Mobile Application Development
Final Report

# Nutrition Recipe Mobile Application


Group 5 – ICT

*Members*

| | |
|---|---|
| BI11-110 | Phùng Gia Huy |
| BI11-195 | Nguyễn Hoài Nam |
| BI11-198 | Hoàng Đức Nghĩa |
| BI11-236 | Bùi Hồng Sơn |
| BI11-273 | Nguyễn Đăng Trung |
| BI11-286 | Nguyễn Xuân Vinh |

*Lecturer*
Dr. Trần Giang Sơn

*ICT Lab*
December, 2022

# Content

# List of Figures

# List of Tables

# I. INTRODUCTION

*1. Context and Motivation*

Nowadays, nutrition in each meal has played an important and essential role in our life. In the past, we might not know exactly the components of nutrition in each food, as well as their quantity and quality. Besides, we could not control the calories in each meal so well. As a result, we could not find it easy to arrange a meal with reasonable nutritional balance, not too many carbohydrates or proteins. Consequently, it could lead to conditions of excess and lack of nutrients or calories. Therefore, we might feel difficult to get a totally healthy body.

As can be seen, balanced nutrition and calories, at least knowing the details of nutrients and calories in each dish is very important. In addition, we should also know how to arrange our meal in a reasonable and proper way. Knowing and understanding the detailed components in each food is one problem, but arranging them is another problem. Suppose that you could know the component of food apparently (thanks to pioneer nutritionist and biologist in the past), and your next mission is making the meal arrangement logically. With the development of technology nowadays, especially the internet and computers, we can search on the internet, write them to the paper or save the text of a nutrition recipe in the computer. From my point of view, that could not be one of the most convenient and comfortable methods. Computers or Laptops can be a portable convenient way. Hence, at least at the moment, we should find a solution to solve this problematic matter in an effective, convenient, enjoyable and maybe not expensive way.

As can be known, the development of mobile phones in contemporary days is really popular and ubiquitous. More and more people tend to use smartphones nowadays. And in this project, a Nutrition Recipe Mobile Application is anticipated to resolve the mentioned problem with contemporary tendency. With this application, many people can easily manage their nutrients and calories in each meal. And it is also much more convenient for a lot of people to monitor their diet regularly every time or everywhere just by using small and pocketable smartphones. By this way, we can see that this method is really advantageous.

*2. Thesis Structure*

The thesis is likely to contain all the information to reproduce the result. Firstly, we are anticipated to introduce the interesting issue in monitoring diet and the solution proposed in **section I: Introduction**. In **section II: Objectives**, we are projected to define expected requirements of the project, provide a concise overview of the function of the system and the reasons for its development and improvement. Then, we will illustrate the scenarios, use cases, object model, and dynamic models for the system in **section III: Requirement Analysis**. This section contains the complete functional particularization, including navigational paths representing the sequence of screens. In **section IV: Methodology**, we are expected to list all the techniques and tools utilized in the project, the reasons why they are opted and the detailed use cases implementation. In **section V: Result**, we are going to list all the functionalities which are implemented in the system. There is **Conclusion and Future Work** in **section VI**. Finally is **section VII: Appendix**, which displays all the user interfaces of our system.

## II. OBJECTIVE

In this section, we will define the expected essential demand of the project, provide a concise overview of the function of the application and the casuals for its development.

*1. Desired Features*

Our application tries to help users be able to build nutrition recipes on their mobile phones. Hence, the main aim of this project is to develop an application with basic following services:

- **Feature 1**: **Authenticate**
  - **Sub-Feature 1.1**: **Register**: allows users to create their own accounts.
  - **Sub-Feature 1.2**: **Login**: allows users to access the system.
- **Feature 2**: **Particular Meal**: allows users to interact with many main functionalities here.
  - **Sub-Feature 2.1**: **Search Food**: allows users to search food and they can add food to the list.
  - **Sub-Feature 2.2: Add - List Food**: allows users to add food here and see the list food.
  - **Sub-Feature 2.3**: **Total Nutrients & Energies**: allows and supports users to calculate the nutrients & energies of a specific meal.
- **Feature 3**: **Menu**:
  - **Sub-Feature 4.1**: **Settings**: allows users to know more about the group doing this application, how to contact this group, give feedback, know more about credits and they can delete their account.
  - **Sub-Feature 4.2**: **Helps**: allows users to debug logs, know more about terms of service and FAQs.
  - **Sub-Feature 4.3**: **Background Music**: allows users to choose and listen to music they want whereas using this application.
  - **Sub-Feature 4.4**: **Calculator**: allows and supports users to calculate the number of energies and nutrients.
  - **Sub-Feature 4.5**: **Log Out**: allows users to log out of account they have used befored

*2. Expected Outcome*

Our system aims at helping users control their diet in an effective, comfortable and convenient way. The particular goals include:

- Develop a backend environment for monitoring data of users like username, password, quantity and quality of calories or nutrients.
- Develop a user interface that could help users interact easily, comfortably and conveniently.
- The application should have all the features mentioned in the part of Desired Feature and make users feel satisfied.
- This application should be able to run in various types of smartphones having the Android operating system.

## III. REQUIREMENT ANALYSIS

Our application is about building the Nutrition Recipe and the application is used on mobile phones. Hence, in this section, we are likely to describe a brief overview of the functions in the project, the prospect and use cases for the system. This part contains the complete functional particularization, including navigational paths representing the sequence of screens.

*1. General System Requirements*

Overall, our application has to satisfy the following main requirements:

- A login and register system with authentication.
- The system could limit users with some feature:
    + Search food.
    + Add food to the list and be able to see the list.
    + Calculate total energies and nutrients in each meal and all day.
    + Delete account.
    + Know about the group of creators and be able to contact them.
    + Give feedback.
    + Know more about credits, terms of service and FAQs.
    + Listen to music.

*2. Use Cases*

2.1. Use case Diagram

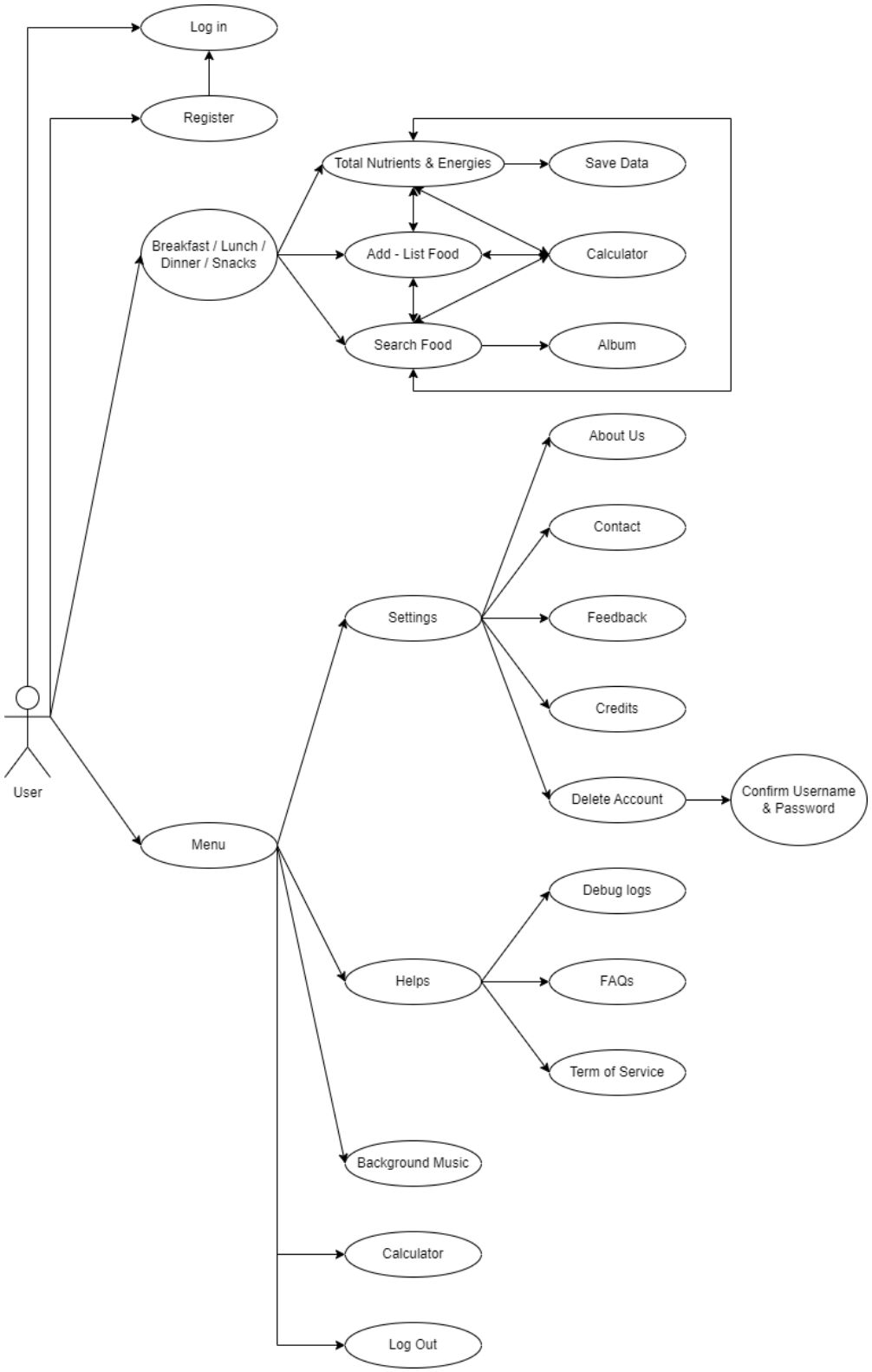


**Figure 1**. Use Case Diagram

## 2.2. Users Characteristics

Our application is building the recipe of nutrition used on the mobile phone. Therefore, almost all users can use this application, regardless of age or gender. The users are expected to have their own requirements.

A user who utilizes the application checks and monitors their diet each meal and everyday.

## 2.3. Use case Description

| No. | Use Case Name | None |
|---|---|---|
| UC.1 | Login | None |
| UC.2 | Register | None |
| UC.3 | Particular Meal | None |
| UC.4 | Menu | None |
| UC.5 | Total Nutrients & Energies | Include case of UC.3, 6, 7, 14 |
| UC.6 | Add - List Food | Include case of UC.3, 5, 7, 14 |
| UC.7 | Search Food | Include case of UC.3, 5, 6, 14 |
| UC.8 | Settings | Include case of UC.4 |
| UC.9 | Helps | Include case of UC.4 |
| UC.10 | Background Music | Include case of UC.4 |
| UC.11 | Calculator | Include case of UC.4 |
| UC.12 | Log Out | Include case of UC.4 |
| UC.13 | Save Data | Include case of UC.5 |
| UC.14 | Calculator | Include case of UC.5, 6, 7 |
| UC.15 | Album | Include case of UC.7 |
| UC.16 | About Us | Include case of UC.8 |
| UC.17 | Contact | Include case of UC.8 |
| UC.18 | Feedback | Include case of UC.8 |
| UC.19 | Credits | Include case of UC.8 |
| UC.20 | Delete Account | Include case of UC.8 |
| UC.21 | Debug logs | Include case of UC.9 |
| UC.22 | FAQs | Include case of UC.9 |
| UC.23 | Term of Service | Include case of UC.9 |
| UC.24 | Confirm Username & Password | Include case of UC.20 |

**Table 1**. Use Case Table

2.4. Use Case and Scenario Description

| Use Case Name | Login | | UC.1 |
|---|---|---|---|
| User/Actor | All kinds of users | Relationship | Include: UC.2 |
| Brief Definition | User logs into the application with username and password. | | |
| Basic Flow | 1) User opens the application.<br>2) User inputs his/her username and password.<br>3) User is directed into the main page view. | | |
| Alternative Flow | If the user inputs the wrong username or password, or does not fill in all the blanks yet; the application will display a message to notify him/her. | | |
| Pre-condition | User hasn't logged into the application before or has logged out with the previous login session. | | |

**Table 2**. Scenario Description of UC.1

| Use Case Name | Register | | UC.2 |
|---|---|---|---|
| User/Actor | All kinds of users | Relationship | Include: None |
| Brief Definition | User creates his/her ow account with his/her own username and password. | | |
| Basic Flow | 1) User opens the application.<br>2) User clicks the button "Sign up".<br>3) User inputs his/her own username and password to register.<br>4) User clicks the button "Register" when done.<br>5) User is directed to the Login User Interface. | | |
| Alternative Flow | If the user does not fill in all the blanks yet; the application will display a message to notify him/her. | | |
| Pre-condition | User hasn't created his/her own account before. | | |

**Table 3**. Scenario Description of UC.2

| Use Case Name | Particular Meal | | UC.3 |
|---|---|---|---|
| User/Actor | All kinds of users | Relationship | Include: UC.5, 6, 7, 13, 14, 15 |
| Brief Definition | User is able to search Food, add Food to the list and calculate the total nutrients and energies. | | |
| Basic Flow | 1) If the user clicks the button -Meal-, user is directed to the Meal User Interface and user can save the data of nutrients and energies by clicking the button "Save" and calculate many numbers by using the calculator.<br>2) If the user clicks the button "ADD FOOD", user is directed to the Add Food User Interface and user could click the button -information- to see the album of food and clicks the button -add- to add food to the list.<br>3) If the user clicks the button "List", user can see the list food added before there. | | |
| Alternative Flow | None | | |
| Pre-condition | Successfully perform UC.1 | | |

**Table 4**. Scenario Description of UC.3

| Use Case Name | Menu | | UC.4 |
|---|---|---|---|
| User/Actor | All kinds of users | Relationship | Include: UC.8, 9, 10, 11, 12, 16, 17, 18, 19, 20, 21, 22, 23, 24 |
| Brief Definition | User is able to do something with Settings, Helps, listen to music, use calculator and Log Out. | | |
| Basic Flow | 1) User clicks the button -Menu- on the top right corner of the Main Page User Interface.<br>2) User is directed to the Menu User Interface.<br>3) If the user clicks the button "Settings", user is directed to the About Us User Interface, Contact User Interface, Feedback User Interface, Credits User Interface and user could delete account by being directed to Delete Account.<br>4) If the user clicks the button "Helps", user is directed to the Debug logs User Interface, FAQs User Interface, Term of Service User Interface.<br>5) If the user clicks the button "Background Music", user is directed to Background Music User Interface and he/she can listen to some musics while utilizing the application.<br>6) If the user clicks the button "Calculator", user is directed to Calculator User Interface. Here, user can use the calculator to calculate some complex numbers.<br>7) If the user clicks the button "Log Out", user will log out with the previous log in session and user is directed to the first Login–Register User Interface. | | |
| Alternative Flow | None | | |
| Pre-condition | Successfully perform UC.1 | | |

**Table 5**. Scenario Description of UC.4

## IV. METHODOLOGY

Our project is about making nutrition recipes on the mobile phones. So, in this section, all the tools and techniques used in the project, the reason why they are opted and the detailed use cases implementation are projected to be listed.

### *1. Tools and Techniques*

#### 1.1. **Figma**

Figma is a browser-based, collaborative user interface design tool that lets users work together to create vibrant and interactive prototypes. Since its release in 2016, Figma has become a popular tool both in the web design industry and in online communities. Users can collaborate and share templates, designs, and widgets with millions of users across the globe.

We use Figma to design our Web User Interface.

#### 1.2. **Android Studio**

Android Studio is an open source code based on Linux Kernel for all mobile devices (smartphones, tablets, smart watches).

Android Studio is also an official IDE and it is used widely. It is developed by Google and Google uses it to create applications that we use everyday.

Android Studio has library software as well as utility tools. It also help us build, test and assist us in debugging Android applications.

Android Studio also has some advantages:

- It is developed by Google which is also the owner of the Android operating system.
- The tools are completely supported and updated.
- The feature is easy to get used to and the interface is friendly => a big plus.

=> It helps us a lot in making the application as we want in the Android operating system.

*2. System Architecture*

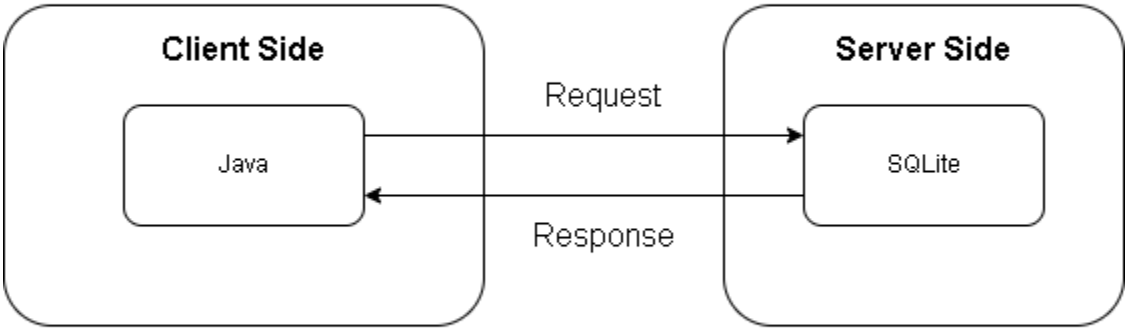


**Figure 2**. System Architecture Diagram

The Nutrition Recipe Mobile Application could be a full side project with a completely developed Client side (Front-end) and Server side (Back-end):

- The Server side receives the request from the Client side, processes and gets data from the Database then returns a response.
- The Client side interacts with the user, retrieves the user request, sends it to the Server side, receives the response from the Server then response data back and displays it to the Client side.

