

PERSONAL DIETICIAN – AN ANDROID APPLICATION

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Ritika Khiria

SPRING
2018

© 2018

Ritika Khiria

ALL RIGHTS RESERVED

PERSONAL DIETICIAN – AN ANDROID APPLICATION

A Project

by

Ritika Khiria

Approved by:

_____, Committee Chair
Dr. Jinsong Ouyang

_____, Second Reader
Prof. Bob Buckley

Date

Student: Ritika Khiria

I certify that this student has met the requirements for format contained in the University format manual and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

_____, Graduate Coordinator _____
Dr. Jinsong Ouyang Date

Department of Computer Science

Abstract
of
PERSONAL DIETICIAN – AN ANDROID APPLICATION
by
Ritika Khiria

In today's lifestyle, people are moving towards achieving a fit and healthy body. This shift has changed the way of living in almost every household. Now everyone craves for healthy and nutritious food to be placed on their plates. Hence, healthy eating and nutritious food have become an essential part of everyone's lifestyle to achieve a balanced and healthy life in such busy and hectic environment.

Hence, to make their fitness path a bit smoother and to enhance their experience, I have created an Android [Personal Dietician] application to provide a broader approach in providing a better living through nutritious and fit diet plan to the users.

In this project, the Personal Dietitian android application will use food ontology APIs, which is a part of knowledge representation and semantic web technology to produce diet plans for the users. Additionally, the app will provide an activity tracker which will track the steps walk, climb, and run by the user. The activity tracker is built using an API which uses accelerometer and gyroscope sensors built into the Android device.

The application will start by signing up or logging the user with the Personal Dietitian application. The signup and login screen which will be useful to the user to manage their activities in the application. The application provides four main user functionalities, namely, the activity tracker, meal planner, reports, and health blog. (1) The Activity Tracker is used for tracking the user steps including the walking, running and stairs time along with the total step taken in a day. (2) The Meal planner activity is used to get the suggested meal plans with the help of the user's general information (Height, Weight, Gender, Age, and Body fat) and the food ontology database APIs. This functionality also allows the user to add foods manually using text search and voice search. (3) The third feature, Progress Report activity is used to present the weekly and monthly reports of the activity tracker and nutrients consumed by the user in the form of a line and pie chart. Moreover, the user can also see his/her weekly progress for the activity tracker and meal planner by just clicking the icons on the activities in the form of bar chart as well as a daily report in a list format. (4) The last activity is the health blog which can be used to see different blogs available online related to the health and life improvement.

_____, Committee Chair
Dr. Jinsong Ouyang

Date

ACKNOWLEDGEMENTS

I would like to express my appreciation and gratitude to Dr. Jinsong Ouyang to be my advisor in the journey of completing my master's project successfully. The continuous guidance and help that he has provided on this project helped me gain in-depth knowledge regarding Android Development as well as an understanding the essence of different project requirements in developing this application. Through the encouragement given by him to move forward in achieving the best design and functionality for my application, I was able to build a real-life project. I would also like to thank Prof. Bob Buckley for his willingness to serve on this committee and taking his time to review my report. I appreciate the invaluable feedback given by him to be able to finish my project report successfully.

Also, I would like to sincerely thank the entire faculty and staff of Department of Computer Science at California State University, Sacramento.

Lastly, I would like to thank my parents and family members for being active support throughout the journey of completing my studies.

TABLE OF CONTENTS

	Page
Acknowledgements	vii
List of Figures	xi
Chapter	
1. INTRODUCTION	1
1.1 Overview	1
1.2 Purposed Solution	2
1.3 Features and Functionalities	2
2. ANDROID OVERVIEW	4
2.1 Android Application Fundamentals	4
2.2 Android Application Components	4
2.2.1 Activities	4
2.2.2 Services	7
2.2.3 Content Provider	9
2.2.4 Broadcast Receivers	9
2.2.5 Fragments	10
2.3 Setting Up Android Environment	10
2.4 Creating Android Project	12
2.5 Run Android Application	14

2.5.1	On the Device	14
2.5.2	Using Android Emulator	15
3.	SYSTEM DESIGN	18
3.1	Application Architecture	18
3.2	Technology Used	19
3.3	Libraries Used	19
3.4	Project Workflow	22
3.4.1	Register/Sign Up	24
3.4.2	User Dashboard	25
3.4.3	User Profile	25
3.4.4	Activity Tracker	26
3.4.5	Meal Planner	27
3.4.6	Activity Tracker: Progress Report	28
3.4.7	Meal Planner: Progress Report	29
3.4.8	Weekly and Monthly Progress Report	30
3.4.9	Health Blog	31
4.	APPLICATION IMPLEMENTATION	32
4.1	User Registration	32
4.2	User Login	36
4.3	Forget Password	39
4.4	User Dashboard	40
4.5	User Profile	43

4.6 Activity Tracker	47
4.6.1 Activity Tracker: Weekly and Daily Report	52
4.7 Meal Planner	56
4.7.1 Meal Planner: Weekly and Daily Progress Report	68
4.8 User Log	72
4.9 Track Me: Activity and Meal Report	74
4.10 Health Blog	83
5. CONCLUSION AND FUTURE WORK	85
5.1 Lesson Learnt	85
5.2 Future Enhancements	85
References	86

LIST OF FIGURES

Figures	Page
1. Activity Life Cycle	6
2. Service Life Cycle	8
3. Fragment Life Cycle	11
4. New Project Screen	12
5. Android Studio Target Screen	13
6. Activity Addition Screen	13
7. Customize the Activity Screen	14
8. Device Debugging Setting Screen	15
9. Android Virtual Device Screen	16
10. Android Virtual Device Manager Screen	17
11. Application Architecture	18
12. MPAndroidChart Library	20
13. Support Dependencies Code Snippet	20
14. Glide Library Dependencies	21
15. Using FitChart Library	21
16. Use Case Diagram of an Application	22
17. Project Flow Diagram	23
18. Login/Registration Flow Diagram	24

19. User Dashboard Flow Diagram	25
20. User Profile Flow Diagram	25
21. Activity Tracker Flow Diagram	26
22. Meal Planner Flow Diagram	27
23. Activity Tracker Report Flow Diagram	28
24. Meal Planner Report Flow Diagram	29
25. Weekly and Monthly Report Flow Diagram	30
26. Health Blog Flow Diagram	31
27. User Registration Screen	32
28. Registration activity code	33
29. onCreate Method of Register Activity	33
30. init() Method of Register Activity	34
31. Registration Validation Code	35
32. WebCall Class to Handle the HTTP Network Connection	35
33. Login Screen	36
34. Show/Hide Password Code Snippet	37
35. Login Activity Code Snippet	38
36. Forgot Password Screen	39
37. Forgot Password Code Snippet	40
38. User Dashboard Screen	41
39. User Dashboard Explicit Intent	42

40. Navigation Drawer Screen	42
41. Navigation Drawer List Initialization	43
42. User Profile Screen	44
43. Profile Saving Data	45
44. Implicit Intent to get the Image	46
45. Decoding Image File	46
46. Glide Library to set the Profile Image	47
47. Activity Tracker Screen	48
48. Activity Tracker onCreate Method	49
49. Activity Tracker AlarmManager Code Snippet	50
50. onSensorChange Method Code Snippet	51
51. Activity Tracker SaveTracking Method	52
52. Activity Tracker Weekly and Daily Progress Report	53
53. Activity Tracker init() Method	54
54. Code Snippet for Step Count BarChart	55
55. Code Snippet to Add the Daily Report	56
56. Meal Planner Main Screen	57
57. Calculating Calories Required Per Day	58
58. Calculation for Left Calories	58
59. init() Method of Meal Planner Activity	59
60. Meal Planner Screen	60

61. Food Add Screen	61
62. Voice Search Screen	62
63. Voice Search Code Snippet	63
64. Text Search Screen	64
65. Text Search Code Snippet	65
66. APIs Calls to get the Food Data	65
67. Barcode Scanner Screen	66
68. Barcode Fragment Code Snippet	67
69. Detail Food Screen	67
70. Meal Planner Weekly and Daily Report	68
71. Code Snippet to Show the Explicit intent	69
72. Code Snippet to Add Bar Graph Report	70
73. Code Snippet to Add Daily Report	71
74. Activity Log Screen	72
75. Meal Log Screen	73
76. Track Me Activity Screen	74
77. Activity Report Screen	75
78. Adding Calorie Burned Goal Limit	76
79. Line Chart Weekly Activity Class	78
80. Food and Nutrition Report Screen	79

81. Calculating Average Nutrition Values	80
82. Setting Up the Data Points for the Food Report	81
83. Method to Set Notification	82
84. Health Blog Screen	83
85. Health Blog Code Snippet	84

1. INTRODUCTION

1.1 Overview

In today's lifestyle, people are moving towards achieving a fit and healthy body. This shift has changed the way people are living right now in almost every household. Hence, healthy eating and nutritious food have become an essential part of everyone's lifestyle to achieve a balanced and healthy life in such busy and hectic environment. On the other hand, an imbalanced diet can lead to disastrous results on one's health, which may lead to diseases such as obesity, diabetes, etc. However, such conventional diseases and their symptoms can be reduced or prevented by active living and by including better nutritional diet. Hence, those in search of healthy and nutritious food have used the internet to research the food's nutrition values. According [1] to the Flurry Analytics, "the mobile development world saw a sudden 62% increase in the usage of health apps in 2016". As a result, many companies and developers introduced healthcare applications on mobile devices.

In the Android market, there are many healthcare applications available that provide diet plans for their clients. However, there is no such app available that can provide a diet plan as well as activity tracker functionality to their user. Hence, to make their fitness path a bit smoother and to enhance their experience, I have built an Android application, Personal Dietitian, to provide a broader approach in providing a healthier living through nutritious diet plan to the users.

1.2 Proposed Solution

In the Proposed solution the Android Mobile Application "Personal Dietitian" will use knowledge representation and semantic web technology in the form of the Food Ontology APIs to produce the diet plans for the users. Additionally, the application will provide an Activity Tracker which will tracks the steps walked, run, and climbed by the user using mobile device inbuilt APIs, that is the accelerometer and gyroscope sensors. Through that sensor, the application will also show the current activity the user is doing at that time, for example, if the user is walking the application will display walking on the activity tracker screen.

The meal planner of the application will help the user to add meal plans for breakfast, lunch, and dinner to the user's food log. The application shows the exact calories the user needs for a day by calculating the user's BMR (Basal Metabolic Rate) through the user's general information (Height, Weight, Gender, and Age) and his/her activity level. Additionally, through this application, the user can also scan the products directly on the market shelves to see the nutrition values and calories that product contains and later add the item to his/her food log. The proposed application will also help the user to see his/her activity by providing the weekly progress report in a bar chart format by simply clicking the icons in both activities (Activity Tracker and Meal Planner). Also, the user can also see the weekly and monthly reports in the form of a line and pie charts. Personal Dietitian also offers different health blogs through a "blog API" to make the user more active and motivated. The notification system of the proposed application also helps the user to add their foods to their food logs throughout the day.

1.3 Features and Functionalities

- Register with the Personal Dietitian application.
- Update the general information using User Profile.

- Count steps using Activity Tracker.
- Counts minutes for walking, running, and stairs.
- Meal Planner
- Search the food using voice and text.
- Barcode Scanner to scan the packed product.
- Calculate the left calories from the activity tracker and meal planner.
- Weekly and Monthly reports on nutrition values and activity tracker.
- Health Blogs
- Notification

2. ANDROID OVERVIEW

2.1 Android Application Fundamental

Android is an open source operating system which can be used to build mobile or other small devices applications [1] and Android applications coded in the JAVA programming language. Android Studio [2] is used for developing Android applications. It supports all the Android SDK tools needed to build, design, maintains, test, debug and publish any Android applications. The Android application compiles into a set of files known as .apk files which hold the information used to run the application on any device or even on the android emulator the .apk file is used to run the application.

2.2 Android Application Components

Android application components are the basic building blocks of any Android applications. In the Manifest file [3] of Android application, all the critical components are present to facilitate the application. The points below describe additional components of the development of the Android application.

2.2.1 Activities

Activities are used to interact with the users through the user interface of the Android application. A single application can hold multiple activities which represent different functionalities of an application. To access these activities, one needs to define them in the Android Manifest file. The Manifest file contains tags to identify various activities. The <application> tag is used to define the name, log, version, etc. of the Android application. The <activity> tag is a child tag inside the <application> tag which is used to declare an activity element in the manifest file. The Application's user interface

can be defined using an XML layout file and later bound with its activity to make the particular UI to work.

The main activity starts when the application launched. An Android application's activity has a series of the life cycles that define its activity class in the Java file. There are also callbacks methods which help the activity navigate between different activity stages [3].

Figure 1 shows the activity lifecycle of the Android development.

- `onCreate()`: During the creation of first activity, the `OnCreate()` method invoked. This method manages the data binding process, background initializations and receiving of bundle objects which contains the previous state's instances [3]. This method followed by the `onStart()` method in the activity lifecycle.
- `onStart()`: This method helps the activity to enter the start stage which makes the particular activity visible to the user. This method helps the activity to run in the foreground process as well as interact with the users. Once this method ends, it's always followed by the `onResume()` method which is activated when the activity returned to the background. Otherwise, it will be followed by the `onStop()` method if the activity is hidden.

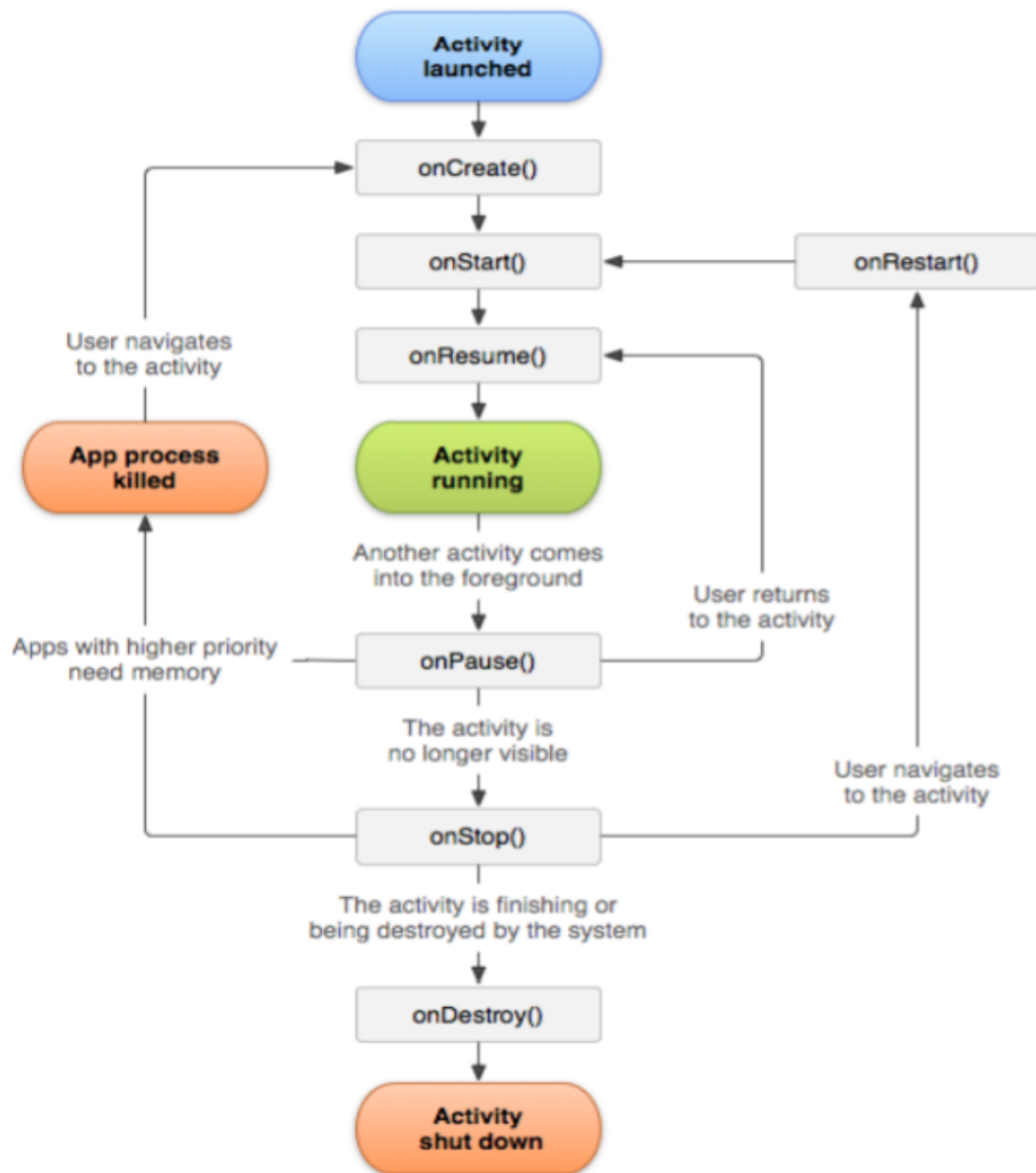


Figure 1: Activity Life Cycle [3]

- `OnResume()`: This method is called when the activity enters in the resumed state, in other words, it will be called once the user returns to the activity. Additionally, this method helps the activity start the interaction between the user and the

application. If the activity encounters any interruption during the process, at that time the activity will enter the paused state by calling the `onPause()` method [3].

The activity will resume when the `onResume()` method calls again.

- `onPause()`: When the user first leaves the current activity to start some other application or the user is interrupted by a phone call, the `onPause()` method evoked. This method will temporarily hold all the operations of the running activity until the interruption is over. The Android system release all the substantial resources that might consume the device battery life during the pause state such as broadcast receivers or sensors like GPS. The activity will remain in the pause state until the system calls the `onResume()` method to start the activity again.
- `onStop()`: An activity enters in the stopped state when it is not visible to the user any longer. During this call, the application will start releasing all the resources that are no longer required by the application.
- `onDestroy()`: This method is called after the `onStop()` method, as it is the final call that activity can make before the destruction of activity. This method is called in two cases; one is when the application has destroyed the activity to save some space and when the activity is already finished running. This method also releases all the resources which unreleased during the `onStop()` method.

2.2.2 Services

Services are capable of performing long-running operations in the background, and that is the reasons they do not have any user interface. During the use of this component, any application can start a new service and can continue the process to run in

the background even when the user switches to some other application. Figure 2, shows the lifecycle of services in the Android Development. Any component can bind with the service to interact with the other Android components as well as to execute interprocess communication (IPC) or send notifications [4].

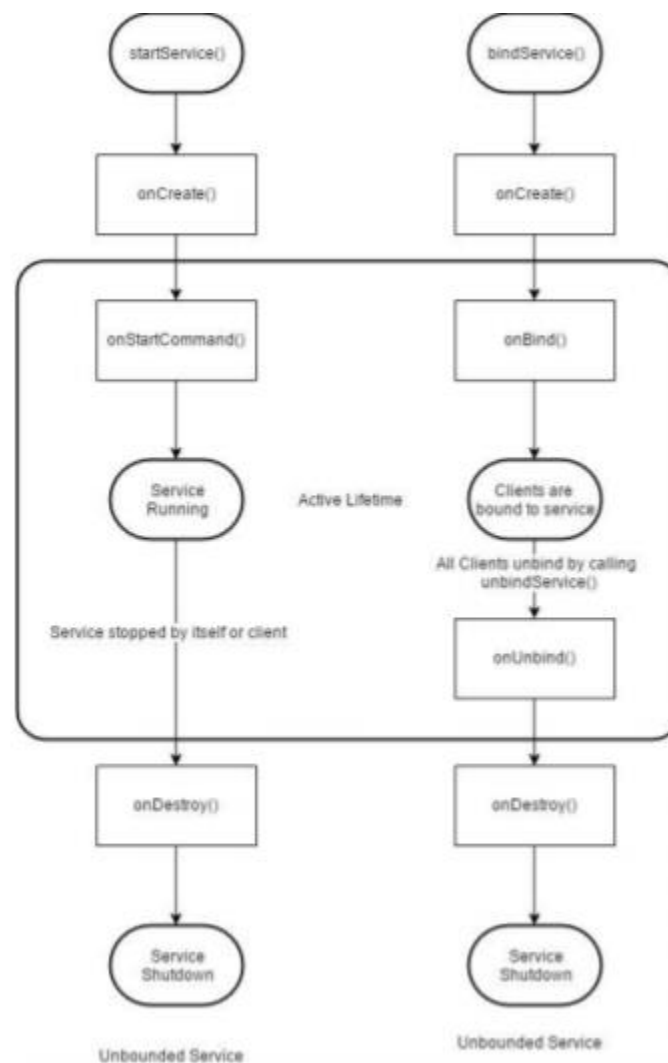


Figure 2: Service Life Cycle [4]

There are three types of services provided by the Android system.

- **Started:** A service will begin once any components or activity evokes the `startService()` method. When the service starts, it will remain running in the background until the application component destroyed.
- **Schedule:** A service is known as Scheduler when an API launches the service. Android applications mostly use the `JobScheduler` class to register jobs and specify the requirements for networking and timing.
- **Bound:** When an Android component links to a service by calling the `bindService()` method, then it is known as Bound. A Bound service provides a means to interact with the different component of the service by sending requests and receiving results.

2.2.3 Content Provider

This component manages permission to structure set of data. This component uses encapsulation to provide security for the data. Content providers enable the user interface to interact with the data among different applications. Their main function is to read the data from the SQLite database on request and also help to store the data [4].

