VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Ibrahim Olanigan

DESIGN AND IMPLEMENTATION OF FOOD MANAGEMENT SYSTEM ON ANDROID PLATFORM WITH QR CODE SUPPORT

Technology and Communication

2013

ACKNOWLEDGEMENT

To Allah ta'ala (God the Exalted), who grants me ease and succour amidst the difficulties of life. To my mother who has taught me that hard work and being patient and prayerful over what one have no control over are vital for success. To my late father, who has instilled in me discipline and trustworthiness, and has been my best role model in this era. To my siblings, who have always supported me and trusted my big decisions in life. To my close friends who were always cheering me on. I am grateful to them all especially Rafiat Sanni, Ibrahim Afolabi, Nimatallah King, Sherifah Alagbe, Lawal Olufowobi, AbdulMajeed Folorunsho to mention a few.

To my supervisor, Yang Liu who made me realise I can always be better. To Santiago Chavez and Jarmo Makelä who made complex topics look easy and to all the amazing teachers I have had the honour of studying with. I am grateful for the wonderful and challenging times you gave me.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Information Technology

ABSTRACT

Author	Ibrahim Olanigan
Title	Design and Implementation of Food Management System on Android Platform with QR Code Support
Year	2013
Language	English
Pages	53
Name of Supervisor	Yang Liu

Food wastage is increasingly becoming a topic of concern due primarily to the negative impact it has on the economic and agricultural industry. Research has shown that in Finland, households seems to be the highest producers of food waste and some of this, is as a result of food being disposed because they are expired.

The main objective of this thesis was to provide a viable solution that allows mobile users to track the life cycle of their food inventory efficiently. This project also provides a demo of better implementation with future enhancement in food packaging. This project was developed for the Android platform, using Facebook integration to simplify the user registration and a web server for storing the information.

Keywords: Android, Food, Client, Server, PHP, MySQL

The mobile application, which acts as the client-side component was developed and built in the Eclipse software environment using the Android SDK alongside external libraries, Facebook SDK for Facebook integration and ZBarScannerLibrary to read QR code. The server-side component was developed using the Notepad++ software for the PHP code and phpMyAdmin for the database management. The end application is able to register user with their Facebook account and food entries can be added and deleted both on the mobile device and the server.

Keywords: Android, Food, Client, Server, PHP, MySQL

CONTENTS

LIST OF ABBREVIATIONS

API	Application programming Interface
AVD	Android Virtual Device
CSV	Comma-separated values
GUI	Graphical User Interface
GPL	GNU General Public License
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
QR	Quick Response
SDK	Software Development Kit
SQL	Structured Query Language
XML	Extensible Markup Language
UI	User Interface

1 INTRODUCTION

The prevalence of food waste has been a subject of interest and discussion in recent years and researches are being done to find effective ways to curb it. It has been identified as a primary issue in the sustainability of food production and consumption, in addition to the sustainability of food supply chains. According to Heta-Kaisa Koivupuro, food waste can be divided into avoidable and unavoidable waste. Avoidable waste includes edible food and spoiled/damaged edible food, while unavoidable waste consists of the inedible food like bones, fruit peels, and egg shells among others./1/

A research shows that in Finland, 5% of purchased food is wasted in households and an average person wasted about 20-30 kg of food in a year. The average total amount of food wasted in households yearly is about 120-160 million kilogrammes./1/ Household wastage could be intentional or not. Many of the food wastage in household could be as a result of forgetfulness or negligence for the food expiry date. In countries like Finland with high cost of living, consumers are inclined to buy food nearing its expiry date due to the discount shop sellers attach periodically.

1.1 Background

The advancement of technology has brought ease to the stressful life of human beings. The prevalence of mobile technologies enables us to constantly be in touch with the world. By it, different aspects of our lives are brought together for easy access. For instance, a person could be making finishing touches with his presentation for next day, discussing with his/her spouse about dinner, booking a flight for a weekend trip, to mention a few, all in the same place and likely simultaneously.

The idea for this project was born with my observation of how easily fellow students dispose expired food products. Due to the high cost of living, many students tend to buy food products which are close to their expiry date and sometimes in large quantity due to their discounted prices, as shops attempt to get

as much of these products out of their inventory to reduce their losses.

With the proliferation of smartphones, I thought of the feasibility of using the smartphone as a lifecycle tracker for our food inventory, and be regularly informed of those products whose expiry date is close by. It is expected that this would help to reduce the amount of food spillage in the households.

1.2 Objective

The objective of this project was to create a mobile application to assist users in managing their food inventory. The application would store and display basic information about the inventory contents and alerts the user of the food products which are due to expire the next day. Consequentially, users may take actions to avoid the concerned products get wasted or spoiled. It is believed that a considerable amount of food waste would be avoided in households if the occupants are well-informed of the timeline of their food stocks. Provisions have also been made to allow for the multi-device use.

Most food management applications available are mainly concerned with helping users watch their weight and food in-take and generally requires lots of information from user. The advantage of this project is the use of the simplest information of food products to monitor the inventory. With an eye on the future, a demo solution was integrated to show compatibility with future advancement in food packaging.

1. INTRODUCTION TO PROJECT TOOLS

This project utilises various technologies and tools. They are Android, QR Code, Hypertext Preprocessor (PHP), MySQL database and phpMyAdmin.

2.1 Android

Android is an Open Source software stack for mobile devices like phones and tablets and others. The stack comprises of a Linux-based kernel, middleware and mobile applications. It is developed by the Open Handset Alliance spearheaded by Google Inc. It is licensed under the Apache Software License, 2.0 , which is commonly abbreviated as “Apache 2.0”.

2.1.1 Application Overview

An Android application is usually made of a number of user interface components, called Activities. An activity is a component that provides a screen for user interaction to perform an action, such as take a photo, or view gallery. Typically, an application often has a main activity by which other activities are called.
An application may also have non-visual components that are essential to its operations. These components are Services and BroadcastReceiver. A service is an Android component used to perform long-running operations in the background, i.e. not visible to the user, and could also be used by an application to expose some of its functionality to other applications./2/ It is registered using the <service> tag in the *AndroidManifest.xml* file.

A BroadcastReceiver is an Android component which receives and handles a broadcast sent by the system or any application. A broadcast is a system message that is sent when an application or system occurs. For instance, a broadcast message may be sent by the orientation of the phone or the battery status changes. It is statically registered in an application using the <receiver> tag in the *AndroidManifest.xml* file.

Communication between the application components is done using Intents. Intent is an abstract description of an operation to be performed. It is mostly used to start an Activity. It can also be used to send a broadcast message and communicate with Services./3/

2.1.2 Android Development

An Android application can be developed using an Android SDK and a compatible software development environment. The Android SDK provides develops with the necessary set of tools and libraries needed to build, test and debug applications on the Android platform./4/ It is readily available for download, along with needed support, on the Android official website.

This project was built in the Eclipse software development environment, which supports multiple programming languages and operating systems, and it is free to use under the Eclipse Public License.

2.1.2.1 AndroidManifest.xml

The requirement for all Android application is to have the *AndroidManifest.xml* file in its root directory. It presents vital information about the application to the Android system that the system requires before running any code of the application

Some of the information found in the *AndroidManifest.xml* file includes,

- The unique package name for the application
- The minimum Android API level required for the application.
- Description of the application's components, i.e. the activities, services, broadcast receivers and content providers that make up the application.
- Lists of libraries linked to the application.
- Declaration of permissions needed to access protected API components, among others./5/

2.1.3 Android Emulator

The Android SDK provides an emulator, which is a virtual mobile device, which runs on the computer and enables the user debug and test applications without a physical device.

The specification of the emulator is defined, and can be edited, by the developer using the AVD Manager, which is a graphical user interface used to configure and manage AVDs.

The AVD can be configured as different devices, screen sizes, Android target levels. For this project, I have configured the AVD as a Nexus device with a screen size of 4.65 inches with a resolution of 720 by 1280 pixels. The AVD runs Android Jelly Bean, version 4.2.2, which is equivalent to API level 17.

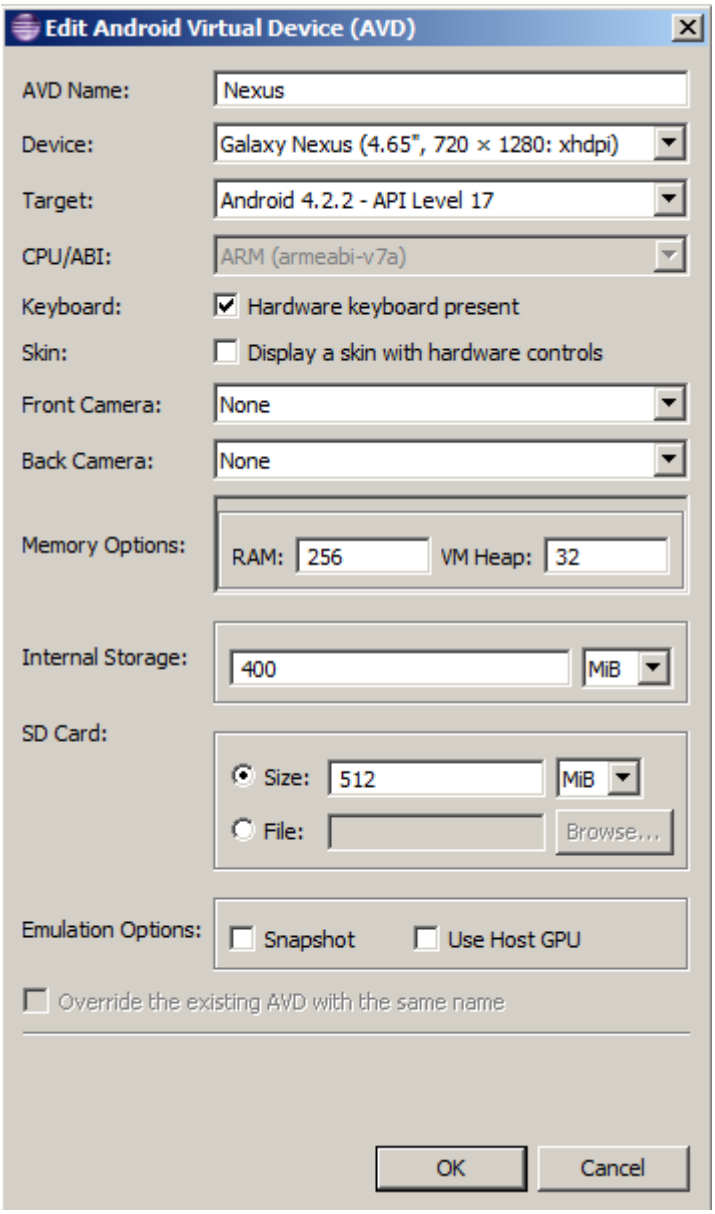

Figure 1.AVD Configuration interface.

