```java
package com.example.CaloriesCalculator;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

/**
 * The Food Data Base class
 */

public class FoodDB extends SQLiteOpenHelper {

    public FoodDB(Context context) {
        super(context, "Food", null, 1);
    }

    /**
     * This method initialize the food data base
     *
     * @param database - the food data base
     */
    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL("create table Food(IdFood text,nameOfFood text, IdCategory text, Calories text , gr text, image text)");
        ContentValues values = new ContentValues();

        values.put("IdFood", "123451");
        values.put("nameOfFood", "tomato");
        values.put("IdCategory", "vegetables");
        values.put("Calories", "78");
        values.put("gr", "100");
        database.insert("Food", null, values);

        values.put("IdFood", "123452");
        values.put("nameOfFood", "cucumber");
        values.put("IdCategory", "vegetables");
```

```
40          values.put("Calories", "60");
41          values.put("gr", "100");
42          database.insert("Food", null, values);
43
44          values.put("IdFood", "123453");
45          values.put("nameOfFood", "banana");
46          values.put("IdCategory", "fruit");
47          values.put("Calories", "90");
48          values.put("gr", "100");
49          database.insert("Food", null, values);
50
51          values.put("IdFood", "123454");
52          values.put("nameOfFood", "apple");
53          values.put("IdCategory", "fruit");
54          values.put("Calories", "80");
55          values.put("gr", "100");
56          database.insert("Food", null, values);
57
58          values.put("IdFood", "123455");
59          values.put("nameOfFood", "Chicken");
60          values.put("IdCategory", "meat");
61          values.put("Calories", "250");
62          values.put("gr", "100");
63          database.insert("Food", null, values);
64
65          values.put("IdFood", "123456");
66          values.put("nameOfFood", "meat ball");
67          values.put("IdCategory", "meat");
68          values.put("Calories", "350");
69          values.put("gr", "100");
70          database.insert("Food", null, values);
71
72          values.put("IdFood", "123457");
73          values.put("nameOfFood", "cheese");
74          values.put("IdCategory", "milk");
75          values.put("Calories", "600");
76          values.put("gr", "100");
77          database.insert("Food", null, values);
78
79          values.put("IdFood", "123458");
80          values.put("nameOfFood", "cream cheese");
```

```
 81          values.put("IdCategory", "milk");
 82          values.put("Calories", "200");
 83          values.put("gr", "100");
 84          database.insert("Food", null, values);
 85
 86          values.put("IdFood", "123459");
 87          values.put("nameOfFood", "yogurt");
 88          values.put("IdCategory", "milk");
 89          values.put("Calories", "133");
 90          values.put("gr", "100");
 91          database.insert("Food", null, values);
 92
 93          values.put("IdFood", "123460");
 94          values.put("nameOfFood", "Yogurt pro");
 95          values.put("IdCategory", "milk");
 96          values.put("Calories", "170");
 97          values.put("gr", "100");
 98          database.insert("Food", null, values);
 99
100          values.put("IdFood", "123461");
101          values.put("nameOfFood", "White cheese");
102          values.put("IdCategory", "milk");
103          values.put("Calories", "210");
104          values.put("gr", "100");
105          database.insert("Food", null, values);
106
107          values.put("IdFood", "123462");
108          values.put("nameOfFood", "Mozzarella");
109          values.put("IdCategory", "milk");
110          values.put("Calories", "250");
111          values.put("gr", "100");
112          database.insert("Food", null, values);
113
114          values.put("IdFood", "123463");
115          values.put("nameOfFood", "Mascarpone");
116          values.put("IdCategory", "milk");
117          values.put("Calories", "400");
118          values.put("gr", "100");
119          database.insert("Food", null, values);
120
121          values.put("IdFood", "123464");
```

```
122         values.put("nameOfFood", "eggplant");
123         values.put("IdCategory", "vegetables");
124         values.put("Calories", "70");
125         values.put("gr", "100");
126         database.insert("Food", null, values);
127
128         values.put("IdFood", "123465");
129         values.put("nameOfFood", "lettuce");
130         values.put("IdCategory", "vegetables");
131         values.put("Calories", "25");
132         values.put("gr", "100");
133         database.insert("Food", null, values);
134
135         values.put("IdFood", "123466");
136         values.put("nameOfFood", "pepper");
137         values.put("IdCategory", "vegetables");
138         values.put("Calories", "66");
139         values.put("gr", "100");
140         database.insert("Food", null, values);
141
142         values.put("IdFood", "123467");