2.2.4 Broadcast Receivers

As the name suggests, the broadcast receivers are intended to receive and listen to the system messages and intents. Broadcast receivers look to the messages from other applications or message within the android system. If there are any new event occurrences, then the receiver will be notified by the Android system through notifications [4].

2.2.5 Fragments

Fragments are the subcomponent of the Activity. They are associated with an activity and interacted via fragment manager [5]. The lifecycle of the fragment(see figure 3) depends on the lifecycle of the Activity as shown in figure 1.

- `onAttach()` – called when the fragment associated with the activity.
- `onCreateView()` – called when the fragment draws its user interface for the first time.
- `onActivityCreated()` – called when the host activity is created and after `onCreateView()` is called.
- `onDestroyView()` – called when the hierarchy associated with the fragments destroyed.
- `onDetach()` - called when the fragment disassociated from the activity.

2.3 Setting up Android Environment

Android application development requires an ADT (Android Development Tools) bundle. This bundle provides all the tools to develop the features of an Android app. This ADT also includes a version of Android Studio. SDK tools downloaded from the SDK manager option of the Android Studio. These SDK tools provide basic functionality to start the code for the application [6]. Main components required in the development of an Android application.

- Java
- XML
- Android Studios

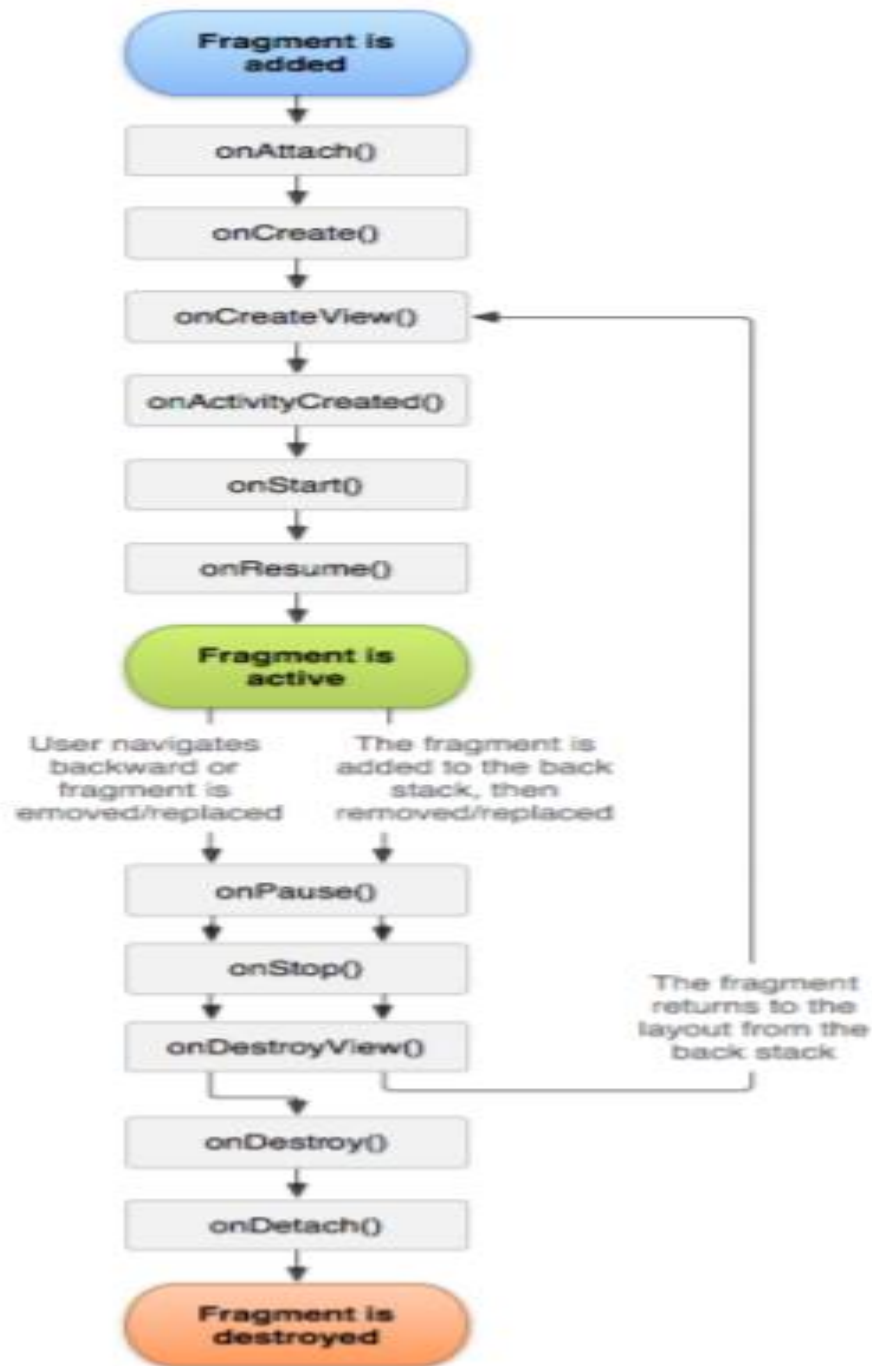


Figure 3: Fragment Life Cycle [5]

2.4 Creating Android Project

- In figure 4, Open the Android Studio, click on ‘New Project’ option under the File menu to create a new project.
- On the New Project screen, fill in the application name and Company domain.

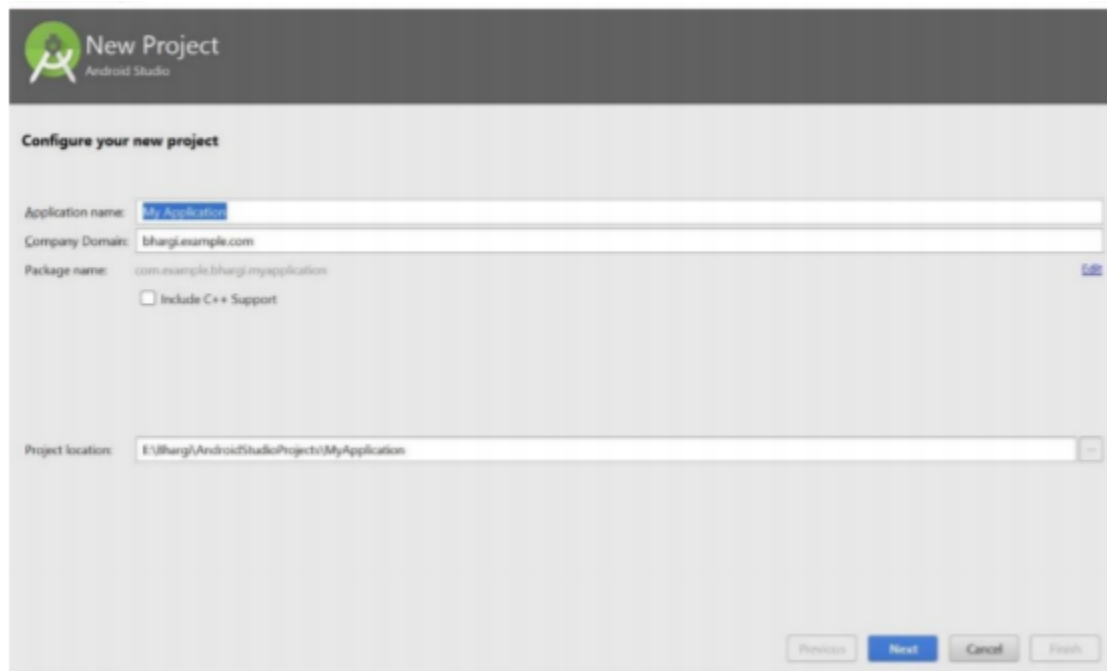


Figure 4: New Project Screen [2]

- Click on Next. Figure 5, asks to select minimum SDK for the application.
- Next, select an activity type required. In Figure 6, the window displays various kinds of activities like Blank activity, Full activity, Google Maps, Navigation Drawer, ActionBar, Settings Activity, etc. A Blank activity can perform same tasks as other activities by adding features to it.

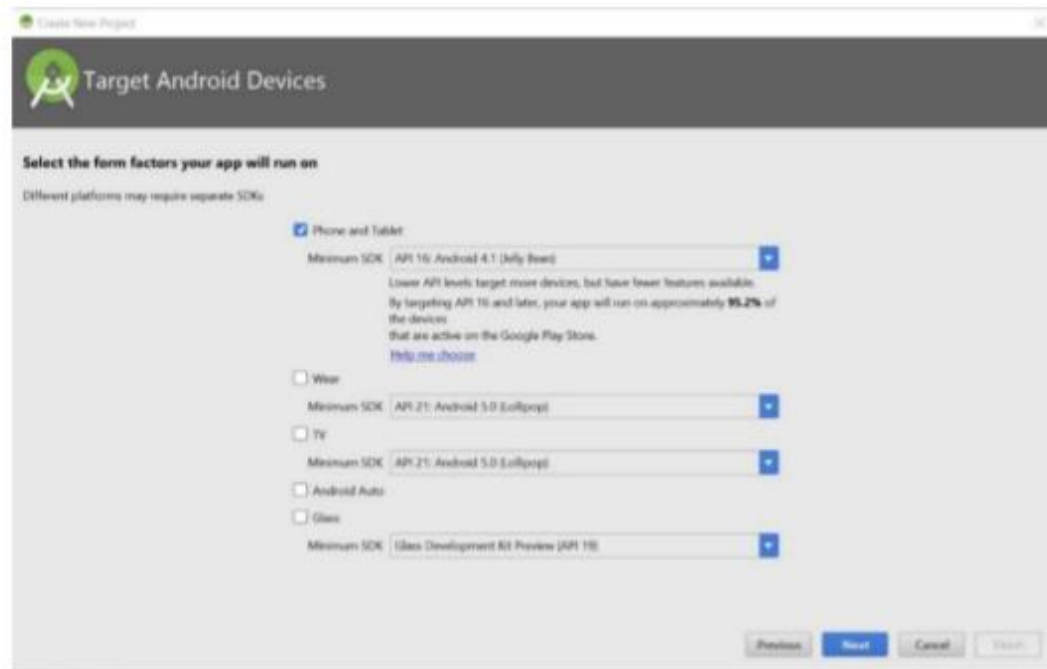


Figure 5: Android Studio Target Screen [2]

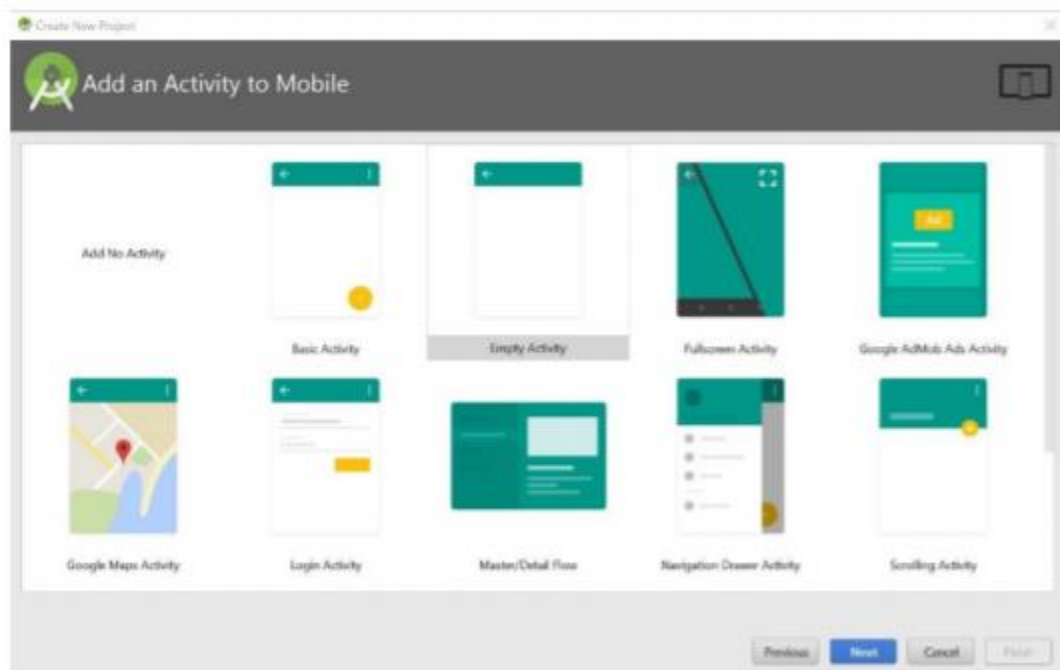


Figure 6: Activity Addition Screen [2]

- In Figure 7, the next window asks to name the activity java file and resource file. The activity layout needs to be selected as well, namely Vertical or Horizontal; by default, the layout is Vertical.

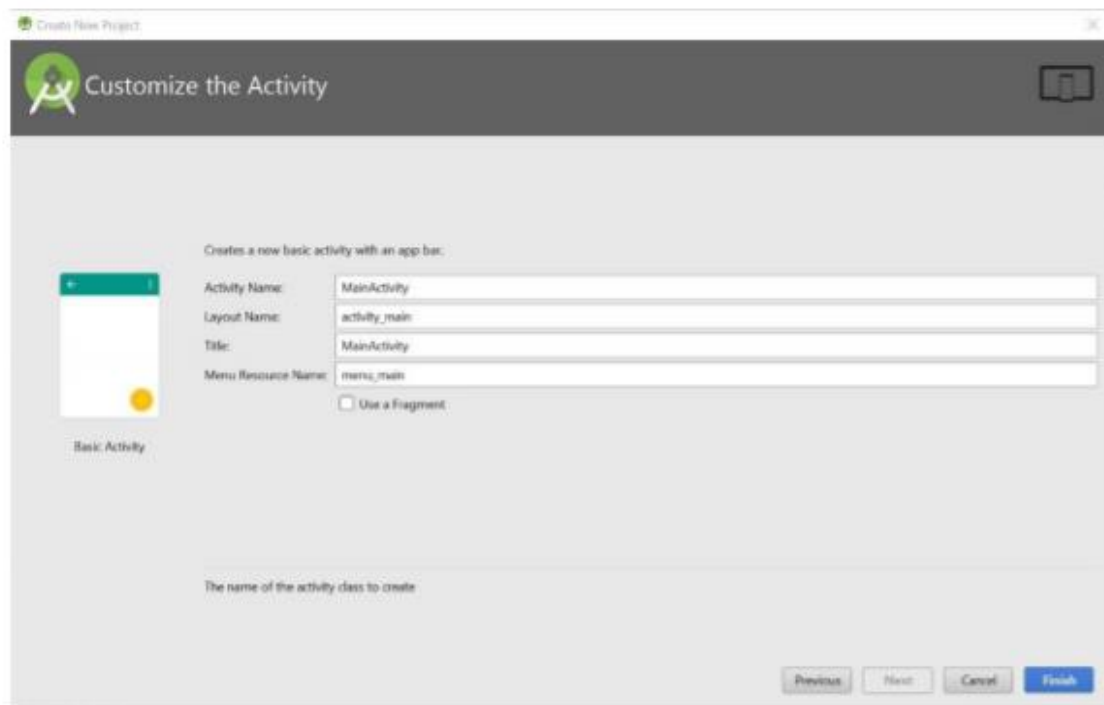


Figure 7: Customize the Activity Screen [2]

2.5. Run Android Application

There are two ways to run the Android Application – On an Android Device or using an Emulator [7].

2.5.1. On the Device

To run an application on the Android device, Developer options should enable the Settings as shown in Figure 8. The Android device then needs to be connected to the laptop with USB cable. After connecting the device, the USB debugging requires an enabling on the device [7].

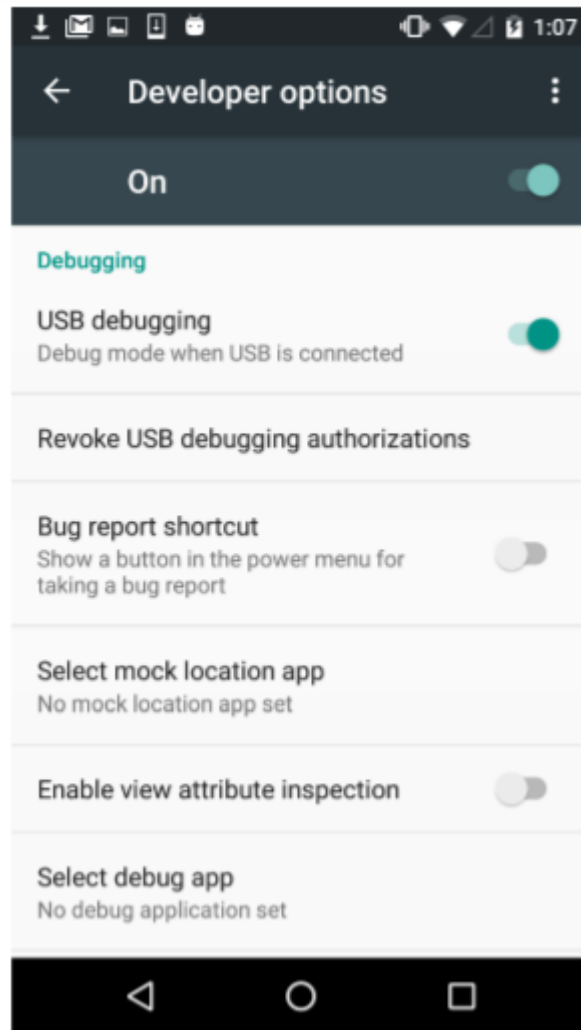


Figure 8: Device Debugging Setting Screen [2]

After successful connection to the device, click on the Run icon in the Android Studio.

The Device Chooser will appear as shown below. The application will install on the machine after selecting the correct device.

2.5.2. Using Android Emulator

To use an emulator to run an app on Android, the user will have to create an Android Virtual Device (AVD) using Android Studio [7].

- Clicking on the AVD icon opens the Android Virtual Device as shown in Figure 9. It shows all the existing/available AVDs. To create a new one, the user must click on the Create Virtual Device.

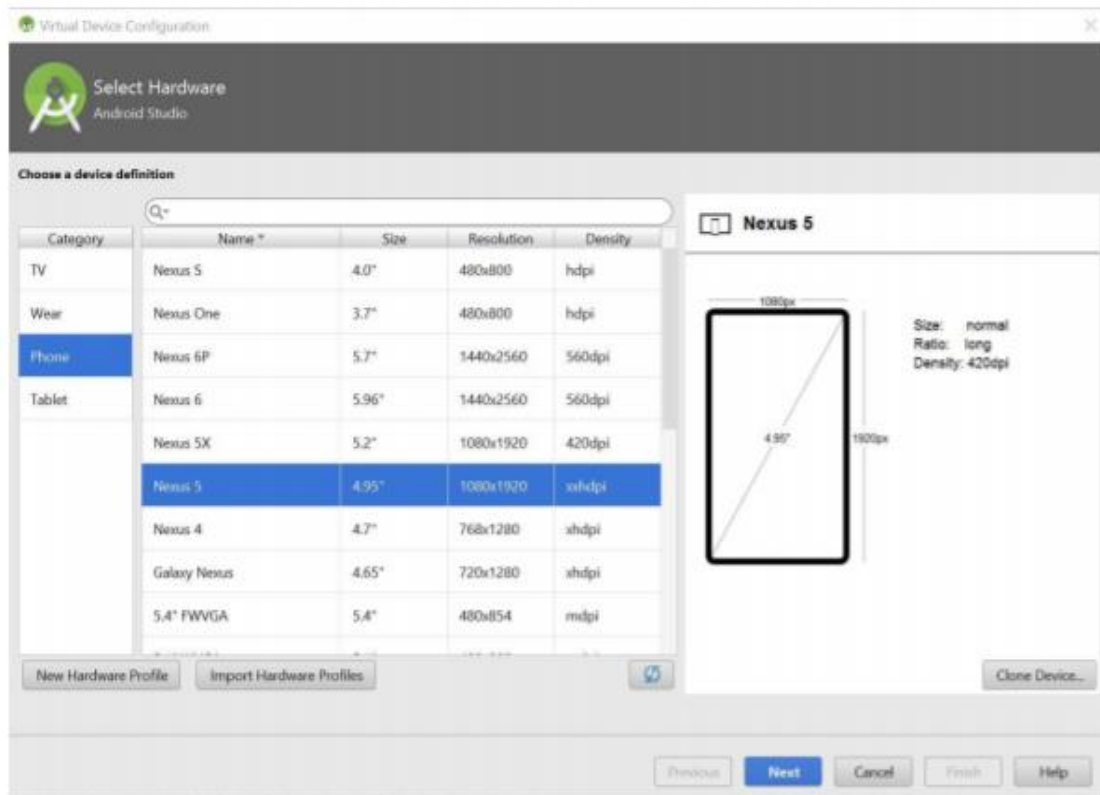


Figure 9: Android Virtual Device Screen [2]

- On the Android Device Configuration window, as shown in Figure 10, the user must select an AVD name, the category of the device (phone, tablet, TV, etc.)



Figure 10: Android Virtual Device Manager Screen [2]

3. SYSTEM DESIGN

The Personal Dietitian Application helps the user use its feature by enabling the user to sign in to the app if the user is already in the system. If the user is new to the application, he/she needs to go to the registration screen to fill out the general information to sign in to the application. After successful login, the application will take the user to his/her dashboard where the user can navigate to different functionalities or activities of the application. The user can choose to update his/her general information through the user profile section.

3.1 Application Architecture

The application architecture based on the client-server architecture. In the Personal Dietitian Mobile Application, the user acts as a client who is consuming services from the server side of the application, where the user data stored in the form of tables with the help of MySQL database.

Figure 11 represents the application's architecture.