Figure 2.AVD (Scaled to original dimensions)

2.1.4 SQLite Database

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional database engine. It is free to use for any purpose, be it private or commercial. It is compact and lightweight hence it is easily deployable to any system. It is supported by the many UNIX and Windows operating systems and can be ported easily to other systems. The data types supported are TEXT (to hold string values), INTEGER (to hold integer values) and REAL (to hold precision floating-point values)./6/

Android provides the SQLite database to allow for data storage in an application. An application in the Android system may have a private database and this can only be accessed and managed within the application code.

2.2 Quick Response Code (QR Code)

The QR code is the trademark for a type of two-dimensional bar code. It is an optical machine-readable label used to store information about the item it is attached to. It was originally designed for use in the automotive industry by a Toyota subsidiary in Japan, but has become widely popular for other usages because it is faster to read and have more storage capacity than standard bar codes.

QR codes are usually used to store contact information, Uniform Resource Locations (URLs), phone numbers, and text.

Figure 3. Sample QR Code, used for testing in the project.

2.3 PHP: Hypertext Preprocessor (PHP)

PHP is a server-side scripting language primarily designed for the production of dynamic pages. It was created by Rasmus Lerdorf in 1995 and it is now being developed by the PHP group. PHP is free software released under the PHP License, which makes it incompatible with the GNU General Public License (GPL) due to restriction on the use of the term PHP./7/

It is cross-platform software mostly used in the server-side web development and it is now being used in the client-side User Interface (UI). It has been used in the creation of many Web content management systems like Drupal, Wordpress and Moodle.

```
<?php
// PHP code goes here
?>
```

Figure 4. Basic PHP syntax

2.4 MySQL

MySQL is a cross-platform open source relational database management system (RDBMS). It was created by Michael Widenius, who partly named it after his daughter, My. It was initially released on the 23rd of May, 1995 under the GPL License. It was originally owned by a Swedish firm, MySQL Ab, which is now owned by Oracle.

It is written in C and C++. For this project, the MySQL database was managed using phpMyAdmin.

2.5 phpMyAdmin

PhpMyAdmin is a free and open source GUI tool written in PHP, which is used for web database administration. It has cross-platform support for the major operating systems and it was first released in the 1998 under the GNU General Public License.

It has an intuitive web interface, and core support for many MySQL features. It also has data management (including import and export) support for many formats like CSV, SQL, PDF, XML, among others./8/

2.6 Facebook Integration

Facebook is a popular social networking platform started in 2004 by Mark Zuckerberg and couple of his friends. It is regularly expanding and boasts of 1 billion users as in October 2012. Due to its large user base, Facebook provides an

avenue of services for developers to tap into its wealth of information.

For the purpose of this project, I have integrated a Facebook login functionality to access basic information about the users, with their permission, for registration on the server-side of the project.

3 SYSTEM OVERVIEW

3.1 Choice of Android

Android has been chosen for this project, primarily for the open-source nature of the platform as well as the ease of development and deployment with the extensive supports provided on the official Android website and major developers' forums, such as the Stack Overflow website.

It also has the largest market share and has native compatibility with tablets. It also supports cross platform application development, i.e. developers can develop Android application in Mac, Windows and many UNIX-based operating systems like Ubuntu.

3.2 Requirements

There are certain requirements the proposed application must fulfill to meet the objectives of the project.

The requirements on the client-side are:

- It must have a user interface
- It must be compatible with most Android devices.
- It must have Facebook integration.
- It must have QR code reading capabilities.
- It should have the ability to store data in the server

The requirements on the server-side are:

- Database must have a user table
- Database must be dedicated food tables for users.
- Database must be able to communicate with client-side application.

3.3 System Architecture

This application consists of an Android application on the client side and PHP-MySQL application on the server side. The Android application is the part visible

to the user and one it interacts with, while the PHP/MySQL-based server-side component serves as an interface between the Android application and the database on the server.

The use case for the client-side application is seen in Figure 5 below, showing all the cases available to the user in the application.

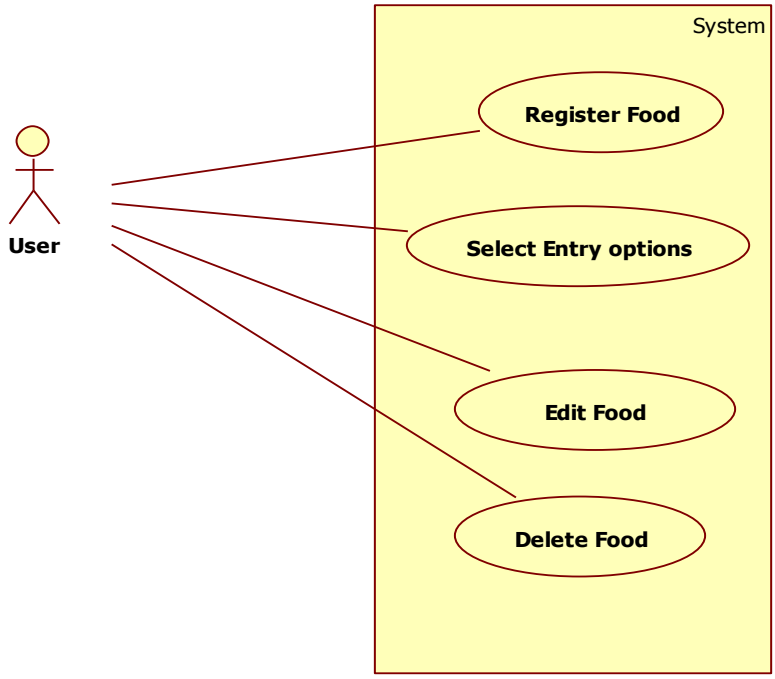

Figure 5. Use case for client-side application

Figure 6 below, shows the use case diagram for the server-side component.

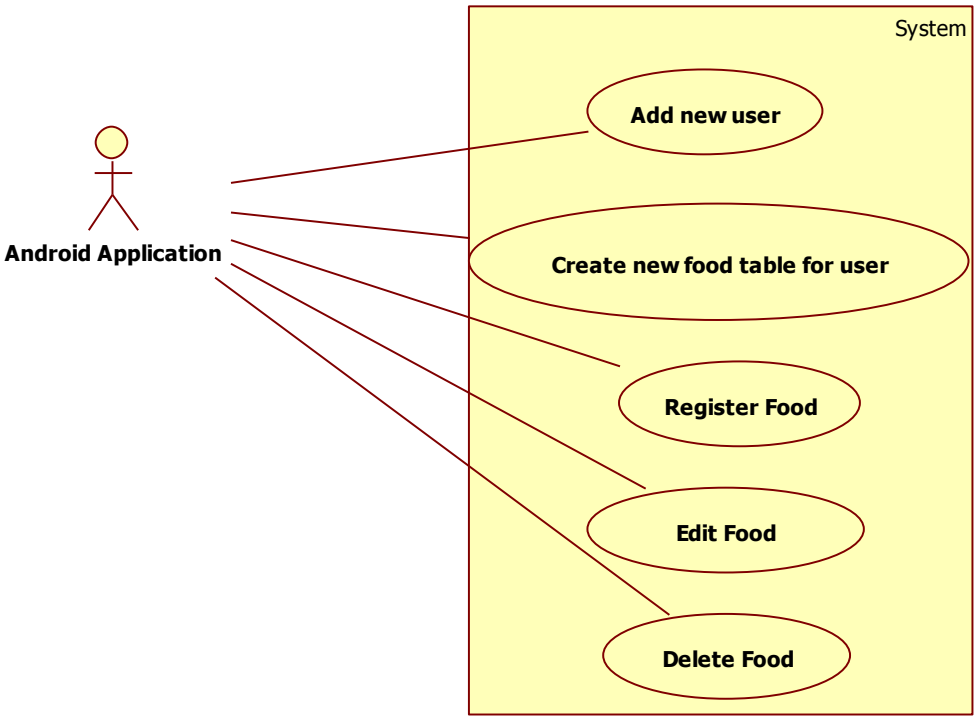

Figure 6. Use case for server-side implementation

4 CLIENT-SIDE DESIGN & IMPLEMENTATION

The client-side application is designed based on the requirements stated in 3.2.1, using the right sets of libraries, database design and programming methods while providing a good user experience.

4.1 Development Overview

The mobile application was developed in the Eclipse software using the Android SDK downloaded from the Android official website. This project uses three java packages, namely:

- com.olanigan.food
- com.olanigan.data
- com.olanigan.utils

The food package contains all the interface-related classes, while the data package contains all data management classes. Utility classes are found under the utils package. The figure below shows the structure of the project in the Eclipse software.

The application is configured to a minimum API level of 8 and declares permissions to use the WAKE_LOCK, INTERNET and CAMERA functionalities of the system. The figures below show the configuration of the *AndroidManifest.xml* file.

All the titles of the application are defined under the <application> tag, along with the list of components and libraries used. The figure below shows a breakdown of the application structure.

