/fence-api-overview>>. [8 April 2020].

FitNow, Inc., 2019. *Lose It! – Calorie Counter*. Mobile app. Version 11.4.601. Available from: <https://play.google.com/store/apps/details?id=com.fitnow.loseit&hl=en>. Accessed 15 August 2019.

*Fused Location Provider API* | *Google Developers*, 2020. Available from: <<https://developers.google.com/location-context/fused-location-provider>>. [8 April 2020].

*Geofencing API* | *Google Developers*, 2020. Available from: <<https://developers.google.com/location-context/geofencing>>. [8 April 2020].

*Google Awareness API* | *Google Developers*, 2020. Available from: <<https://developers.google.com/awareness>>. [8 April 2020].

*GPS Vs A-GPS* | *Difference Between GPS And GPS-A*, 2012. Available from: <<https://www.rfwireless-world.com/Terminology/GPS-vs-AGPS.html>> [8 April 2020].

Khan, N & Khan, F 2013, *Context Based Reminder System Supporting Persons Using Smart Phone Accelerometer Data*. Master thesis, Blekinge Institute of Technology.

MyFitnessPal, Inc., 2019. *Calorie Counter - MyFitnessPal*. Mobile app. Version 19.80. Available from: <https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=en>. Accessed 15 August 2019.

*Penzu*, 2020. Available from: <<https://penzu.com/food-diary>>. [8 April 2020].

*Sensors* | *Android Developers*, 2019. Available from: <<https://developer.android.com/guide/topics/sensors>>. [9 April 2020].

*Ubiquitous Computing*, 2020. [online] Available from: <[https://en.wikipedia.org/wiki/Ubiquitous\\_computing](https://en.wikipedia.org/wiki/Ubiquitous_computing)>. [8 April 2020].

*What's The Awareness API? | Google Awareness API*, 2020. Available from: <<https://developers.google.com/awareness/overview>>. [9 April 2020].

*Wi-Fi Positioning System*, 2020. Available from: <[https://en.wikipedia.org/wiki/Wi-Fi\\_positioning\\_system](https://en.wikipedia.org/wiki/Wi-Fi_positioning_system)>. [8 April 2020].

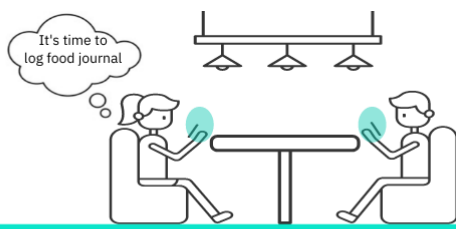
## APPENDICES

### Context-Aware Reminder Responsiveness Testing Results

Testing 1	Testing 2
2 min 24 sec	2 min 33 sec
2 min 25 sec	2 min 34 sec
2 min 22 sec	2 min 32 sec
2 min 23 sec	2 min 33 sec
2 min 23 sec	2 min 33 sec

### Battery Consumption When User Device is Receiving Location Update at 30 seconds and 120 seconds Interval

Time running user device location tracking in the background (minute)	Battery consumption for 30s location update interval (mAh)	Battery consumption for 120s location update interval (mAh)
2	4	4
4	8	7
6	12	11
8	16	15
10	20	18
12	24	21
14	28	25
16	33	28
18	37	31
20	41	34
22	45	38
24	49	41
26	53	45
28	57	48
30	62	51
32	66	54
34	70	57
36	74	61
38	78	64
40	82	67



# CONTEXT-AWARE REMINDER for FOOD JOURNAL MOBILE APP

Notify user who is **sitting** in a **restaurant** within  
specific **mealtime**

## INTRODUCTION

A food journal helps people in tracking what they ate daily to achieve several purposes such as weight loss goals, maintain healthy eating habits or monitor food allergies.

Most of the existing food journal apps in the market provide reminder to notify users for logging a meal at specific meal time each day.