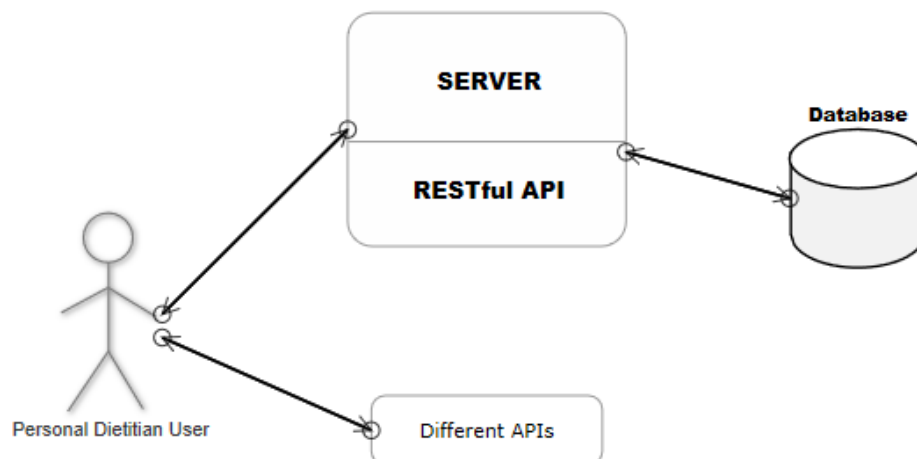


Figure 11: Application Architecture [2]

3.2 Technology Used

The architecture of the personal dietitian application represents three main parts as shown in Figure 11; first, as the client of the application that is using the app. The client and server communication enables through a series of HTTP network calls. The user interacts with the server to request the services such as user registration, login, updating a user profile, storing different user data, etc. using a simple user interface.

Secondly, the server side can define as a middleman between the client and the database side. The server provides the requests and responds to the client using the server-side scripting language, PHP is used to create RESTful API for serving applications in this project as it is carried out in the JSON format. The server side is built using a web hosting side, namely, 000webhost, which provides a phpMyAdmin tool to create PHP files and to maintain as well as create the database using the MySQL database system. The MySQL database is used as it perform CRUD operation with ease and to provide the appropriate responses to the client's requests.

Lastly, the user can make calls to different APIs, such as Nutrinitionix API and Edamam API, to get the food data from the Food Ontology database.

3.3 Libraries Used

Circular Cuboid button library – This library is used to enhance the look of the android application [8].

Volley library – This library used to make the HTTP networking calls easy for Android applications. It is also known as HTTP library or merely networking library. Through volley, the network calls become more straightforward and faster [9]. Volley mostly supports JSON connections to make requests calls, to handle responses and to set up RequestQueue the latter is one of the features Volley library provides for Android development. It works asynchronously.

MPAndroid Chart library – This library is one of the dominant chart libraries provided by Android. Figure 12 show how we can add this library to our build.gradle file. It an easy to use chart library for Android applications, it supports line-, bar-, scatter-, candlestick, bubble-, pie- etc. [10].

```
dependencies {
    compile 'com.github.PhilJay:MPAndroidChart:v3.0.2'
}
```

Figure 12: MPAndroidChart Library [10]

Support library – To improve the user interface of the Android application, the support library provides classes for application components and user interface widgets. This library offers backward compatible implementation as a central core platform features [11]. In this project, I have used Coordinate layout, CardView, CircularImage Button, as well as the following (Figure 13), to enhance the user interface design.

```
compile 'com.android.support:appcompat-v7:25.3.1'
compile 'com.android.support.constraint:constraint-layout:1.0.2'
compile 'com.android.support:design:25.3.1'
compile 'com.android.support:support-v4:25.3.1'
compile 'com.android.support:multidex:1.0.1'
compile 'com.android.support:cardview-v7:25.3.1'
```

Figure 13: Support Dependencies Code Snippet

Glide library – Figure 14 shows the Glide library dependency in the gradle file. It is a fast and efficient media management and image loading framework for Android [12]. It supports animation and images GIFs, fetching, decoding and displaying video stills, etc.

```

repositories {
    mavenCentral()
    maven { url 'https://maven.google.com' }
}

dependencies {
    compile 'com.github.bumptech.glide:glide:4.3.1'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.3.1'
}

```

Figure 14: Glide Library Dependencies [12]

Zxing library – Figure 15, shows the Zebra Crossing barcode scanning library is an open source, a multi-format 1D//2D barcode scanner for Android application [13].

FitChart library – This is a wheel chart library used to enhance the user interface design for the Android application [14].

```

dependencies {
    ...
    compile 'com.txusballesteros:FitChart:1.0'
}

```

```

<com.txusballesteros.widgets.FitChart
    android:layout_width="200dp"
    android:layout_height="200dp" />

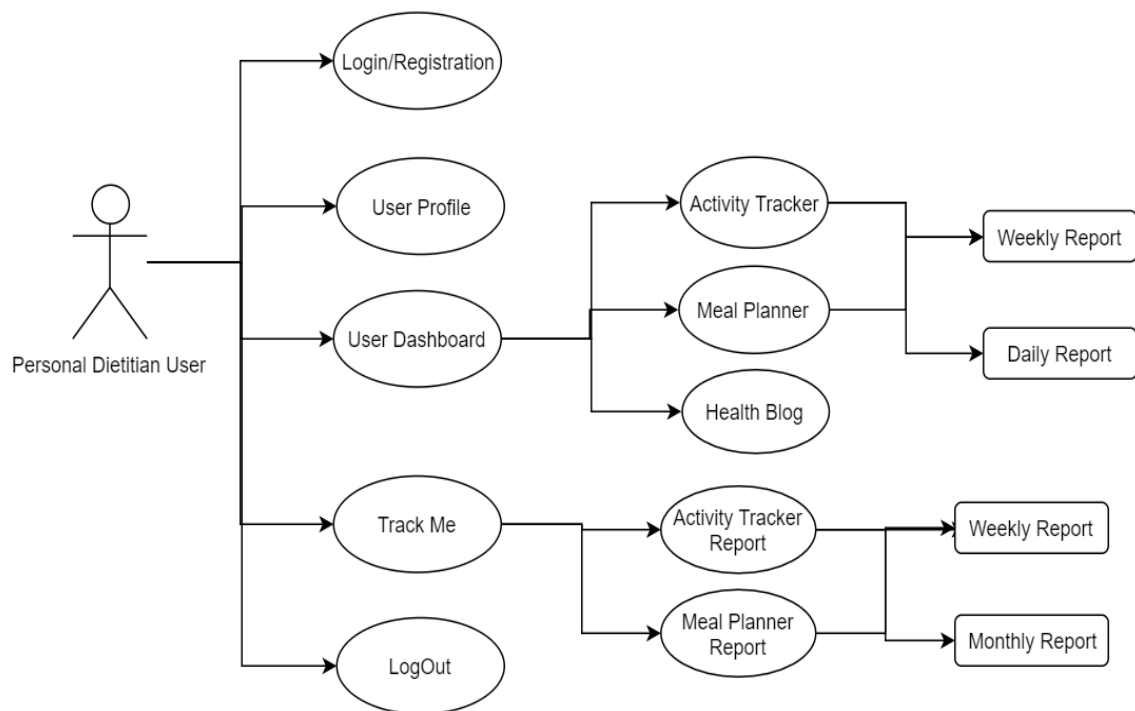
```

Figure 15: Using FitChart Library [14]

3.4 Project Workflow

Figure 17, on page 23, shows the complete workflow of the proposed application. On the start of the application, the login screen will display to the user to start the application, or if the user is

new to the application, they can navigate themselves to the registration page to register themselves with the application. There is another choice namely forgotten password which can be used when the user forgot their password. After successful sign-in, the user is permitted to use the functionalities of the Personal Dietitian application. The user dashboard provides the user the abilities to choose which activity or feature he/she want to use.



Figures 16: Use Case Diagram of an Application

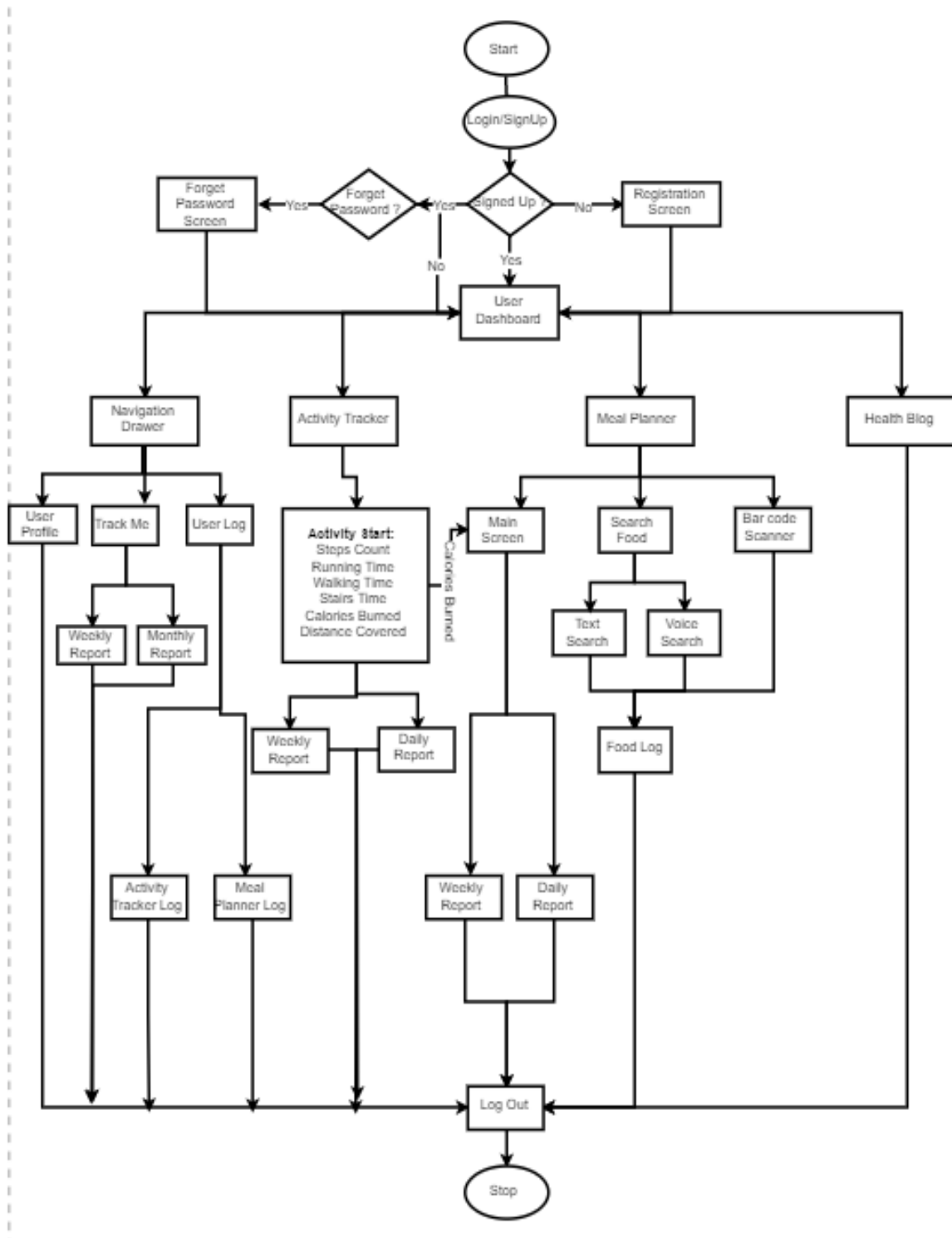


Figure 17: Project Flow Diagram

3.4.1 Registration/Sign Up

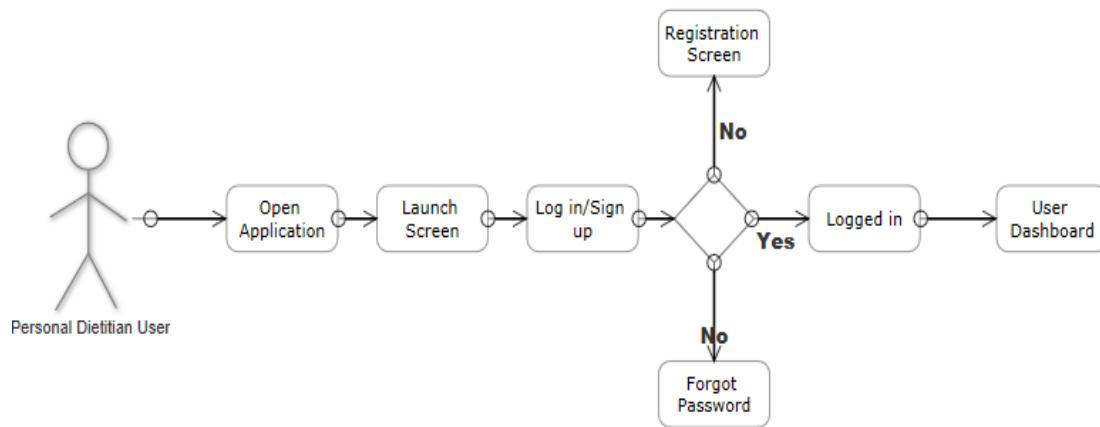


Figure 18: Login/Registration Flow Diagram

In Figure 18, When the application is first launched it will present the login/sign up screen. The user can either login into the application or register themselves. If they are new to the application, they can register by filling out a registration form from the register option for the app. To register themselves with the Personal Dietitian Application, the user will then be asked to provide general information, such as name, email, weight, height, age, gender, activity level, allergy, etc. Once the user registers, they can start using the application to login to it. The login screen has more options such as show & hide the password. If the user forgets their password, he/she can regain the password by requesting the password sent to their provided email address.

3.4.2 User Dashboard

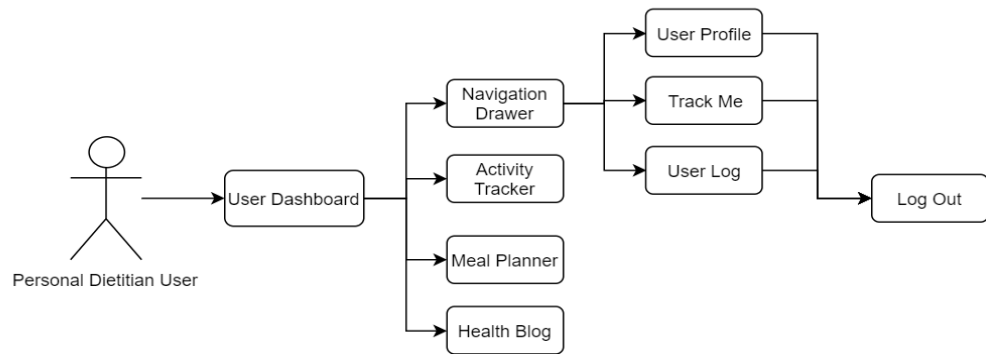


Figure 19: User Dashboard Flow Diagram

After the user successfully logged in the application, the user will come directly to their dashboard screen from where they can choose a different activity to jump over and use them accordingly as shown in Figure 19. Mainly there are four activities on the dashboard; navigation drawer, activity tracker, meal planner, and health blog. Also, the user can choose the navigation drawer to select the profile option to make some updates, can see the user log and, can track themselves, seeing the weekly and monthly progress report.

3.4.3 User Profile

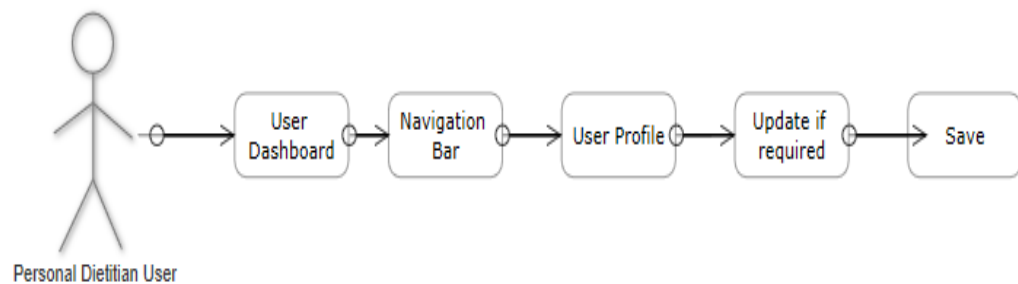


Figure 20: User Profile Flow Diagram

In Figure 20, the user can choose the user profile option from the navigation drawer/bar to look into their general information, or they can make updates if required and later save the details in the database for future reference. The user can also choose a profile image over the user profile screen using the implicit intent to use the image gallery of the mobile to upload a picture.

3.4.4 Activity Tracker

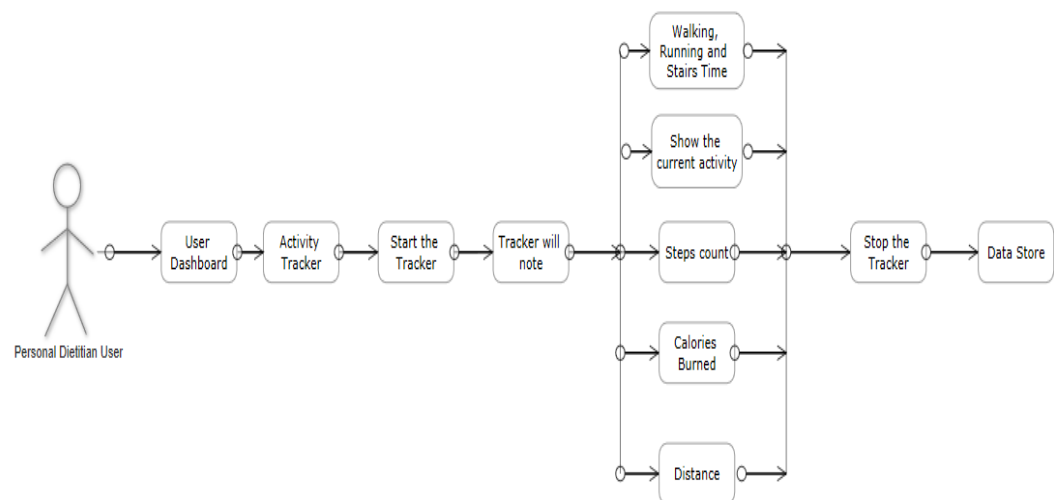


Figure 21: Activity Tracker Flow Diagram

In Figure 21, the Activity Tracker will track the steps count at the set alarm using the AlarmManager method. Also, the tracker will note and show the current activity performed by the user at that times running, walking, stairs time, total step count, calories burned and lastly the distance covered by the user using the Activity tracker functionality. The user can also look into the weekly progress in the form of a bar chart by simply clicking the different icons such as steps count, walking min, running min,

stairs min, calories burned and distance. This will open a new activity page where the user can see the bar chart for his/her weekly progress and also the older saved data according to the dates in the RecyclerView.

3.4.5 Meal Planner

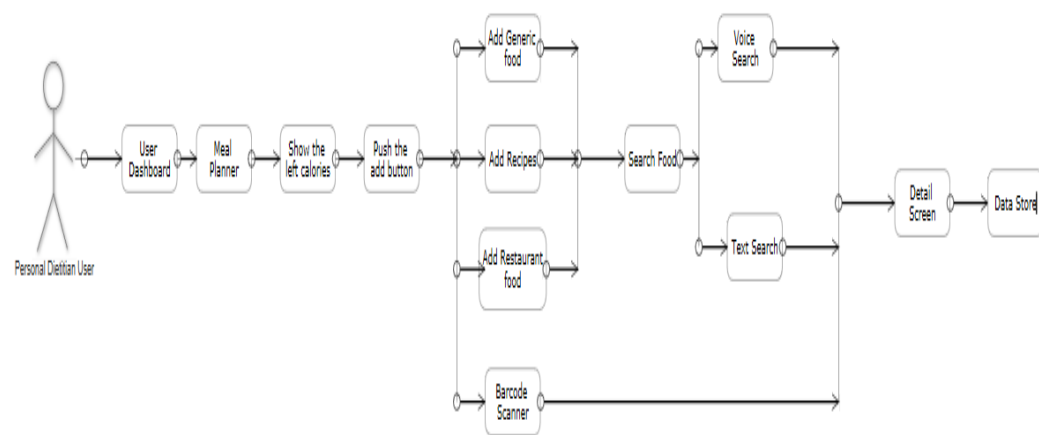


Figure 22: Meal Planner Flow Diagram

In Figure 22, the user can choose the option of Meal Planner Activity from the dashboard screen. In the Meal Planner, the activity will calculate the left calories on the top of the screen, which will take the total calorie required by the user, that can be figured using general information and the current activity level the user has such as sedentary or light active, etc. The calculation also takes the total calories intake by the user using the food calories from the Meal Planner screen and the calories burned using the Activity Tracker is also used in the Meal Planner activity to calculate the left calories that the user can consume. The user can search the food items using the voice search as well as text search to add the food item to his/her meal plan. Similar to the Activity Tracker, Meal

Planner will also provide a separate weekly bar chart for the total consumed food calories by the user. Moreover, the user can also click the breakfast, lunch, and dinner to get to see the calories consumed by the user in the form of a bar chart, as well as showing the calories consumed data chronologically, in a RecyclerView below the bar chat.