*3. Database Design*

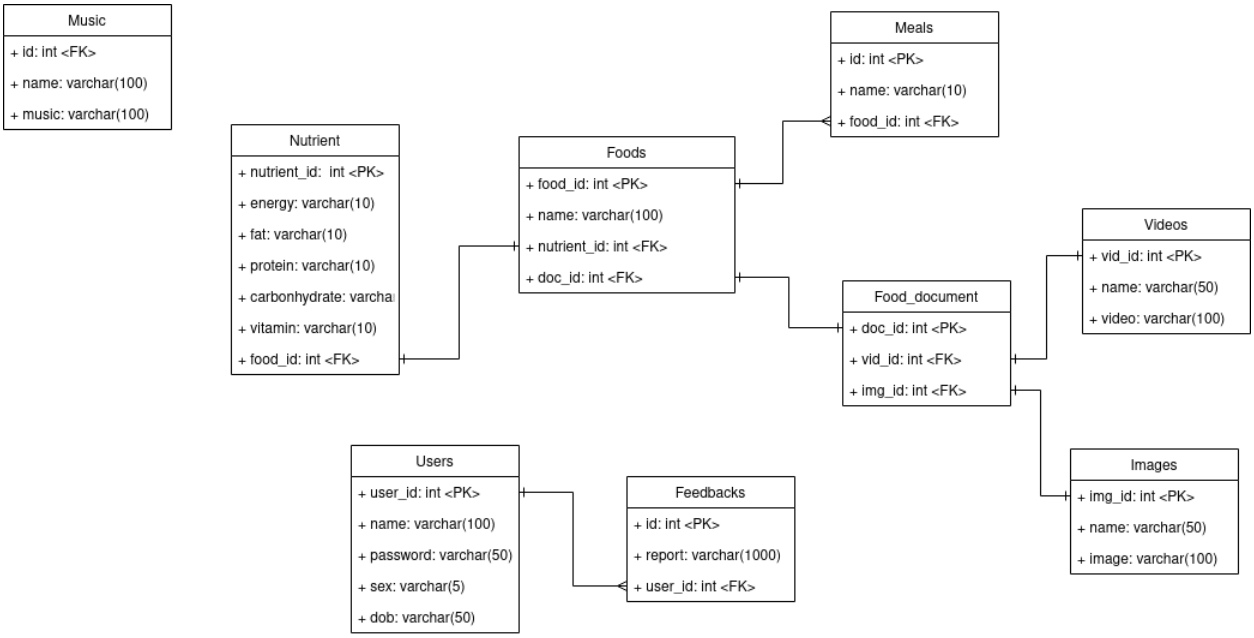


**Figure 3**. Overall Database Design

| Music |
|---|
| + id: int <FK> |
| + name: varchar(100) |
| + music: varchar(100) |

This table displays all information about the music used in the application.

**Table 6**. Music Database Design

| Nutrient |
|---|
| + nutrient_id: int <PK> |
| + energy: varchar(10) |
| + fat: varchar(10) |
| + protetin: varchar(10) |
| + carbohydrate: varchar(10) |
| + vitamin: varchar(10) |
| + food_id: int <FK> |

This table displays all information about the food. Each food has each own information about nutrients or energies. Hence, it has a one-to-one relationship with table Foods.

**Table 7**. Nutrient Database Design

| Users |
| --- |
| + user_id: int <PK> |
| + name: varchar(100) |
| + password: varchar(50) |
| + sex: varchar(50) |
| + dob: varchar(50) |

**Table 8**. Users Database Design

This table displays all information about the user. Each user can write one or many feedbacks. Therefore, it has a one-to-many relationship with table Feedbacks.

| Foods |
| --- |
| + food_id: int <PK> |
| + name: varchar(100) |
| + nutrient_id: int <FK> |
| + doc_id: int <FK> |

**Table 9**. Foods Database Design

This table displays all information about the food. Each food can be used in many meals. So, it has a one-to-many relationship with table Meals. In addition, each food has their own information like video or image. Therefor, it has a one-to-one relationship with table Food Document.

| Feedbacks |
| --- |
| + id: int <PK> |
| + report: varchar(1000) |
| + user_id: int <FK> |

**Table 10**. Feedbacks Database Design

This table displays all information about the feedbacks sent by the users. A lot of feedbacks could be sent by a user. Hence, it has a many-to-one relationship with table Users.

| Meals |
| --- |
| + id: int <PK> |
| + name: varchar(10) |
| + food_id: int <FK> |

**Table 11**. Meals Database Design

This table displays all information about the meals. One food can appear in many meals. Therefore, it has a many-to-one relationship with table Foods.

| Food_document |
| --- |
| + doc_id: int <PK> |
| + vid_id: int <FK> |
| + img_id: int <FK> |

**Table 12**. Food Document Database Design

This table displays all information about the food document. Each food has an own private food document. So, each video or each image is relative to one kind of food.

| Videos |
| --- |
| + vid_id: int <PK> |
| + name: varchar(50) |
| + video: varchar(100) |

**Table 13**. Videos Database Design

This table displays all information about the videos of the food. Each food document has an own private video. Hence, it has a one-to-one relationship with table Food Document.

| Images |
| --- |
| + img_id: int <PK> |
| + name: varchar(50) |
| + image: varchar(100) |

**Table 14**. Images Database Design

This table displays all information about the images of the food. Each food document has an own private image. Therefore, it has a one-to-one relationship with table Food Document.

*4. Use Cases Implementation*

4.1. Authentication

This use cases illustrates how a user log into the application and create for his/her own account.

User could be whoever people around the world.

To build for his/her nutrition recipe, he/she needs to have their own account first. To possess an account, he/she needs to register initially. First of all, when the user opens the application, he/she is directed to the first User Interface. If the user clicks the button “Sign up”, he/she is directed to the Register User Interface. To register, the user needs to fill the username and password. The user does not succeed in registering if the re-entered password is not same as before password or there are still some blank not filled yet. There is a notification sent to the user. When done, the user can click the button “Register”. The data is sent to the Database to process and return. Then, the user is directed to the Log in User Interface. Otherwise, user can be directed to the Login User Interface from the first User Interface by clicking the button “Log in”. Same as the Register part, the user needs to fill all the blanks and confirm the exact password. The user could click the button “Log in” when done. Then, the data is sent to the Database to process and return. Finally, if the user logs into the application successfully, he/she will be directed to the Main Page User Interface.

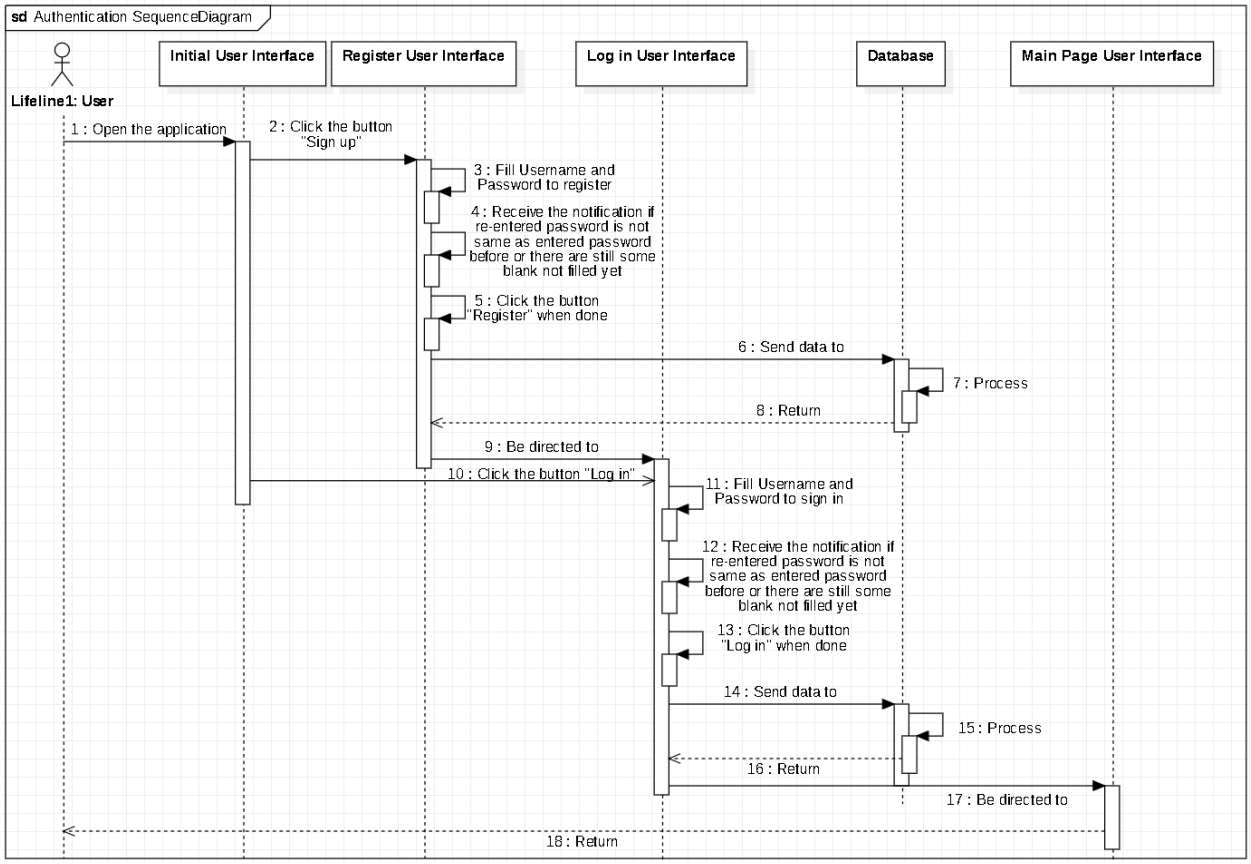


**Figure 4**. Authentication Sequence Diagram

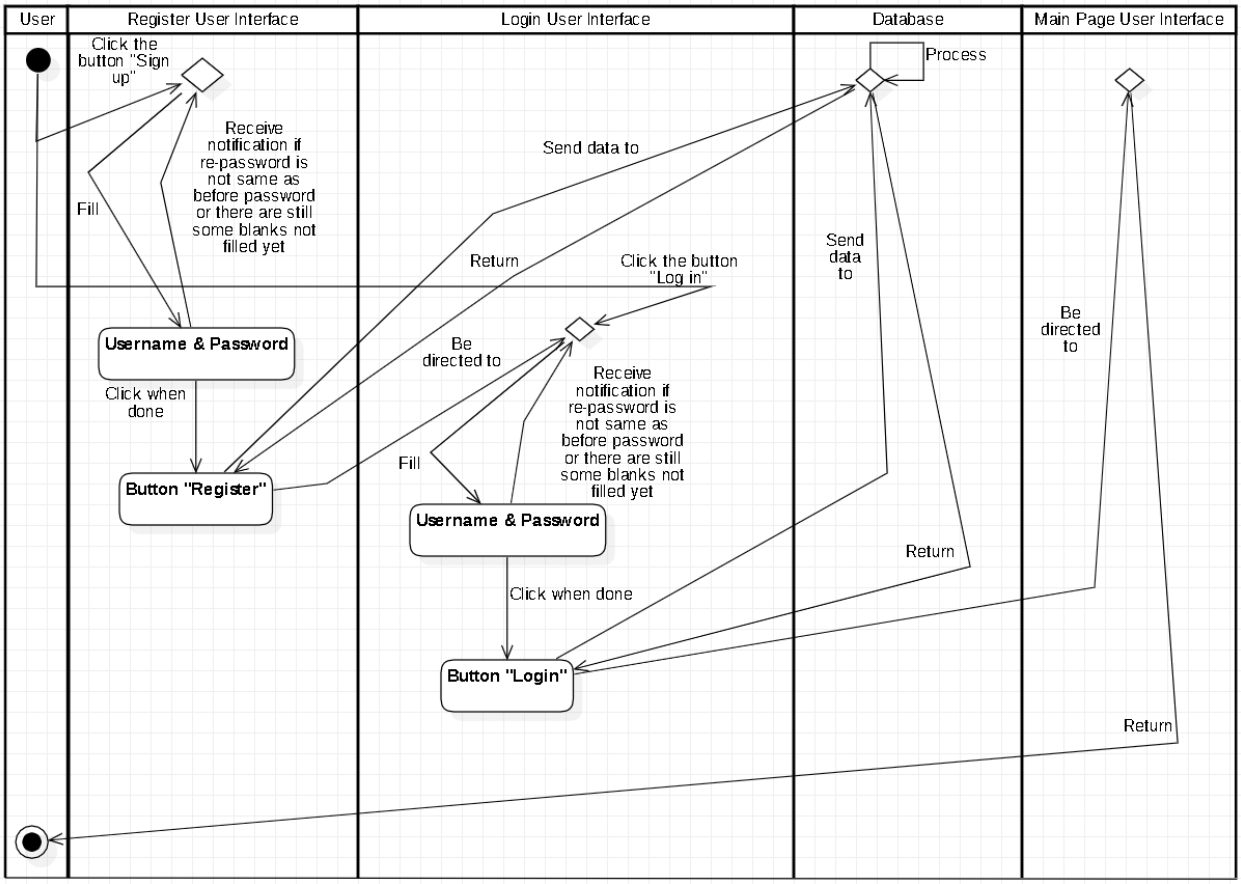


**Figure 5**. Authentication Activity Diagram

4.2. Particular Meal

This use cases illustrates how a user interact with the part of Particular Meal.
User could be whoever people around the world.
To interact with the part of Particular Meal, like breakfast, firstly, user can search Food by click the button “Add Food”. Here, user can search the food by typing the name of the food on the SearchView. There will be some items of food below. User can see the album of food by clicking the button -Info- on each item. User can add food to the list by clicking the button -Add-. User can see the list food by clicking the button “List Food” from the Main Page User Interface or clicking the item “List” on the menu on the right top corner. The Menu on the right top corner include some items: List, Calculator, Add, Delete, Total. Hence, user can easily change to other layout or activity. User can delete the List Food by clicking the item “Delete” on the menu on the right top corner. User can be directed to Total Nutrients & Energies User Interface by clicking the button -Particular Meal- in Main Page User Interface or clicking the item “Total” on the menu on the right top corner. Here, user is able to input the quantity of nutrients or energies and save the data inputted. User can use calculator to be supported calculating many numbers together. User can be directed to the Calculator User Interface by clicking the item “Calculator” on the menu on the right top corner.

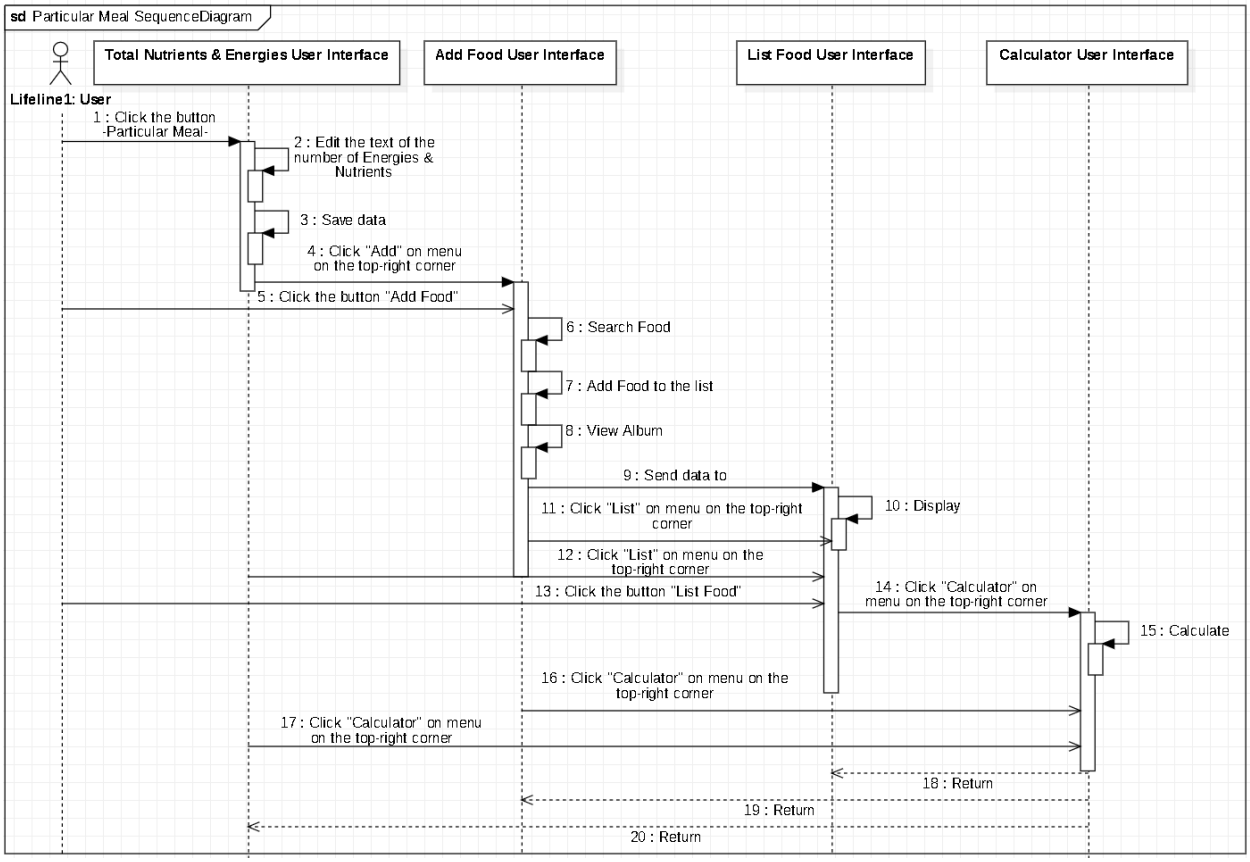


**Figure 6**. Particular Meal Sequence Diagram

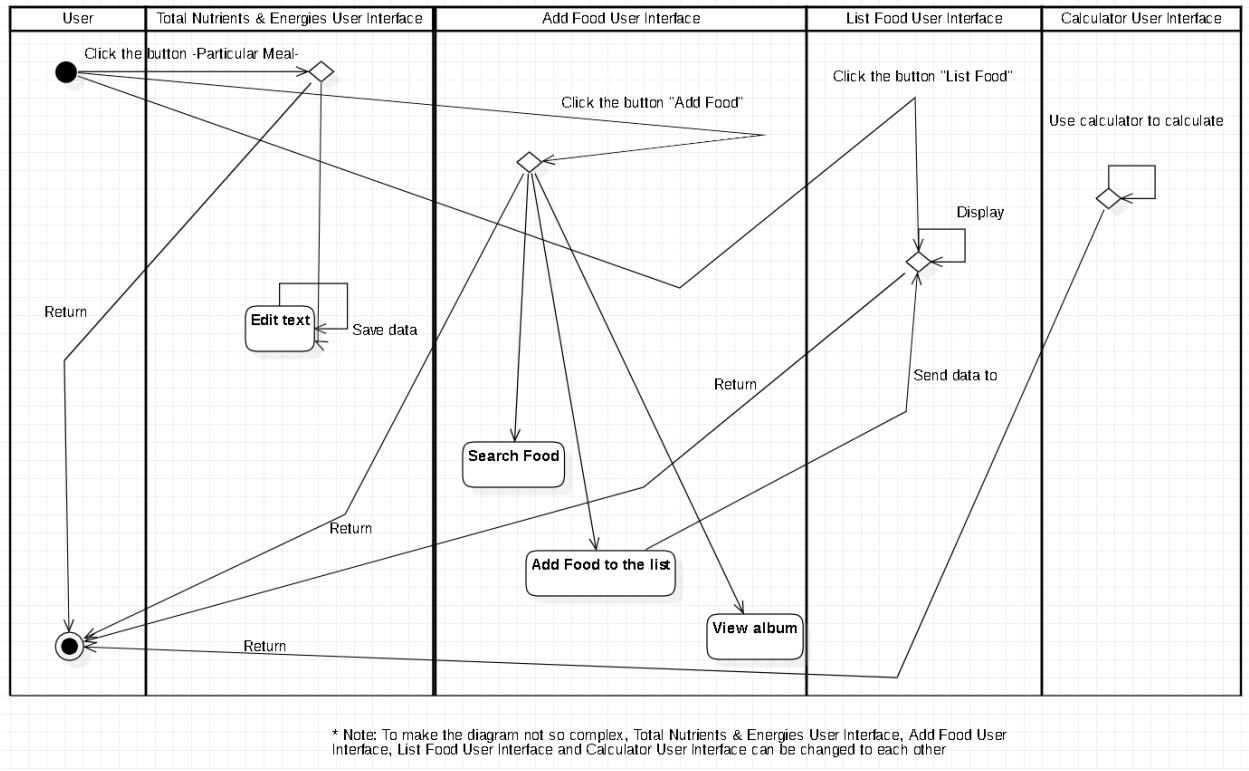


**Figure 7**. Particular Meal Activity Diagram

4.3. Menu Layout – Activity

This use cases illustrates how a user interact with the Menu Layout - Activity

User could be whoever people around the world.