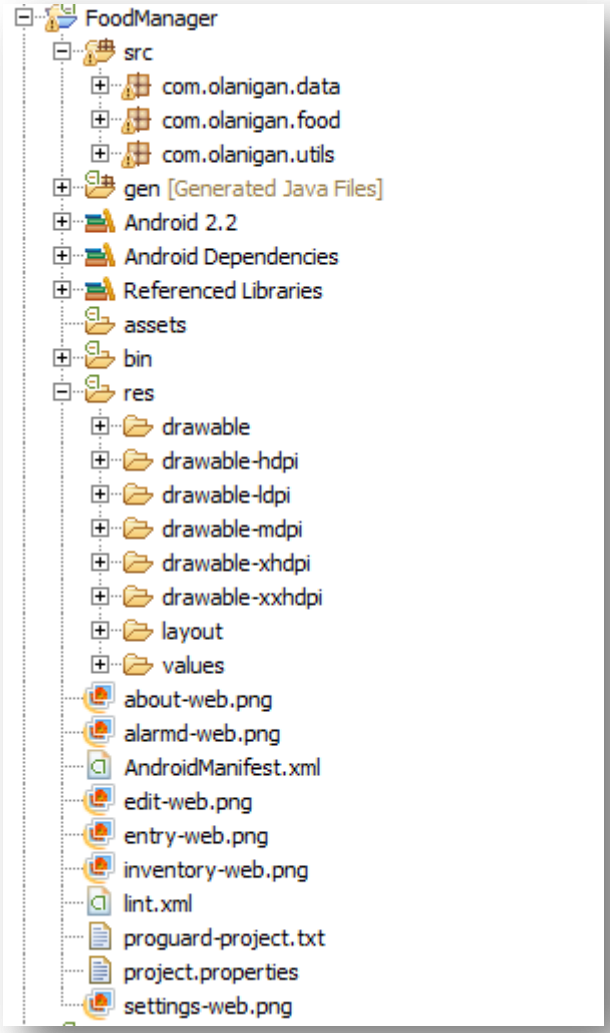

Figure 7. Overview of project structure in Eclipse

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.olanigan.food"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature android:name="android.hardware.camera" />
    <uses-sdk android:minSdkVersion="8" />

    <application..

</manifest>
```

Figure 8. Overview of the *AndroidManifest.xml* file

```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name"
    android:allowBackup="true"
    >
    <activity
        android:name="com.olanigan.food.FbLoginActivity"
        android:label="@string/facebook" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name="com.olanigan.food.MainActivity"
            android:label="@string/app_name" />
    <activity android:name="com.olanigan.food.NewEntryActivity"
            android:label="@string/newentry" />
    <activity android:name="com.dm.zbar.android.scanner.ZBarScannerActivity"
                android:screenOrientation="landscape"
                android:label="@string/zbar" />
    <activity android:name="com.facebook.LoginActivity"
                android:label="@string/facebook" />
    <activity android:name="CodeReader"
            android:label="@string/zbar" />