3.4.6 Activity Tracker: Progress Report

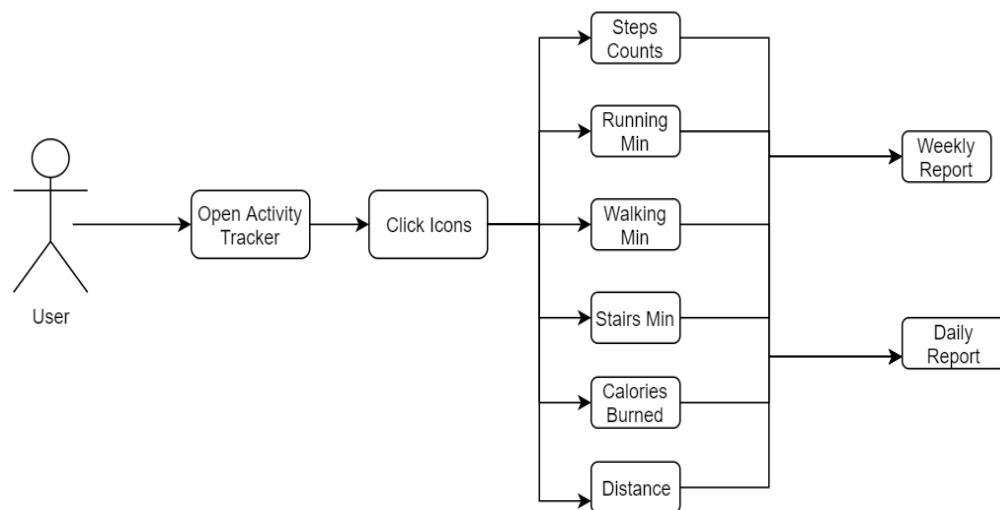


Figure 23: Activity Tracker Progress Report Flow Diagram

Through the Activity Tracker activity, as shown in Figure 23, the user can see their progress on the weekly or daily report basis. The Activity Tracker report is represented in the form of a bar chart and shows the calories burned, distance, running, walking, climbing time and steps count for that day as well as for the whole week. User's report can be viewed by clicking the icons on the Activity Tracker page. For example, if the user clicks the steps count circle, then the new activity opens, which shows the weekly bar chart report and daily date wise report in the RecyclerView for the steps count.

Similarly, the user can see the other weekly and daily report by clicking the other icons from the Activity Tracker page.

3.4.7 Meal Planner: Progress Report

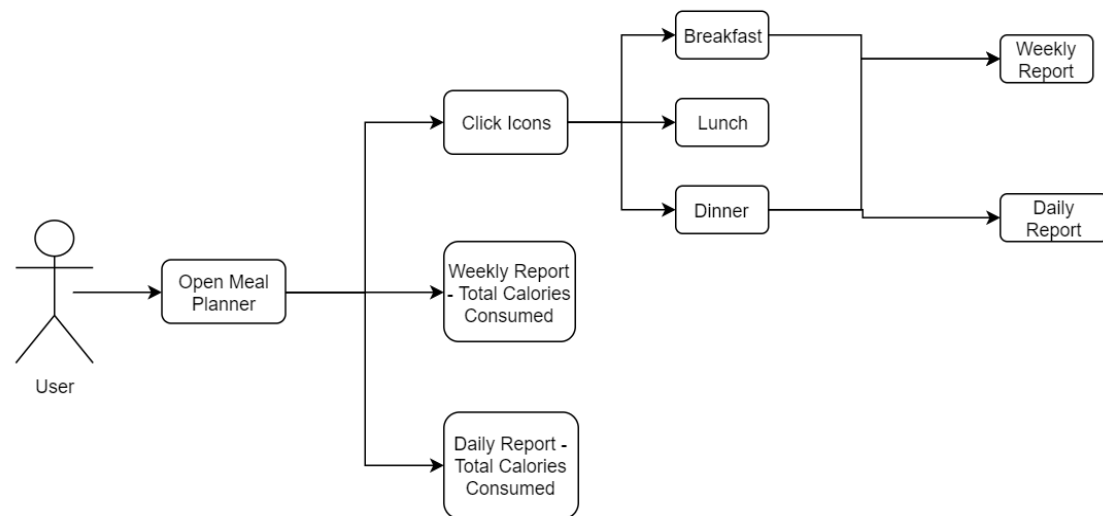


Figure 24: Meal Planner Report Flow Diagram

Through the Meal Planner activity, as shown in Figure 24, the user can see their progress on the weekly and daily report basis. Similar to the Activity Tracker, the Meal Planner activity can also help the user to see the weekly and daily progress report by clicking the icons on the activity page. When the user clicks the breakfast icon a new activity page opens, which lets the user see his/her progress report in the form of bar chart. The weekly report is represented in the bar chart manner, while the daily chronological report is shown in the RecyclerView below the bar chart. This will help the user to look at the individual data of intake calories taken during breakfast, lunch, and dinner.

3.4.8 Weekly and Monthly Progress Report

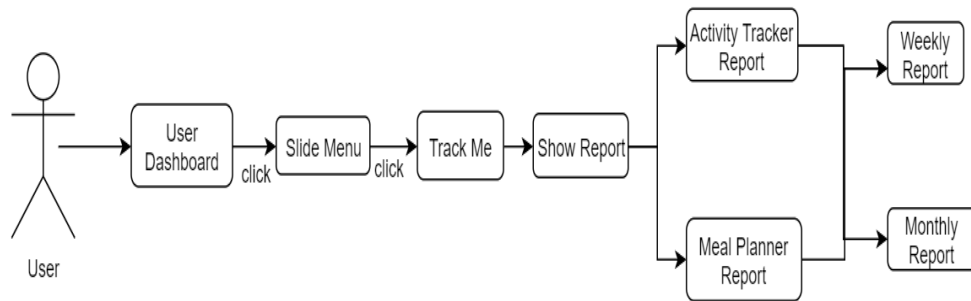


Figure 25: Weekly and Monthly Report Flow Diagram

Through the Track Me report activity, as shown in Figure 25, the user can see their progress on the weekly or monthly report basis. The user can access the progress report by clicking the slide menu bar from the dashboard and then choose the Track Me option from the list to see the weekly and monthly progress report. The Activity Tracker report represented in the form of a line chart and showed the calories burn, running, walking, stairs time and steps count data for that day. Also, the Activity Tracker Report also shows the minimum, maximum and average calories burned by the user on a weekly and monthly basis. Whereas, the Meal Planner Report represented in the form of the pie chart and is used to show the weekly and monthly report on the nutrition consumed by the user over the week and month. Moreover, the Meal Planner Report also shows the total nutritional values of different food items as well as the average values of different nutritional intake by the user on the weekly and monthly basis.

3.4.9 Health Blog

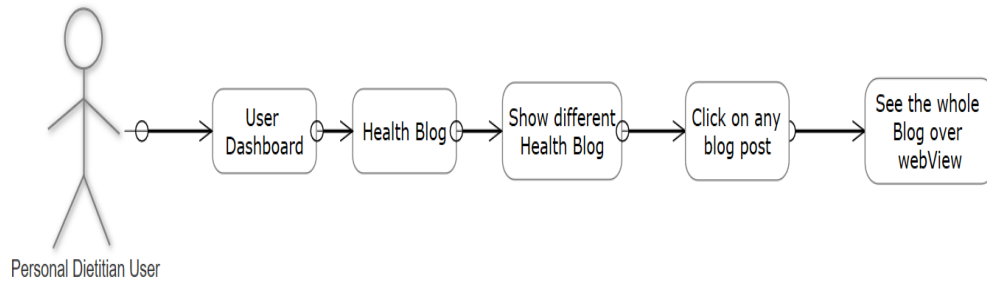


Figure 26: Health Blog Flow Diagram

In Figure 26, the user can select this option from the User Dashboard as well as from the navigation drawer. This activity is used to motivate the user by reading the latest health blog related to the nutritional or other health problem and their solution. The activity can be expanded to view the blog over the browsers.

4. APPLICATION IMPLEMENTATION

4.1 User Registration

To use the application services, one needs to register themselves with the Personal Dietician application. As shown in Figure 27, to register every user needs to provide general information about themselves such as a valid email address, full name, weight, height, gender, age, activity level, food habit, and allergy. All the fields need to be written accurately otherwise the application will generate an error. For example, the phone number should be precisely ten digits or the email address is in the incorrect format.

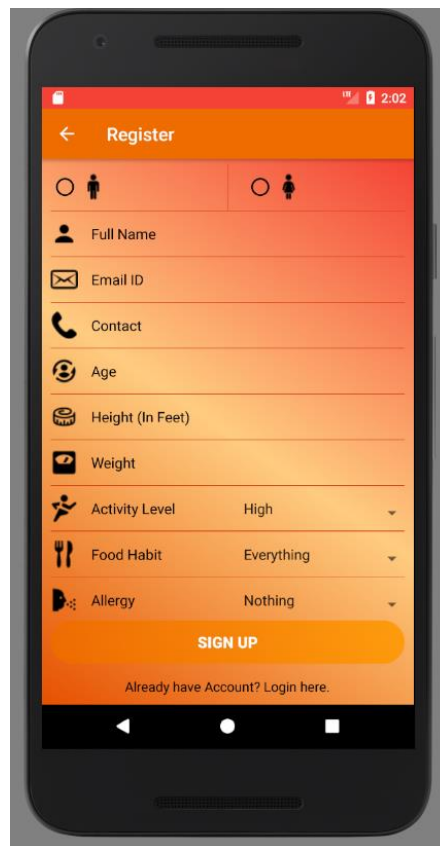


Figure 27: User Registration Screen

When the user filled out the registration page with the valid values and clicked on the sign-up button, a request is sent to the server with the user values in the JSON format, including the valid Registration endpoint API URL. All the credentials entered by the user will be saved in the database. The code to accomplish this activity is shown in Figure 28. After successful registration, the application will take the user to the next screen, that is the Login screen to enter into the application.

```
public class Register extends AppCompatActivity implements OnWebCall {

    RadioGroup radioGroup_gender, radioGroup_foodhabit;
    Button login_signup;
    TextView already_user;
    EditText user_name, user_email, user_phone, user_weight, user_height, user_age;
    String user_nameString, user_emailString, user_phoneString, user_weightString, user_heightString, user_ageString,
        user_activityString, user_genderString, user_foodHabitString;
    Spinner spiActivityLevel, spiFoodHabit, spiUserAllergy;
    private final String Fill_user_name = "Please enter user name";
    //private final String Fill_user_nu_meal = "Please enter number of meal";
    private final String Fill_user_correct_email = "Please enter valid email";
    private final String Fill_user_email = "Please enter user email";
    private final String Fill_user_phone = "Please enter user phone number";
    private final String Fill_user_phone_invalid = "Please enter valid number";
    private final String Fill_user_weight = "Please enter user weight";
    private final String Fill_user_height = "Please enter user height";
    private final String Fill_user_gender = "Please enter user gender";
    private final String Fill_user_age = "Please enter user age";
    private final String Fill_user_foodhabit = "Please enter user food habit";
    private final String Fill_user_allergy = "Please enter user allergy";
    private final String Fill_user_activity = "Please enter user activity level";
    private final String Network_not_connected = "Internet Not Connected! Please Try Again";
    private String TAG = Register.class.getSimpleName();
```

Figure 28: Registration Activity Code

In the above code the 'Register' class, contains the code for the registration process of the user with the application. The activity is implemented using 'onClickListner' interface on the sign-up button. The 'OnCreate()' method initialized the activity (Figure 29).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);
    init();
}
```

Figure 29: onCreate Method of Register Activity


```

private void init() {

    user_name = (EditText) findViewById(R.id.user_name);
    user_email = (EditText) findViewById(R.id.user_email);
    user_phone = (EditText) findViewById(R.id.user_phone);
    user_height = (EditText) findViewById(R.id.user_height);
    user_weight = (EditText) findViewById(R.id.user_weight);
    user_age = (EditText) findViewById(R.id.user_age);
    //user_allergy = (EditText) findViewById(R.id.user_allergy);
    spiActivityLevel = (Spinner) findViewById(R.id.userActivityLevel);
    spiFoodHabit = (Spinner) findViewById(R.id.userFoodHabit);
    spiUserAllergy = (Spinner) findViewById(R.id.userAllergy);
    //user_activityLevel = (EditText) findViewById(R.id.user_activity_le
    //user_foodHabit = (EditText) findViewById(R.id.user_food_habit);
    user_genderString = "";
    radioGroup_gender = (RadioGroup) findViewById(R.id.userGender);
    //radioGroup_foodhabit = (RadioGroup) findViewById(R.id.radioGroup_fo
    radioGroup_gender.setOnCheckedChangeListener((group, checkedId) → {
        if (checkedId == R.id.male) {
            user_genderString = "male";
            Log.d(TAG, "Male selected" + user_genderString);
        } else if (checkedId == R.id.female) {
            user_genderString = "female";
            Log.d(TAG, "Female selected" + user_genderString);
        }
    });
}

```

Figure 30: init() Method of Register Activity

This method contains the XML layout file of the activity to show the user interface along with the module init() method. In the code shown in Figure 30, module init() method shows that the ‘findViewById’ is used to connect the user interface programmatically to provide the interaction. This code also generates the validation logic for the registration process to check if the enter values is a valid entry or not.

In the code shown in Figure 31 and 32, when the user clicks the sign-up button which in turns calls the ‘onCreate()’ method, which will take the credentials entered by the user. It also checks all parameters, such as the validity of the email, phone number, age, connectivity to the internet, etc.

Once the validation test pass, the user entry will be stored in the database using the HTTP network calls to take the JSON objects. In my case, I have created a separate class to make this connection so that I need to call the object of this class to get the JSON object and store it in the database.

```
login_signup = (Button) findViewById(R.id.signup_button);
login_signup.setOnClickListener((view) -> {

    user_nameString = user_name.getText().toString().trim();
    user_emailString = user_email.getText().toString().trim();
    user_phoneString = user_phone.getText().toString().trim();
    user_weightString = user_weight.getText().toString().trim();
    user_heightString = user_height.getText().toString().trim();
    user_ageString = user_age.getText().toString().trim();
    user_allergyString = (String) spiUserAllergy.getSelectedItem();
    user_activityString = (String) spiActivityLevel.getSelectedItem();
    user_foodHabitString = (String) spiFoodHabit.getSelectedItem();

    if (user_nameString.equalsIgnoreCase("")) {
        Logger.showToast(Register.this, Fill_user_name);
    } else if (user_emailString.equalsIgnoreCase("")) {
        Logger.showToast(Register.this, Fill_user_email);
    } else if (!(isValidEmail(user_emailString))) {
        Logger.showToast(Register.this, Fill_user_correct_email);
    } else if (user_phoneString.equalsIgnoreCase("")) {
        Logger.showToast(Register.this, Fill_user_phone);
    } else if (user_phoneString.length() != 10) {
        Logger.showToast(Register.this, Fill_user_phone_invalid);
    } else if (user_weightString.equalsIgnoreCase("")) {
        Logger.showToast(Register.this, Fill_user_weight);
    } else if (user_heightString.equalsIgnoreCase("")) {
```

Figure 31: Registration Validation Code

```
private void handleRegister(String response) {
    try {
        JSONObject jobj = new JSONObject(response);
        String _response = jobj.getString("response");
        String message = jobj.getString("message");
        if (OnWebCall != null) {
            if (_response.equalsIgnoreCase("200")) {
                OnWebCall.OnWebCallSuccess(message);
            } else {
                OnWebCall.OnWebCallError(message);
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        if (OnWebCall != null)
            OnWebCall.OnWebCallError(ex.toString());
    }
}
```

Figure 32: WebCall Class to Handle the HTTP Network Connection

4.2 User Login

Any registered user can log in with their valid credentials and enter into the Personal Dietitian application as shown in Figure 33. After the successful login, the application will take the user to his/her dashboard. In the dashboard, the user has a choice to jump into different activities using the navigation drawer as well as the through the main screen of user dashboard. However, with the unsuccessful login, the user will encounter an error message from the application. Also if the user is unable to remember their password, they can use the “forgot password” link to have an email sent from the application to retrieve their password.

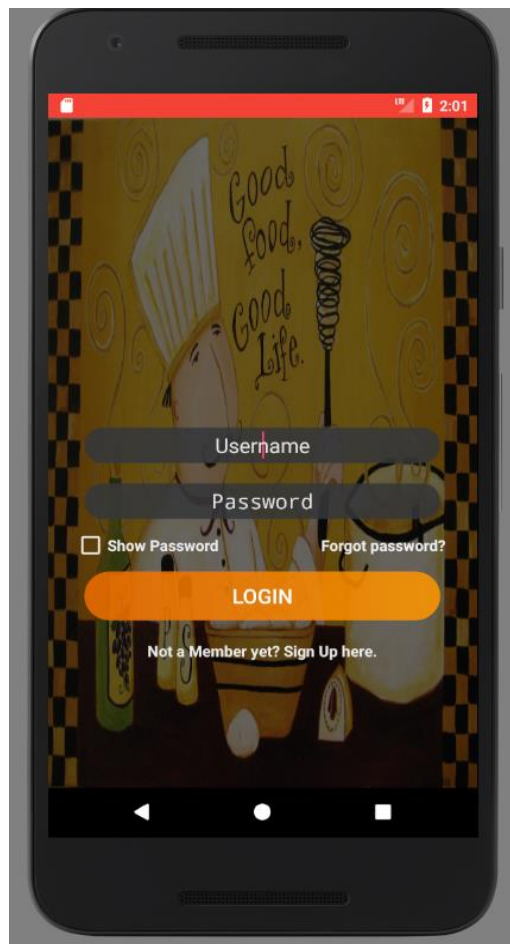


Figure 33: Login Screen

In the login screen, the user can also find another option to hide or show his/her password while trying to login to the application. The code for this functionality shown in Figure 34 below.

```

/* Set check listener over checkbox for showing and hiding password. */
show_hide_password = (CheckBox) findViewById(R.id.show_hide_password);
show_hide_password.setOnCheckedChangeListener((button, isChecked) → {

    // If it is checked then show password else hide password
    if (isChecked) {
        show_hide_password.setText("Hide Password"); // change checkbox text

        password.setInputType(InputType.TYPE_CLASS_TEXT);
        password.setTransformationMethod(HideReturnsTransformationMethod.getInstance()); // show password
    } else {
        show_hide_password.setText("Show Password"); // change checkbox text

        password.setInputType(InputType.TYPE_CLASS_TEXT
            | InputType.TYPE_TEXT_VARIATION_PASSWORD);
        password.setTransformationMethod(PasswordTransformationMethod.getInstance()); // hide password
    }
});

```

Figure 34: Show/Hide Password Code Snippet

This code is using the inbuilt method which is `HideReturnsTransformationMethod` to get the instance of the created password by the user to hide from the login screen and show the star symbols in place of the password.

In the code snippet shown in Figure 35, the login button implemented with the `setOnClickedListener()` method which will invoke the `onClick()` method to send the requested data to the database to check the credentials of the user. Also, it will store the data in the “login model.” To maintain the structure of the data Android architecture requires a model structure defined in an application. Inside a model class, there are mainly three methods [16].

- Constructor: A bridge between model and adapter class.
- Getter method: Method used for getting values of any property.
- Setter method: Method used for setting the value of any property.

```
login_signup = (TextView) findViewById(R.id.login_signup);
login_signup.setOnClickListener((view) → { sendToRegister(); });

login_button = (Button) findViewById(R.id.login_button);
login_button.setOnClickListener((view) → {
    userNameString = user_name.getText().toString().trim();
    passwordString = password.getText().toString().trim();

    if (userNameString.equalsIgnoreCase("")) {
        Logger.showToast(Login.this, "Please enter username");
    } else if (passwordString.equalsIgnoreCase("")) {
        Logger.showToast(Login.this, "Please enter password");
    } else {
        LoginModel model = new LoginModel();
        model.setEmail(userNameString);
        model.setPassword(passwordString);

        WebCalls webCalls = new WebCalls(Login.this);
        webCalls.showProgress(true);
        webCalls.setWebCallListner(Login.this);
        webCalls.login(model);
    }
});
```

Figure 35: Login Activity Code Snippet

4.3 Forgot Password

In Figure 36, the activity can be used by the user when he/she was unable to remember their password. Through, this activity the user can regain their password by typing their valid and registered email address on the forgot password screen. After submitting the request, the database will respond with their password as an email to the user.

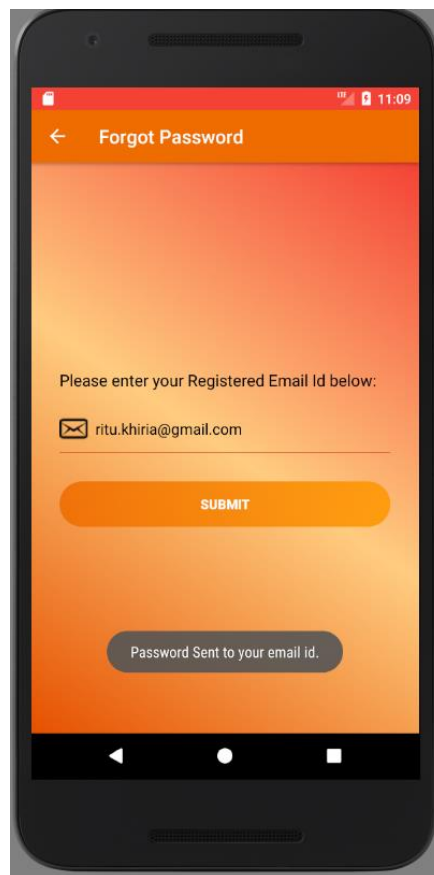


Figure 36: Forgot Password Screen

In the code snippet shown in Figure 37, the onCreate method defines an everyday HTTP network connection class object to use its methods to connect to the database or in other words to the

server side. The `setWebCallListner()` method of `WebCalls` class is establishing a connection between the forgot password data request and database.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_forgot_password);
    email = (EditText) findViewById(R.id.email_forgot);
    submit = (Button) findViewById(R.id.submit);
    submit.setOnClickListener((v) -> {
        String emailStr = email.getText().toString().trim();
        if(emailStr.equalsIgnoreCase("")){
            Logger.showToast(ForgotPassword.this, Fill_user_email);
        } else if(!Utils.isValidEmail(emailStr)){
            Logger.showToast(ForgotPassword.this, Fill_user_correct_email);
        } else {
            ForgotModel model = new ForgotModel();
            model.setEmail(emailStr);
            WebCalls webCalls = new WebCalls(ForgotPassword.this);
            webCalls.setWebCallListner(ForgotPassword.this);
            webCalls.showProgress(true);
            webCalls.ForgotPassword(model);
        }
    })
}
```

Figure 37: Forgot Password Code Snippet

4.4 User Dashboard

The user dashboard activity, shown in Figure 38, is the main screen or home screen for the registered user of the Personal Dietitian application. User dashboard provides four different activities namely Navigation Drawer, Activity Tracker, Meal Planner, and Health blog section. So the user can choose which functionality he/she wants to use at that time. In the navigation drawer, the user can navigate to different activity including user profile, track me, user log activity screen and log out button. The dashboard will even show the user's profile image which can update the user profile by the user.