143         values.put("nameOfFood", "cabbage");
144         values.put("IdCategory", "vegetables");
145         values.put("Calories", "30");
146         values.put("gr", "100");
147         database.insert("Food", null, values);
148
149         values.put("IdFood", "123468");
150         values.put("nameOfFood", "Onion");
151         values.put("IdCategory", "vegetables");
152         values.put("Calories", "40");
153         values.put("gr", "100");
154         database.insert("Food", null, values);
155
156         values.put("IdFood", "123469");
157         values.put("nameOfFood", "Carrot");
158         values.put("IdCategory", "vegetables");
159         values.put("Calories", "85");
160         values.put("gr", "100");
161         database.insert("Food", null, values);
162
```

```
163         values.put("IdFood", "123470");
164         values.put("nameOfFood", "Grapes");
165         values.put("IdCategory", "fruit");
166         values.put("Calories", "85");
167         values.put("gr", "100");
168         database.insert("Food", null, values);
169
170         values.put("IdFood", "123471");
171         values.put("nameOfFood", "Strawberries");
172         values.put("IdCategory", "fruit");
173         values.put("Calories", "120");
174         values.put("gr", "100");
175         database.insert("Food", null, values);
176
177         values.put("IdFood", "123472");
178         values.put("nameOfFood", "watermelon");
179         values.put("IdCategory", "fruit");
180         values.put("Calories", "180");
181         values.put("gr", "100");
182         database.insert("Food", null, values);
183
184         values.put("IdFood", "123473");
185         values.put("nameOfFood", "pear");
186         values.put("IdCategory", "fruit");
187         values.put("Calories", "90");
188         values.put("gr", "100");
189         database.insert("Food", null, values);
190
191         values.put("IdFood", "123474");
192         values.put("nameOfFood", "peach");
193         values.put("IdCategory", "fruit");
194         values.put("Calories", "100");
195         values.put("gr", "100");
196         database.insert("Food", null, values);
197
198         values.put("IdFood", "123475");
199         values.put("nameOfFood", "coconut");
200         values.put("IdCategory", "fruit");
201         values.put("Calories", "85");
202         values.put("gr", "100");
203         database.insert("Food", null, values);
```

```
204
205         values.put("IdFood", "123476");
206         values.put("nameOfFood", "Kiwi");
207         values.put("IdCategory", "fruit");
208         values.put("Calories", "90");
209         values.put("gr", "100");
210         database.insert("Food", null, values);
211
212         values.put("IdFood", "123477");
213         values.put("nameOfFood", "persimmon");
214         values.put("IdCategory", "fruit");
215         values.put("Calories", "110");
216         values.put("gr", "100");
217         database.insert("Food", null, values);
218
219         values.put("IdFood", "1234578");
220         values.put("nameOfFood", "");
221         values.put("IdCategory", "meat");
222         values.put("Calories", "350");
223         values.put("gr", "100");
224         database.insert("Food", null, values);
225
226         values.put("IdFood", "1234579");
227         values.put("nameOfFood", "hamburger");
228         values.put("IdCategory", "meat");
229         values.put("Calories", "500");
230         values.put("gr", "100");
231         database.insert("Food", null, values);
232
233         values.put("IdFood", "1234580");
234         values.put("nameOfFood", "Shawarma");
235         values.put("IdCategory", "meat");
236         values.put("Calories", "600");
237         values.put("gr", "100");
238         database.insert("Food", null, values);
239
240         values.put("IdFood", "1234581");
241         values.put("nameOfFood", "");
242         values.put("IdCategory", "meat");
243         values.put("Calories", "350");
244         values.put("gr", "100");
```

```
245         database.insert("Food", null, values);
246
247         values.put("IdFood", "1234582");
248         values.put("nameOfFood", "kebab");
249         values.put("IdCategory", "meat");
250         values.put("Calories", "350");
251         values.put("gr", "100");
252         database.insert("Food", null, values);
253
254         values.put("IdFood", "1234583");
255         values.put("nameOfFood", "hot dog");
256         values.put("IdCategory", "meat");
257         values.put("Calories", "250");
258         values.put("gr", "100");
259         database.insert("Food", null, values);
260
261         values.put("IdFood", "1234584");
262         values.put("nameOfFood", "chicken liver");
263         values.put("IdCategory", "meat");
264         values.put("Calories", "330");
265         values.put("gr", "100");
266         database.insert("Food", null, values);
267
268         values.put("IdFood", "1234585");
269         values.put("nameOfFood", "chicken breast");
270         values.put("IdCategory", "meat");