User can interact with many things in Menu. If user clicks the button “Settings”, user can view About Us, Contacts, Credits or send feedbacks. User can also delete account here. If user clicks the button “Delete Account”, user will receive an alert to delete or not. If user confirms, user will be directed to the Delete Account User Interface. Here, user confirms username and password to delete account. From Menu Layout User Interface, if user clicks the button “Helps”, user can view Debug logs, FAQs, Term of Service. From Menu Layout User Interface, if user clicks the button “Background Music”, user will be directed to the Background Music User Interface. Here, user can listen to list of musics while using the application. User can also pause, stop or continue the music. From Menu Layout User Interface, if user clicks the button “Calculator”, user will be directed to the Calculator User Interface and use calculator to calculate a lot of complex numbers.

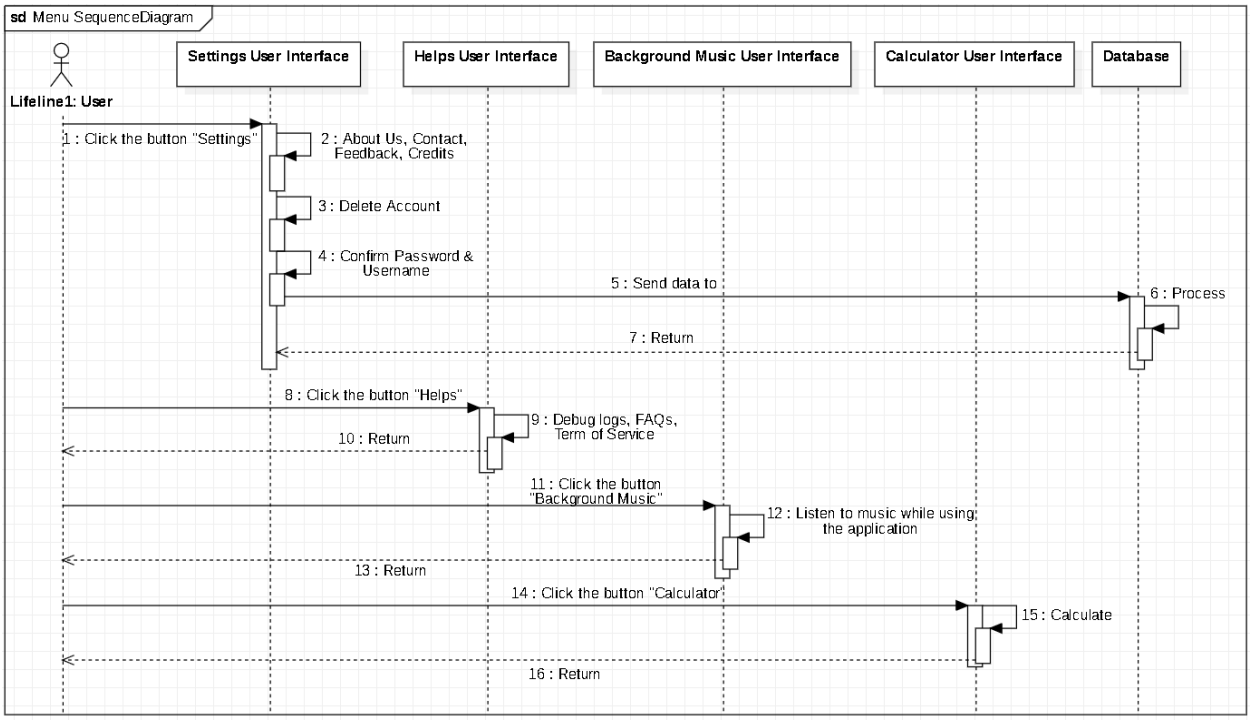


**Figure 8**. Menu Sequence Diagram

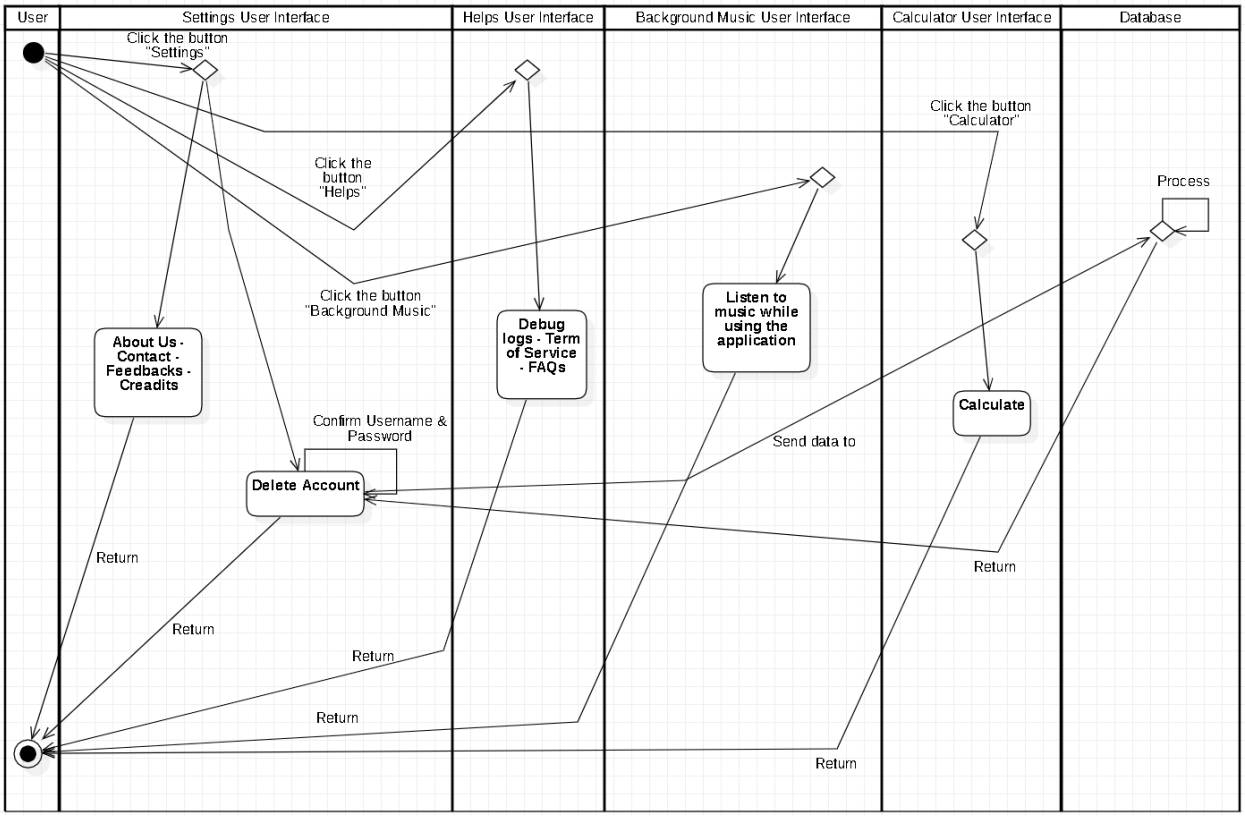


**Figure 9**. Menu Activity Diagram

*5. Code Method*

5.1. Layouts

Firstly, we set nothing on our screen by change the code in the values\themes.xml and night\themes.xml

```
<style name="Theme.NutritionRecipe" parent="Theme.AppCompat.Light.NoActionBar">
```

Then, we set code of colors in colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#03A9F4</color>
    <color name="gray">#CCCCCC</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue">#0099FF</color>
    <color name="purple_700">#7B03F4</color>
</resources>
```

We design the icon, then we save it in ic_launcher in mipmap. Then, we change the logo of our application in AndroidManifest.xml

```
        android:icon="@mipmap/ic_launcher"
        android:label="Nutrition Recipe"
        android:roundIcon="@mipmap/ic_launcher_round"
```

We set the name of our application in strings.xml

```
<string name="app_name">Nutrition Recipe</string>
```

We make icon by creating Vector Asset in drawable. This is our example of syntax to make an icon

```
<vector android:height="40dp" android:tint="#0099FF"
    android:viewportHeight="24" android:viewportWidth="24"
    android:width="40dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="@android:color/white" android:pathData="M19,13h-6v6h-2v-6H5v-2h6V5h2v6h6v2z"/>
</vector>
```

We make the menu in folder menu. This our example of syntax to make the menu

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/item1"
        android:title="Total"
        android:icon="@drawable/ic_total_food"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/item2"
        android:title="Delete"
        android:icon="@drawable/ic_delete_food"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/item3"
        android:title="Add"
        android:icon="@drawable/ic_baseline_add_food"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/item4"
        android:title="Calculator"
        android:icon="@drawable/ic_calculator"
        app:showAsAction="ifRoom"/>
</menu>
```

We use both LinearLayout and RelativeLayout: LinearLayout for some layouts and RalativeLayout for other layouts.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context=".LoginFragment"
    android:background="#0099FF"
    android:gravity="center"
    android:orientation="vertical"
    >
```

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/gray"
    tools:context=".TotalSnacksActivity">
```

We use ScrollView for some long contents in some layouts

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/gray"
    tools:context=".BreakfastListActivity">
```

We use MaterialCardView to display each item in the list

```
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:cardElevation="10dp"
    app:cardCornerRadius="20dp"
    app:strokeWidth="3dp"
    app:cardBackgroundColor="@color/white"
    android:layout_marginRight="10dp"
    android:layout_marginLeft="10dp"
    app:strokeColor="#0099FF"
    tools:ignore="MissingClass">
```

We use custom action bar androix.appcompat.widget.Toolbar to create Toolbar in Layout instead of using the default one.

```
<androidx.appcompat.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#0099FF"
    android:elevation="4dp"
    >
```

We use TextView to display the text

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Feedback"
    android:textSize="30dp"
    android:textColor="@color/black"
    android:layout_gravity="center"
    />
```

We use Button to create button

```
<Button
    android:id="@+id/arrow_feedback"
    android:layout_width="50dp"
    android:layout_height="30dp"
    android:background="@drawable/back_arrow"
    android:layout_gravity="left"
    />
```

We use EditText to edit the text

```
<EditText
    android:layout_width="match_parent"
    android:layout_marginLeft="280dp"
    android:layout_marginRight="75dp"
    android:layout_height="60dp"
    android:hint="0"
    android:layout_marginTop="210dp"
    android:background="@color/white"
    android:textColor="@color/black"
    android:textSize="16dp"
    android:paddingLeft="10dp"
    android:id="@+id/edittext_carbohydrate_breakfast"
    />
```

We use the ImageView to display the image

```
<ImageView
    android:layout_marginTop="20dp"
    android:layout_centerHorizontal="true"
    android:id="@+id/beans"
    android:layout_width="300dp"
    android:layout_height="200dp"
    android:src="@drawable/beans"
    android:layout_below="@+id/tool"
    />
```

We use CheckBox to display the Check Box

```
<CheckBox
    android:id="@+id/checkbox_debug_logs"
    android:layout_width="fill_parent"
    android:layout_height="60dp"
    android:layout_marginTop="70dp"
    android:paddingLeft="10dp"
    android:text="Debug Logs"
    android:textSize="30dp"
    android:layoutDirection="rtl"
    android:background="@color/white"
    android:textColor="@color/black"
    android:checked="false"/>
```

Besides, we also use many differents syntaxes to design more beautifully and eye-catchingly like layout_margin, padding, background,….

## 5.2. Functions

To make the button be able to change layout or activity, this is our example of syntax

```
View btn = findViewById(R.id.arrow_album_breakfast);
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        startActivity(new Intent( packageContext: AlbumActivity.this, BreakfastActivity.class));
    }
});
```

This is our example of syntax to make an alert

```
delete_btn = (Button) findViewById(R.id.delete_account);
builder = new AlertDialog.Builder( context: this);
delete_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        builder.setTitle("Delete User?")
                .setMessage("Do you really want to delete your account?\n\nNote: You can not undo this action!")
                .setPositiveButton(Html.fromHtml( source: "<font color='#FF453A'>Confirm</font>"), new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        Intent intent = new Intent( packageContext: SettingsActivity.this, DeleteChoiceActivity.class);
                        startActivity(intent);
                    }
                }).setNegativeButton( text: "No", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        dialogInterface.dismiss();
                    }
                })
                .show();
    }
});
```

To make and connect the database, we have some examples of syntax

Firstly, we create the database

```
public static final String DBNAME = "Bocchi.db";
public DBHelper(Context context) { super(context,  name: "Bocchi.db",  factory: null,  version: 1); }
```

Then, we create a table in the database

```
public static final String TABLE_NAME1 = "users";
@Override
public void onCreate(SQLiteDatabase MyDB) {
    String table1 = "CREATE TABLE "+TABLE_NAME1+"(username TEXT PRIMARY KEY, password TEXT)";
    MyDB.execSQL(table1);

}
```

To update the table in the database, we have the example of syntax

```
@Override
public void onUpgrade(SQLiteDatabase _db, int oldVer, int newVer) {
    _db.execSQL("DROP TABLE IF EXISTS " +TABLE_NAME1);
    onCreate(_db);
}
```

To insert data to the table in the database, we have the example of syntax

```
public Boolean insertData(String username, String password){
    SQLiteDatabase MyDB = this.getWritableDatabase();
    ContentValues contentValues= new ContentValues();
    contentValues.put("username", username);
    contentValues.put("password", password);
    long result = MyDB.insert(TABLE_NAME1, nullColumnHack: null, contentValues);
    if(result==-1)
        return false;
    else
        return true;
}
```

To verify the data is exist or not, we have the example of syntax

```
public Boolean checkusername(String username) {
    SQLiteDatabase MyDB = this.getWritableDatabase();
    Cursor cursor = MyDB.rawQuery( sql: "Select * from "+TABLE_NAME1+" where username = ?", new String[]{username});
    if (cursor.getCount() > 0)
        return true;
    else
        return false;
}
```

To delete data in the database, we have the example of syntax

```
public void deleteAccount(String username, String password){
    SQLiteDatabase MyDB = this.getWritableDatabase();
    MyDB.delete(TABLE_NAME1, whereClause: "username=?", new String[]{username});
    MyDB.delete(TABLE_NAME1, whereClause: "password=?", new String[]{password});
    MyDB.close();
}
```

We have the functions to register with get the data from EditText

```
// Get ID of EditText and Button
username = (EditText) findViewById(R.id.username_txt);
password = (EditText) findViewById(R.id.password_txt);
repassword = (EditText) findViewById(R.id.retype_password_txt);
signup = (Button) findViewById(R.id.enter_sign_up);
// Get the DB
DB = new DBHelper( context: this);
```

```
// create function of button "Register"
signup.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String user = username.getText().toString();
        String pass = password.getText().toString();
        String repass = repassword.getText().toString();
```

```
// condition in the function
if(user.equals("")||pass.equals("")||repass.equals(""))
    Toast.makeText( context: SignUpActivity.this, text: "Please enter all the fields", Toast.LENGTH_SHORT).show();
else{
    if(pass.equals(repass)){
        Boolean checkuser = DB.checkusername(user);
        if(checkuser==false){
            Boolean insert = DB.insertData(user, pass);
            if(insert==true){
                Toast.makeText( context: SignUpActivity.this, text: "Registered successfully", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(getApplicationContext(),LoginActivity.class);
                startActivity(intent);
            }else{
                Toast.makeText( context: SignUpActivity.this, text: "Registration failed", Toast.LENGTH_SHORT).show();
            }
        }
        else{
            Toast.makeText( context: SignUpActivity.this, text: "User already exists! please sign in", Toast.LENGTH_SHORT).show();
        }
    }else{
        Toast.makeText( context: SignUpActivity.this, text: "Passwords not matching", Toast.LENGTH_SHORT).show();
    }
} }
```

We also have the functions of Login

```
username = (EditText) findViewById(R.id.username_txt);
password = (EditText) findViewById(R.id.password_txt);
btnlogin = (Button) findViewById(R.id.enter_log_in);
DB = new DBHelper( context: this);
btnlogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String user = username.getText().toString();
        String pass = password.getText().toString();
        if(user.equals("")||pass.equals(""))
            Toast.makeText( context: LoginActivity.this,  text: "Please enter all the fields", Toast.LENGTH_SHORT).show();
        else{
            Boolean checkuserpass = DB.checkusernamepassword(user,pass);
            if(checkuserpass==true){
                Toast.makeText( context: LoginActivity.this,  text: "Sign in successfully", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(getApplicationContext(), MainPageActivity.class);
                startActivity(intent);
            }else{
                Toast.makeText( context: LoginActivity.this,  text: "InvalidCredentials", Toast.LENGTH_SHORT).show();
            }
        }
    }
});
```

We use SharedPreferences to save the data when we change to another layout or activity

```
// Set the ID
Save = (Button) findViewById(R.id.save_total_breakfast);
text_energy = (EditText) findViewById(R.id.edittext_energy_breakfast);
text_fat = (EditText) findViewById(R.id.edittext_fat_breakfast);
text_carbohydrate = (EditText) findViewById(R.id.edittext_carbohydrate_breakfast);
text_protein = (EditText) findViewById(R.id.edittext_protein_breakfast);
text_vitamin = (EditText) findViewById(R.id.edittext_vitamin_breakfast);
```

```
// to Retrieve the Data from Sharepreferences
SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences( context: this);
String st1 = pref.getString( s: "st1breakfast",  s1: "");
text_energy.setText(st1);
String st2 = pref.getString( s: "st2breakfast",  s1: "");
text_fat.setText(st2);
String st3 = pref.getString( s: "st3breakfast",  s1: "");
text_carbohydrate.setText(st3);
String st4 = pref.getString( s: "st4breakfast",  s1: "");
text_protein.setText(st4);
String st5 = pref.getString( s: "st5breakfast",  s1: "");
text_vitamin.setText(st5);
```

```
// make function for button
Save.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        st_1 = text_energy.getText().toString();
        st_2 = text_fat.getText().toString();
        st_3 = text_carbohydrate.getText().toString();
        st_4 = text_protein.getText().toString();
        st_5 = text_vitamin.getText().toString();

        // To save the data
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences( context: TotalBreakfastActivity.this);
        SharedPreferences.Editor editor = prefs.edit();

        editor.putString( s: "st1breakfast", st_1);
        editor.putString( s: "st2breakfast", st_2);
        editor.putString( s: "st3breakfast", st_3);
        editor.putString( s: "st4breakfast", st_4);
        editor.putString( s: "st5breakfast", st_5);
        editor.apply();
        Toast.makeText( context: TotalBreakfastActivity.this,  text: "Data Saved Successfully", Toast.LENGTH_SHORT).show();
    }
});
```