Figure 38: User Dashboard Screen

In the code, shown in Figure 39, the dashboard activity class is making an explicit intent call to start different activities on the click of the desired button. An explicit intent is used to call or start any other activity class within the application. Whereas, in Figure 40, the code handles the view of items that need to add to the navigation drawer. Also, Figure 41, shows the navigation drawer items or list initialization where these items are also using the explicit intent functionality to help the user jumps into different activities, as the user clicks one of them it will start the clicked activity in a new screen.


```

private void init() {
    session = Session.getSession(MainDashboard.this);
    Logger.log("user id::" + session.getUserId());

    mealPlanner = ((CuboidButton) findViewById(R.id.meal_cuboid_button));
    mealPlanner.setOnClickListener((view) → {
        startActivity(new Intent(MainDashboard.this, DietType.class));
    });

    activityTracker = ((CuboidButton) findViewById(R.id.activity_cuboid_button));
    activityTracker.setOnClickListener((view) → {
        startActivity(new Intent(MainDashboard.this, MySteps.class));
    });

    healthBlog = ((CuboidButton) findViewById(R.id.blog_cuboid_button));
    healthBlog.setOnClickListener((view) → {
        startActivity(new Intent(MainDashboard.this, HealthBlog.class));
    });
}

```

Figure 39: User Dashboard Explicit Intent

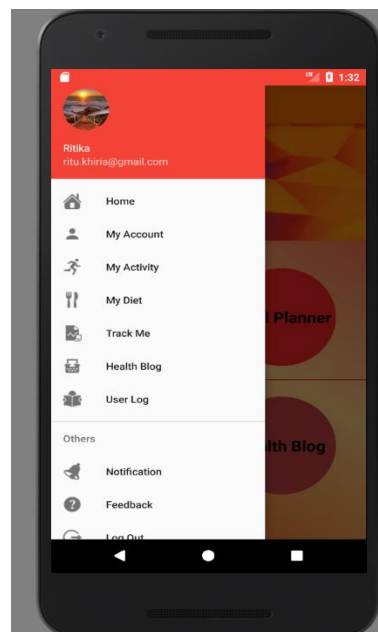


Figure 40: Navigation Drawer Screen

```

@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.home) {
        startActivity(new Intent(MainDashboard.this, MainDashboard.class).setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP));
    } else if (id == R.id.my_account) {
        startActivity(new Intent(MainDashboard.this, Profile.class));
    } else if (id == R.id.my_activity) {
        startActivity(new Intent(MainDashboard.this, MySteps.class));
    } else if (id == R.id.my_activity) {
        startActivity(new Intent(MainDashboard.this, HistoryActivity.class));
    } else if (id == R.id.my_diet) {
        startActivity(new Intent(MainDashboard.this, DietType.class));
    } else if (id == R.id.track_me_report) {
        startActivity(new Intent(MainDashboard.this, TrackMeNewActivity.class));
    } else if (id == R.id.blog_post) {
        startActivity(new Intent(MainDashboard.this, HealthBlog.class));
    } else if (id == R.id.my_log) {
        startActivity(new Intent(MainDashboard.this, UserLogActivity.class));
    } else if (id == R.id.help) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("plain/text");
        intent.putExtra(Intent.EXTRA_EMAIL, new String[] { "feedback@admin.address" });
        startActivity(Intent.createChooser(intent, ""));
    } else if (id == R.id.nav_logout) {
        session.setUserId("");
        startActivity(new Intent(MainDashboard.this, Login.class));
    }
}

```

Figure 41: Navigation Drawer List Initialization

4.5 User Profile

In Figure 42, the user can update his/her general information provided during the registration process. The user can use this activity using the navigation drawer on the user dashboard activity. The user requests the server to process the POST request with the user's data that he/she has changed using the profile activity.

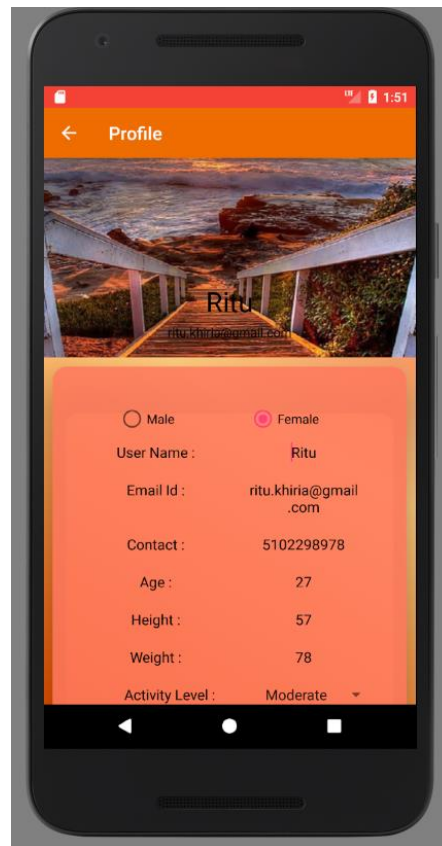


Figure 42: User Profile Screen

In Figure 43 code snippet, when the user clicks the save button, the `onClick` method evokes and collects all the data entered by the user and passed to the database as a JSON object in the form of POST request.

```

save = (Button)findViewById(R.id.save_button);
save.setOnClickListener((view) → {
    // int no_meals = Integer.parseInt(edit_diet_plan.getText().toString());
    //int setAlarm = (Integer) (18/no_meals);
    user_nameString = edit_name.getText().toString().trim();
    user_emailString = edit_email.getText().toString().trim();
    user_phoneString = edit_phone.getText().toString().trim();
    user_weightString = edit_weight.getText().toString().trim();
    user_heightString = edit_height.getText().toString().trim();
    user_ageString = edit_age.getText().toString().trim();
    /*user_allergyString = edit_allergy.getText().toString().trim();
    user_activityString = edit_activityLevel.getText().toString().trim();
    user_foodhabitString = edit_foodHabit.getText().toString().trim();*/

    if(user_nameString.equalsIgnoreCase("")) {
        Logger.showToast(Profile.this, Fill_user_name);
    } else if(user_emailString.equalsIgnoreCase("")) {
        Logger.showToast(Profile.this, Fill_user_email);
    } else if(!isValidEmail(user_emailString)) {
        Logger.showToast(Profile.this, Fill_user_correct_email);
    } else if(user_phoneString.equalsIgnoreCase("")) {
        Logger.showToast(Profile.this, Fill_user_phone);
    } else if(user_phoneString.length() != 10) {
        Logger.showToast(Profile.this, Fill_user_phone_invalid);
    } else if(user_weightString.equalsIgnoreCase("")) {
        Logger.showToast(Profile.this, Fill_user_weight);
    } else if(user_heightString.equalsIgnoreCase("")) {
        Logger.showToast(Profile.this, Fill_user_height);
    } else if(user_ageString.equalsIgnoreCase("")) {

} else {
    if(isOnline(Profile.this)) {
        Log.d(tag, "Your profile updated successfully");
        EditModel model = new EditModel();
        model.setEmail(user_emailString);
        model.setContact(user_phoneString);
        model.setName(user_nameString);
        model.setAge(user_ageString);
        model.setHeight(user_heightString);
        model.setWeight(user_weightString);
        model.setGender(user_genderString);
        // model.setFoodHabit(user_foodhabitString);
        model.setFoodHabit(user_foodhabitString);
        model.setAllergy(user_allergyString);
        model.setActivityLevel(user_activityString);
        model.setId(session.getUserId());
        WebCalls webCalls = new WebCalls(Profile.this);
        webCalls.setWebCallListner(Profile.this);
        webCalls.showProgress(true);
        webCalls.EditUser(model);
    }
    else
    {
        Logger.showToast(Profile.this, Network_not_connected);
    }
}
}

```

Figure 43: Profile Saving Data

In this activity, the user can also select their profile image using their device gallery. To choose a profile image the activity is using an implicit intent to call the image gallery store in the device as shown in Figure 44. To, upload the picture a library the glide library [12] is used. The Glide Android library is used to wrap the pictures or video decoding, memory and disk caching and more into a simple and easy to use interface.

```

/* Use to get the image from the phone gallery. */
profileImage.setOnClickListener(
    (arg0) → {
        if (hasPermissions()) {
            // Intent to gallery
            Intent in = new Intent(Intent.ACTION_PICK);
            in.setType("image/*");
            startActivityForResult(in, select_photo); // start activity to get the image
        } else {
            //our app doesn't have permissions, So i m requesting permissions.
            requestPerms();
        }
    });

```

Figure 44: Implicit Intent to get the Image

```

public static Bitmap decodeUri(Context context, Uri uri,
    final int requiredSize) throws FileNotFoundException {
    BitmapFactory.Options o = new BitmapFactory.Options();
    o.inJustDecodeBounds = true;
    BitmapFactory.decodeStream(context.getContentResolver().openInputStream(uri), null, o);

    int width_tmp = o.outWidth, height_tmp = o.outHeight;
    int scale = 1;

    while (true) {
        if (width_tmp / 2 < requiredSize || height_tmp / 2 < requiredSize)
            break;
        width_tmp /= 2;
        height_tmp /= 2;
        scale *= 2;
    }

    BitmapFactory.Options o2 = new BitmapFactory.Options();
    o2.inSampleSize = scale;
    return BitmapFactory.decodeStream(context.getContentResolver().openInputStream(uri), null, o2);
}

```

Figure 45: Decoding Image File

The code shown in Figure 45, is used to convert the image size using the inbuilt method Bitmap decodeUri.

```

, ...adding up user profile image...
public void setProfilePicture() {

    if(!session.getUserProfile().equals("")) {
        File imgFile = new File(session.getUserProfile());

        if(imgFile.exists()){

            Bitmap myBitmap = BitmapFactory.decodeFile(imgFile.getAbsolutePath());

            profileImage.setImageBitmap(myBitmap);

        }
    }else {
        Log.d(tag, "session Url()" + session.getUserProfileUrl());
        Glide.with(this)
            .load(session.getUserProfileUrl())
            .diskCacheStrategy(DiskCacheStrategy.ALL)
            .dontAnimate()
            .placeholder(getDrawable(R.drawable.bg))
            .skipMemoryCache(true)
            .into(profileImage);
    }
}

```

Figure 46: Glide Library to set the Profile Image

In Figure 46, the `setProfileImage()` method is called to set the profile image over the `ImageView`. If the image string is not empty, then the method will reset the `imgFile` variable with that input and check if the image already presents in the database or not; if not, it will decode the image size and stores it in the database. If the image already has been decoded, then it will use the glide library to get the image file from the database and show it on the screen.

4.6 Activity Tracker

In Figure 47, the activity is used to track the user steps using the motion sensors APIs. Every device support inbuilt sensors. The Android development platform provides a way to gain data from the motion sensors and later use this data in the Android applications. Mainly there are two types of hardware-based sensors offered by Android, accelerometer, and gyroscope [17]. The one

that monitors user activity is an accelerometer, as it is used to measure the acceleration force applied to the mobile devices on the three-dimensional physical axes [17]. This sensor includes the gravitational force on the device, to calculate just the acceleration of the user, gravitational force, i.e., 9.8 could subtract from the total acceleration.

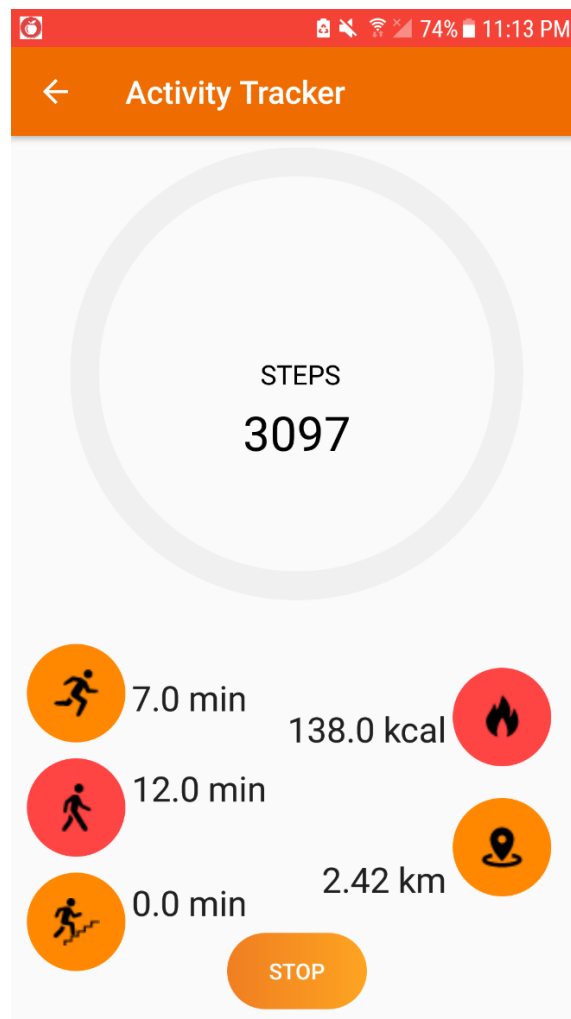


Figure 47: Activity Tracker Screen

```

public void onCreate() {
    super.onCreate();
    intent = new Intent(BROADCAST_ACTION);
    pref = getSharedPreferences(Common.STEP_OF_DAY, MODE_PRIVATE);
    session = Session.getSession(this);
    synchronized (this) {
        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        simpleStepDetector = new StepDetector();
        simpleStepDetector.registerListener(this);
    }
    List<HistoryModel> d = DBHelper.getInstance(this).getTodayTrackingData(Utils.todayDate());
    if (d != null && d.size() > 0) {
        steps = d.get(0).getSteps() + "";
        distance = d.get(0).getDistance() + "";
        calories = d.get(0).getCaloriesBurned() + "";
        runningCount = d.get(0).getRunning();
        stairCount = d.get(0).getStairs();
        walkingSteps = d.get(0).getWalking();
        preSteps = runningCount + stairCount + walkingSteps;
        long startTime = System.currentTimeMillis();
        long endTime = System.currentTimeMillis();
        MyStepModel stepModel = new MyStepModel();
        stepModel.setDistance(Float.parseFloat(distance));
        stepModel.setSteps(Long.parseLong(steps));
        stepModel.setCaloriesBurned(Double.parseDouble(calories));
        int totalWalkTime = walkingSteps + (int)(endTime - startTime);
        walkTime = totalWalkTime / 133;
        stepModel.setWalking(walkTime);
        int totalRunTime = runningCount + (int)(endTime - startTime);
    }
}

```

Figure 48: Activity Tracker onCreate Method

In Figure 48, the onCreate() method is setting up the sensor manager and step detector class object to set the sensors and registering the user with the sensors. In addition, it will also help the activity to broadcast the process in the background using BoardcastReciever[18] that is using the onRecieve() method to start the activity in the background process, when the user moves to different applications. The Activity Tracker will keep running in the background and save the collected data in the database. However, to make the activity keep running in the background and not to be killed by the process, we need to schedule a JobService from the receiver using the JobScheduler, so the system knows that the process is still running or active[18].


```

SensorManager s = (SensorManager) getSystemService(SENSOR_SERVICE);
Sensor countSensor = s.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
SharedPreferences.Editor editor = sharedPreferences.edit();
if (countSensor != null) {
    if (start_stop.getText().toString().equalsIgnoreCase(anotherString: "START")) {
        start_stop.setText("STOP");
        startService(new Intent(getBaseContext(), StepCountingService.class));
        editor.putBoolean("isServiceStopped", false);
        editor.apply();
        startActivity();
    } else {
        start_stop.setText("START");
        stopService(new Intent(getBaseContext(), StepCountingService.class));
        editor.putBoolean("isServiceStopped", true);
        editor.apply();
        stopActivity();
        AppJobManager.getJobManager().addJobInBackground(new SendTrackingDataToServer(),
        reset();
    }
}

private void startActivity() {
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(System.currentTimeMillis());
    calendar.set(Calendar.HOUR_OF_DAY, 7);

    AlarmManager alarmManager = (AlarmManager) this.getSystemService(this.ALARM_SERVICE);
    Intent intent = new Intent(packageContext: this, ActivityRecognizedService.class);
    PendingIntent pendingIntent = PendingIntent.getService(context: this, requestCode: 0, intent, PendingIntent.FLAG_UPDATE_
    assert alarmManager != null;
    alarmManager.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, pen
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates(mApiClient, 1000, pendingIntent);
}

```

Figure 49: Activity Tracker AlarmManager Code Snippet

As shown in Figure 49, in the `onClick()` method the activity tracker will start by calling the `startActivity()` method, which will automatically start the Activity Tracker at seven in the morning using the `AlarmManager` and will repeat the process for the next day using the `setInexactRepeating()` method[19].

```

public void onSensorChanged(SensorEvent event) {
    Sensor sensor2 = event.sensor;
    i++;
    if (sensor2.getType() == Sensor.TYPE_STEP_COUNTER) {
        /*
         * A step counter event contains the total number of steps since the listener
         * was first registered. We need to keep track of this initial value to calculate the
         * number of steps taken, as the first value a listener receives is undefined.
         */
        if (counterSteps < 1) {
            // initial value
            counterSteps = (int) event.values[0];
        }

        // Calculate steps taken based on first counter value received.
        numSteps = (int) event.values[0] - counterSteps;

        if (isStair) {
            runningCount++;
        } else if (isRunning) {
            stairCount++;
        }

        Log.d(tag, "steps : " + numSteps);
        String steps = "" + numSteps + TEXT_NUM_STEPS;
        steps_count.setText(steps);
        lblDistance.setText(getDistanceRun(numSteps) + "");
        lblCaloriesBurned.setText(Utils.caloriesBurnedForWalking(Float.parseFloat(session.getWeight()), numSteps)

```

Figure 50: onSensorChange Method Code Snippet

A `SensorEventListener` creates `onSensorChanged()` method in Figure 50; this method invoked whenever there is a change in the sensor values of the mobile sensors. This activity also calculates the calories burned, as well as running, walking and stairs steps using the total step counts. In the below code snippet in Figure 51, The data will be stored in the database once the broadcast service stops, which will then evoke the `saveTracking()` method. The `SaveTracking()` method also calculates the walking, running and, stairs times from the walking, running and stairs steps and saves all the values in the user database. This method will also update the display screen with the correct data continuation while the activity is running in the background.

```

private void broadcastSensorValue() { saveTracking(); }

private void saveTracking() {
    MyStepModel stepModel = new MyStepModel();
    if (!DBHelper.getInstance(this).getTodayTracking()) {
        distance = "0";
        steps = "0";
        calories = "0.0";
        walkingSteps = 0;
        runningCount = 0;
        stairCount = 0;
        numSteps = 0;
        stepModel.setDistance(Float.parseFloat(distance));
        stepModel.setSteps(Long.parseLong(steps));
        stepModel.setCaloriesBurned(Double.parseDouble(calories));
        long startTime = System.currentTimeMillis();
        long endTime = System.currentTimeMillis();
        int totalWalkTime = walkingSteps * (int)(endTime - startTime);
        walkTime = totalWalkTime / 133;
        stepModel.setWalking(walkTime);
        int totalRunTime = runningCount * (int)(endTime - startTime);
        runTime = totalRunTime / 178;
        stepModel.setRunning(runTime);
        int totalStairsTime = stairCount * (int)(endTime - startTime);
        stairsTime = totalStairsTime / 90;
        stepModel.setStairs(stairsTime);
        stepModel.setStatus(0);
        DBHelper.getInstance(this).insertTrackingNew(stepModel);
    }
}

```

Figure 51: Activity Tracker SaveTracking Method

4.6.1 Activity Tracker: Weekly and Daily Report

As shown in Figure 52, the Activity Tracker main screen the user can see the weekly and daily progress report for each activity, that is being calculated when the Activity Tracker starts by simply clicking the different activity icons. Icons such as total steps, running time, walking time, stairs time, calories burned, and distance covered by the user can be seen in the form of bar chart for the day or the whole week. These data are saved in the database and represented in the form of a bar chart for the weekly data report. For the daily data report, the data is presented in the form of the list which is represented in front of the user in a RecyclerView below the bar graph.