### Problem ?

All these reminders are considering time factor only. As the users presently scheduling a reminder may not be able to predict a specific time, the reminder is mostly set to trigger at arbitrarily selected time which as it turns out, not so efficient.

### Motivation

This project proposes to develop a context-aware reminder for food journal mobile app, that utilises factors of **time**, **device location**, and **user activity recognition** to notify the users in a more efficient way.

## PROPOSED METHOD



### 1 Alarm Trigger

Once the user sets a reminder for specific mealtime, the app creates two alarms at the start and end of meal time: to trigger step 2, and to stop user device location & step counter tracking for the app reminder respectively



### 2 Device Location Tracking

The app starts tracking user device location using Fused Location Provider API at specific interval.

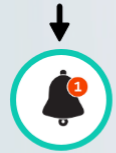
If the user device stays in a restaurant location circular region with certain radius within certain period, proceed to step 3. Else, repeat this step.



### 3 Step Counter Sensor Tracking

Whenever the device detects user is walking, the step counter value is recorded.

If the user current step counter remains unchanged and staying in the restaurant region within certain period, proceed to step 4. Else, repeat this step.



### 4 Notification & Journal Logging

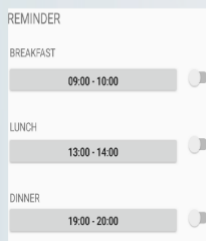
The app assumes user is sitting in a restaurant for specific mealtime, therefore stops the device location and step counter tracking, then vibrates the device and shows notification. Once user taps it, he will be prompted to camera screen of the app for logging the journal photo and details, which are later saved into app-specific storage and local Room database.

### Context-awareness

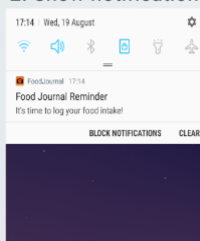
refers to the property of mobile devices that is able to adapt behaviours according to the information gathered from its environment at any given time.

## RESULTS

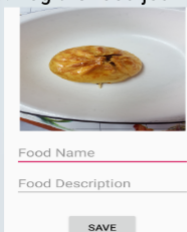
### 1. Set reminder



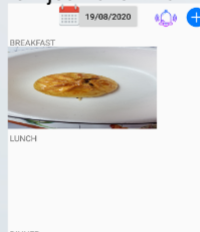
### 2. Show notification



### 3. Log the food journal



### 4. Show journal on main screen

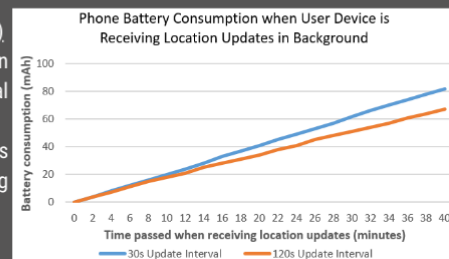


## DISCUSSION

### Fused Location Provider API (for User Device Location Tracking)

The phone battery consumes about 4 mAh per 2 minutes when user device is receiving location updates at 30 seconds interval and with high priority/ accuracy setting.

Setting location update request to longer interval saves negligible amount of battery, unless location tracking is running for long period of time such as several hours.



### Step Counter Value (for User Sitting Activity Recognition)

Step counter may be not accurate when user is NOT walking in constant rate, but doesn't matter. As long as the counter value is increasing while user is moving, developer can decide himself what is the duration of step counter remains unchanged that assumes user is sitting.



## CONCLUSION

Those time-based reminder provided by most of the existing food journal apps in the market to notify users for logging a meal at specific mealtime are not efficient enough, as the users presently scheduling a reminder may not be able to predict a specific time. The concept of context-awareness gives inspiration to this project to develop a food journal app with context-aware reminder, which utilizes more type of context signal.

Aside from supporting food journal app, the context-aware reminder can become an independent app to remind user of important things to do by just showing message. The reminder can let user choose flexible and different combination of context signals to trigger the message set by himself. By doing so, this context-aware reminder app not only remind user of daily or personal matters, but more significantly it serves as the universal reminder for all other apps, so that those apps no need to enhance the built-in reminder itself or purposely add reminder feature.