    <receiver  android:name="Notifier" />
    <meta-data android:name="com.facebook.sdk.ApplicationId"
        android:value="@string/appId"> </meta-data>
</application>
```

Figure 9. Overview of the application structure

4.2 User Interface Classes

The user interface in Android is displayed using classes that extend the Activity class directly or indirectly. The classes used for user interaction in this application are MainActivity, NewEntryActivity and FbLoginActivity.

The MainActivity class is the main user interface for the application. It displays the registered food inventory of the user and has the main menu by which other activities can be accesses.

The class diagram for the MainActivity is shown below.

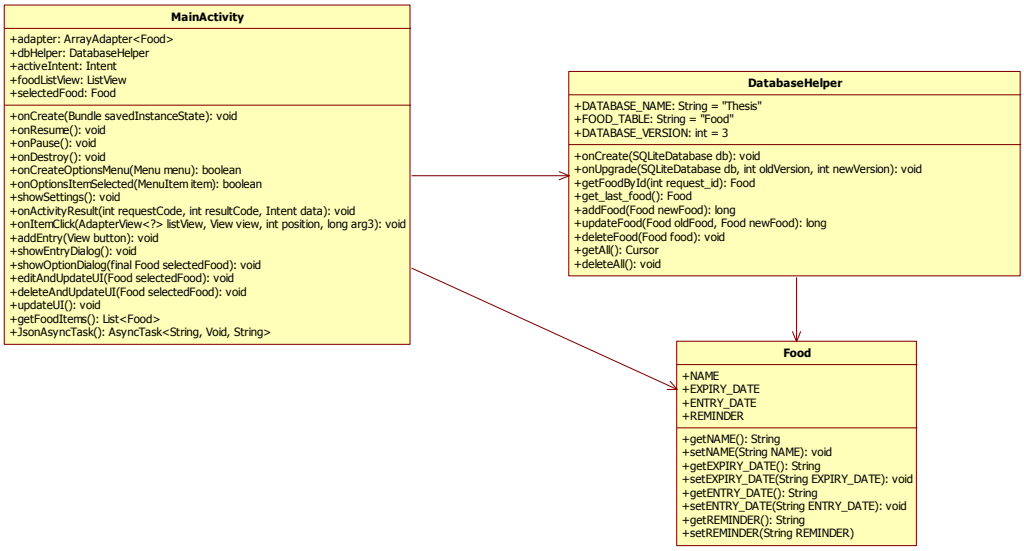

Figure 10. MainActivity Class diagram

The NewEntryActivity class handles both the manual entry as well as the QR code entry of the information about the food to be monitored. It is called by the MainActivity class and returned to it after the entry is completed. It displays a form that requests information about the name of the food, its expiry date and reminder time.

The class displays the result of the QR code scanning initiated after the user chooses the QR code entry in the MainActivity interface. The customized QR code used for this project, contains information about the name of the new food,

and best-before date, and is scanned using the ZBarScannerActivity class, which is called from an open source QR code scanning library, ZBarScannerLibrary.

After the user fills in the form completely, the class handles the storing of the new food data into the database, as well as set an alarm notification to the user-defined time a day before the expiry date. After the completion of its task execution, it returns the user to the MainActivity class where the updated food listing is displayed.

The diagram below shows the class diagram for the NewEntryActivity class.

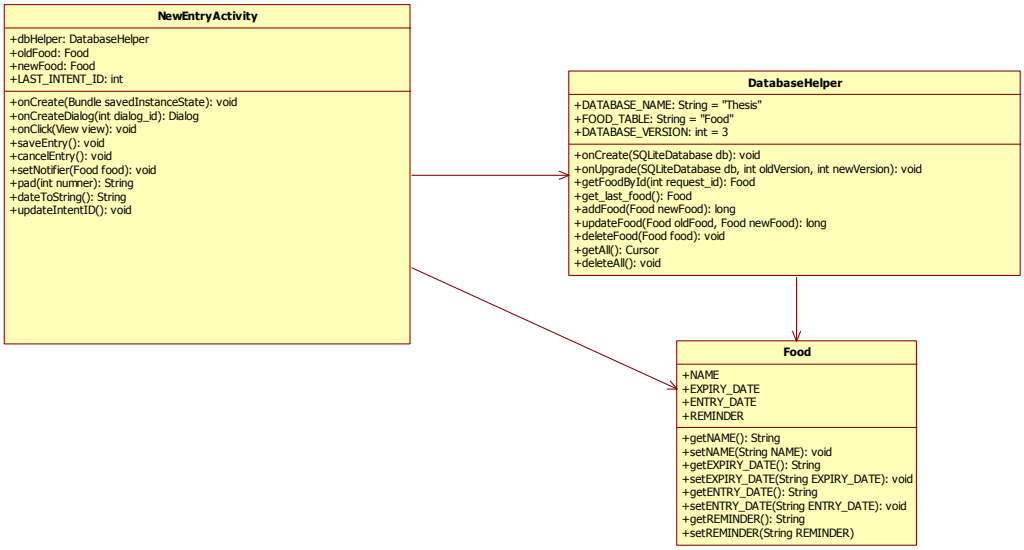

Figure 11. NewEntryActivity Class Diagram

The FbLoginActivity class is the entry point into the application and it handles the login and logout of the user using Facebook authentication. Unlike the other activities, it extends FragmentActivity and acts as the main display for three Fragment classes, which are the SplashFragment, InfoFragment and UserSettingsFragment classes.

The SplashFragment class displays the Facebook-custom login button. When clicked, the button calls the Facebook login dialog from the Facebook SDK library. The InfoFragment class is called after a successful Facebook login by the user. It retrieves basic information about the user and this information is used

either to register the new user into the application database or to retrieve the latest food listing from the application database. The InfoFragment class redirects the user to the MainActivity UI.

The userSettings class displays the Facebook logout button by which the user exits from the main application and redirects to the Facebook login page after execution. This class is provided by the Facebook library.

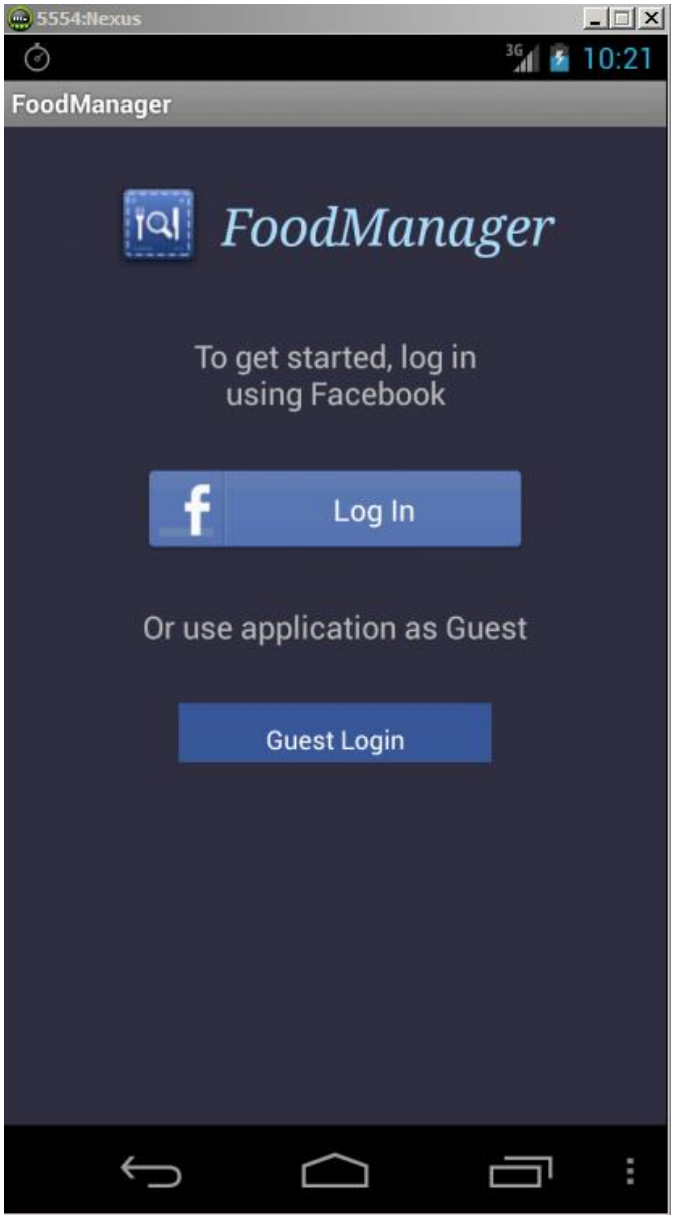

Figure 12. Application Login Screen

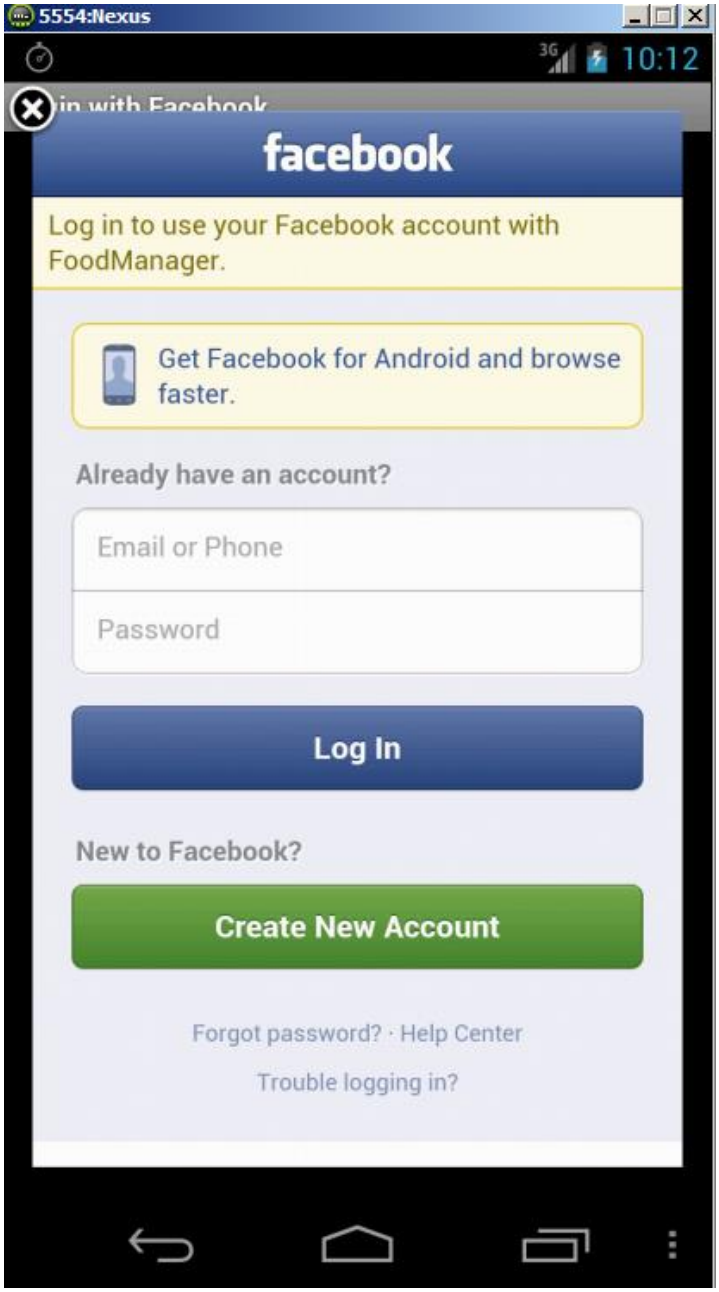

Figure 13. Facebook Login Dialog

4.3 Data Management Classes

4.3.1 Data Modeling

A Food class acts as the model object for the application. It was created to enable uniformity and ease of data management. The properties of the model include the

name of the food, its expiry date, and its date of entry as well as the chosen time to be reminded of its expiry.

The structure of the model is displayed in Figure 14 below.

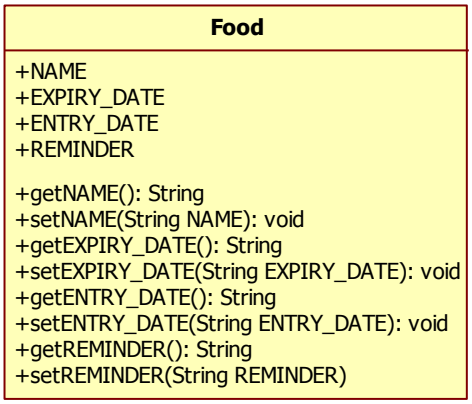

Figure 14. Food model of application

The table below highlights the attributes of the model.

Table 1. Description of Food model attributes

Attribute	Description
Name	Name of the food to be stored.
Expiry Date	Best-before date of the food product
Entry Date	Date of entry of food information
Reminder	Time chosen by user to be reminded a day before expiry date

4.3.2 Database Management

Data persistence in an Android application is done primarily with the SQLite database provided as a library component in the Android system. Applications that utilise the SQLite database, usually have dedicated classes to handle the

management of the database. The DatabaseHelper class was created for that purpose. It extends the SQLiteOpenHelper class and handles all the internal database functions of the application including opening and closing the database, executing queries and handles queries with the Food model.

It also does operation on the server database using the methods defined in the *UrlHandler* class.

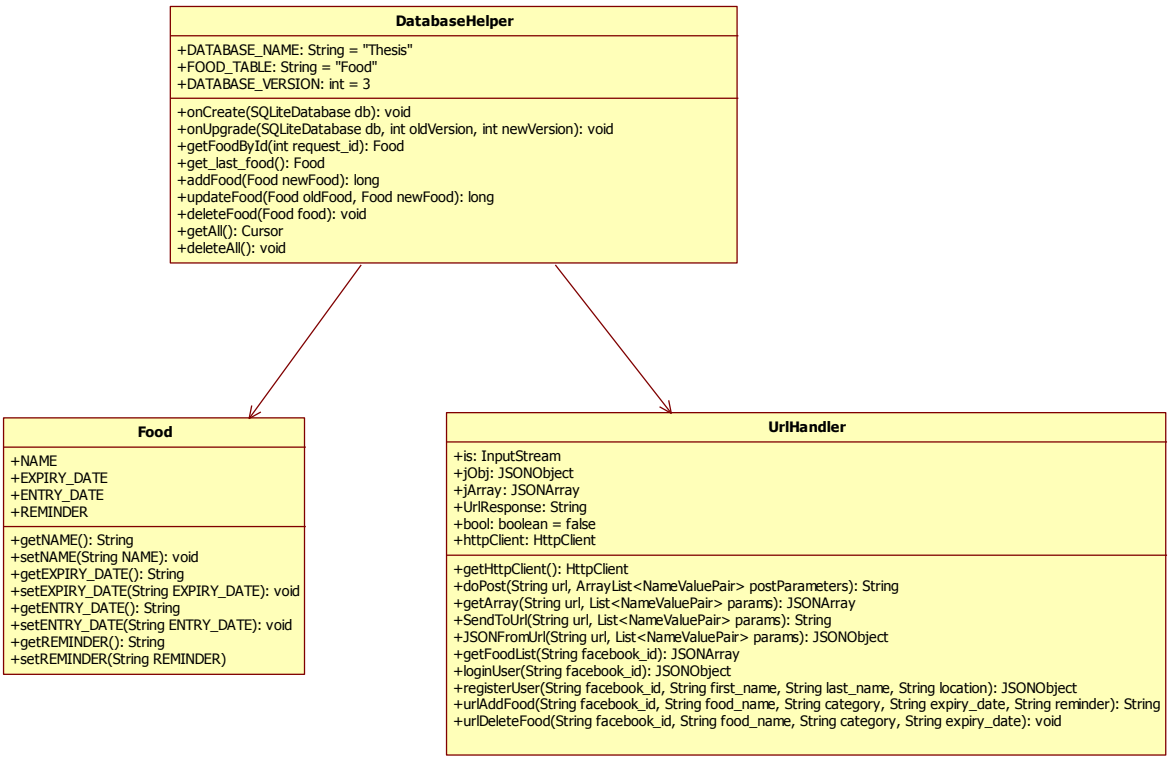

Figure 15. DatabaseHelper Class Diagram

For this project, the database was named “Thesis” and defined in the *DatabaseHelper* class as *DATABASE_NAME.* Also, only one table was created in the database. The table is named “Food” and defined in the *DatabaseHelper* class as *FOOD_TABLE.* The table is created when the database is first created and can only be structurally modified when the database is upgraded. It is used to store the food entries of the user and the final version of the database is three (3) due to the structural changes it has undergone during development.

4.4 Server Communication Classes

Data communication between the application and external server is handled by the UrlHandler class. Its operations include retrieving and registering user and food inventory information.

The class has numerous functions which sends HTTP request to the server. The requests consist of the URL address of the *index.php* file on the server and parameters that contain the information of the user and food entries. The table below highlights the common parameters used in the request.

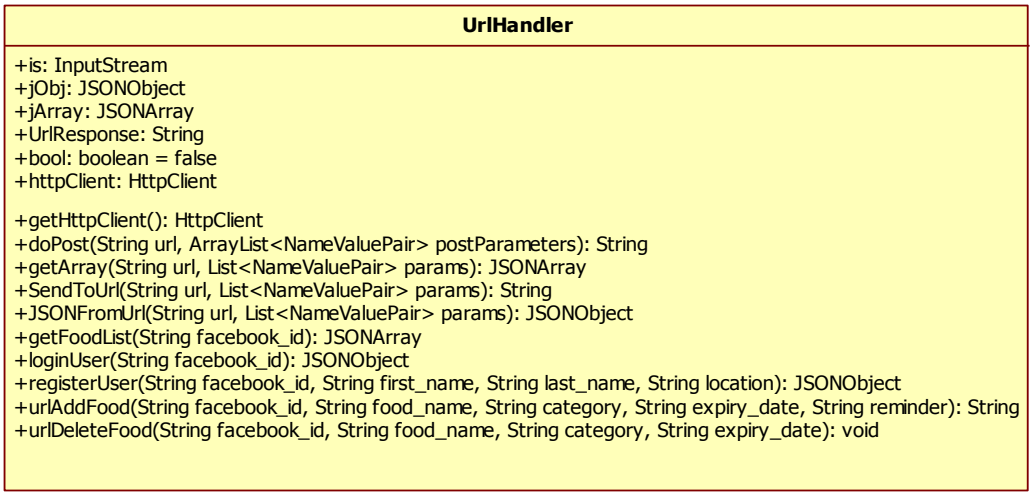

Figure 16. UrlHandler Class Diagram

Table 2. Description of Parameters used in HTTP request to server

HTTP Parameters	Description
tag	Tag for different operation
facebook_id	ID of user from Facebook server
first_name	First name of user
last_name	Last name of user

location	Primary location of user
food_name	Name of the food to be stored.
expiry _date	Best-before date of the food product
entry_date	Date of entry of food information
reminder	Time chosen by user to be reminded a day before expiry date

The number of parameters that accompany a request differs depending on the desired operation to be performed on the server. However, the tag parameter is the primary parameter that informs the server what operation to perform. In this application, a list of tags was created as string and the outline of these tags can be found in the table below.

Table 3. Description of different tag types

Tag Name	Value (string)	Description of requested operation
list_tag	list	Return the list of all food entries by current user
login_tag	login	Check if user information exists in database
register_tag	register	Register new user and create dedicated table for user in the database
add_tag	add	Add new food entry to database
delete_tag	delete	Delete selected food entry from database

4.5 Utility classes

The Notifier class acts as a Broadcast Receiver for the application. It inherits a BroadcastReceiver class.

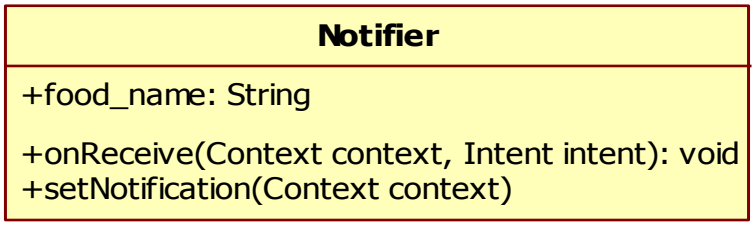

Figure 17. Notifier Class Diagram

The class is notified by the Android system when an alarm WAKE_UP event occurs after the countdown value set in the NewEntryActivity or QREntryActivity classes elapses. The onReceive function of the class is automatically called to handle the event. The setNotification function is then called to create a new notification in the system notification bar, informing the user of the name of the food that is expected to expire the next day.

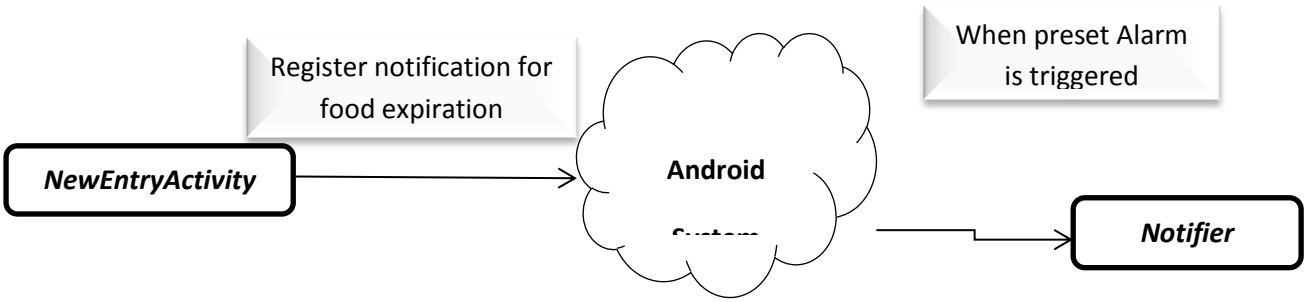

Figure 18. Relationship between NewEntryActivity and Notifier Classes

5 SERVER-SIDE DESIGN AND IMPLEMENTATION

This chapter deals with data retrieval and storage from the database, initiated by HTTP request from the client-side application. PHP is used to handle the request from the application and performs appropriate tasks on the MYSQL database. It also informs the application of success or failure of the application.

5.1 Development Overview

The PHP development was done in the Notepad++ software, which is a free and open Windows multi-language editor. It provides colored support for native functions, as well as code indentation. It displays the edited files in tabs for ease of accessing and editing multiple files simultaneously.

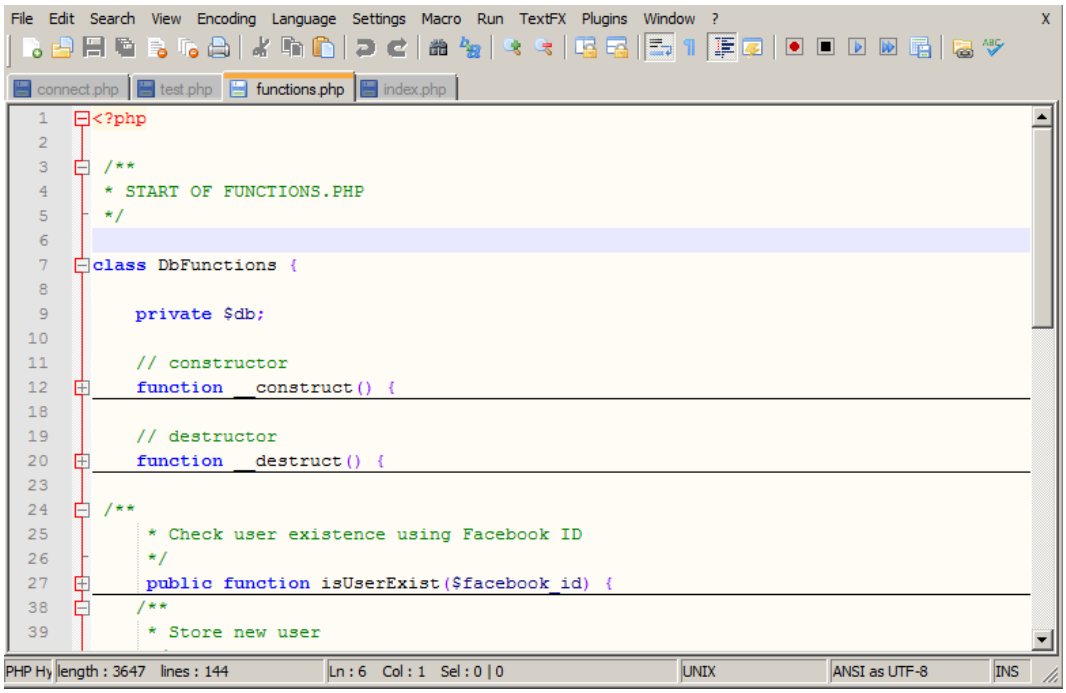

Figure 19.PHP Development in Notepad++ software

The server database was managed by a free PHP-based GUI tool named phpMyAdmin which was readily available on the server.

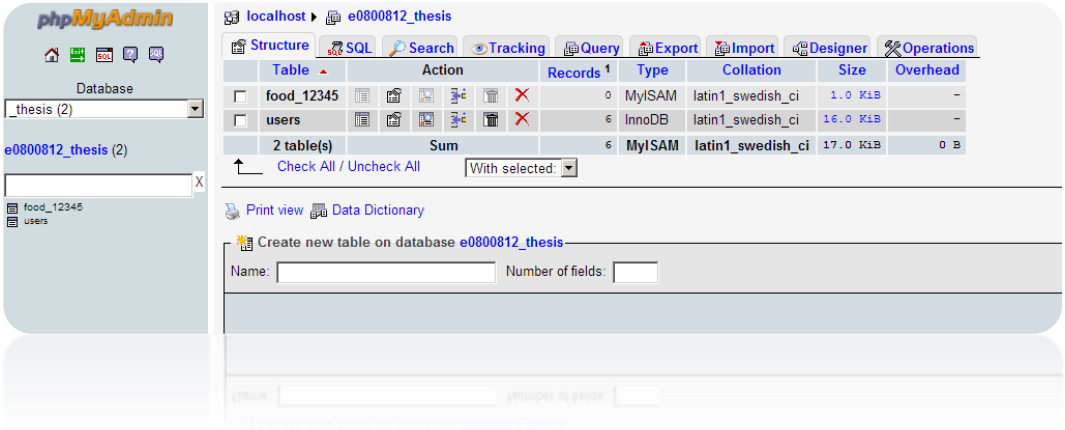

Figure 20.phpMyAdmin interface on the server

5.2 Database Design

The database for the project was designed based on the requirements listed in chapter 3.2. A table named “users” is created to store the basic information of users that uses the application. This information includes the Facebook ID, first name and the last name of the user, which are provided by the Facebook server via the mobile application.

The table has four columns. They are *facebook_id* for the Facebook ID of the users, *first_name*, *last_name* for the first and last name of the user respectively and *created_at* for the date the user information was stored in the database. The *facebook_id* column acts as the primary key for the table.

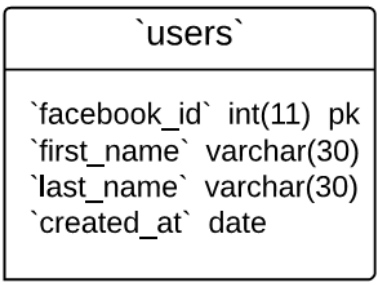

Figure 21. Entity diagram for *users* table

Each user is expected to have a dedicated table in the database to store food entries. In order to have this, a dynamic naming convention is used for these tables. The Facebook ID of the user is used as an underscore suffix to the word,

hence the name of each food table is in the format *food_[Facebook ID]*.For instance, if the Facebook ID of a user is 12345, the food table for this user would be named “food_12345”.

The table has five columns; *id* column which acts as the primary key, *food_name* column for the name of the food, *expiry_date* and *entry_date* columns for the expiry date as well as the date of entry for the food, and the *reminder* column for the time set by the user to be reminded the day before the expiry date.

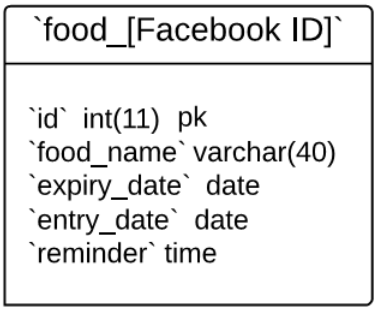

Figure 22. Entity diagram for food table

5.3 PHP Classes/Files

Three PHP files were created to handle the logical and data management operations. The main PHP file is the *index.php* file which handles the HTTP requests from the application. The other files are the *functions.php* file, which contains the declaration of the functions used in the *index.php* file, and the *connect.php* file which contains the login details for authorized access to the database. These files were stored in my student account on the *cc.puv.fi* server.

5.3.1 Connect PHP file

The *connect.php* file contains the administrative configuration of the MySQL database required to do operations on it. Due to the administrative rights granted by this file, the user application may make changes to the content of the database.