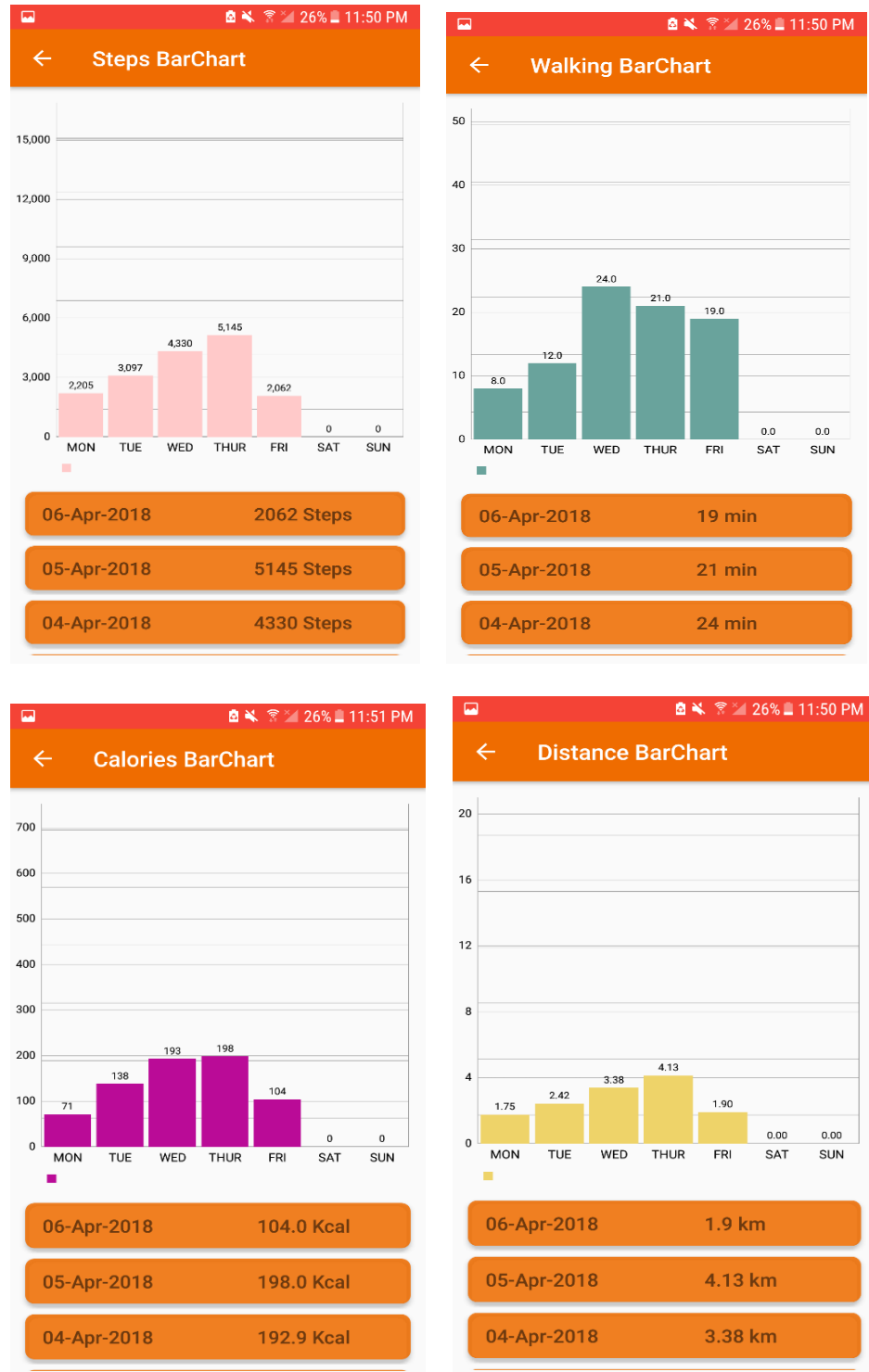


Figure 52: Activity Tracker Weekly and Daily Progress Report

The below code snippet in Figure 53, shows how the user can click icons to see the weekly and daily progress reports from the main Activity Tracker screen. The init() method, uses the explicit intent to make the ImageView of different activities able to open another activity when the user clicks them.

```
private void init() {
    isServiceStopped = sharedPreferences.getBoolean( key: "isServiceStopped", defValue: true);
    lblDistance = (TextView) findViewById(R.id.lblDistance);
    lblActivity = (TextView) findViewById(R.id.lblActivity);
    lblCaloriesBurned = (TextView) findViewById(R.id.lblCaloriesBurned);
    lblWalkingCount = (TextView) findViewById(R.id.textWalking);
    lblRunningCount = (TextView) findViewById(R.id.textRunning);
    lblStairCount = (TextView) findViewById(R.id.textStairs);
    steps_count = (TextView) findViewById(R.id.steps_count);
    start_stop = (Button) findViewById(R.id.start_stop);

    mfitChart = (FitChart) findViewById(R.id.fitChart);
    mfitChart.setOnClickListener((v) -> {
        Intent stepsIntent = new Intent( packageContext: MySteps.this, StepsBarGraph.class);
        startActivity(stepsIntent);
    });

    ImgRunning = (ImageView) findViewById(R.id.imageRunning);
    ImgRunning.setOnClickListener((v) -> {
        Intent runningIntent = new Intent( packageContext: MySteps.this, RunningBarGraph.class);
        startActivity(runningIntent);
    });

    ImgWalking = (ImageView) findViewById(R.id.imageWalking);
    ImgWalking.setOnClickListener((v) -> {
        Intent walkingIntent = new Intent( packageContext: MySteps.this, WalkingBarGraph.class);
        startActivity(walkingIntent);
    });
}
```

Figure 53: Activity Tracker init() Method

For example, when the user clicks the total step's icon from the Activity Tracker screen, that action will lead the user into a new activity page where they can see the daily and weekly report on their progress in the form of bar chart using the MPAndroidChart library. Similarly, the user can see all the weekly and daily report for each activity by simply clicking their respective icons such as walking time, running time, stairs time,

calories burned and distance covered to see the values in the form a bar graph. In Figure 54, the setData() method is used to create the datasets in the MPAndroidChart library to form a bar chart. So, the bar chart can be passed by creating the array list and adding the values back to the list. The values stored in the list are stored in the data set and passed to the bar chart class. To display the required value in the form of a bar, I have created one data set using the single array list to display the required value. In the below case, total steps are used to show in the bar char for a weekly report.

```
private void setData() {

    ArrayList<BarEntry> stepsArrayList = new ArrayList<>();

    ArrayList<HistoryModel> mlist = new ArrayList<>(DBHelper.getInstance(StepsBarGraph.this).weeklyData());

    for (int i = 0; i < 7; i++) {
        if (i < mlist.size()) {
            // Log.d(tag, "save Date : " + mlist.get(i).get("saveDate") + "Value of day : " + getDay(mlist.get(i).getSaveDate()));
            Log.d(tag, "totalSteps : " + mlist.get(i).getSteps());
            Log.d(tag, msg: "steps : " + mlist.get(i).getSteps());
            Log.d(tag, msg: "Day : " + mlist.get(i).getSaveDate());
        }
    }

    for (int j = 0; j < 7; j++) {
        Log.d(tag, msg: "Value of J before : " + j);
        for (int k = 0; k < mlist.size(); k++) {
            if (j == getDay(mlist.get(k).getSaveDate())) {

                Log.d(tag, msg: "Value of K : " + k);
                Log.d(tag, msg: "SIZE OF mlist(k) : " + mlist.size());
                Log.d(tag, msg: "DATE : " + getDay(mlist.get(k).getSaveDate()));
                Log.d(tag, msg: "Value of steps : " + mlist.get(k).getSteps());

                stepsArrayList.add(new BarEntry(j, mlist.get(k).getSteps()));

                totalLeftXis = totalLeftXis + mlist.get(k).getSteps();
            }
        }
    }
}
```

Figure 54: Code Snippet for Step Count BarChart

In Figure 55, the code shows the StepBarGraph class's adaptor which is used to attach the total steps count values from the database in a RecyclerView, and to represent the daily progress report for the total steps counts in the form of a list. This list data is placed

below the bar graph which shows the daily progress report of the user by letting them know the number of steps taken on that day.

```
public class StepsHistoryAdapter extends RecyclerView.Adapter<StepsBarGraph.StepsHistoryAdapter.MyViewHolder> {

    private Context context;

    public StepsHistoryAdapter(Context context) {
        super();
        this.context = context;
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {
        private TextView tvTimestamp, tvSteps;

        public MyViewHolder(View view) {
            super(view);
            tvTimestamp = (TextView) view.findViewById(R.id.tvTimestamp);
            tvSteps = (TextView) view.findViewById(R.id.tvSteps);
        }
    }

    /* Inflating the view to show the Activity Tracker history data. */
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.list_steps_bar_chart, parent, attachToRoot: false);
    }
}
```

Figure 55: Code Snippet to Add the Daily Report

4.7 Meal Planner

In Figure 56, the activity is used to create the meal plans for the user using the Food Ontology APIs that is Nutrinitonix API in case of Personal Dietitian Application. Through this activity, the user can add the intake of food items during the three-meal interval of the day an easy and convenient way. The main screen of the Meal Planner activity shows different functionality allowing the user to know more about what they are consuming and other related information. The first part is where the user can look into the number of his/her goal calories that is calculated according to the user's initial information feed during the registration process. Next, to that value, the user can see the total consumed calories for that day and the calories burned using the

Activity Tracker as well as the remaining calories that they can be used to chuck some more good things.

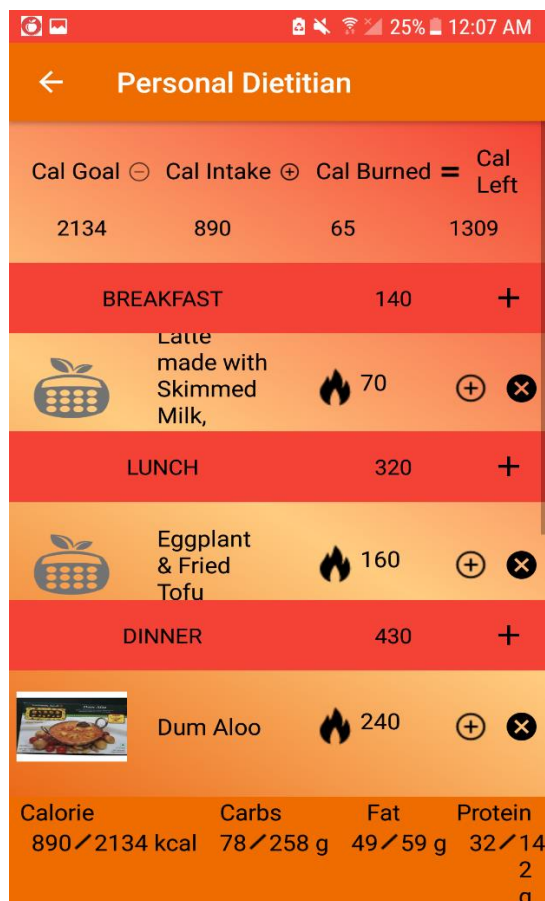


Figure 56: Meal Planner Main Screen

The way that the Meal planner activity calculates the total calories needed by the person in a day is calculated by calculating the BMR of the user. Hence, to calculate the total calories required by a person, the application will use the general information of the user provided during the registration process, such as weight, height, age, gender and activity level of the user to get the BMR and other related calculations as shown in Figure 57.


```

public static double calculateCaloriesRequired(String gender, String activityLevel, double height, double weight, int
double calculatedBMR;

if (gender.equals("male")) {
    calculatedBMR = 66.5 + (13.7*weight) + (5*height) - (6.76*age);
    if (activityLevel.equals("Low")) {
        caloriesPerDay = calculatedBMR * 1.375;
    } else if (activityLevel.equals("Moderate")) {
        caloriesPerDay = calculatedBMR * 1.55;
    } else {
        caloriesPerDay = calculatedBMR * 1.725;
    }
} else {
    calculatedBMR = 655 + (9.56*weight) + (1.8*height) - (4.68*age);
    if (activityLevel.equals("Low")) {
        caloriesPerDay = calculatedBMR * 1.375;
    } else if (activityLevel.equals("Moderate")) {
        caloriesPerDay = calculatedBMR * 1.55;
    } else {
        caloriesPerDay = calculatedBMR * 1.725;
    }
}
return caloriesPerDay;
}

```

Figure 57: Calculating Calories Required Per Day

Leftover calories or remaining calories can be calculated by using the total calories required, that is goal calories, food calories consumed by the user and the calories burned by the user using the activity tracker as shown in Figure 58.

```

public static int getLeftCalorie(Context context, String name, String defaultValue, float weight, int steps) {
    int left;
    left = (int) caloriesPerDay - (Integer.parseInt(getCalorieShared(context, name, defaultValue))) + ((int) calories
    Log.d(tag, "left calories, goal, food, exercise: "+left + goalCalorie + foodCalorie + exerciseCalorie);
    return left;
}

```

Figure 58: Calculation of Left Calories

```

private void init() {
    session = Session.getSession(context);

    myFoods_B = new ArrayList<>();
    myFoods_D = new ArrayList<>();
    myFoods_L = new ArrayList<>();

    setBreakfastView();
    setLunchView();
    setDinnerView();

    ArrayList<HashMap<String, String>> mlist = new ArrayList<>(DBHelper.getInstance(getActivity()).weeklyNutritionDataInfo);

    String prev = Utils.getCalorieShared(context, name: "Calorie", defaultValue: "0");
    Log.d(tag, msg: "nutritionModel : "+nutritionModel.getId());
    Calories.setText(prev);

    // Showing the total calories intake by the user in day

    for (int j = 0; j < 7; j++) {
        Log.d(tag, msg: "Value of J before : " + j);
        for (int k = 0; k < mlist.size(); k++) {
            if ((j == getDay(mlist.get(k).get("date")))) {
                totalCalForADay = Integer.valueOf(mlist.get(k).get("totalCalories"));
                Calories.setText(mlist.get(k).get("totalCalories"));
            }
        }
    }

    // Showing the total carbs, protein and fat intake by the user in day
    for (int j = 0; j < 7; j++) {
        Log.d(tag, msg: "Value of J before : " + j);
        for (int k = 0; k < mlist.size(); k++) {
            if ((j == getDay(mlist.get(k).get("date")))) {
                intakeCal.setText(mlist.get(k).get("totalCalories"));
                intakeCarbs.setText(mlist.get(k).get("totalCarbs"));
                intakeProtein.setText((mlist.get(k).get("totalProtein")));
                intakeFat.setText((mlist.get(k).get("totalFat")));
            }
        }
    }

    int goalOfCal = (int) Utils.calculateCaloriesRequired(String.valueOf(session.getGender()), String.valueOf(session.getActivity()),
        Float.parseFloat(session.getHeight()), Float.parseFloat(session.getWeight()), Integer.parseInt(session.getAge()));

    goal.setText(String.valueOf(goalOfCal));

    totalCalorie.setText(String.valueOf((int) Utils.calculateCaloriesRequired(String.valueOf(session.getGender()), String.valueOf(session.getActivity()),
        Float.parseFloat(session.getHeight()), Float.parseFloat(session.getWeight()), Integer.parseInt(session.getAge())));
    intakeCal.setText(prev);

    protein.setText(String.valueOf(Math.round(Utils.getProtein(Float.parseFloat(session.getWeight())))) + " g");
    int totalProteinVal = (int) ((Float.parseFloat(mlist.get(0).get("protein"))) + (Float.parseFloat(mlist.get(1).get("protein"))));
    intakeProtein.setText(String.valueOf(totalProteinVal));

    fat.setText(String.valueOf(Math.round(Utils.getFat())) + " g");
}

```

Figure 59: init() Method of Meal Planner Activity

The next part of the Meal Planner screen includes the three-meal interval. Namely, the Breakfast, Lunch and Dinner section allows the user to see the added food items consumed during that period. This section also shows the total calories consumed for the individual section, for

example, in the Breakfast section, the user can see the total calories consumed during breakfast, the name of the food item as well as the food image provided by the API. The value of consumed calories for this section is retrieved from the user database and is visible once the user adds the food item in the database using the circular add button in the given sections as shown in Figure 60.

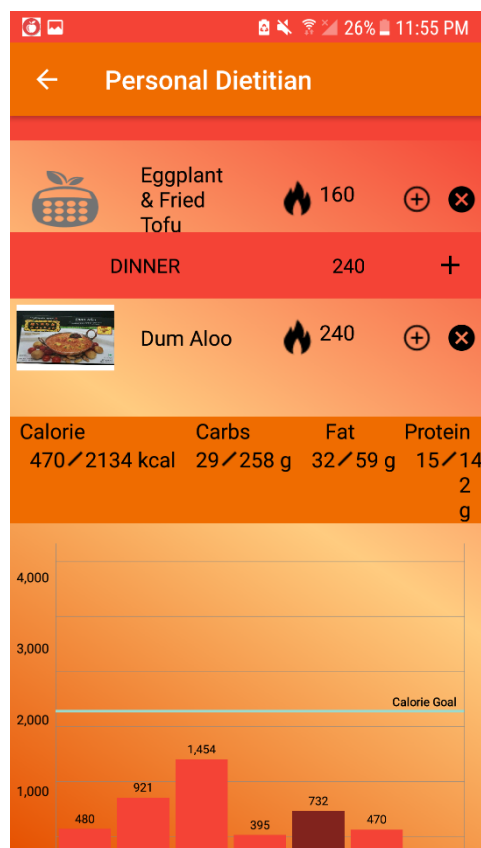


Figure 60: Meal Planner Screen

This will also store the food nutrition values in the local database and shown in the below section of the Meal Planner screen. The nutrition values are compared with the required nutrition values needed by the person. The next part of the activity shows the total weekly calories consumed by

the user in a bar graph formate, as well as the daily progress report, can also be provided to the user below the bar graph.

The user can investigate the API database and add food items by clicking the Add button near the breakfast, lunch or dinner section of the meal planner activity as shown in Figure 60. The application will take the user to a new screen where the user can see the list of food items. In that screen, there are four fragment classes with the tabs layout that the user can use namely, the food, recipe, restaurant and barcode scanner tabs as shown in Figure 61.

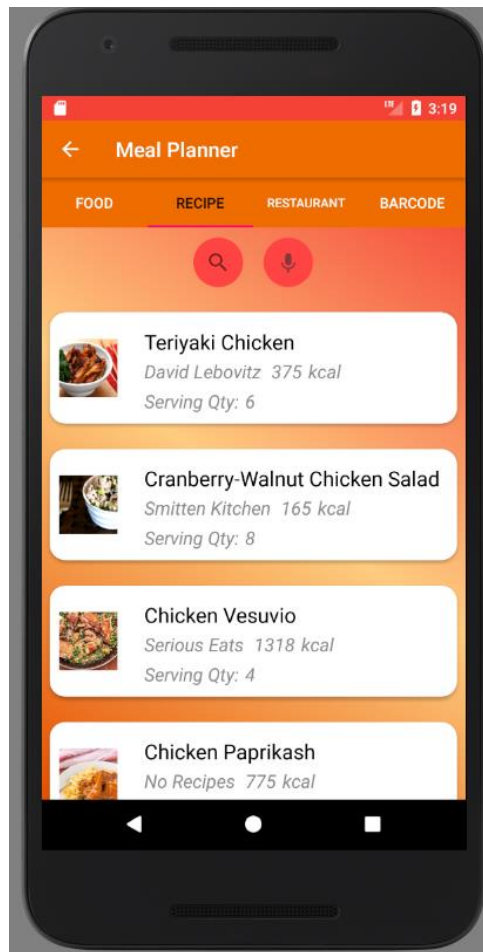


Figure 61: Add Food Screen

Through this screen, the user can search the food items using voice search (Figure 62) as well as text search (Figure 64). The application is using the “inbuilt intents” to make the voice and text search work. The code in Figure 63, will show the codes required for the voice search functionality to work for this application. When the user clicks on the voice button, it will invoke the `onClick()` method through the `setOnClickListener()` method. This method will call another method `startSpeechToText()` method, which will help to start the “voice intent,” most commonly a Google voice search intent. Then the voice intent input will pass with the speech recognition code in the `startActivityResult()` method to make use of the voice input according to the application requirement. To make the voice input into text, the `activityForResult()` method is used to receive voice input in the application. Hence, to find the food item the voice turn text input will pass in the `searchRecipe()` method to search the food ontology API database.