271         values.put("Calories", "280");
272         values.put("gr", "100");
273         database.insert("Food", null, values);
274
275         values.put("IdFood", "1234586");
276         values.put("nameOfFood", "Lamb chops");
277         values.put("IdCategory", "meat");
278         values.put("Calories", "480");
279         values.put("gr", "100");
280         database.insert("Food", null, values);
281
282         values.put("IdFood", "1234587");
283         values.put("nameOfFood", "Chicken patties");
284         values.put("IdCategory", "meat");
285         values.put("Calories", "220");
```

```
286         values.put("gr", "100");
287         database.insert("Food", null, values);
288 
289         values.put("IdFood", "1234588");
290         values.put("nameOfFood", "Meat pastry");
291         values.put("IdCategory", "meat");
292         values.put("Calories", "330");
293         values.put("gr", "100");
294         database.insert("Food", null, values);
295 
296     }
297 
298     /**
299      * The method gets a food name and returns the
    calories of the food
300      *
301      * @param food - the food name
302      * @return the calories of the food on the input
303      * if the food does not exist - return 0
304      */
305     public int caloriesByFood(String food) {
306 
307         int count = 0;
308         SQLiteDatabase db = this.getReadableDatabase
    ();
309         String pro[] = {"nameOfFood", "Calories"};
310         Cursor c = db.query("Food", pro, null, null,
     null, null, null);
311         c.moveToPosition(0);
312         if (c.getString(0).equals(food.trim())) {
313             return Integer.parseInt(c.getString(1));
314         } else {
315             while (c.moveToNext()) {
316                 Log.i("what c", c.getString(0));
317                 if (c.getString(0).equals(food.trim(
    ))) {
318                     return Integer.parseInt(c.
    getString(1));
319                 }
320             }
321         }
```

```
322             return 0;
323         }
324
325
326         /**
327          * The method gets a name of category and
    returns all the food from that category
328          *
329          * @param category - category of food
330          * @return cursor of the food from a specific
    category
331          */
332
333         public Cursor allFoodBycategort(String category)
     {
334             SQLiteDatabase db = this.getReadableDatabase
    ();
335             Log.i("cat", category);
336             String pro[] = {"IdFood", "nameOfFood", "
    IdCategory", "Calories", "gr"};
337             Cursor c;
338             if (category.equals("milk")) {
339                 c = db.rawQuery(
340                         "SELECT IdFood, nameOfFood From
    Food WHERE IdCategory = 'milk'",
341                         null);
342             } else if (category.equals("meat")) {
343                 c = db.rawQuery(
344                         "SELECT IdFood, nameOfFood From
    Food WHERE IdCategory = 'meat'",
345                         null);
346             } else if (category.equals("fruit")) {
347                 c = db.rawQuery(
348                         "SELECT IdFood, nameOfFood From
    Food WHERE IdCategory = 'fruit'",
349                         null);
350             } else {
351                 c = db.rawQuery(
352                         "SELECT IdFood, nameOfFood From
    Food WHERE IdCategory = 'vegetables'",
353                         null);
```

```
354         }
355
356         return c;
357     }
358
359     /**
360      * That method is called when version of DB
    changed
361      *
362      * @param db         - the food data base
363      * @param oldVersion - the old version of the
    data base
364      * @param newVersion - the new version of the
    data base
365      */
366
367     @Override
368     public void onUpgrade(SQLiteDatabase db, int
    oldVersion, int newVersion) {
369         db.execSQL("DROP TABLE IF EXISTS Food");
370         onCreate(db);
371     }
372
373 }
```

```java
package com.example.CaloriesCalculator;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * The Meals Data Base class
 */
public class MealsDB extends SQLiteOpenHelper {


    public MealsDB(Context context) {
        super(context, "Meals1", null, 1);
    }

    /**
     * This method initialize the meals data base
     *
     * @param database - the meals data base
     */
    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL("create table Meals1(IdUser text,MealTime text,nameOfFood text,Date text,Calories text)");
        ContentValues values = new ContentValues();
    }

    /**
     * That method is called when version of DB changed
     *
     * @param db         - the meals data base
```

```
39      * @param oldVersion - the old version of the
   data base
40      * @param newVersion - the new version of the
   data base
41      */
42     @Override
43     public void onUpgrade(SQLiteDatabase db, int
   oldVersion, int newVersion) {
44         db.execSQL("DROP TABLE IF EXISTS Meals1");
45         onCreate(db);