```
3   # Define MySQL configuration
4   define("MYSQL_HOST", "mysql.cc.puv.fi");
5   define("MYSQL_USER", "e0800812");
6   define("MYSQL_PASS", "[redacted]");
7   define("MYSQL_DB", "e0800812_thesis");
```

Figure 23. MySQL configuration

The figure shows that the MySQL used for this project is hosted on *mysql.cc.puv.fi* which is the official MySQL server for VAMK University of Applied Sciences. The server is only accessible locally. For this project, I have used my student account on the server and a VPN connection to allow access to the PHP files as well as managing the database remotely.

The file also defines the *Connect* class which handles connection to the database. The figures below shows the class diagram as well as the PHP code for the *Connect* class.

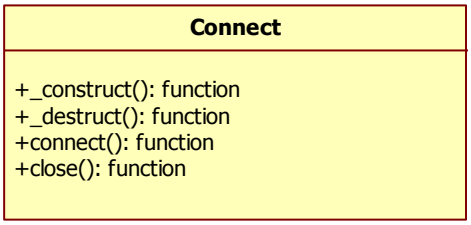

Figure 24. Connect class diagram

The class has a defined constructor and destructor. A class constructor is a method defined to be called when a new instance of the class is created and in the *Connect* class, it calls the *connect* method. Therefore, a connection is made to the database whenever an instance of the *Connect* class is created.
When a class is no longer referenced, the destructor method is called. The destructor of this class calls the *close* method, which closes connection to the database.