To make the Menu on the right corner of screen activate, we have the syntax

```
// create and get ID for the menu
menuBuilder = new MenuBuilder( context: this);
MenuInflater inflater = new MenuInflater( context: this);
inflater.inflate(R.menu.menu_in_total, menuBuilder);
menu_show = (Button) findViewById(R.id.menu_in_total);
```

```
// function of button menu
menu_show.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        MenuPopupHelper optionMenu = new MenuPopupHelper( context: TotalBreakfastActivity.this, menuBuilder, view);
        optionMenu.setForceShowIcon(true);
        // condition for each item in menu
        menuBuilder.setCallback(new MenuBuilder.Callback() {
            @Override
            public boolean onMenuItemSelected(@NonNull MenuBuilder menu, @NonNull MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.item1_in_total:
                        startActivity(new Intent( packageContext: TotalBreakfastActivity.this, BreakfastListActivity.class));
                        return true;
                    case R.id.item2_in_total:
                        startActivity(new Intent( packageContext: TotalBreakfastActivity.this, CalculatorBreakfastActivity.class));
                        return true;
                    case R.id.item3_in_total:
                        startActivity(new Intent( packageContext: TotalBreakfastActivity.this, BreakfastActivity.class));
                        return true;
                    default:
                        return false;
                }
            }
            @Override
            public void onMenuModeChange(@NonNull MenuBuilder menu) {
            }
        });
        optionMenu.show(); // display menu
    }
});
```

To make the Calculator, we have the syntax

```
// get and assign the ID in onCreate
resultTv = findViewById(R.id.result_tv);
solutionTv = findViewById(R.id.solution_tv);

assignId(buttonC,R.id.button_c);
assignId(buttonBrackOpen,R.id.button_open_bracket);
assignId(buttonBrackClose,R.id.button_close_bracket);
assignId(buttonDivide,R.id.button_divide);
assignId(buttonMultiply,R.id.button_multiply);
assignId(buttonPlus,R.id.button_plus);
assignId(buttonMinus,R.id.button_minus);
assignId(buttonEquals,R.id.button_equals);
assignId(button0,R.id.button_0);
assignId(button1,R.id.button_1);
assignId(button2,R.id.button_2);
assignId(button3,R.id.button_3);
assignId(button4,R.id.button_4);
assignId(button5,R.id.button_5);
assignId(button6,R.id.button_6);
assignId(button7,R.id.button_7);
assignId(button8,R.id.button_8);
assignId(button9,R.id.button_9);
assignId(buttonAC,R.id.button_ac);
assignId(buttonDot,R.id.button_dot);
```

```
// function of setOnClick for all instead of setting one by one
void assignId(Button btn,int id){
    btn = findViewById(id);
    btn.setOnClickListener(this);
}
```

```java
// solutionTv to input, resultTv to output
// function input output
@Override
public void onClick(View view) {
    Button button =(Button) view;
    String buttonText = button.getText().toString();
    String dataToCalculate = solutionTv.getText().toString();

    if(buttonText.equals("AC")){
        solutionTv.setText("");
        resultTv.setText("0");
        return;
    }
    if(buttonText.equals("=")){
        solutionTv.setText(resultTv.getText());
        return;
    }
    if(buttonText.equals("C")){
        dataToCalculate = dataToCalculate.substring(0,dataToCalculate.length()-1);
    }else{
        dataToCalculate = dataToCalculate+buttonText;
    }
    solutionTv.setText(dataToCalculate);

    String finalResult = getResult(dataToCalculate);

    if(!finalResult.equals("Err")){
        resultTv.setText(finalResult);
    }

}
```

```java
    // function get result
    String getResult(String data){
        try{
            Context context  = Context.enter();
            context.setOptimizationLevel(-1);
            Scriptable scriptable = context.initStandardObjects();
            String finalResult =  context.evaluateString(scriptable,data, sourceName: "Javascript", lineno: 1, securityDomain: null).toString();
            if(finalResult.endsWith(".0")){
                finalResult = finalResult.replace( target: ".0", replacement: "");
            }
            return finalResult;
        }catch (Exception e){
            return "Err";
        }
    }

}
```

To make the button having sound, firstly, we save the file .mp3 in the raw, then we have the syntax

```
// get the ID
final MediaPlayer dragon = MediaPlayer.create( context: this,R.raw.snoringsound);
Button press_dragon_tight = (Button) this.findViewById(R.id.signUp);

final MediaPlayer siu = MediaPlayer.create( context: this,R.raw.siu);
Button press_siu = (Button) this.findViewById(R.id.login);

// function to make the button having sound
press_siu.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        siu.start();
        startActivity(new Intent( packageContext: MainActivity.this, LoginActivity.class));
    }
});
```

To make the list of musics, firstly, we create the file .java BackgroundSoundService with the syntax

```
MediaPlayer mediaPlayer; // implement Mediaplayer

@Nullable
@Override
public IBinder onBind(Intent intent) { return null; } // get intent

// get the music
@Override
public void onCreate(){
    super.onCreate();
    mediaPlayer = MediaPlayer.create( context: this,R.raw.backgroundmusic);
    mediaPlayer.setLooping(true);
    mediaPlayer.setVolume( leftVolume: 100, rightVolume: 100);
}
```

```
// start and pause
public int onStartCommand(Intent intent, int flags, int startId) {
    if (!mediaPlayer.isPlaying()){
        mediaPlayer.start();
        Toast.makeText( context: this,  text: "Now Playing: Background music", Toast.LENGTH_SHORT).show();
    }else{
        mediaPlayer.pause();
        Toast.makeText( context: this,  text: "Paused: Background music", Toast.LENGTH_SHORT).show();
    }
    return START_STICKY;
}
```

```java
// stop
@Override
public void onDestroy() {
    super.onDestroy();
    mediaPlayer.stop();
    Toast.makeText( context: this,  text: "Stopped Playing: Background music", Toast.LENGTH_SHORT).show();
    mediaPlayer.release();
    mediaPlayer = MediaPlayer.create( context: this, R.raw.backgroundmusic);
}
```

Then, in the activity displaying the layout having music, we have the syntax

```java
// function to create music in this layout or activity
public void PlaySound1(View view){
    Intent intent = new Intent( packageContext: BackgroundMusicActivity.this, BackgroundSoundService.class);
    startService(intent);
}
```

Then, to listen to music, we code in AndroidManifest.xml

```xml
<service android:name=".BackgroundSoundService" />
```

In particular meal, we have some functions: search item in the list, add item to the list, see the list and delete item from the list.

Firstly, we create file Food.java to save the information of the food and this is a place to call from to display.

```java
package vn.edu.usth.nutrition_recipe;

import java.io.Serializable;

public class Food implements Serializable{
    // call attribute
    private String mName;
    private Integer mCalo;
    private Integer mFat;
    private Integer mCarbohydrate;
    private Integer mVitamin;
    private Integer mProtein;

    // return attribute
    public Food(String mName, Integer mCalo, Integer mFat, Integer mCarbohydrate, Integer mProtein, Integer mVitamin) {
        this.mName = mName;
        this.mCalo = mCalo;
        this.mFat = mFat;
        this.mCarbohydrate = mCarbohydrate;
        this.mVitamin = mVitamin;
        this.mProtein = mProtein;
    }
    public String getmName() { return mName; }

    public Integer getmCalo() { return mCalo;}

    public Integer getmFat() { return mFat; }

    public Integer getmCarbohydrate() { return mCarbohydrate;}

    public Integer getmVitamin() { return mVitamin; }

    public Integer getmProtein() { return mProtein;}
}
```

Then, we create file FoodAdapter.java to connect to file .xml to create each item

```
    // call the attribute
    TextView currentFoodName,
            btn_album,
            currentCalo,
            currentFat,
            currentCarbohydrate,
            currentProtein,
            currentVitamin,
            addItem;

    // get the attribute
    public FoodViewHolder(View itemView) {
        super(itemView);
        currentFoodName = (TextView) itemView.findViewById(R.id.selected_food_name);
        currentCalo = (TextView) itemView.findViewById(R.id.selected_food_amount);
        currentFat = (TextView) itemView.findViewById(R.id.selected_food_amount_fat);
        currentCarbohydrate = (TextView) itemView.findViewById(R.id.selected_food_amount_carbohydrate);
        currentProtein = (TextView) itemView.findViewById(R.id.selected_food_amount_protein);
        currentVitamin = (TextView) itemView.findViewById(R.id.selected_food_amount_vitamin);
        addItem = (TextView) itemView.findViewById(R.id.add_item);
        btn_album = (TextView) itemView.findViewById(R.id.food_album);
    }
}
```

```
// function of the list
public FoodAdapter(List<Food> myOrders, IClickItemUserListener listener, IClickItemUserListener2 listener2) {
    this.list = myOrders;
    this.iClickItemUserListener = listener;
    this.iClickItemUserListener2 = listener2;
}
// display each item in the list
@NonNull
@Override
public FoodViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.activity_food_adapter, parent, attachToRoot: false);
    FoodViewHolder fvh = new FoodViewHolder(v);
    return fvh;
}
```

```java
// get the data and put it to each item to the list
@Override
public void onBindViewHolder(@NonNull FoodViewHolder holder, int position) {
    final Food currentFood = list.get(position);
    holder.currentFoodName.setText(currentFood.getmName());
    holder.currentCalo.setText(currentFood.getmCalo() + " Calo, ");
    holder.currentFat.setText(currentFood.getmFat() + " g Fat, ");
    holder.currentCarbohydrate.setText(currentFood.getmCarbohydrate() + " g Carbohydrate, ");
    holder.currentProtein.setText(currentFood.getmProtein() + " g Protein, ");
    holder.currentVitamin.setText(currentFood.getmVitamin() + " g Vitamin");

    holder.btn_album.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            iClickItemUserListener2.onClickItemUser2(currentFood);
        }
    });

    holder.addItem.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { iClickItemUserListener.onClickItemUser(currentFood);
    });
}

// "lọc" item in SearchView
@Override
public int getItemCount() { return list.size(); }

public void filterList(ArrayList<Food> filteredList) {
    list = filteredList;
    notifyDataSetChanged();
}
```

Then, we code in the main file .java, this is the syntax of SearchView
Firstly, we create functions outside onCreate

```java
// "lọc" item in the SearchView
private void filter(String text) {
    ArrayList<Food> filteredList = new ArrayList<>();

    for (Food item : orders) {
        if (item.getmName().toLowerCase().contains(text.toLowerCase())) {
            filteredList.add(item);
        }
    }

    foodAdapter.filterList(filteredList);
}
```

```java
// function to create the list
private void createExampleList() {
    orders = new ArrayList<>();
    orders.add(new Food( mName: "10 g Rice",  mCalo: 30,  mFat: 1,  mCarbohydrate: 5,  mProtein: 1,  mVitamin: 3));
    orders.add(new Food( mName: "15 g Beans",  mCalo: 40,  mFat: 3,  mCarbohydrate: 3,  mProtein: 2,  mVitamin: 7));
    orders.add(new Food( mName: "12 g Bread",  mCalo: 35,  mFat: 1,  mCarbohydrate: 6,  mProtein: 1,  mVitamin: 4));
    orders.add(new Food( mName: "14 g Meat",  mCalo: 50,  mFat: 2,  mCarbohydrate: 1,  mProtein: 8,  mVitamin: 3));
    orders.add(new Food( mName: "20 g Salad",  mCalo: 32,  mFat: 1,  mCarbohydrate: 6,  mProtein: 1,  mVitamin: 12));
}
// support to display the list
private void buildRecyclerView() {
    recyclerView = findViewById(R.id.list_item);
    recyclerView.setHasFixedSize(true);
    layoutManager = new LinearLayoutManager( context: this);
    foodAdapter = new FoodAdapter(orders, new IClickItemUserListener() {
        @Override
        public void onClickItemUser(Food currentItem) { onClickGoToDetail(currentItem); }
    }, new IClickItemUserListener2() {
        @Override
        public void onClickItemUser2(Food currentItem) { onClickGoToDetail2(currentItem); }
    });

    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setAdapter(foodAdapter);
}
```

```java
// create function for each button in each item in the list
private void onClickGoToDetail(Food currentItem) {
    Intent intent = new Intent( packageContext: BreakfastActivity.this, BreakfastListActivity.class);
    Bundle bundle = new Bundle();
    // add data to bundle
    bundle.putSerializable("data1breakfast", currentItem);
    bundle.putSerializable("data2breakfast", currentItem);
    bundle.putSerializable("data3breakfast", currentItem);
    bundle.putSerializable("data4breakfast", currentItem);
    bundle.putSerializable("data5breakfast", currentItem);
    bundle.putSerializable("data6breakfast", currentItem);
    intent.putExtras(bundle);
    startActivity(intent);
    Toast.makeText( context: this, text: "New Food Added", Toast.LENGTH_SHORT).show();
}
private void onClickGoToDetail2(Food currentItem) {
    startActivity(new Intent( packageContext: BreakfastActivity.this, AlbumActivity.class));
}
```

Then, we call the functions inside onCreate

```
// call the functions of SearchView
createExampleList();
buildRecyclerView();

EditText editText = findViewById(R.id.search_breakfast);

editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {

    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {

    }

    @Override
    public void afterTextChanged(Editable s) { filter(s.toString()); }
});
```

To support displaying the list of Food added, we have the syntax in file ViewAdapter.java

```
// put in the ArrayList
public BreakfastViewAdapter(Context context, ArrayList foodname_id, ArrayList calo_id, ArrayList fat_id, ArrayList carbohydrate_
    this.context = context;
    this.foodname_id = foodname_id;
    this.calo_id = calo_id;
    this.fat_id = fat_id;
    this.carbohydrate_id = carbohydrate_id;
    this.protein_id = protein_id;
    this.vitamin_id = vitamin_id;
}
// Display the list
@NonNull
@Override
public BreakfastViewAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(context).inflate(R.layout.breakfastentry,parent, attachToRoot: false);
    return new MyViewHolder(v);
}
// get the data and put it to each item to the list
@Override
public void onBindViewHolder(@NonNull BreakfastViewAdapter.MyViewHolder holder, int position) {
    holder.foodname_id.setText(String.valueOf(foodname_id.get(position)));
    holder.calo_id.setText(String.valueOf(calo_id.get(position)));
    holder.fat_id.setText(String.valueOf(fat_id.get(position)));
    holder.carbohydrate_id.setText(String.valueOf(carbohydrate_id.get(position)));
    holder.protein_id.setText(String.valueOf(protein_id.get(position)));
    holder.vitamin_id.setText(String.valueOf(vitamin_id.get(position)));
```

```java
// get the attribute
@Override
public int getItemCount() { return foodname_id.size(); }
public class MyViewHolder extends RecyclerView.ViewHolder {
    TextView foodname_id, calo_id, fat_id, carbohydrate_id, protein_id, vitamin_id;
    public MyViewHolder(@NonNull View itemView) {
        super(itemView);
        foodname_id = itemView.findViewById(R.id.text_breakfast_foodname);
        calo_id = itemView.findViewById(R.id.text_breakfast_calo);
        fat_id = itemView.findViewById(R.id.breakfast_text_fat_gram);
        carbohydrate_id = itemView.findViewById(R.id.breakfast_text_carbohydrate_gram);
        protein_id = itemView.findViewById(R.id.breakfast_text_protein_gram);
        vitamin_id = itemView.findViewById(R.id.breakfast_text_vitamin_gram);
    }
}
```

Then, we code in the main file .java, this is the syntax of List Food. Here, we also have the function of adding and deleting food.

```java
// load data from preferences
loadData();
// get ID and attribute
recyclerView = findViewById(R.id.view_breakfast);
adapter = new BreakfastViewAdapter( context: this, foodname, calo, fat, carbohydrate, protein, vitamin);
recyclerView.setAdapter(adapter);
recyclerView.setLayoutManager(new LinearLayoutManager( context: this)); // display list


Bundle bundle = getIntent().getExtras(); // get data from other activity through bundle

if (bundle == null) {
    return;
}
// call the attribute and put data to the attribute
Food food1 = (Food) bundle.get("data1breakfast");
Food food2 = (Food) bundle.get("data2breakfast");
Food food3 = (Food) bundle.get("data3breakfast");
Food food4 = (Food) bundle.get("data4breakfast");
Food food5 = (Food) bundle.get("data5breakfast");
Food food6 = (Food) bundle.get("data6breakfast");
// add data to the list
foodname.add(food1.getmName());
calo.add(food2.getmCalo());
fat.add(food3.getmFat());
carbohydrate.add(food4.getmCarbohydrate());
protein.add(food5.getmProtein());
vitamin.add(food6.getmVitamin());
saveData();
```

```java
// delete Food
private void deleteBreakfast() {
    SharedPreferences sharedPreferences = getSharedPreferences( name: "shared preferences breakfast", MODE_PRIVATE);
    sharedPreferences.edit().clear().commit();
}
// save the list by using preferences
private void saveData() {
    SharedPreferences sharedPreferences = getSharedPreferences( name: "shared preferences breakfast", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    String json1 = gson.toJson(foodname);
    String json2 = gson.toJson(calo);
    String json3 = gson.toJson(fat);
    String json4 = gson.toJson(carbohydrate);
    String json5 = gson.toJson(protein);
    String json6 = gson.toJson(vitamin);
    editor.putString( s: "task list 1 breakfast", json1);
    editor.putString( s: "task list 2 breakfast", json2);
    editor.putString( s: "task list 3 breakfast", json3);
    editor.putString( s: "task list 4 breakfast", json4);
    editor.putString( s: "task list 5 breakfast", json5);
    editor.putString( s: "task list 6 breakfast", json6);
    editor.apply();
}
```

```java
// load data from preferences
private void loadData() {
    SharedPreferences sharedPreferences = getSharedPreferences( name: "shared preferences breakfast", MODE_PRIVATE);
    Gson gson = new Gson();
    String json1 = sharedPreferences.getString( s: "task list 1 breakfast",  s1: null);
    String json2 = sharedPreferences.getString( s: "task list 2 breakfast",  s1: null);
    String json3 = sharedPreferences.getString( s: "task list 3 breakfast",  s1: null);
    String json4 = sharedPreferences.getString( s: "task list 4 breakfast",  s1: null);
    String json5 = sharedPreferences.getString( s: "task list 5 breakfast",  s1: null);
    String json6 = sharedPreferences.getString( s: "task list 6 breakfast",  s1: null);
    Type type1 = new TypeToken<ArrayList<String>>() {
    }.getType();
    Type type2 = new TypeToken<ArrayList<String>>() {
    }.getType();
    Type type3 = new TypeToken<ArrayList<String>>() {
    }.getType();
    Type type4 = new TypeToken<ArrayList<String>>() {
    }.getType();
    Type type5 = new TypeToken<ArrayList<String>>() {
    }.getType();
    Type type6 = new TypeToken<ArrayList<String>>() {
    }.getType();
    foodname = gson.fromJson(json1, type1);
    calo = gson.fromJson(json2, type2);
    fat = gson.fromJson(json3, type3);
    carbohydrate = gson.fromJson(json4, type4);
    protein = gson.fromJson(json5, type5);
    vitamin = gson.fromJson(json6, type6);
    // if null => display nothing
    if (foodname == null) {
        foodname = new ArrayList<>();
    }
```

## V. RESULT AND DISCUSSION

*1. Result*

After some couples of months of development and upgradation, our nutrition recipe application has basically completed. We have almost achieved our goals we have achieved our goals we have proposed at first. To be more specific, the application has been designed with the following functions

- **Authentication**: **Register** & **Login**:
    + Be able to allow users to create an account and access into the system.
    + Be able to allow users to access based on the account that has been provided before.
- **Particular Meal**: Be able to allow users to search Food, add Food to the list, see list Food and use calculator to calculate the quantity of energies and nutrients.
- **Menu**: Be able to allow users to interact with Settings, Helps, Background Music, Calculator and Log Out:
    + Be able to allow users to know more about About Us, Contact, Credits.
    + Be able to send whatever Feedbacks users want to write.
    + Be able to delete users' own account
    + Be able to know more about Debug logs, FAQs, Term of Service.
    + Be able to use calculator and listen to music.
    + Be able to log out of the account used before.