46     }
47
48     /**
49      * The method gets a user id and returns all his
   meals from today
50      *
51      * @param idUser - the id of a specific user
52      * @return string of ListOfBreakfast, ListOfLunch
   ,  ListOfDinner, ListOfSnacks
53      */
54     public String allMealsPerUserTime(final String
   idUser) {
55         SQLiteDatabase dbMeals = this.
   getReadableDatabase();
56         String pro[] = {"IdUser", "MealTime", "
   nameOfFood", "Date"};
57         Cursor c = dbMeals.query("Meals1", pro, null,
    null, null, null, null);
58         c.moveToPosition(0);
59         String listOfBreakfast = "";
60         String listOfLunch = "";
61         String listOfDinner = "";
62         String listOfSnacks = "";
63         DateFormat dateFormat = new SimpleDateFormat(
   "yyyy/MM/dd");
64         Date date = new Date();
65
66         try {
67             if (c.isNull(0) == false) {
68                 Log.i("firstmeal1", c.getString(0) +
   " " + c.getString(1) + " " + c.getString(2) + " " + c
```

```
68 .getString(3));
69                 Log.i("id user", idUser);
70                 Log.i("time", dateFormat.format(date
   ));
71                 if (c.getString(0).equals(idUser) &&
    c.getString(3).equals(dateFormat.format(date))) {
72                     Log.i("firstmeal2", c.getString(
   0) + " " + c.getString(1) + " " + c.getString(2) +
   " " + c.getString(3));
73                     switch (c.getString(1)) {
74                         case "Breakfast":
75                             listOfBreakfast =
   listOfBreakfast + c.getString(2) + ",";
76                             break;
77                         case "Lunch":
78                             listOfLunch =
   listOfLunch + c.getString(2) + ",";
79                             break;
80                         case "Dinner":
81                             listOfDinner =
   listOfDinner + c.getString(2) + ",";
82                             break;
83                         case "Snacks":
84                             listOfSnacks =
   listOfSnacks + c.getString(2) + ",";
85                             break;
86                     }
87                 }
88                 while (c.moveToNext()) {
89                     Log.i("firstmealbeforif", c.
   getString(0) + " " + c.getString(1) + " " + c.
   getString(2) + " " + c.getString(3));
90                     if (c.getString(0).equals(idUser
   ) && c.getString(3).equals(dateFormat.format(date))
    {
91                         Log.i("firstmealinwhile", c.
   getString(0) + " " + c.getString(1) + " " + c.
   getString(2) + " " + c.getString(3));
92                         switch (c.getString(1)) {
93                             case "Breakfast":
94                                 listOfBreakfast =
```

```
 94 listOfBreakfast + c.getString(2) + ",";
 95                                     break;
 96                                 case "Lunch":
 97                                     listOfLunch =
    listOfLunch + c.getString(2) + ",";
 98                                     break;
 99                                 case "Dinner":
100                                     listOfDinner =
    listOfDinner + c.getString(2) + ",";
101                                     break;
102                                 case "Snacks":
103                                     listOfSnacks =
    listOfSnacks + c.getString(2) + ",";
104                                     break;
105                             }
106                         }
107                     }
108                 }
109             } catch (Exception e) {
110             }
111             Log.i("break", listOfBreakfast);
112             Log.i("lunch", listOfLunch);
113             Log.i("dinner", listOfDinner);
114             Log.i("snacks", listOfSnacks);
115             return listOfBreakfast + "?" + listOfLunch +
     "?" + listOfDinner + "?" + listOfSnacks;
116         }
117
118         /**
119          * The method gets a user id and returns the
    number of calories he ate today
120          *
121          * @param idUser - id of a specific user
122          * @return Calories - string of the number of
    calories the user ate until now
123          */
124         public String caloriesAllMealsPerDay(final
    String idUser) {
125             DateFormat dateFormat = new SimpleDateFormat
    ("yyyy/MM/dd");
126             Date date = new Date();
```

```
127
128         SQLiteDatabase dbMeals = this.
    getReadableDatabase();
129         String pro[] = {"IdUser", "Date", "Calories"
    };
130         Cursor c = dbMeals.query("Meals1", pro, null
    , null, null, null, null);
131         c.moveToPosition(0);
132         int calories = 0;
133         try {
134             if (c.isNull(0) == false) {
135                 if (c.getString(0).equals(idUser) &&
     c.getString(1).equals(dateFormat.format(date))) {
136                     calories = calories + Integer.