```
 9  class Connect {
10
11      // constructor
12      function __construct() {
13          $this->connect();
14      }
15
16      // destructor
17      function __destruct() {
18          $this->close();
19      }
```

Figure 25. Constructor and destructor for Connect class

In the *connect* method, the native MySQL function, *mysql_connect* is used to connect to the database server using the pre-defined configuration values and the function, *mysql_select_db* is used to select the specific database for this project, as shown in line 25 and 27 of the figure below.

The *close* function is defined to close the database connection by calling the native MySQL function, *mysql_close.*

```
21      //Open connection
22      public function connect() {
23
24          // connect to Database
25          $connection = mysql_connect(MYSQL_HOST, MYSQL_USER, MYSQL_PASS);
26          // selecting database
27          mysql_select_db(MYSQL_DB);
28
29          // return database handler
30          return $connection;
31      }
32
33      // Close connection
34      public function close() {
35          mysql_close();
36      }
```

Figure 26.*connect* and *close* methods for Connect class

5.3.2 Functions PHP file

The *functions.php* file defines the *DbFunctions* class. The *DbFunctions* class defines methods for database operations which are called from the *index.php*

based on the value of the tag parameter in the HTTP request. The figure below shows the class diagram for the class.

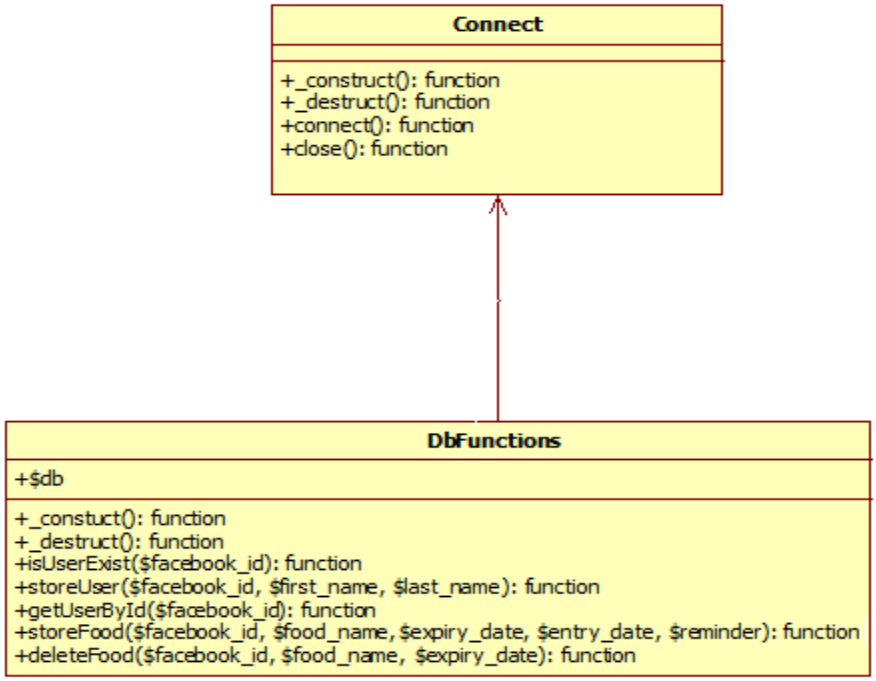

Figure 27.*DbFunctions* Class Diagram

A global variable *db* is declared and used as an instance of the *Connect* class. The assignment of the variable *db* as a new instance of *Connect* class is done in the constructor method of the class, and the *connect.php* was included to achieve this. The class has no defined destructor method.

```php
class DBFunctions {

    private $db;

    // constructor
    function __construct() {
        require_once 'connect.php';
        // connecting to database
        $this->db = new Connect();

    }

    // destructor
    function __destruct() {

    }
```

Figure 28. Global variable and constructor method

The class has five methods defined for database operation, namely: *isUserExist*, *storeUser*, *getFoodById*, *storeFood* and *deleteFood*. The *isUserExist* method is used to check if a user exists using its unique Facebook ID as an argument. The method calls a MySQL query that selects the *facebook_id* column from the *users* table with a conditional statement to check if the Facebook ID of the user exists in the column.

```php
/**
 * Check user existence using Facebook ID
 */
public function isUserExist($facebook_id) {
    $result = mysql_query("SELECT facebook_id from users WHERE
                           facebook_id = '$facebook_id'");
    $no_of_rows = mysql_num_rows($result);
    if ($no_of_rows > 0) {
        // user exists
        return true;
    } else {
        // user doesn't exist
        return false;
    }
}
```

Figure 29. Code for *isUserExist* method

The *storeUser* method is used to store information about a new user in the *users* table, and also used to create a dedicated food table for the user using a dynamic naming system. It takes three arguments, namely: the Facebook ID, first name and the last name of the user. It first stores the new user information in the *users* table, and on successful completion, it creates a new table for the user using the Facebook ID.

```
36  /**
37   * Store new user
38   */
39  public function storeUser($facebook_id,$first_name, $last_name) {
40
41      $result = mysql_query("INSERT INTO users
42              (facebook_id, first_name,last_name, created_at )
43              VALUES('$facebook_id', '$first_name', '$last_name', NOW())");
44
45      /* check for successful store*/
46      if ($result) {
47          $result1 = mysql_query("create table food_".$facebook_id."
48                  (`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
49                    `food_name` VARCHAR(40) NOT NULL ,
50                    `expiry_date` DATE NOT NULL ,
51                    `entry_date` DATE NOT NULL ,
52                    `reminder` TIME NOT NULL)")
53                    or die(mysql_error());
54
55          return $result1;
56      } else {
57          return false;
58      }
59
60  }
```

Figure 30. Code for *storeUser* method

The *getFoodById* method is used to retrieve all the food entries of a particular user using the Facebook ID of the user as argument.

```
62  /**
63   * Returns all food entries of user using Facebook ID
64   */
65  public function getFoodById($facebook_id) {
66      $result = mysql_query("SELECT * FROM food_".$facebook_id."")
67                                  or die(mysql_error());
68      // check for result
69      $no_of_rows = mysql_num_rows($result);
70      if ($no_of_rows > 0) {
71          return $result;
72      } else {
73          return false;
74      }
75  }
```

Figure 31. Code for *getFoodById* method

The *storeFood* method is defined to store food entries from the mobile application to the server. It takes five (5) arguments, one of which is the Facebook ID of the user which is used to determine the table to store the entry into based on the dynamic table naming system. The other arguments are the food information provided by the mobile application. They include the name of the food, its expiry date, its entry date and reminder time for the food.