Moreover, we succeed in making an application that allows users to build their nutrition recipe on their mobile phones.

*2. Discussion*

When we begin to do the project, we move up step by step slowly. Our aims or objectives are always the desire to succeed in function of building the nutrition recipe on the mobile phones.

First of all, we have some problems with beginning using Android Studio. We have not known about the Android Studio before, and we spent a period of time on getting used to it. We also spent a period of time on downloading and setting the app in our computer. Android Studio takes a lot of space on our device, too.

Secondly, the project of Mobile Application Development was happening at the same time with the Group Project. So, we have to do two projects parallelly. Hence, we have some difficulties in dividing the work for the members of our group. Some people have to focus on Group Project and just do a little work on Mobile Project and vice versa.

Thirdly, there are many files and many parts in our projects. Therefore, we usually have problematic issues or errors or bugs in our code. It costs us a lot of time to debug or fix errors, often just a little errors.

Fourthly, we can not connect and handle entirely with database yet. Until now, we are not able to put the data into the database and get it in other place. We spent a lot of time on the database but it is still impossible for us at the moment.

Fifthly, with the part of Register & Login, the Authentication is not entirely completed. We do not focus on the security, so the users' accounts may not be protected. We also have made the Admin User Interface in time yet. Without Admin User Interface, we may have a lot of difficulties in managing our application between many accounts.

Sixthly, we have some problems on networking and pie chart. Initially, we intend to put the piechart into our project. But we can not find the reasonable way to do it later. In addition, we have not found the method to put the data into pie chart in time yet. Futhermore, about networking, we do not really understand how it works and how it helps our project. Because of that, we decide not to put the networking into our project.

Lastly, we may have some other problematic matters or issues like too much gasoline to move, our computer running out of battery fast or broken,... or we may not work well as a real team or group, but we always try to overcome difficulties anytime, anywhere.

To conclude, while we are doing the project, some problems happen and we may have difficulties. They are invaluable experiences we have ever had and we try to do better next time.

## VI. CONCLUSION AND FUTURE WORK

*1. Conclusion*

To sum up, almost the purposes and objectives of our project known as Nutrition Recipe Mobile Application have been achieved. The user can build his/her own nutrition recipe. By providing a friendly, comfortable and simple user interface on the application, the interaction between the system and users can become more convenient. Managing data or databases could become more possible and effective. The main functions of the Nutrition Recipe Mobile Application have been implemented:

- Users can create their own accounts and log into the application.
- Users can build their own nutrition recipe by search Food, add Food to the list and see the list Food.
- Users can calculate the total nutrients and energies.
- Users can interact with the Settings, Helps, Calculators.
- Users can delete their own account if they want.
- Users can listen to the list of musics whereas using the application.

*2. Future Work*

To further improve this mobile application in the future, the following tasks need to be done:

- Fix all errors and debug all bugs.
- Improve and develop the application security.
- Make a Guideline for using the application.
- Improve and develop the performance or the user interface of the system to be more responsive and eye-catchingly.
- Make a function to automatically calculate the total calories and total grams in each nutrient and energy for each meal (connect to the database).
- Set each image and also video for each food instead of simple food album.
- Design Admin UI for managing and monitoring the mobile application.
- Set up firebase for user authentication so that the user has more choices for logging or signing up.
- Set up networking for our application.
- Make a pie chart to show the result after calculation.

If this application is forecast to be perfectly completed in the future, it is expected to be a good supporting application for those who want to have a suitable diet on the portable and convenient devices. From my point of view, I think our application could be used widely and become more ubiquitous as a cheap, convenient and efficient method