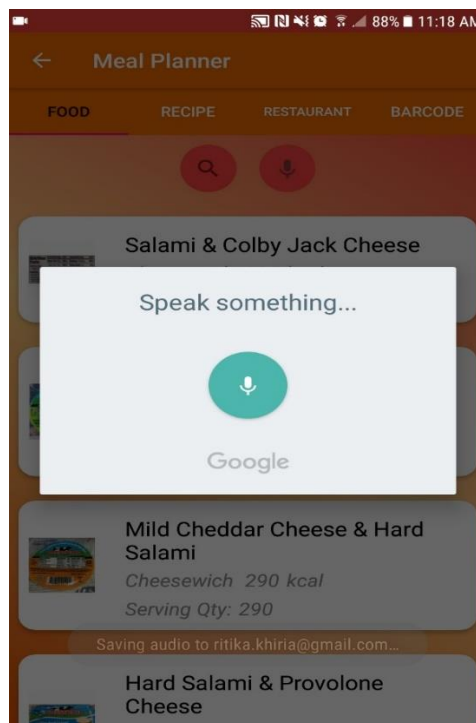


Figure 62: Voice Search Screen

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    view = inflater.inflate(R.layout.fragment_tab_restaurant, container, false);
    restaurantSearch = (SearchView) view.findViewById(R.id.restaurant_search);
    restaurantVoiceButton = (ImageButton) view.findViewById(R.id.voice_restaurant);
    restaurantVoiceButton.setOnClickListener((v) -> { startSpeechToText(); });
    initAdapter();
    searchRecipe("pizza");

    restaurantSearch.setOnQueryTextFocusChangeListener((v, hasFocus) -> {
        // TODO Auto-generated method stub
        //Toast.makeText(activity, String.valueOf(hasFocus), Toast.LENGTH_SHORT).show();
    });

    restaurantSearch.setOnQueryTextListener(new SearchView.OnQueryTextListener()
    {
        @Override
        public boolean onQueryTextSubmit(String query) {

private void startSpeechToText() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
        "Speak something...");
    try {
        startActivityForResult(intent, SPEECH_RECOGNITION_CODE);
    } catch (ActivityNotFoundException a) {
        Toast.makeText(getApplicationContext(),
            "Sorry! Speech recognition is not supported in this device.",
            Toast.LENGTH_SHORT).show();
    }
}

/**
 * Callback for speech recognition activity
 * */
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        case SPEECH_RECOGNITION_CODE: {
            if (resultCode == RESULT_OK && null != data) {
                ArrayList<String> result = data
                    .getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                String text = result.get(0);
                restaurantFoodList.clear();
                searchRecipe(text);
            }
            break;
        }
    }
}
}

```

Figure 63: Voice Search Code Snippet

In the case of text search, the user clicks on the text search button to open the keyboard to enter the text in the search field as shown in Figure 64. The searchView set with the `setQueryTextListener()` method which evokes the `onQueryTextSubmit()` method to run the query based on text entered by the user through searchView as shown in Figure 65. To receive the food item from the API, the text string is passed to the `searchRecipe()` method which will make the calls to the Food ontology APIs.

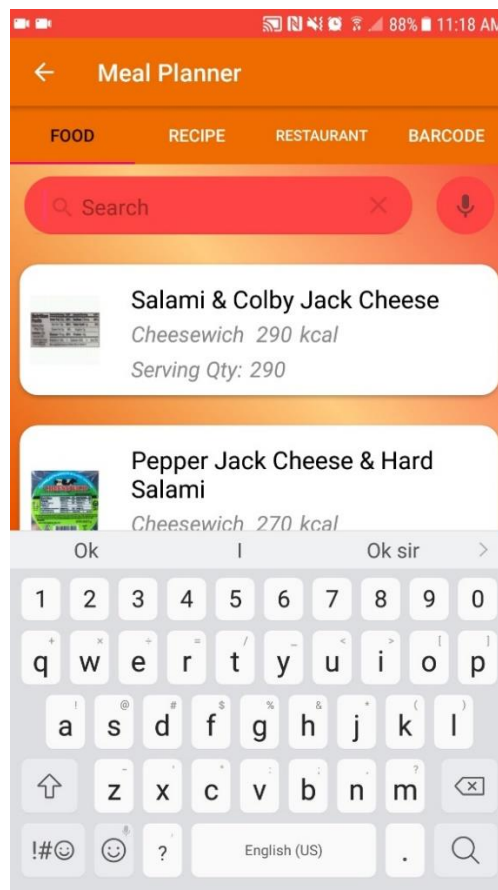


Figure 64: Text Search Screen

```

restaurantSearch.setOnQueryTextFocusChangeListener((v, hasFocus) -> {
    // TODO Auto-generated method stub
    //Toast.makeText(activity, String.valueOf(hasFocus),Toast.LENGTH_SHORT).show();
});

restaurantSearch.setOnQueryTextListener(new SearchView.OnQueryTextListener()
{
    @Override
    public boolean onQueryTextSubmit(String query) {

        // Reset SearchView
        restaurantFoodList.clear();
        searchRecipe(query);
        restaurantSearch.clearFocus();
        restaurantSearch.setQuery("", false);
        restaurantSearch.setIconified(true);
        restaurantSearch.collapseActionView();
        return true;
    }

    @Override
    public boolean onQueryTextChange(String query) {

```

Figure 65: Text Search Code Snippet

The result for the food items is coming from the Nutrinitionix APIs calls. In figure 66, the code is used to call the API objects and store it in the model class.

```

private void searchRecipe(String recipeName) {
    SearchModel searchModel = new SearchModel();
    searchModel.setRecipeName(recipeName);
    webCalls = new WebCalls(getActivity());
    webCalls.showProgress(false);
    webCalls.getRestaurantList(searchModel);
    webCalls.setWebCallListner(new OnWebCall() {

        @Override
        public void OnWebCallSuccess(String userFullData) {
            //Log.d(TAG,"Recipe search : "+userFullData);
            try {
                JSONObject jsonObject = new JSONObject(userFullData);
                JSONArray jsonArray = jsonObject.getJSONArray("branded");
                for (int i = 0; i < jsonArray.length(); i++) {
                    RestaurantFood restaurantModel = new RestaurantFood();
                    JSONObject jsonObjectItems = jsonArray.getJSONObject(i);
                    restaurantModel.setRestaurantFoodName(jsonObjectItems.getString("food_name"));
                    restaurantModel.setServingQuantity(jsonObjectItems.getString("serving_qty"));
                    restaurantModel.setRestaurantFoodCalories(jsonObjectItems.getString("nf_calories"));
                    restaurantModel.setRestaurantBrandName(jsonObjectItems.getString("brand_name"));
                    restaurantModel.setRestaurantFoodNixItemId(jsonObjectItems.getString("nix_item_id"));
                    Log.d(TAG, "Food Name : "+jsonArray.length());
                    JSONObject jsonObjectPhoto = jsonObjectItems.getJSONObject("photo");
                    Log.d(TAG, "Food Name : "+jsonObjectItems.getString("food_name")+"\n Food Image Url : "+jsonObj
                    restaurantModel.setRestaurantFoodImage(jsonObjectPhoto.getString("thumb"));
                    restaurantFoodList.add(restaurantModel);
                }
            } catch (Exception e) {

```

Figure 66: APIs Calls to get the Food Data

The barcode fragment is used to scan the packed food products from the market to know the detail of its nutrition contains. For this project, the Zxing (Zebra Crossing) library [13] is used to scan the product barcode. This library supports various kinds of barcode formats including UPC-A, UPC-E, EAN-8, EAN-13, Code 39, and more [13]. To use this library in the application, we need to define the zxing jar files in the application build.gradle file. When the user clicks the barcode button (Figure 67 and Figure 68) from the barcode fragment, a camera permission dialog box will appear. By accepting the permission, the user can now scan the barcode of the food item. The Nutritionix API can access the nutrition values of the particular product by providing the UPC (Universal Product Code) value in the query form the API endpoints. After scanning the product, the user can see the result on a separate screen (Figure 69) which will show the food image, food name, calories, serving size, weight, fat, carbohydrate, protein and sugar values.



Figure 67: Barcode Scanner Screen

```

public class BarcodeFragment extends Fragment {

    public static final String TAG = BarcodeFragment.class.getSimpleName();
    View view;
    Button barcodeScan;

}
@Override
}
public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

    view = inflater.inflate(R.layout.fragment_barcode, container, false);
    barcodeScan = (Button) view.findViewById(R.id.btn_barcode);
    barcodeScan.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            barcodeScanner();
        }
    });
    return view;
}
}
public void barcodeScanner() {
    new IntentIntegrator(getActivity()).initiateScan(); // 'this' is the current Activity
}
}

```

Figure 68: Barcode Fragment Code Snippet

The user can add any food items from the list by clicking each. This action takes the user to a new screen where they can see the complete information on the selected food item as shown in Figure 69. And from that screen, he/she can add the food to its meal plan.

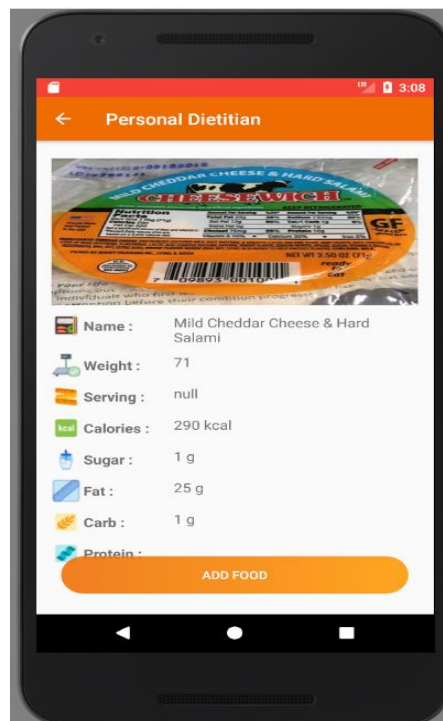


Figure 69: Detail Food Screen

4.7.1 Meal Planner: Weekly and Daily Report

As shown in Figure 70, the Meal Planner's main screen can be used to see the progress reports for the consumed calories by the user in the form of bar chart for the day or the whole week. The data is saved in the database and represented in the form of a bar chart for the weekly data report. For the daily chronological data report, it is presented in the form of the list which is represented in front of the user in a RecyclerView below the bar graph. The Personal Dietician application provides the progress report, not only for the total consumed calories for that day or whole week but also as separate progress reports for each meal interval (breakfast, lunch, and dinner).

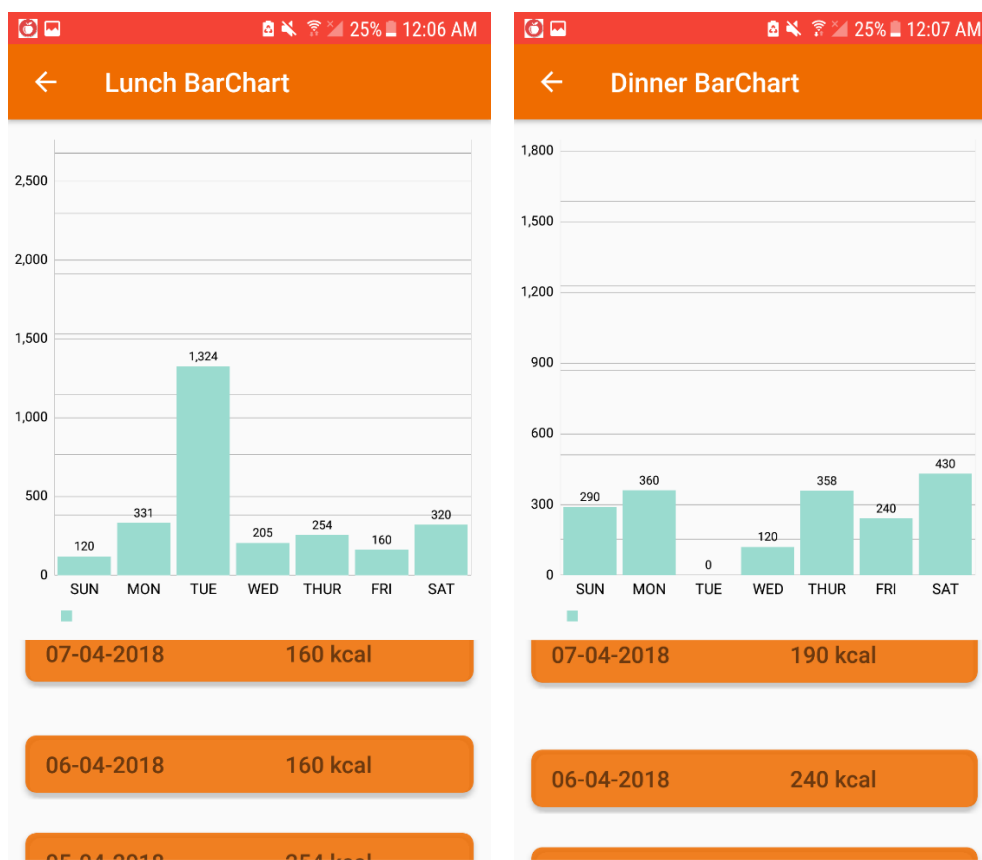


Figure 70: Meal Planner Weekly and Daily Report

In Figure 71, the code show how the explicit intent is attached to the TextView to trigger the new activity. This occurs when the user clicks the breakfast, lunch or dinner text view from the main Meal Planner screen using `setOnClickListener()` method. This action will open a new activity page where the user can see the weekly progress report in a bar chart format and the daily report in a list format represented in the RecyclerView this allows the user to scroll more data from the local database.

```
// intent to graph
graphBreakfast = (TextView) v3.findViewById(R.id.break_fast);
graphBreakfast.setOnClickListener((v) → {
    foodTypeId = 1;
    comm.respond(id: 1);
    Intent breakfastIntent = new Intent(getActivity(), BreakFastBarGarph.class);
    startActivity(breakfastIntent);
});

graphLunch = (TextView) v3.findViewById(R.id.lunch);
graphLunch.setOnClickListener((v) → {
    foodTypeId = 2;
    comm.respond(id: 2);
    Intent lunchIntent = new Intent(getActivity(), LunchBarGraph.class);
    startActivity(lunchIntent);
});

graphDinner = (TextView) v3.findViewById(R.id.dinner);
graphDinner.setOnClickListener((v) → {
    foodTypeId = 3;
    comm.respond(id: 3);
    Intent dinnerIntent = new Intent(getActivity(), DinnerBarGraph.class);
    startActivity(dinnerIntent);
});
```

Figure 71: Code Snippet to Show the Explicit Intent

In Figure 72, the `setData()` method is used to create the datasets in the MPAndroidChart library to form a bar chart. The bar chart can be passed by creating the array list and adding the values back to the list. The values stored in the list are used to store in the data set and passed to the bar chart class. To display the required value in the form of a bar chart, I have

created one data set using the single array list to display the required value; in Figure 72, dinner calories are used to produce the bar char for a weekly report.

```
private void setData() {

    ArrayList<BarEntry> dinnerArrayList = new ArrayList<>();

    ArrayList<HashMap<String,String>> mlist = new ArrayList<>(DBHelper.getInstance(this).nutritionWeeklyDataDinner());
    ArrayList<NutritionModel> mlist = new ArrayList<>(DBHelper.getInstance(this).nutritionWeeklyDataDinner());

    ArrayList<Foods> mlist1 = new ArrayList<Foods>();

    for (int i = 0; i < 7; i++) {
        if (i < mlist.size()) {
            getDay(mlist.get(i).get("date"));
            Log.d(tag, msg: "Timestamp : " + mlist.get(i).get("timestamp"));
            Log.d(tag, msg: "Day : " + mlist.get(i).get("date"));
            getDay(mlist.get(i).getDate());
            Log.d(tag, "Timestamp : " + mlist.get(i).getTimestamp());
            Log.d(tag, "Day : " + mlist.get(i).getDate());
        }
    }

    for (int j = 0; j < 7; j++) {
        Log.d(tag, msg: "Value of J before : " + j);
        for (int k = 0; k < mlist.size(); k++) {

            totalLeftXis = (int) (totalLeftXis + (Float.parseFloat(mlist.get(k).get("totalCalories"))));

            grapAdded = true;
            break;
        } else {
            grapAdded = false;
        }
    }

    if (!grapAdded) {
        // Log.d(tag, "Value of J After : " + j);
        dinnerArrayList.add(new BarEntry(j, 0, getDayName(j)));
    }

    Log.d(tag, msg: "left xis : " + totalLeftXis);
    //set Left Axis

}

leftAxis.setAxisMaximum(totalLeftXis + 50);

BarDataSet setDinner;
```

Figure 72: Code Snippet to add Bar Graph Report

In Figure 73, the code shows the DinnerBarGraph class's adaptor which is used to attach the total calories consumed in the dinner from the database in a list format represented in a RecyclerView. This list data is placed below the bar graph which shows the daily progress report of the user by letting them know the value of calories taken by them on that day.

```
public class DinnerHistoryAdapter extends RecyclerView.Adapter<DinnerBarGraph.DinnerHistoryAdapter.MyViewHolder> {

    private Context context;

    public DinnerHistoryAdapter(Context context) {
        super();
        this.context = context;
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {
        private TextView tvTimestamp, tvCalories;
        private LinearLayout mLinearLayout;

        public MyViewHolder(View view) {
            super(view);
            mLinearLayout = (LinearLayout) view.findViewById(R.id.stepsDate);
            tvTimestamp = (TextView) view.findViewById(R.id.tvTimestamp);
            tvCalories = (TextView) view.findViewById(R.id.tvSteps);
        }
    }

    /* Inflating the view to show the Activity Tracker history data. */
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.list_meal_bar_chart, parent, attachToRoot: false);

        return new MyViewHolder(itemView);
    }

    /* Populating the Activity Tracker data on the view of RecyclerView */
    @Override
    public void onBindViewHolder(final MyViewHolder holder, final int position) {
        NutritionModel nutritionModel = nutritionModels.get(position);

        if (nutritionModel.getTypeOfFood().equals(3) && (nutritionModel.getTypeOfFood() != 1) && (nutritionModel.getTypeOfFood() != 2)) {
            Log.d(tag, "Name of food : " + mFoods.getName());
            holder.mLinearLayout.setVisibility(View.VISIBLE);
            holder.tvTimestamp.setText(String.valueOf(nutritionModel.getDate()));
            holder.tvCalories.setText(nutritionModel.getCalories() + " kcal");
        } else {
            holder.mLinearLayout.setVisibility(View.GONE);
        }
    }