```
77  /**
78   * Store new food entry
79   */
80  public function storeFood($facebook_id,$food_name,$expiry_date, $entry_date,$reminder) {
81
82      $result = mysql_query("INSERT INTO food_".$facebook_id."
83                             (food_name,expiry_date,entry_date,reminder)
84                             VALUES('$food_name', '$expiry_date','$entry_date','$reminder')")
85                             or die(mysql_error());
86
87      // check for successful store
88      if ($result) {
89          return true;
90      } else {
91          return false;
92      }
93  }
```

Figure 32. Code for *storeFood* method

The *deleteFood* method is defined to delete a particular food entry. It takes three arguments which are the Facebook ID of the user, the name of the food and its expiry date. The Facebook ID is used to select the table and the other arguments are used to select the specific entry to delete. The name of the food and its expiry date are together distinctive of each food entry. The ID of the table was not used for the deletion due to the possibility of being different from its corresponding value in the client-side application.

```
96  /**
97   * Delete food entry
98   */
99  public function deleteFood($facebook_id,$food_name,$producer,$expiry_date) {
100
101     $result = mysql_query("DELETE FROM food_table WHERE food_name='$food_name'
102                            AND expiry_date='$expiry_date'") or die(mysql_error());
103
104      // check for successful deletion
105      if ($result) {
106          return true;
107      } else {
108          return false;
109      }
110  }
```

Figure 33. Code for *deleteFood* method

5.3.3 Index PHP file

This file that handles communication between the mobile application and the server database. It performs logical operations based on the value of the tag parameter of the HTTP request and encodes the response in JSON format, which is handled by the mobile application. It uses the *DbFunctions* class, defined in the *functions.php* file, to perform operation on the database.

```
1  <?php
2
3  if (isset($_POST['tag']) && $_POST['tag'] != '') {
4      //Store values of default paramters
5      $tag = $_POST['tag'];
6      $facebook_id=$_POST['facebook_id'];
```

Figure 34. Validity check for the tag parameter

All the logical operations are enclosed in an IF conditional statement which ensures that *tag* parameter is set and it is not empty, as shown on line 3 of the figure above. The *tag* parameter is used to select which operation the application intends to perform on the database whenever it sends an HTTP request while holding the value of the Facebook ID of the user provided by the Facebook server.

All HTTP requests must have the *tag* and *facebook_id* parameters. Lines 5 and 6 show the storing of the values of these parameters in their corresponding variables.

```
8       // include DbFunctions class
9       require_once 'functions.php';
10      $dbFunctions = new DbFunctions();
```

Figure 35. Inclusion of DbFunctions class

The *require_once* statement on line 10 is used to include the *functions.php* file in the code and ensures it is only included once. The variable *dbFunctions* is declared on line 11, as an instance of *DbFunctions* class declared in the *functions.php* file.

```
12      // Response Array
13      $response = array("tag" => $tag, "success" => 0, "error" => 0);
```

Figure 36. Declaration of the response array

Line 13 shows the declaration of the *response* variable. It is declared as an array and primarily contains the value of the tag parameter in the HTTP request and varying values of success and error, which are determined at the end of each tag-based logical operation. The values of the success and error tags are both zero (0) by default, and only one of these tags changes its value to one (1) after the execution of the tag-based operations. The value of the success tag changes to one (1) if the operation was executed successfully. Otherwise, the value of the error tag changes to one (1).

```
16  if ($tag == 'login') {
17
18      //Check if user already exists
19      if($dbFunctions->isUserExist($facebook_id)){
20          $response["success"] = 1;
21          $response["id"] = $user["facebook_id"];
22          echo json_encode($response);
23      }else{
24          $response["error"] = 1;
25          $response["error_msg"] = "Incorrect ID";
26          echo json_encode($response);
27      }
28
29  }
```

Figure 37. Code for handling login tag

In the case when the mobile application sends an HTTP request with the login tag, the request would only contain the *tag* and *facebook_id* parameters. The *facebook_id* variable is used on line 19, as an argument in the *isUserExist* function of the *DBFunctions* class to check if the information about the user's Facebook ID exists in the database. If the user information is present, a success message is sent back as response as shown on line 22 or else, an error message is returned as shown on line 26. All the response messages are encoded using the JSON format.

```
30  elseif($tag == 'register'){
31
32      $first_name=$_POST['first_name'];
33      $last_name=$_POST['last_name'];
34
35      // check if user is already existed
36      if ($dbFunctions->isUserExist($facebook_id)) {
37          // User already exist - error response
38          $response["error"] = 2;
39          $response["error_msg"] = "User already exists";
40          echo json_encode($response);
41      } else {
42          // store user
43          $user = $dbFunctions->storeUser($facebook_id,$first_name, $last_name);
44          if ($user) {
45              // user stored successfully
46              $response["success"] = 1;
47              $response["id"] = $user["facebook_id"];
48              echo json_encode($response);
49          } else {
50              // user failed to store
51              $response["error"] = 1;
52              $response["error_msg"] = "Error occured in Registration";
53              echo json_encode($response);
54          }
55      }
56  }
```

Figure 38. Code for handling register tag

If the user application wants to register a new user, an HTTP request is made with the register tag. This request contains the default parameter as well as the *first_name* and *last_name* parameters which represent the first name and the last name of the user respectively. This information is provided by the Facebook server to the mobile application.

The *isUserExist* function is used to check if the user already exists in the database on line 36. If the user already exists, an error message is returned. Otherwise, the new user information is stored in the database using the *storeUser* function on line 43. On line 44, a conditional statement is used in check if registration was successful and if it was, a success message is returned as shown on line 51. If the registration failed, an error message is returned as shown on line 53.

.

```
57      elseif($tag == 'add'){
58
59          $food_name=$_POST['food_name'];
60          $entry_date=$_POST['entry_date'];
61          $expiry_date=$_POST['expiry_date'];
62          $reminder=$_POST['reminder'];
63
64          // add new food
65          $food = $dbFunctions->storeFood($facebook_id,$food_name,$expiry_date
66                                          ,$entry_date,$reminder);
67          if ($food) {
68              // food stored successfully
69              $response["success"] = 1;
70              $response["id"] = $food["food_id"];
71              $response["expiry_date"] = $food["expiry_date"];
72              echo json_encode($response);
73          } else {
74              // food failed to store
75              $response["error"] = 1;
76              $response["error_msg"] = $user;
77              echo json_encode($response);
78          }
79      }
```

Figure 39. Code for handling add tag

An HTTP request containing the add tag, is an intent to add a new food entry to the database. This request includes parameters for the name of the food, its entry date, its expiry date as well as the user-set reminder time, alongside the default parameters.

The *storeFood* function of the *DbFunctions* class is called to add the information about the new entry in the database as shown on line 65. All the parameters of the HTTP, except the *tag* parameter, supply the values of the required arguments for the *storeFood* function. The conditional statement on line 67 is used to check for successful database entry. A success message is returned if a new entry was made, and returns an error message to the user application if the entry failed.

```
80      elseif($tag == 'delete'){
81
82          $food_name=$_POST['food_name'];
83          $entry_date=$_POST['entry_date'];
84
85          $result = $dbFunctions->deleteFood($facebook_id,$food_name,
            $expiry_date);
86
87          if ($result) {
88                  // food deleted successfully
89                  $response["success"] = 1;
90                  echo json_encode($response);
91          }else{
92                  // food failed to delete
93                  $response["error"] = 1;
94                  $response["error_msg"] = $result;
95                  echo json_encode($response);
96              }
97
98      }
```

Figure 40. Code for handling delete tag

If the end user intends to delete a food entry, the application sends an HTTP request with the delete tag. This request contains the name of the food and the expiry date which are stored in the *food_name* and *entry_date* parameters respectively. The *deleteFood* function of the *DbFunctions* class is called to delete a certain food entry whose name and expiry date are provided, as shown on line 85. The outcome of the operation is used to check its success or failure with an IF conditional statement, as displayed on line 87 and either a success message or an error message is sent back to the user application as appropriate.

```
 99      elseif($tag == 'list'){
100
101         $food = $dbFunctions->getFoodById($facebook_id);
102         $records = array();
103
104         while($row = mysql_fetch_assoc($food)) {
105             $records[] = $row;
106         }
107
108         echo json_encode($records);
```

Figure 41. Code for handling list tag

The list tag is used if the application intends to get the list of all the food entries for a specific user from the database. The HTTP request only contains the *tag* and *facebook_id* parameters. The *getFoodById* function is called with the Facebook ID of the concerned user as its argument. The function then returns the query result which has the current food listings of the user stored on a dedicated table.

An array named *records* and a *while* statement are used to reorganise the query result into an array, which is then sent to application using JSON encoding.

The table below shows the relationship between the various kinds of tag parameter used in the HTTP request from the mobile application and the PHP classes on the server.

Table 4. Methods employed to handle different tag values

Tag Name (from HTTP request)	Value (handled by *index.php*)	Methods called from DBFunctions class (in *functions.php*)
list_tag	list	*getFoodById()*
login_tag	login	*isUserExist()*
register_tag	register	*isUserExist(),storeUser()*
add_tag	add	*storeFood()*
delete_tag	delete	*deleteFood()*

6 TESTING

6.1 Client-Side Testing

Testing on the mobile application was done primarily with the Android Virtual Device (AVD), provided by the Android SDK. It was used to test all the functionalities of the mobile application except the QR code reading functionality which was done using an Android device. The client-side testing was done primarily within the virtual device.

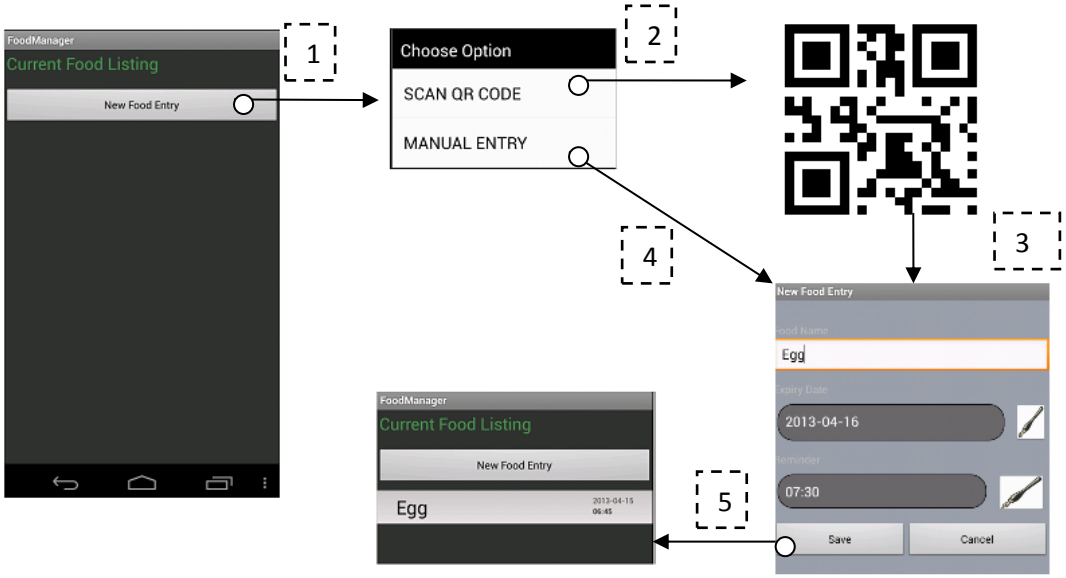

Figure 42. Application sequence for adding new entry.

The figure above depicts the sequence of registering a new entry. Sequence 1 shows the menu dialog that appears when the “Add New Entry” button is clicked. It displays the option of adding the entry either through scanning a QR code or using a manual entry. Sequences 2 and 3 show the path taken if the user decides to scan the QR code. Sequence 2 invokes the ZBarScannerLibrary to scan the QR code and the outcome, if successful, is shown in the NewEntryActivity UI in stage 3.

Sequence 4 shows an empty form on the NewEntryActivity UI for user to fill, and Sequence 5 shows the new listing on the MainActivity UI.

6.2 Server-side Testing

The testing of the server-side was done on a web browser, by making HTTP calls to the server. A new PHP file, named *test.php*, was written specifically for testing purposes. It includes the *functions.php* file to allow access to the methods defined for the *DbFunctions* class. Only the user tag is used for testing.

The *user* tag is used here for testing user login and registration. It is used alongside three other parameters which acts as a demo for the user information retrieved from the Facebook server. The Google Chrome browser was used for the tests.

Below are figures for the testing code as well as test results displayed on the browser and the phpMyAdmin interface.