## VII. APPENDIX – USER INTERFACE


**Figure 10**. Initial User Interface

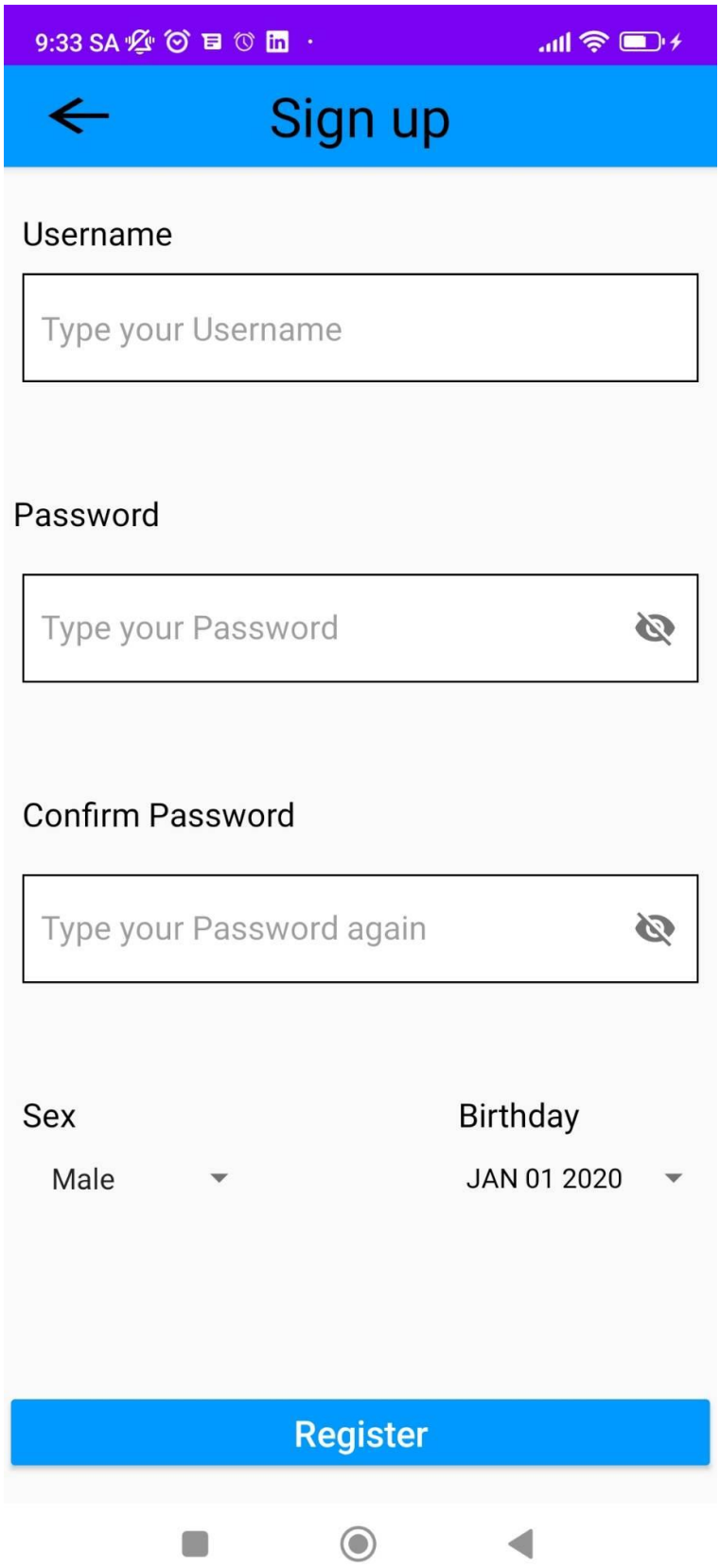


**Figure 11**. Register User Interface

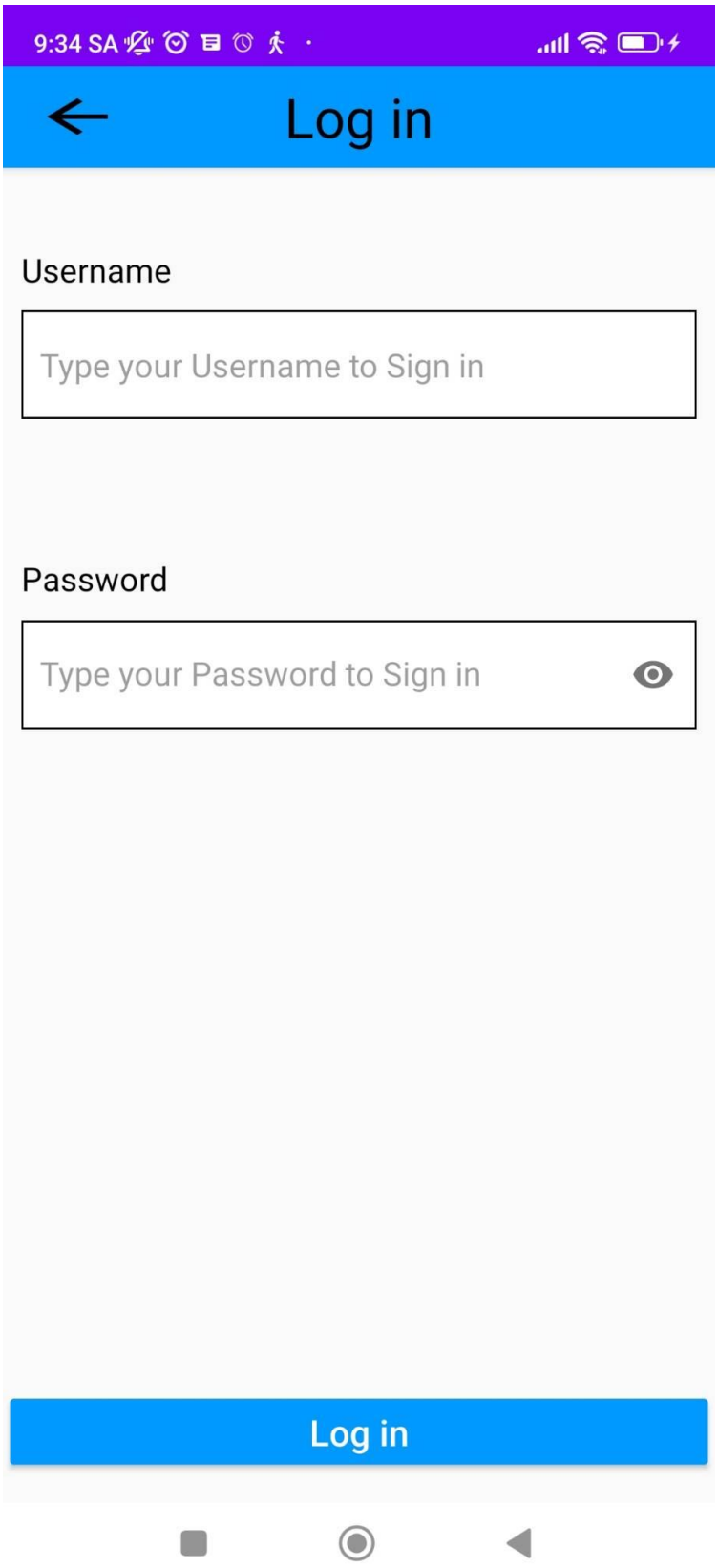


**Figure 12**. Login User Interface

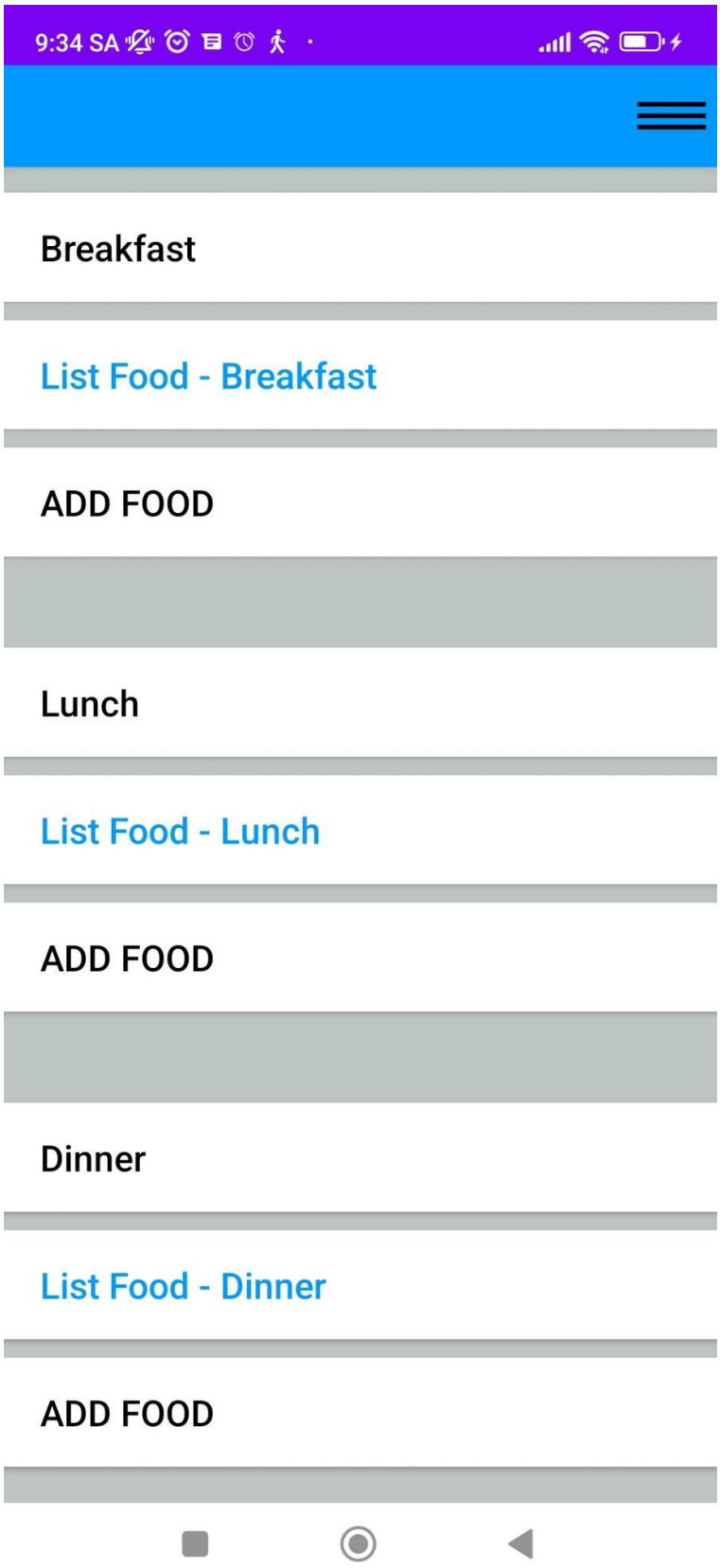


**Figure 13**. Main Page User Interface

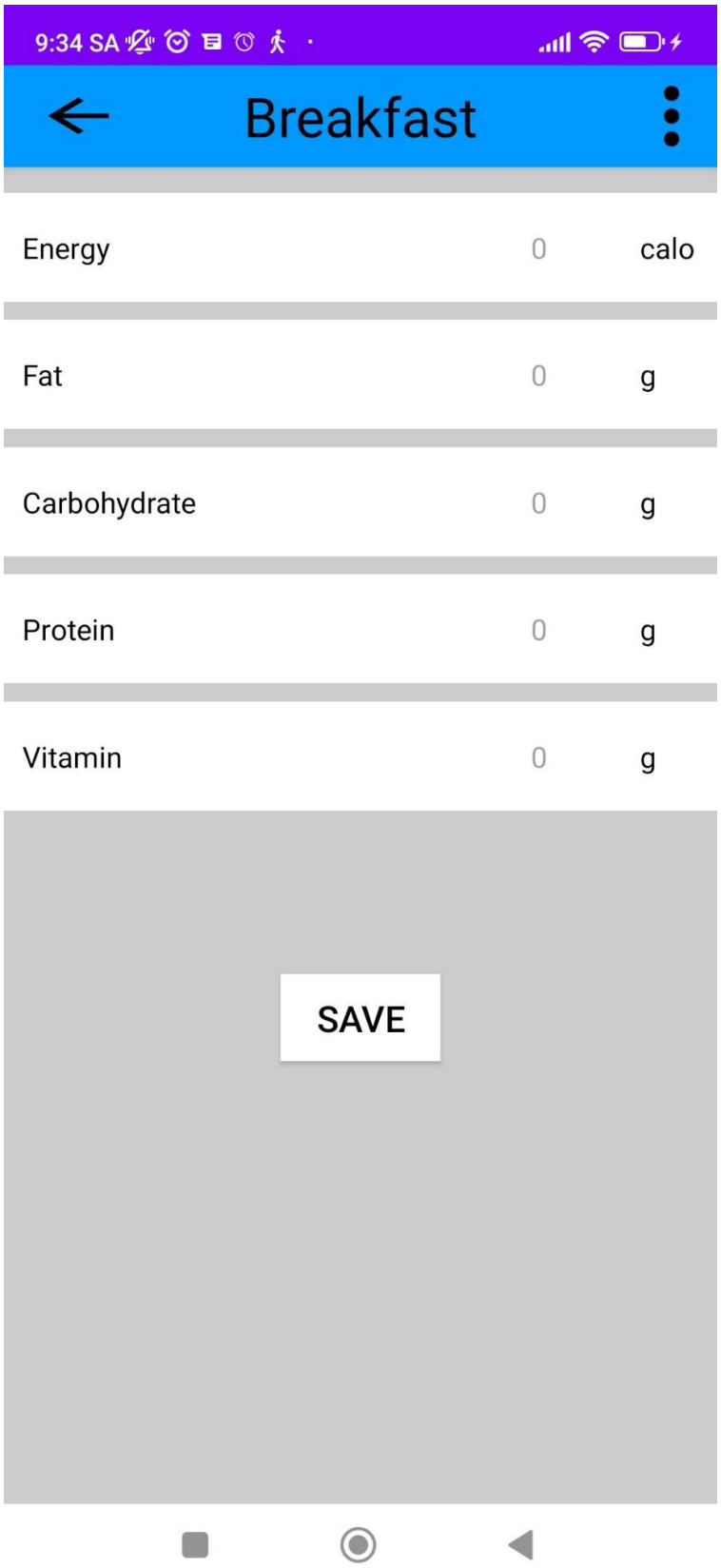


**Figure 14**. Total Energies & Nutrients User Interface

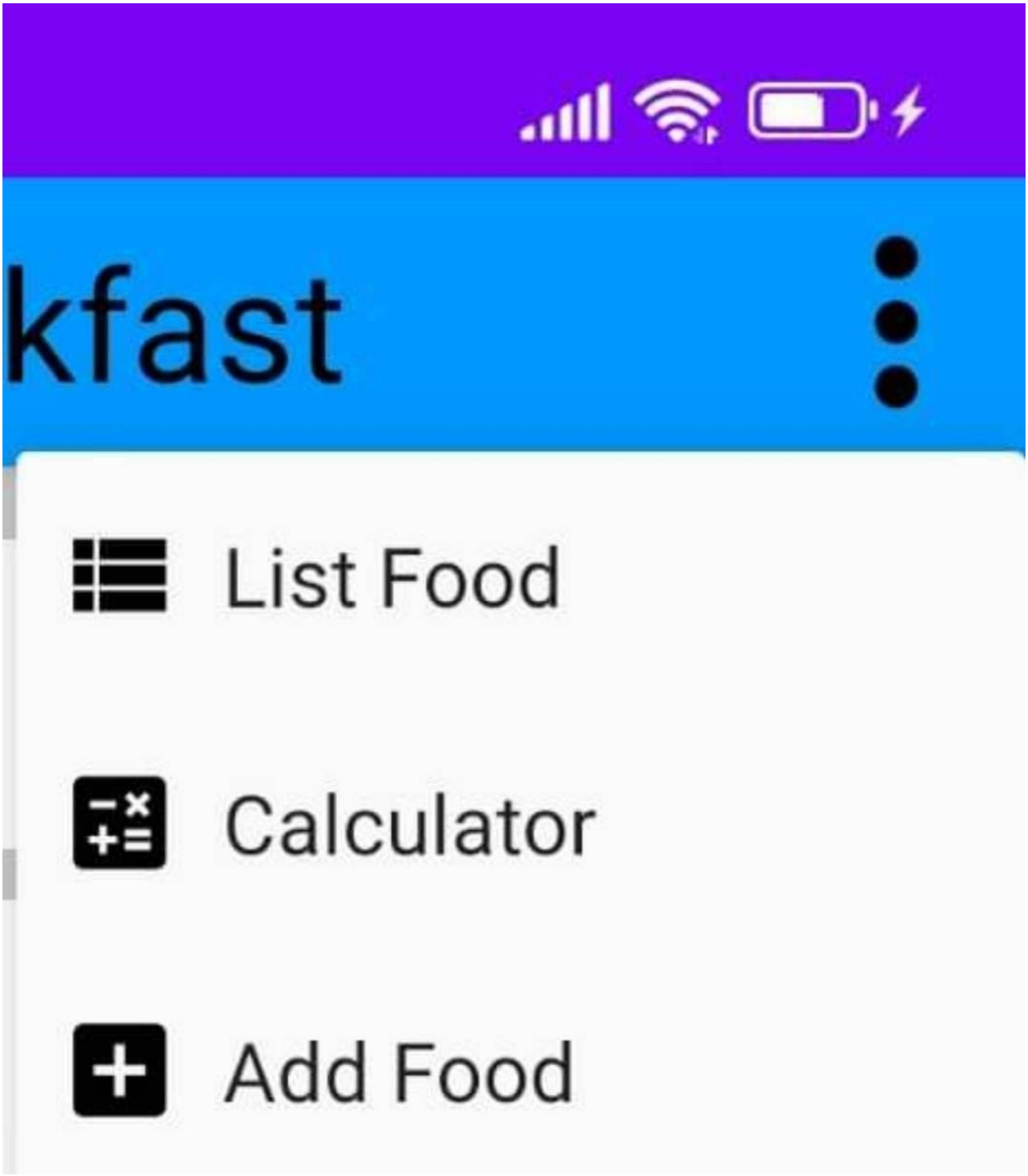


**Figure 15**. Menu – Top Right Corner User Interface

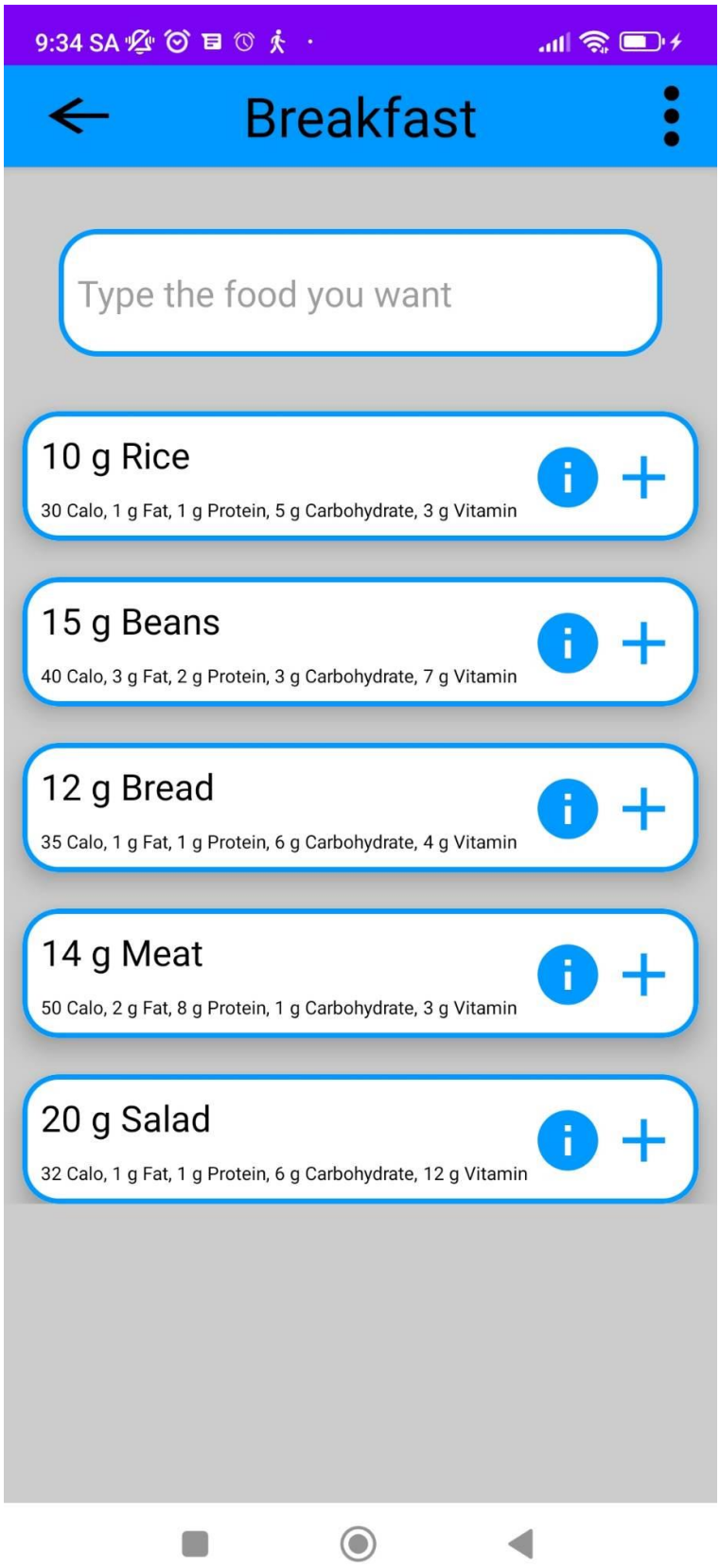