# TURNITIN REPORT FOR PLAGIARISM CHECKING

preferences

**turnitin**  
Originality Report

Processed on: 09-Sep-2020 17:08 +08  
ID: 1381437102  
Word Count: 10400  
Submitted: 4

**FYP2**  
By Hoe Chia Yong

Similarity by Source	
Similarity Index	6%
Internet Sources:	4%
Publications:	3%
Student Papers:	4%

Document Viewer

exclude quoted exclude bibliography exclude small matches

mode: show highest matches together Change mode

ABSTRACT Most of the existing food journal apps in the market provide reminder functionality to notify users for logging a meal at specific mealtime each day. Yet, all these time-based reminders are not sophisticated enough because the users presently scheduling a reminder may not be able to predict a specific time. Since context-awareness in mobile computing is concerned with gathering information about the user current situation, this project proposes to develop a context-aware reminder for food journal mobile application, that utilises factors of time, device location, and user activity recognition to notify the users in a more efficient way. Provided the user has set the context-aware reminder at certain time interval, for instance 2 - 3 pm for Lunch. At 2 pm, the app will start tracking user device location using Fused Location Provider API. If the user is detected inside a restaurant circular region of specified radius for certain period, the app starts tracking step counter sensors value for activity recognition purposes. Then, if the user current step counter remains the same and still stay inside the restaurant region for certain period, the app assumes user is sitting for a meal in a restaurant within specific mealtime. The app then vibrates the device and shows a notification, which will prompt user for logging the meal he is taking. This project also showed the phone battery consumption when user device is receiving location updates using Fused Location Provider API, and tracking step counter sensor value is such simple yet effective way to detect sitting activity. ii

**CHAPTER 1: INTRODUCTION 1.1 Problem Statement and Motivation A** **24**

food journal is a great tool to help people in tracking what they ate daily to achieve several purposes such as weight loss goals, maintain healthy eating habits or monitor food allergies. In this digital era, the journal logging process become

- 1% match (student papers from 08-Jun-2020)  
[Submitted to University of Hertfordshire](#)
- < 1% match (student papers from 12-May-2020)  
[Submitted to Dundalk Institute of Technology](#)
- < 1% match (student papers from 29-Jul-2020)  
[Submitted to University of St. Gallen](#)
- < 1% match (Internet from 05-Feb-2020)  
<https://dspace.cvut.cz/bitstream/handle/10466/2020-Valenta-Tadeas-thesis.pdf?isAllowed=y&sequence=-1>
- < 1% match (student papers from 12-Apr-2014)  
[Submitted to Florida Virtual School](#)
- < 1% match (student papers from 31-Jul-2020)  
[Submitted to RMIT University](#)



<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



## FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

<b>Full Name(s) of Candidate(s)</b>	HOE CHIA YONG
<b>ID Number(s)</b>	17ACB05946
<b>Programme / Course</b>	BACHELOR OF COMPUTER SCIENCE (HONS)
<b>Title of Final Year Project</b>	CONTEXT-AWARE REMINDER FOR FOOD JOURNAL MOBILE APPLICATION

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: 6 %</b>  <b>Similarity by source</b> Internet Sources: 4 % Publications: 3 % Student Papers: 4 %	
<b>Number of individual sources listed of more than 3% similarity: 0</b>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***

Signature of Supervisor

Name: DR. OOI BOON YAIK

Date: 9/9/2020

Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_



## UNIVERSITI TUNKU ABDUL RAHMAN

### FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

#### CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	17ACB05946
Student Name	HOE CHIA YONG
Supervisor Name	DR. OOI BOON YAIK

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Front Cover
✓	Signed Report Status Declaration Form
✓	Title Page
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 9/9/2020

Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction.

(Signature of Supervisor)

Date: 9/9/2020