```
6   echo "<html>";
7
8   $tag = $_GET['tag'];
9   $facebook_id = $_GET['facebook_id'];
10
11  if($tag == "user")
12  {
13      $first_name = $_GET['first_name'];
14      $last_name = $_GET['last_name'];
15
16      $result = $db->isUserExist($facebook_id);
17          if($result)
18              echo "<p>User ".$facebook_id." exists</p>";
19          else{
20              echo "<p>User doesn't exist, Creating New User...</p>";
21              $user = $db->storeUser($facebook_id,$first_name, $last_name);
22              if($user)
23                  echo "<p>User ".$facebook_id." has been successfully registered</p>";
24              else
25                  echo "<p>User ".$facebook_id." can't be registered</p>";
26          }
27  }
28
29  echo "</html>";
```

Figure 43. Code for testing user tag

Figure 44. First Test, Web browser showing an HTTP request to the server

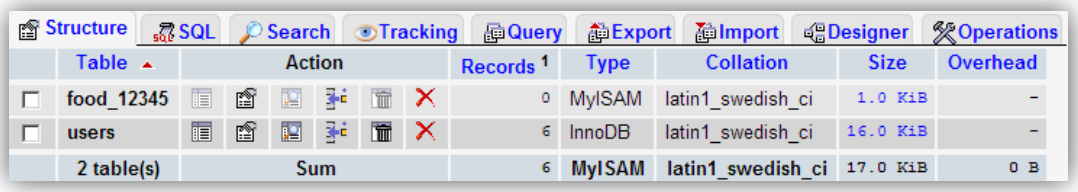

Figure 45.phpMyAdmin interface showing the list of tables.

Figure 46. Second test, Web browser showing an HTTP request to the server.

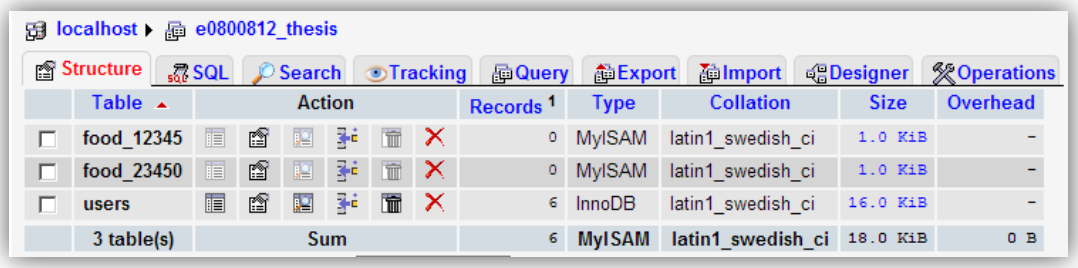

Figure 47.phpMyAdmin interface showing the result of second test.

	facebook_id	first_name	last_name	created_at
	12345	Adee	George	2013-03-14
	23450	Bayo	Ola	2013-04-08

Figure 48. Content of users table after the tests.

6.3 Overall Testing

In order to test the client-side and the server-side simultaneously, a dummy user profile created in the server-side testa was used for testing. The Guest mode in the mobile application was turned off, to allow the application interact with the server. The figure below shows the sequence of testing new food entry and other functionalities were tested likewise.

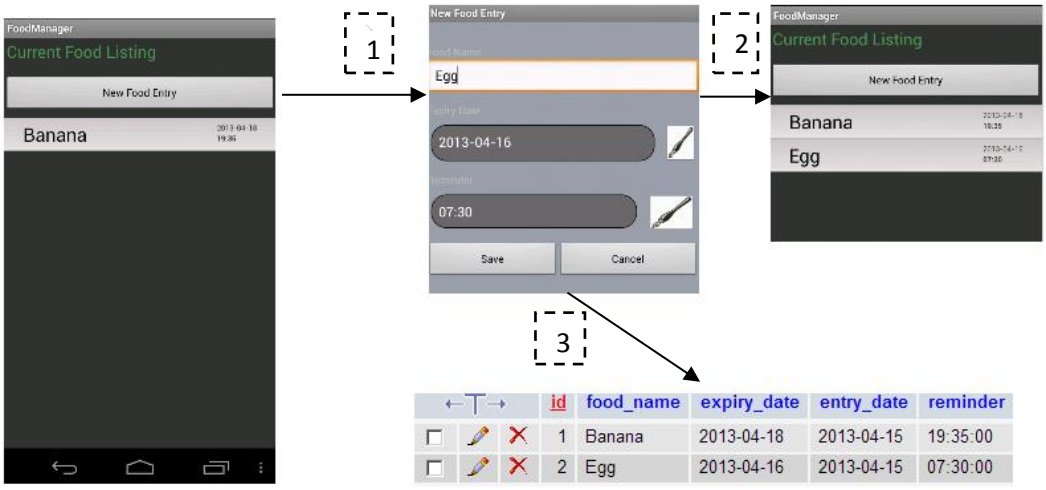

Figure 49. Application sequence for adding new entry locally and to server

Sequence 1 shows the transition to the new entry screen. Sequence 2 and 3 show the new entry updated both in the mobile application and in the database on the server. It should be noted that users are expected to have internet connection when making new entries in user mode.

7 CONCLUSION

7.1 Challenges

A major problem faced by developers for applications like this, is how to effectively manage data between the mobile devices and the server. As previously mentioned, data persistence could be done both locally and externally, hence the issue of synchronizing data while minimizing resources used becomes a serious concern.

This was resolved in this application by giving users the right to choose either to store the information locally on their devices or ability to access it on various devices by storing the information on the server. An internet connection is a requirement for the latter. Hence, it is expected that the user who chooses this option has an internet connection.

It is possible that multiple users have access to a single device. Hence, the dilemma over naming the database table in local application arose. Provision was made on the server to create separate food inventory tables for users, but that is expected for the database on the server.

It is expected that smartphones and tablets are personal items, hence there does not seem to be a need to create separate tables for each user that logs in into the application on a device, as this would be a rarity. Therefore, a single name has been chosen for the table on the mobile application, while a dynamic naming convention is applied on the server. However, if multiple users do use the application on a single device with separate login details, the application deletes the food inventory table when the user logs out and is able to retrieve pre-stored information on the server when the user logs in again.

7.2 Possible Improvements

The solution presented in this project is useful enough to combat food waste through expiration. However, it may appear cumbersome for many users to register their inventories manually into the application.

At the time of writing, there was no standard food information system on food packages that gives the user the information of both the name of the food, as well as its expiry date. The viable improvement would be get the food name from the product bar code and read the expiry date using OCR tools. However, the level of ease of using this option is only slightly greater than using the manual option of filling the food information.

Some companies have started trials with using QR code on their food packages to provide detailed information. Notwithstanding, there is still lot of hurdles to pass for it to become a standard. But for the meantime, this application presents a viable and effective solution.

8 REFERENCES

/1/ Koivupuro, Heta-Kaisa 2011, FOODSPILL – Food wastage and environmental impacts, Henvi Seminar Series, Food and Environment – Sustainable food cycle, MTT Agrifood Research Finland

/2/ Service. Official Android Developer Reference website, 5th March 2013, http://developer.android.com/reference/android/app/Service.html.

/3/ Intent. Official Android Developer Reference website, 5th March 2013, http://developer.android.com/reference/android/content/Intent.html.

/4/ Official website for Android SDK, 5th March 2013, http://developer.android.com/sdk/index.htm.

/5/ The AndroidManifest.xml File, Official Android API Guides website, 6th March 2013,

http://developer.android.com/guide/topics/manifest/manifest-intro.html.

/6/ About SQLite, Official SQLite website, 6th March 2013,

http://www.sqlite.org/about.html

/7/ GNU Operating System, GPL-Incompatible Free Software Licenses, 6th March 2013,

http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses

/8/ Official phpMyAdmin website, 6th March 2013,

http://www.phpmyadmin.net/