**Figure 16**. Add Food User Interface

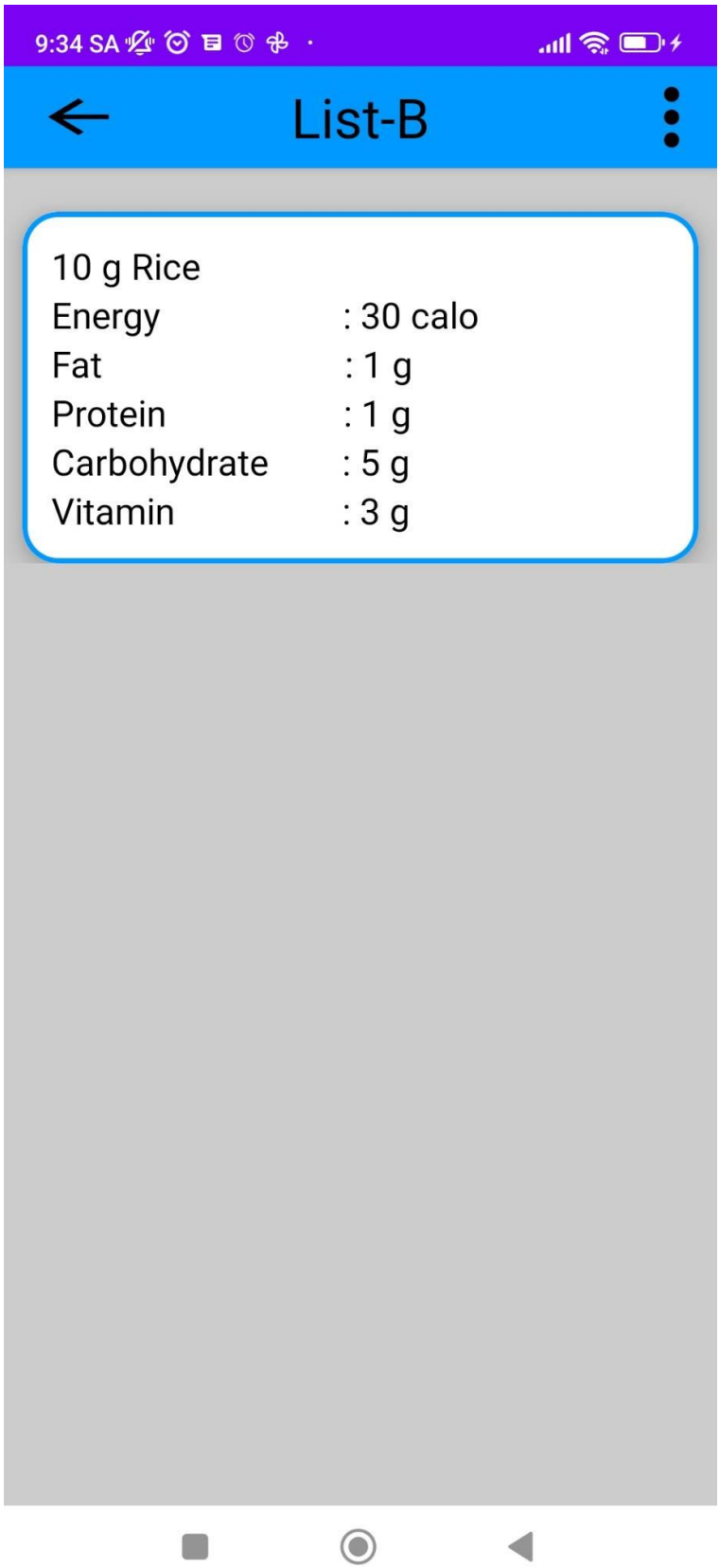


**Figure 17**. List Food User Interface

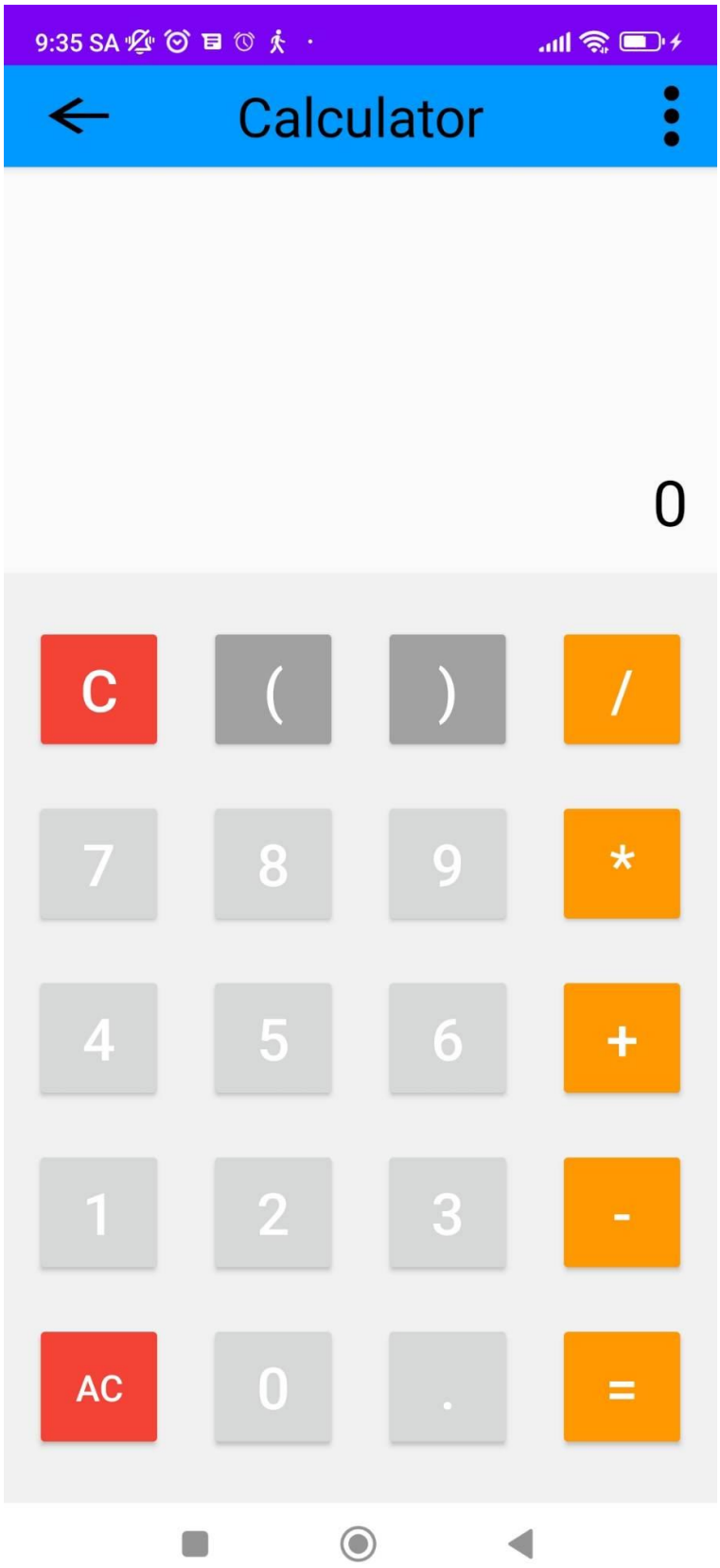


**Figure 18**. Calculator User Interface

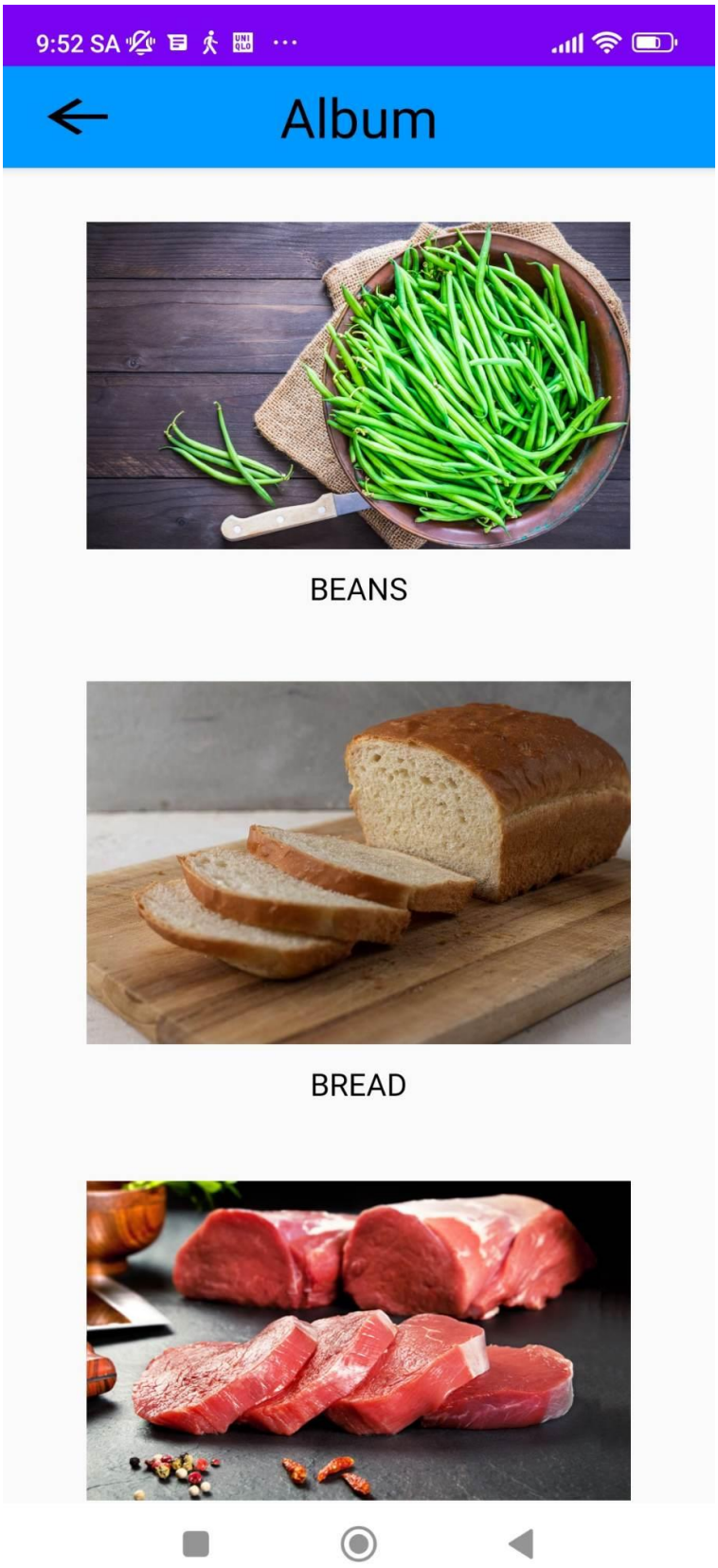


**Figure 19**. Album User Interface

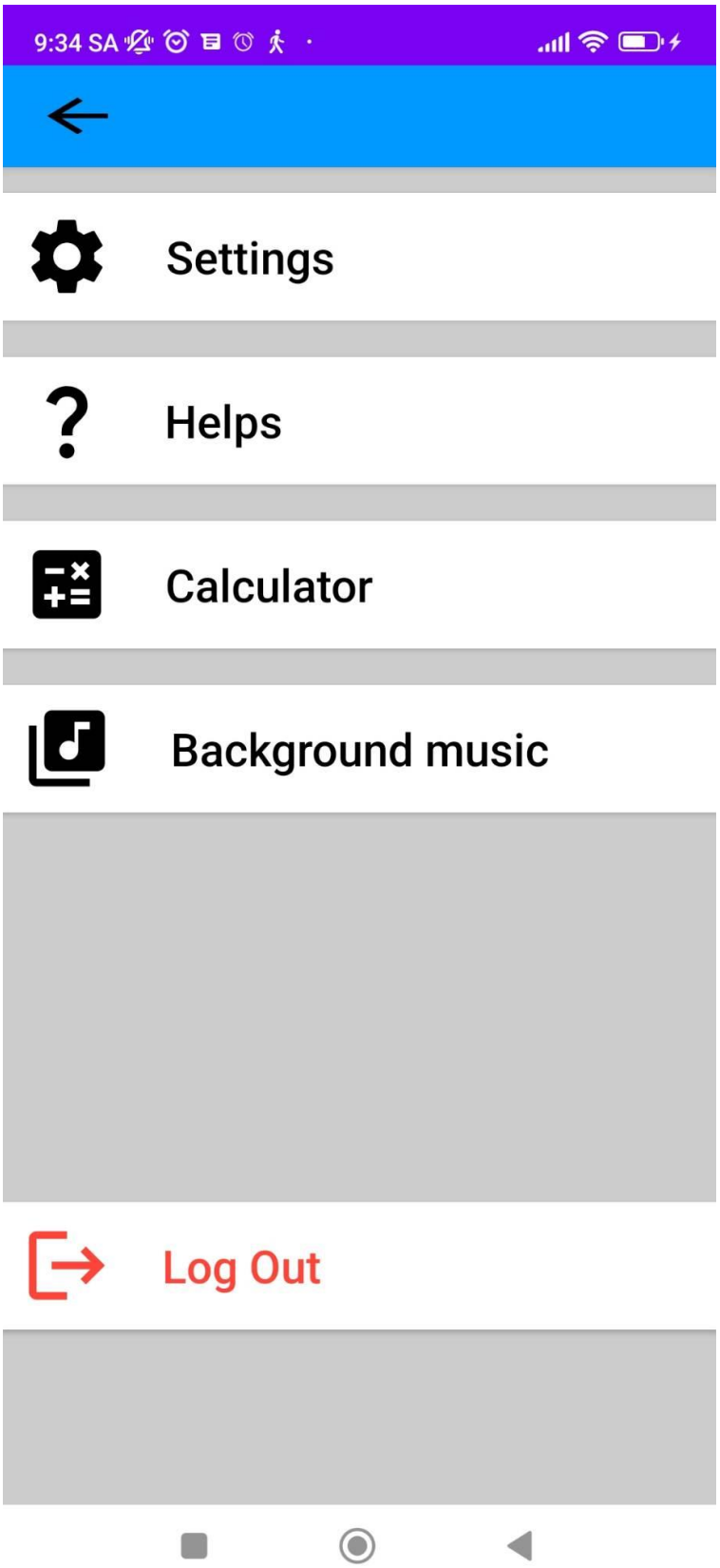


**Figure 20**. Menu Choice User Interface

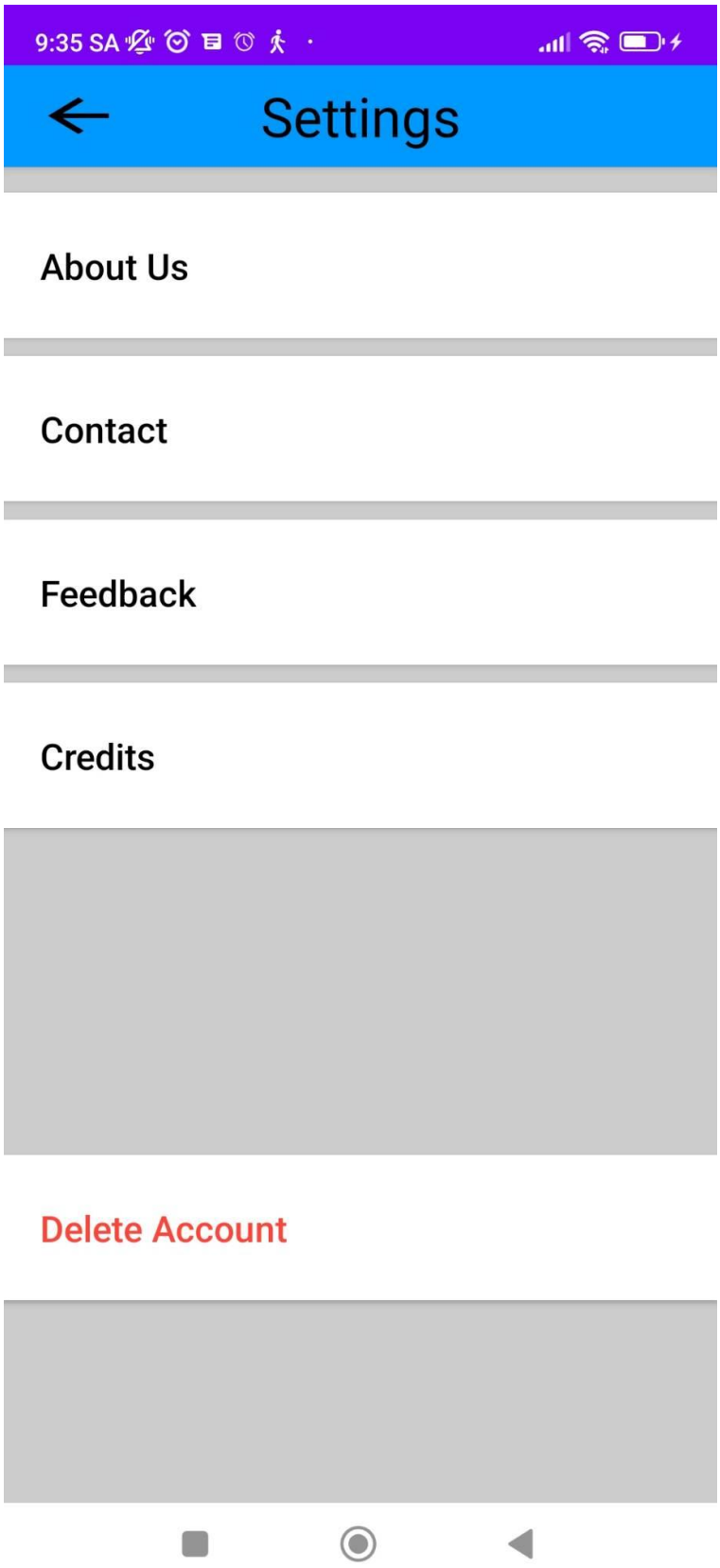


**Figure 21**. Settings User Interface


**Figure 22**. About Us User Interface

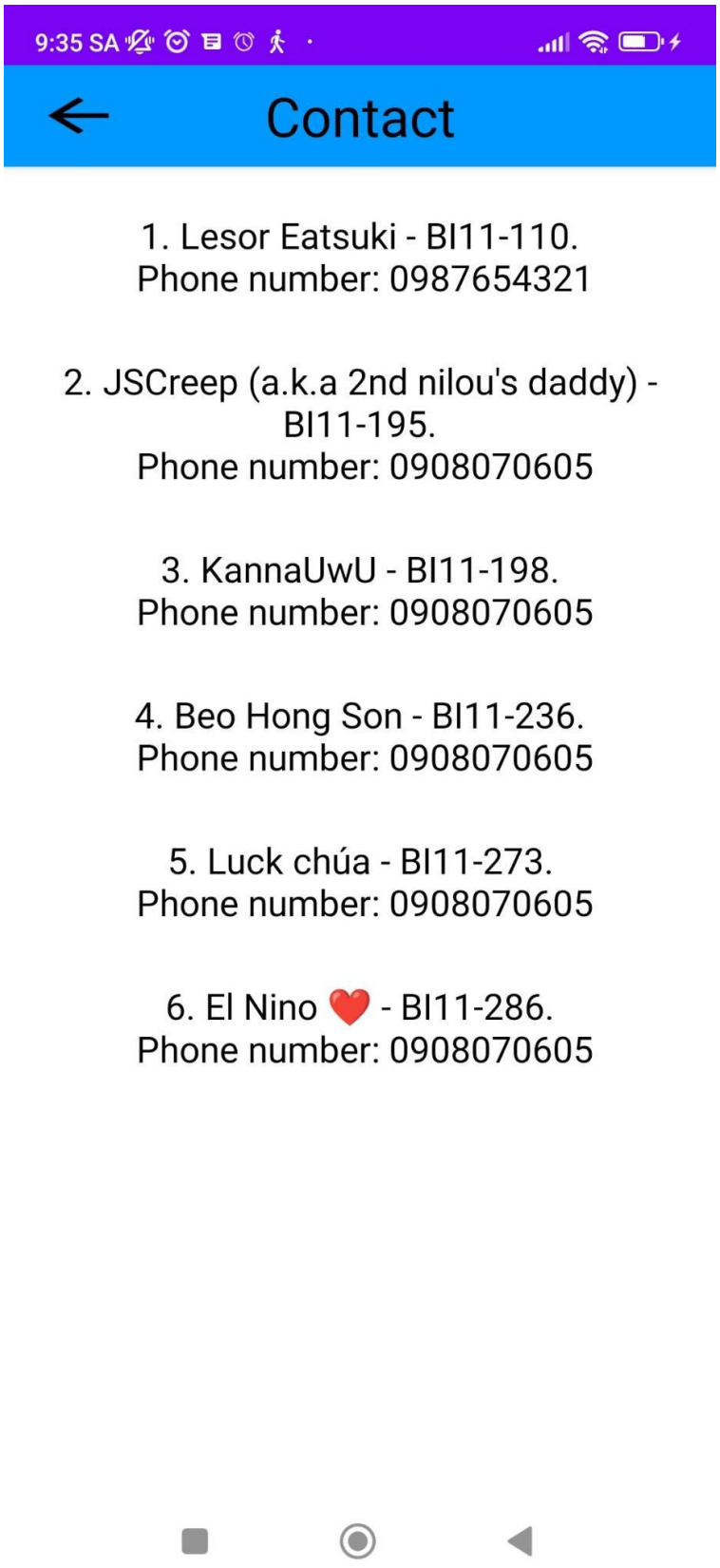


**Figure 23**. Contact User Interface


**Figure 24**. Feedback User Interface

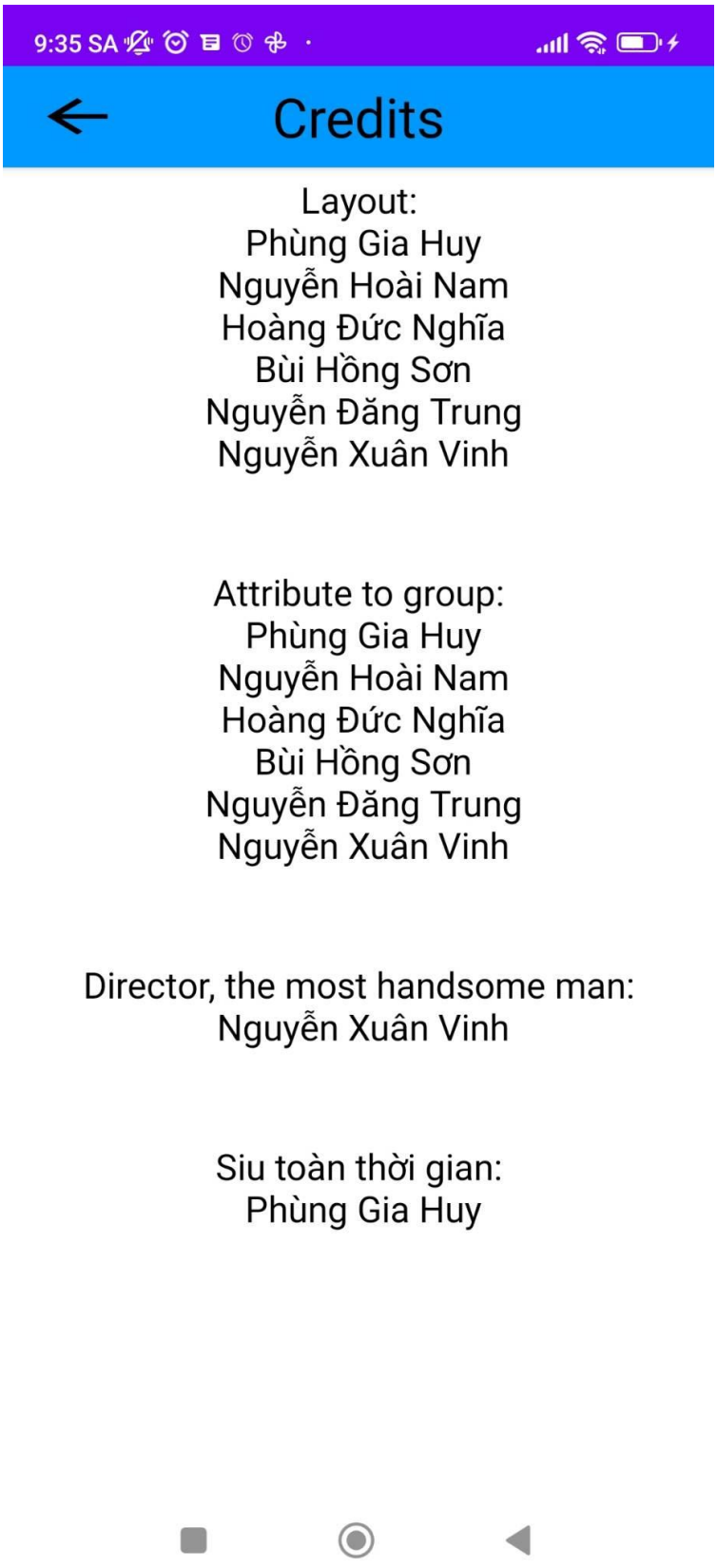