    @Override
    public int getItemCount() { return nutritionModels.size(); }
}
```

Figure 73: Code Snippet to Add Daily Report

4.8 User Log

In Personal Dietician Application, the user can see their log by navigating themselves from the navigation drawer and clicking the “user log” option from it to see the log data from the local database. This activity will provide both the log data from the Activity Tracker and Meal Planner activities. This page contains two tabs to choose from one is using to see the stored data from the Activity Tracker and another one is showing the stored data from the Meal Planner activity. Figure 74, shows the Activity Tracker Log data from the local database such as total steps, distance, calories burned, walking, running, stairs time and date.

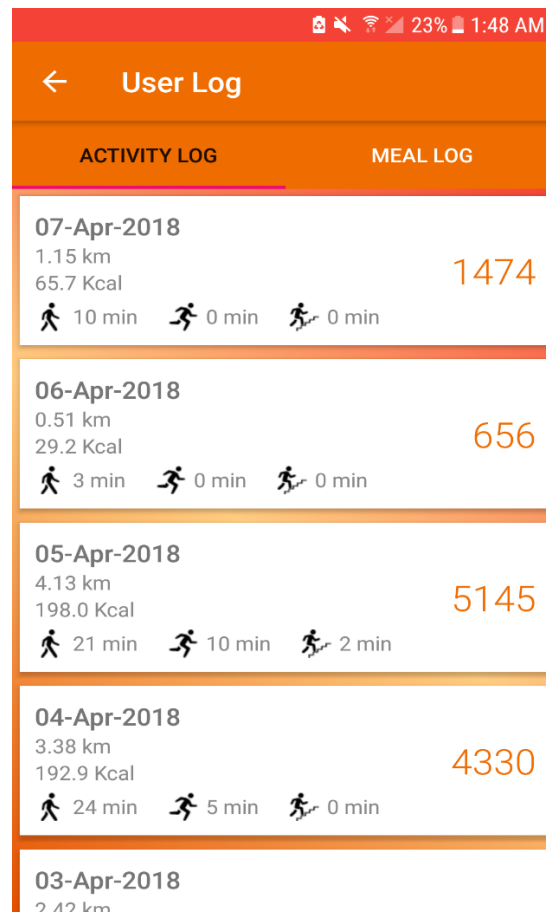


Figure 74: Activity Log Screen

Similarly, for the Meal Log activity, the screen provides a list of the food items values that are stored by the user while using the Meal Planner activity. The nutrition values of the food items are pulled from the local database and displayed in a RecyclerView. The screen shows the date when the food item is added to the database, including its nutrition values such as calories, carbohydrate, fat, and protein. It also shows the values and whether the food item was added during breakfast, lunch or dinner as shown in Figure 75.

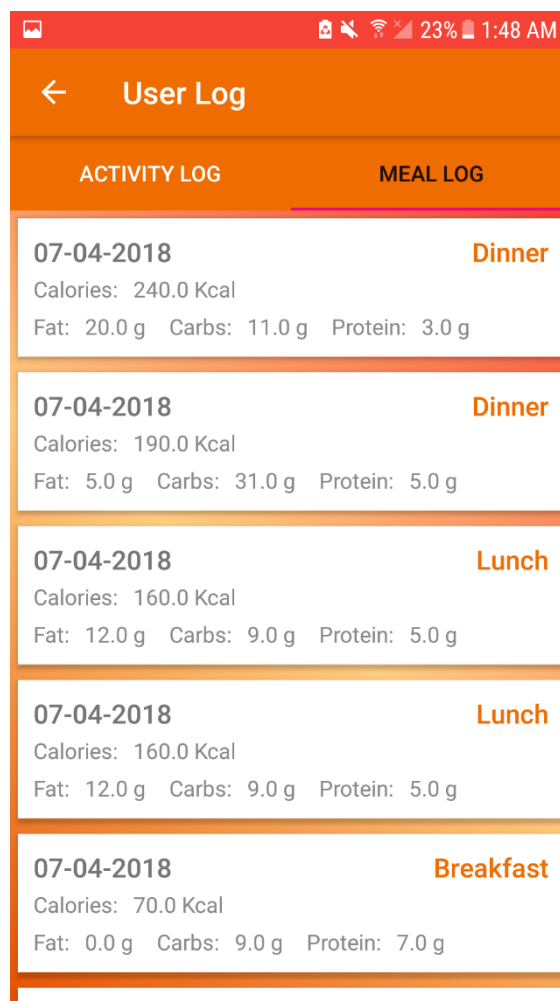


Figure 75: Meal Planner Log

4.9 Track Me: Activity and Meal Report

In Figure 76, the activity is used to see the progress report in the form of line chart and bar chart for the Activity Tracker as well as for the Meal Planner. The reports are generated on the weekly and monthly basis. For the Activity Tracker, the report produced in the form of a line chart, and for the Meal Planner, the report represented in the form of the pie chart. The MPAndroidChart library [10] is used to generate these charts for this application.



Figure 76: Track Me Activity Screen

MPAndroidChart library [10] provides distinct types of charts to use in the application and also offers animation, zoom, pinch and more as defined inside the library.

When the user clicks, the “track me” button from the “navigation drawer” an explicit intent is called to start the new activity. In this case, it starts the “TrackMe” activity class. In this class, the user will get two options to see the progress reports form the Activity Tracker and Meal Planner. One button is used for the Activity Report whereas the other button is used to generate the Meal Report. For example, if the user clicks the activity report button, then a new activity will open, which will project the line chart for the Activity Tracker.

The main screen of this activity contains two fragments which are used to display the weekly and monthly report of the user progress as shown in Figure 77.

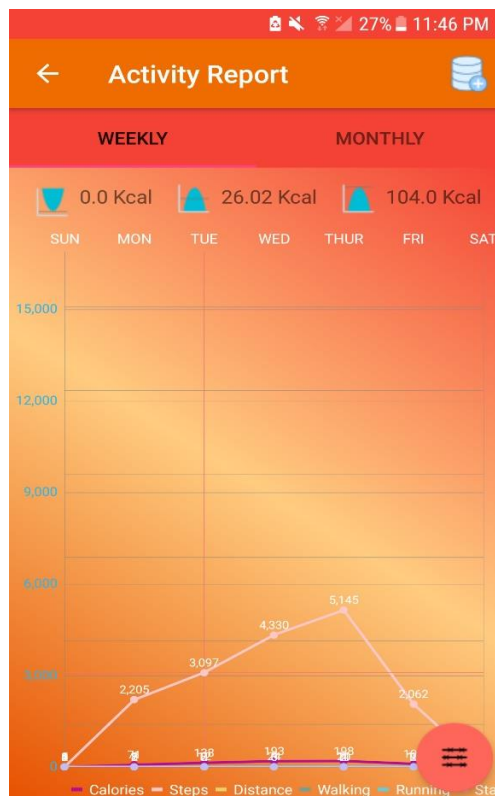


Figure 77: Activity Report Screen

This activity will also show the average, minimum and maximum calories burned by the user on a weekly and monthly basis. The limit line for the goal calories burned can be set by clicking the floating button on the screen as shown in Figure 78. This button will call the `showDialog()` method to open the dialog box with Discrete seek bar [19] with the minimum value as zero and maximum value as the calories required by the person, which would have been calculated earlier in `Utils` class.

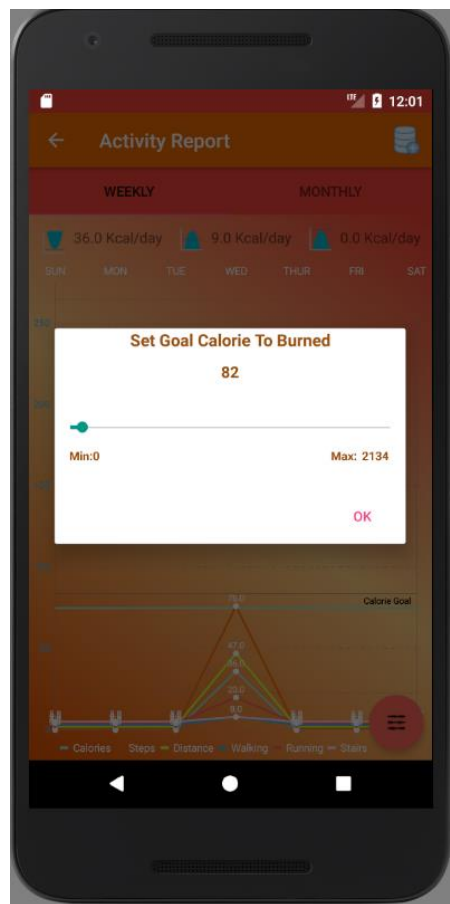


Figure 78: Adding Calorie Burned Goal Limit

The code shown in Figure 79, shows that at the end of the day, all the counters are reset, and the data is stored in the local database. Once the data is stored in the database, it can be quickly retrieved and display in the weekly and monthly reports in the `Track Me` activity class. This class

will take the data set for the value of x-axis and y-axis and represent the data in the line chart format. Values for the line chart can be passed by creating the array list and adding the values back to the list. The values stored in the list are used to store in the data set and passed to the line chart class.

A total of six data sets are created using the array list to display the steps count, total calories, calories burned, walking, running and stairs time for the day.

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    view = inflater.inflate(R.layout.fragment_line_chart_weekly, container, false);
    initChart();
    return view;
}

private void initChart() {
    mChart = (LineChart) view.findViewById(R.id.chart1);
    mChart.setOnChartValueSelectedListener(this);
    // no description text
    mChart.getDescription().setEnabled(false);

    // enable touch gestures
    mChart.setTouchEnabled(true);

    mChart.setDragDecelerationFrictionCoef(0.9f);

    // enable scaling and dragging
    mChart.setDragEnabled(true);
    mChart.setScaleEnabled(true);
    mChart.setDrawGridBackground(false);
    mChart.setHighlightPerDragEnabled(true);

    // if disabled, scaling can be done on x- and y-axis separately
    mChart.setPinchZoom(true);
}

```

```

private void setData(int count, float range) {
    ArrayList<Entry> caloriesArrayList = new ArrayList<>();
    ArrayList<Entry> stepsArrayList = new ArrayList<>();
    ArrayList<Entry> distanceArrayList = new ArrayList<>();
    ArrayList<Entry> walkingArrayList = new ArrayList<>();
    ArrayList<Entry> runningArrayList = new ArrayList<>();
    ArrayList<Entry> stairsArrayList = new ArrayList<>();
    ArrayList<HashMap<String, String>> mList = new ArrayList<>(DBHelper.getInstance(getActivity()).weeklyData());
    for (int i = 0; i < count; i++) {
        if(i<mList.size()) {
            Log.d(tag, "save Date : " + mList.get(i).get("saveDate") + "Value of day : " + getDay(mList.get(i).get("saveDate")));
            Log.d(tag, "totalSteps : " + mList.get(i).get("totalSteps"));
            Log.d(tag, "data : " + mList.get(i).get("data"));
            Log.d(tag, "calories : " + mList.get(i).get("calories"));
            Log.d(tag, "distance : " + mList.get(i).get("distance"));
            Log.d(tag, "steps : " + mList.get(i).get("steps"));
            Log.d(tag, "walking : " + mList.get(i).get("walking"));
            Log.d(tag, "running : " + mList.get(i).get("running"));
            Log.d(tag, "stairs : " + mList.get(i).get("totalStairs"));
        }
    }
}

```

```

LineDataSet setCalories, setSteps, setDistance, setWalking, setRunning, setStairs;

```

```

// create a dataset and give it a type
setCalories = new LineDataSet(caloriesArrayList, "Calories");
setCalories.setAxisDependency(YAxis.AxisDependency.LEFT);
setCalories.setColor(getResources().getColor(R.color.colorCalories));
setCalories.setCircleColor(Color.WHITE);
setCalories.setLineWidth(2f);
setCalories.setCircleRadius(3f);
setCalories.setFillAlpha(65);
setCalories.setFillColor(ColorTemplate.getHoloBlue());
setCalories.setHighLightColor(Color.rgb(244, 117, 117));
setCalories.setDrawCircleHole(false);

setSteps = new LineDataSet(stepsArrayList, "Steps");
setSteps.setAxisDependency(YAxis.AxisDependency.LEFT);
setSteps.setColor(getResources().getColor(R.color.chart_value_2));
setSteps.setCircleColor(Color.WHITE);
setSteps.setLineWidth(2f);
setSteps.setCircleRadius(3f);
setSteps.setFillAlpha(65);
setSteps.setFillColor(ColorTemplate.getHoloBlue());
setSteps.setHighLightColor(Color.rgb(244, 117, 117));
setSteps.setDrawCircleHole(false);

```

Figure 79: Line Chart Weekly Activity Class

Similarly, the Meal Log activity will also show the progress report for the weekly and monthly food items stored in the database as shown in Figure 80. The pie chart is used to represent the

progress report for the meal planner data, which will show the values of fat, carbohydrate, protein, and calories of the food added by the user in its log activity.

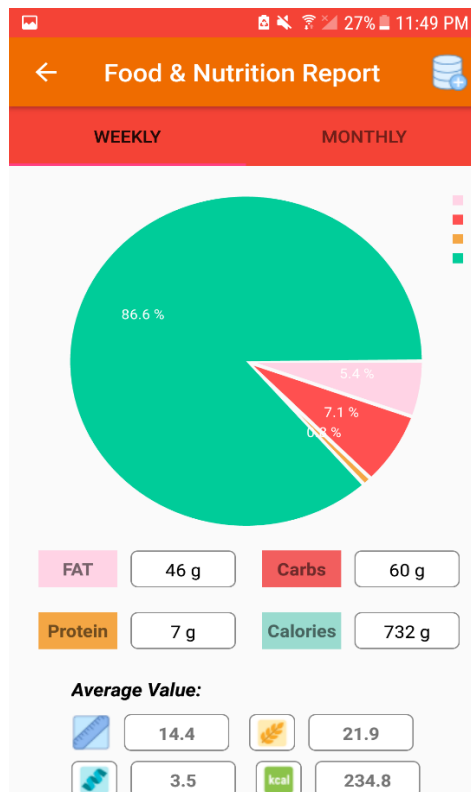


Figure 80: Food & Nutrition Report Screen

Food and nutrition report activity is also calculated for the average values of fat, carbohydrate, protein, and calories of the added food. Figure 81, shows the code for adding the nutrition values from the database and representing the pie chart using that data. It also calculates the average value of all the nutrition values including the food calorie value.

```

private void setData(int count, float range) {

    float mult = range;
    float avgFatValue = 0, avgCarbsValue = 0, avgSugarValue = 0, avgCalVal = 0;

    ArrayList<PieEntry> entries = new ArrayList<>();

    // NOTE: The order of the entries when being added to the entries array determines their position around the center of
    // the chart.
    String[] mParties = new String[] {
        "", "", "", "", "", ""
    };

    ArrayList<HashMap<String, String>> mlist = new ArrayList<>(DBHelper.getInstance(getActivity()).weeklyNutritionData());
    Log.d(tag, msg: "Size of array : "+getThisMonthFirstDate());
    for (int j=0; j < 3; j++) {
        for (int i = 0; i < mlist.size(); i++) {
            if (mlist.get(i).get("totalFat") != null) {
                totalFat = Float.parseFloat(mlist.get(i).get("totalFat"));
                tvFat.setText(new DecimalFormat( pattern: "##.##").format(totalFat) + " g");
                for (int j = 0; j < 3; j++) {
                    if (totalFat > 1) {
                        avgFatValue = avgFatValue + totalFat;
                        avgFatValue = avgFatValue / (j + 1);
                    }
                }
            }
            if (mlist.get(i).get("totalCarbs") != null) {
                totalCarbs = Float.parseFloat(mlist.get(i).get("totalCarbs"));
            }
        }
    }
}

```

Figure 81: Calculating Average Nutrition Values

Like the activity chart, the food chart also passes the values to the pie chart method by creating an array list and later adding those values to the list as shown in Figure 82. Then these lists are stored in data which will pass to the pie chart class.

```

private void setData(int count, float range) {

    float mult = range;

    ArrayList<PieEntry> entries = new ArrayList<>();

    // NOTE: The order of the entries when being added to the entries array determines their position around the ce.
    // the chart.
    String[] mParties = new String[] {
        "", "", "", "", "", ""
    };
    ArrayList<HashMap<String, String>> mlist = new ArrayList<>(DBHelper.getInstance(getActivity()).weeklyNutritionD);
    Log.d(tag, "Size of array : "+getThisMonthFirstDate());
    for (int i = 0; i < mlist.size(); i++) {
        Log.d(tag, "totalCalories : " + mlist.get(i).get("totalCalories"));
        Log.d(tag, "data : " + mlist.get(i).get("data"));
        Log.d(tag, "calories : " + mlist.get(i).get("calories"));
        Log.d(tag, "distance : " + mlist.get(i).get("distance"));
        Log.d(tag, "steps : " + mlist.get(i).get("steps"));
        if (mlist.get(i).get("totalFat") != null) {
            totalFat = Float.parseFloat(mlist.get(i).get("totalFat"));
            tvFat.setText(new DecimalFormat("###.##").format(totalFat) + " g");
        }
        if (mlist.get(i).get("totalCarbs") != null) {
            totalCarbs = Float.parseFloat(mlist.get(i).get("totalCarbs"));
            tvCarbs.setText(new DecimalFormat("###.##").format(totalCarbs) + " g");
        }
        if (mlist.get(i).get("totalSugar") != null) {
            totalSugar = Float.parseFloat(mlist.get(i).get("totalSugar"));
        }

        if (mlist.get(i).get("totalCalories") != null) {
            totalOther = Float.parseFloat(mlist.get(i).get("totalCalories")) - (totalFat + totalCarbs + totalSugar);
            tvOthers.setText(new DecimalFormat("###.##").format(totalOther) + " g");
        }
    }

    for (int i = 0; i < count ; i++) {
        entries.add(new PieEntry(totalOfNutrition(i),
            mParties[i % mParties.length],
            getResources().getDrawable(R.drawable.ic_add_black_24dp)));
    }

    PieDataSet dataSet = new PieDataSet(entries, "");

    //dataSet.setDrawIcons(false);

    dataSet.setSliceSpace(3f);
    //dataSet.setIconsOffset(new MPPointF(0, 40));
    dataSet.setSelectionShift(5f);

    // add a lot of colors

    ArrayList<Integer> colors = new ArrayList<>();

    for (int c : COLORFUL_COLORS)
        colors.add(c);
    colors.add(ColorTemplate.getHoloBlue());

    dataSet.setColors(colors);
    //dataSet.setSelectionShift(0f);

```

Figure 82: Setting Up the Data Points for the Food Report

Figure 83, shows that the application generates a notification to make the user aware that he/she has taken the food during the meal intervals. This application will send the notification updates to the user in the different interval of time throughout the day. Through the notification, the user can also add the food item in his food log or local database if they had the food item in the meal planner. The user can also manually add the food items from the meal planner activity using the circular add button shown in Figure 56.

```

sharedpreferences = getSharedPreferences("Mypref", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedpreferences.edit();
if(!sharedpreferences.getBoolean("alarmSet", false)) {
    setAlarm(context, 9, 0, 0, 1); //For Breakfast
    setAlarm(context, 13, 0, 0, 2); //For launch
    setAlarm(context, 17, 16, 0, 3); //For Dinner
    editor.putBoolean("alarmSet", true);
    editor.commit();
}

```

Figure 83: Method to Set Notification

So, wherever the notification comes, and the user clicks the yes button, then the food item currently stored in the breakfast section will be stored in the database. Later, it will display the values of fat, protein, carbohydrate and other nutrition values in the pie chart report. The report will be generated as a weekly and a monthly report showing the nutrition value intake by the user over the week or month. Similarly, for the lunch and dinner time notification store the values of nutrition from their appropriate section and later displays them in a pie chart.

4.10 Health Blog

In the Health blog activity, shown in Figure 84, the user can read latest blogs related to health and fitness information. The code is shown in Figure 85, to start this activity the user needs to click the health blog button on the user dashboard. This click will evoke the explicit intent to start the Health Blog activity. The swipeRefreshLayout layout is used to load the blog feed whenever the user refreshes the screen. The user can also click the recyclerView values and get a detailed page with the link to the site related to the clicked blog which can then be read by clicking on the mentioned link.



Figure 84: Health Blog Screen

```

public class HealthBlog extends AppCompatActivity {
    private RecyclerView mRecyclerView;
    private ArticleAdapter mAdapter;
    private SwipeRefreshLayout mSwipeRefreshLayout;
    //private String urlString = "http://www.androidcentral.com/feed";
    private String urlString = "https://medlineplus.gov/feeds/news_en.xml";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_health_blog);

        mRecyclerView = (RecyclerView) findViewById(R.id.list);
        mRecyclerView.setLayoutManager(new GridLayoutManager(this, 2));
        mRecyclerView.setItemAnimator(new DefaultItemAnimator());
        mRecyclerView.setHasFixedSize(true);

        mSwipeRefreshLayout = (SwipeRefreshLayout) findViewById(R.id.container);
        mSwipeRefreshLayout.setColorSchemeResources(R.color.colorPrimary, R.color.colorPrimaryDark);
        mSwipeRefreshLayout.canChildScrollUp();
        mSwipeRefreshLayout.setOnRefreshListener(() -> {

            mAdapter.clearData();
            mAdapter.notifyDataSetChanged();
            mSwipeRefreshLayout.setRefreshing(true);
            loadFeed();

        });
    }

    public void loadFeed() {

        Parser parser = new Parser();
        parser.execute(urlString);
        parser.onFinish(new Parser.OnTaskCompleted() {
            //what to do when the parsing is done
            @Override
            public void onTaskCompleted(ArrayList<Article> list) {
                //list is an Array List with all article's information
                //set the adapter to recycler view
                mAdapter = new ArticleAdapter(list, R.layout.row, HealthBlog.this);
                mRecyclerView.setAdapter(mAdapter);
                mSwipeRefreshLayout.setRefreshing(false);
            }

            //what to do in case of error
            @Override
            public void onError() {

                runOnUiThread(() -> {
                    if (!mSwipeRefreshLayout.isRefreshing())
                        mSwipeRefreshLayout.setRefreshing(false);
                    Toast.makeText(HealthBlog.this, "Unable to load data. Swipe down to retry.",
                        Toast.LENGTH_SHORT).show();
                    Log.i("Unable to load ", "articles");
                });
            }
        });
    }
}

```

Figure 85: Health Blog Code Snippet

5. CONCLUSION

5.1 Lesson Learnt

In the journey of developing this Android application, I got an opportunity to learn the detail process of developing Android applications with the help of material design concepts to built beautiful yet elegant user interface. In this process I also able to learn how to develop and use a RESTful API from scratch with the help of PHP. Moreover, I even get a chance to learn how to use different Android libraries such as MPAndroidChart, Zxing, FitChart and so on. I also learn about the various food ontology APIs on how to use them and even by developing the client-server application helped me to understand the communication calls between them. I got know more about the JSON data formats which used for exchanging data between client and server as well as to get the data from the food APIs.

5.2 Future Enhancements

In the future scope, the application can enhance its functionality by adding image recognition which can be used to analyze the food image and produce the result with the nutrition values contained in that particular food item. A Google map can be added to track the distance covered by the user using the Activity Tracker to provide a more visual representation of the activity to the user. The activity tracker can also be updated using the Google fit API for the more accurate result.

REFERENCES

- [1] “Leading health and fitness,” April 2017 [Online]. Available:
<https://www.statista.com/statistics/650748/health-fitness-app-usage-usa>
- [2] Android Development Guide Site, April 2017 [Online].
Available: <https://developer.android.com>
- [3] Android Developer: Activity, April 2017 [Online].
Available: <https://developer.android.com/reference/android/app/Activity.html>
- [4] Android Developer: Service page, April 2017 [Online].
Available: <https://developer.android.com/guide/components/services.html>
- [5] Android Developer: Fragment, April 2017 [Online].
Available: <https://developer.android.com/guide/components/fragments.html>
- [6] Android SDK Environment, April 2017 [Online].
Available: <https://developer.android.com/studio/index.html>
- [7] Run Android Application, April 2017 [Online].
Available: <https://developer.android.com/training/basics/firstapp/running-app.html>
- [8] Circular Cuboid Button, June 2017 [Online].
Available: <https://github.com/MuhammadArsalanChishti/CuboidCircle-Button>
- [9] Volley Library, June 2017 [Online].
Available: <https://developer.android.com/training/volley/index.html>
- [10] MPAndroidChart library, Aug 2017 [Online].
Available: <https://github.com/PhilJay/MPAndroidChart>
- [11] Support library, June 2017 [Online].
Available: <https://developer.android.com/topic/libraries/support-library/index.html>

- [12] Glide library, June 2017 [Online]. Available: <https://github.com/bumptech/glide>
- [13] Zxing Library, June 2017 [Online]. Available: <https://github.com/zxing/zxing>.
- [14] FitChart library, June 2017 [Online].
Available: <https://github.com/txusballesteros/fit-chart>.
- [15] HideReturnsTransformationMethod [Online]. Available:
<https://developer.android.com/reference/android/text/method/HideReturnsTransformationMethod.html>
- [16] GMT Blog (2017), April 2017 [Online]. Available: <http://www.gkmit.co/articles/how-to-use-model-class-in-android>
- [17] Nutritionix API, June 2107 [Online].
Available:<https://www.nutritionix.com/business/api>
- [18] Broadcast Reciever, Jan 2018[Online]. Avaiable:
<https://developer.android.com/guide/components/broadcasts.html>
- [19] Sunil Android (2017), “Custom Seek Bar (Discrete Seek Bar) in Android, 16 Jan 2017 [Online]. Available: <http://www.sunilandroid.com/2017/01/custom-seekbar-discrete-seekbar-in.html>
- [20] Mukasine Angelique (2014). “Ontology-Based Personalized System to Support Patients at Home.” Research paper [Online]. Available:
<https://brage.bibsys.no/xmlui/bitstream/handle/11250/221227/IKT-590%20Spring%20Matster's%20thesis%20Angelique%20MUKASINE.pdf?sequence=1>
- [21] Udacity Android Development Course, May 2017 [Online]. Available:
<https://www.udacity.com/course/android-basics-nanodegree-by-google--nd803>