**Figure 25**. Credits User Interface

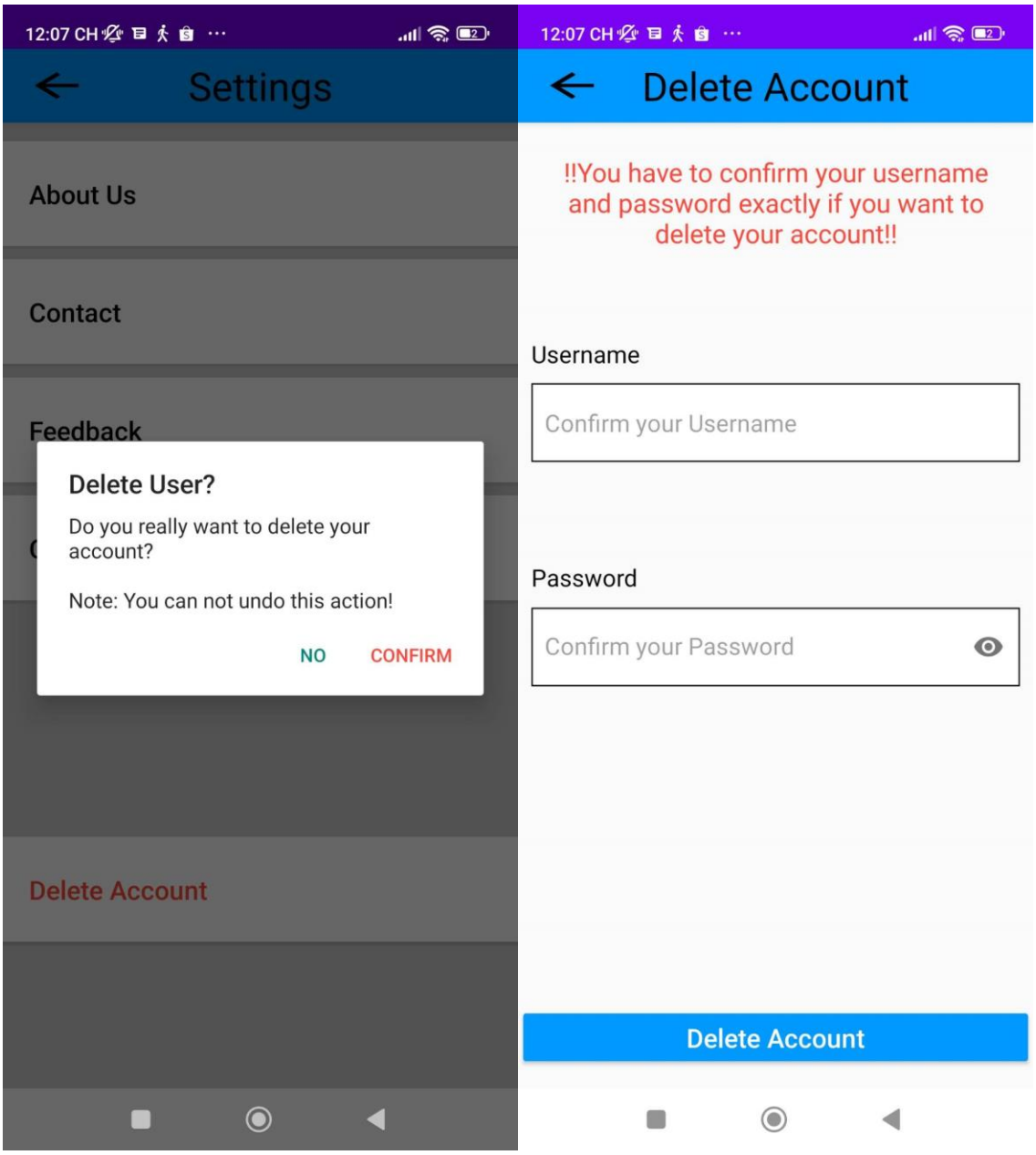


**Figure 26**. Delete Account User Interface

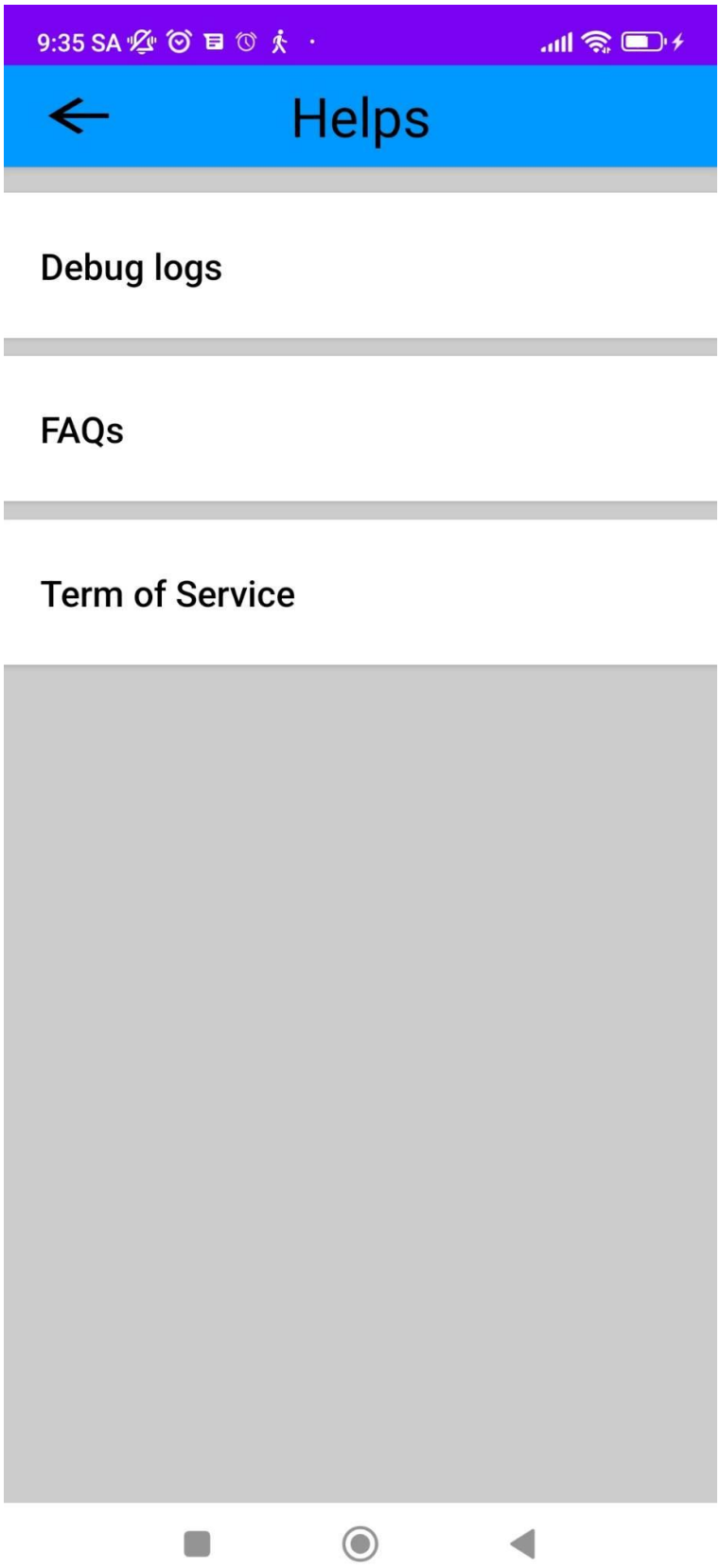


**Figure 27**. Helps User Interface

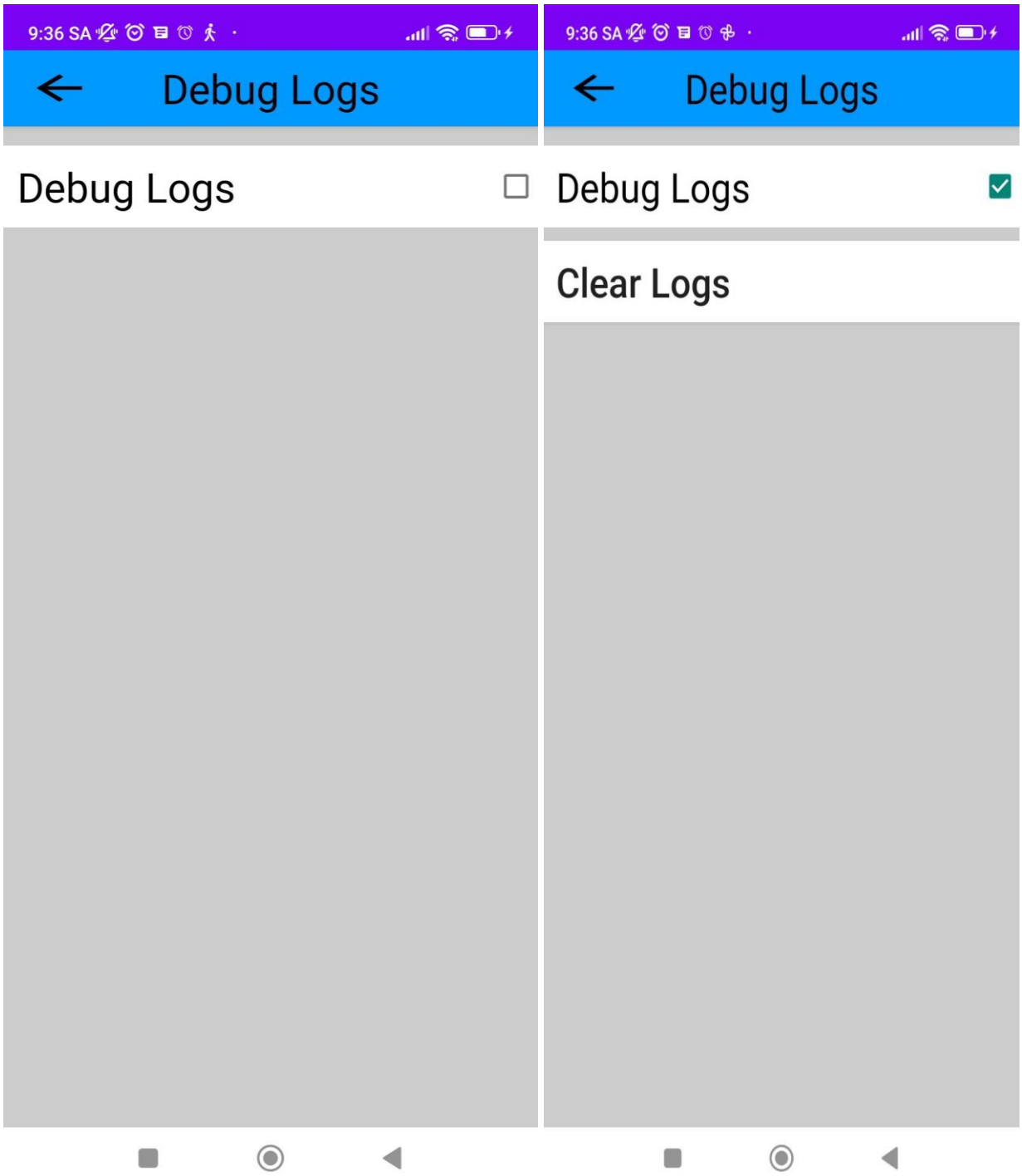


**Figure 28**. Debug logs User Interface

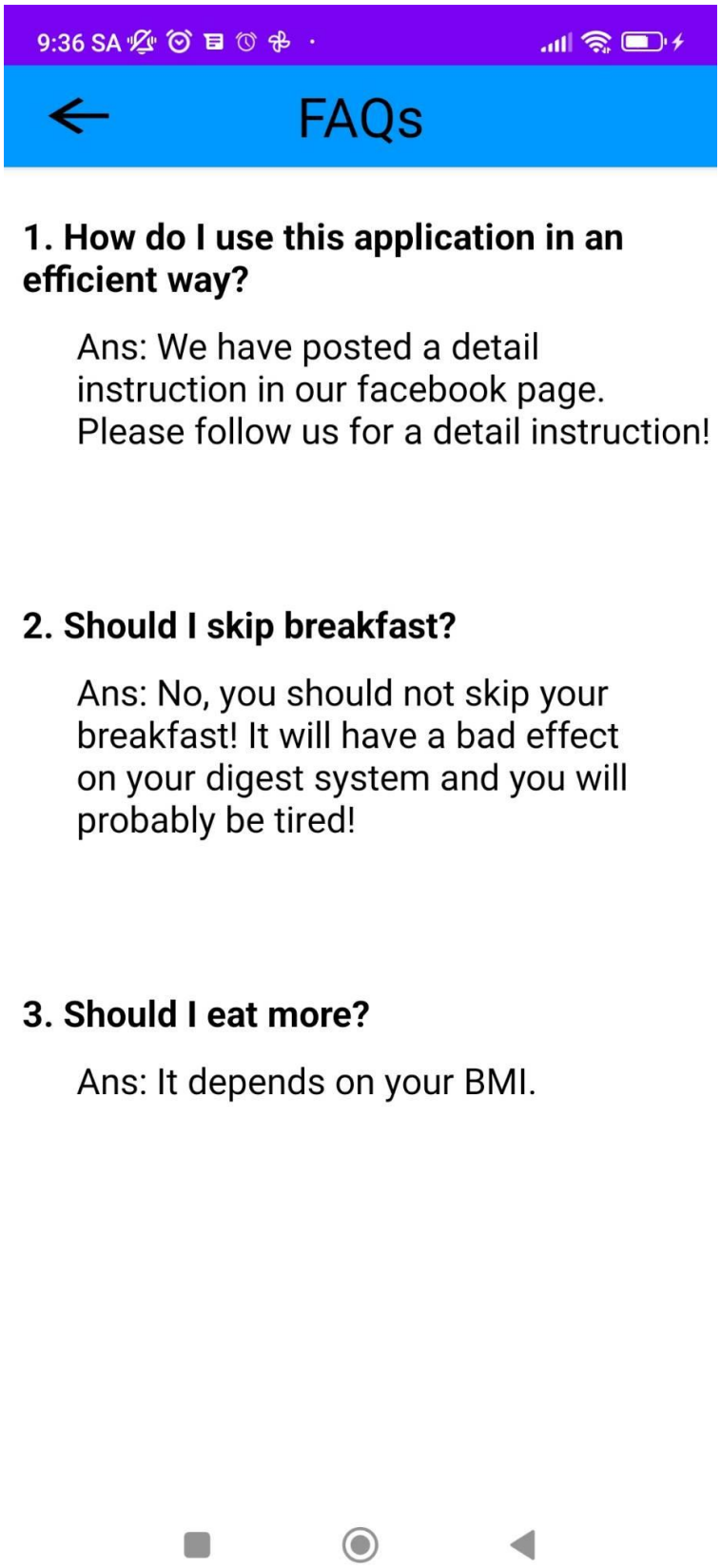


**Figure 29**. FAQs User Interface

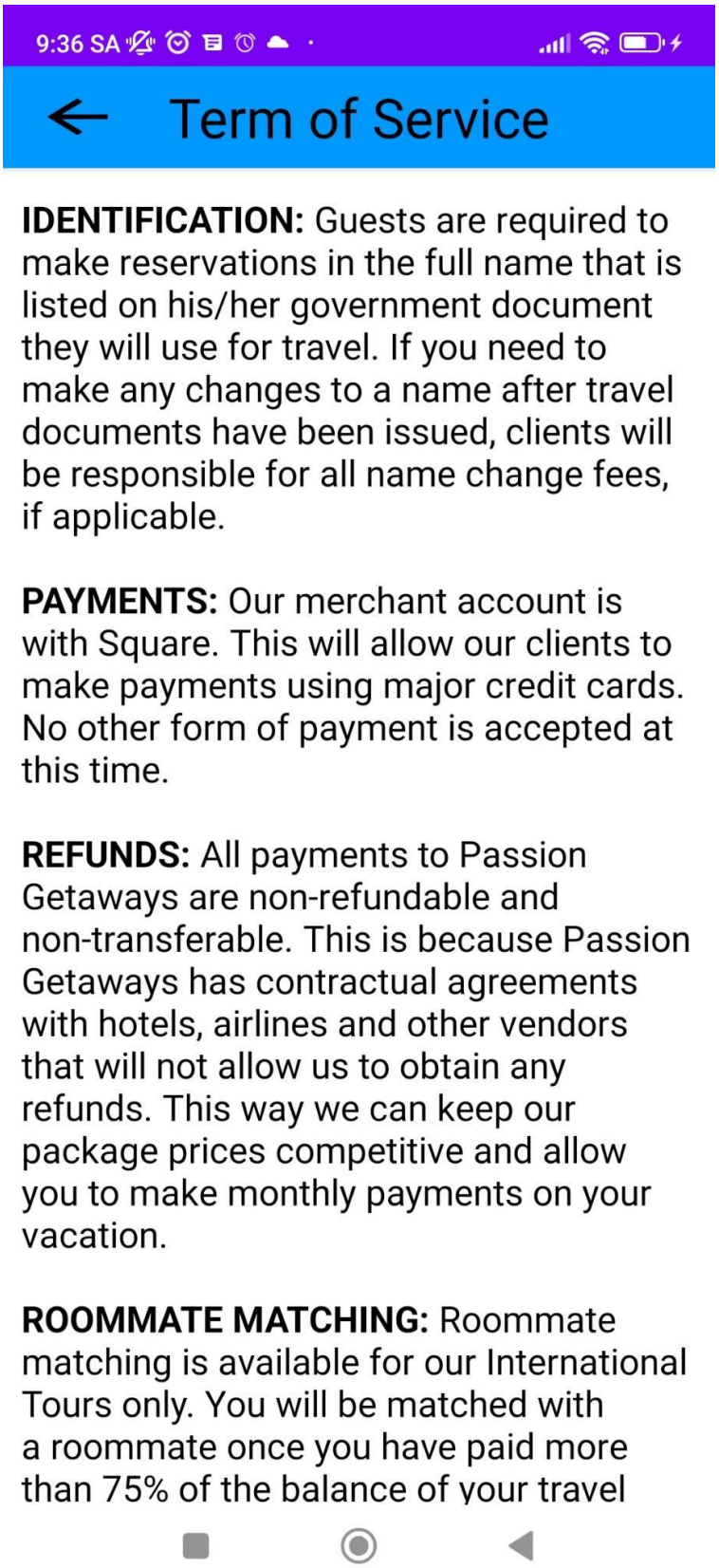


**Figure 30**. Term of Service User Interface

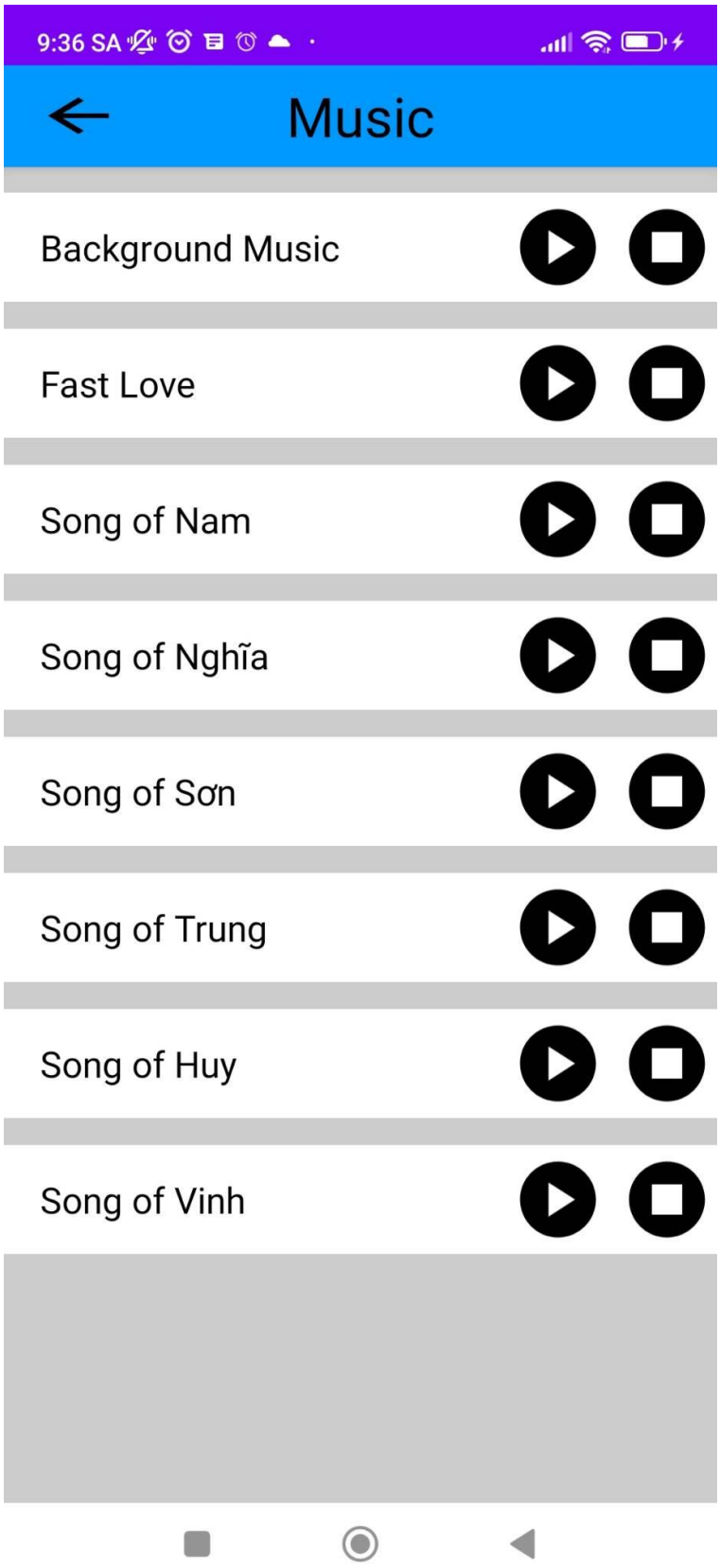


**Figure 31**. Background Music User Interface