    parseInt(c.getString(2));
137                 }
138                 while (c.moveToNext()) {
139                     if (c.getString(0).equals(idUser
    ) && c.getString(1).equals(dateFormat.format(date)))
     {
140                         calories = calories +
    Integer.parseInt(c.getString(2));
141                     }
142                 }
143
144             }
145         } catch (Exception e) {
146         }
147         return Integer.toString(calories);
148     }
149
150 }
```

```java
package com.example.CaloriesCalculator;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

/**
 * The User Data Base class
 */
public class UsersDB extends SQLiteOpenHelper {

    public UsersDB(Context context) {
        super(context, "UsersDB", null, 1);
    }

    /**
     * This method initialize the users data base
     *
     * @param db - the user data base
     */
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table UsersDB (sid text, sname text,sweight text,sheight text,sgender text, sage text)");
        ContentValues values = new ContentValues();
        values.put("sid", "123456");
        values.put("sname", "mor yemin");
        values.put("sweight", "60");
        values.put("sheight", "170");
        values.put("sgender", "female");
        values.put("sage", "20");
        db.insert("UsersDB", null, values);

        values.put("sid", "112233");
        values.put("sname", "mor mor");
        values.put("sweight", "60");
        values.put("sheight", "170");
```

```
40          values.put("sgender", "female");
41          values.put("sage", "20");
42          db.insert("UsersDB", null, values);
43
44      }
45
46      /**
47       * That method is called when version of DB
   changed
48       *
49       * @param db         - the user data base
50       * @param oldVersion - the old version of the
   data base
51       * @param newVersion - the new version of the
   data base
52       */
53      @Override
54      public void onUpgrade(SQLiteDatabase db, int
   oldVersion, int newVersion) {
55          db.execSQL("drop table if exists UsersDB");
56          onCreate(db);
57
58      }
59
60      /**
61       * The method returns all the users from the data
    base
62       *
63       * @return Cursor of all the users in the DB
64       */
65      public Cursor allusers() {
66          SQLiteDatabase db = this.getReadableDatabase(
   );
67          String pro[] = {"sid", "sname", "sweight", "
   sheight", "sgender", "sage"};
68          Cursor c = db.query("UsersDB", pro, null,
   null, null, null, null);
69          c.moveToPosition(0);
70          Log.i("value of the id in db1", c.getString(0
   ));
71          return c;
```

```
72         }
73
74  }
```

```java
package com.example.CaloriesCalculator;

import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.util.Log;
import android.webkit.WebChromeClient;
import android.webkit.WebView;
import android.widget.Toast;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * The Main Activity class
 *
 * @author Chen ravia - 314884412
 * @author Moshe Davila - 308459841
 * @author Mor Yemin - 316046093
 */
public class MainActivity extends AppCompatActivity {

    UsersDB dbUsers;
    FoodDB dbFood;
    MealsDB dbMeals;

    /**
     * The Model class
     */
    class Model {

        /**
         * The method gets a string of food names and
```

```
41   calculate their total calories
42            *
43            * @param FoodName - string of food named
    that the client marked
44            * @return the total calories of the food the
     client marked
45            */
46
47          public String calculateCalArray(String
    FoodName) {
48              return Integer.toString(dbFood.
    caloriesByFood(FoodName));
49          }
50
51          /**
52           * The method gets a category and returns a
    list of food by the category
53           *
54           * @param Category - category of food
55           * @return arrayFood - array of food by the
    category
56           */
57          public ArrayList listOfFood(String Category)
    {
58              Cursor cursorFood = dbFood.
    allFoodBycategort(Category);
59              ArrayList<String> arrayFood = new
    ArrayList<String>();
60              while (cursorFood.moveToNext()) {
61                  Log.i("food ", cursorFood.getString(1
    ));
62                  arrayFood.add(cursorFood.getString(1)
    );
63              }
64              return arrayFood;
65          }
66
67
68          /**
69           * The method gets a user details and add him
     to the users data base
```

```
 70          *
 71          * @param id     - the user's id
 72          * @param name   - the user's name
 73          * @param weight - the user's weight
 74          * @param height - the user's height
 75          * @param gender - the user's gender
 76          * @param age    - the user's age
 77          */
 78         public void AddUser(final String id, final
    String name, final String weight, final String
    height, final String gender, String age) {
 79             SQLiteDatabase dataBaseUsers = dbUsers.
    getWritableDatabase();
 80             ContentValues values = new ContentValues
    ();
 81
 82             values.put("sid", id);
 83             values.put("sname", name);
 84             values.put("sweight", weight);
 85             values.put("sheight", height);
 86             values.put("sgender", gender);
 87             values.put("sage", age);
 88             dataBaseUsers.insert("UsersDB", null,
    values);
 89
 90         }
 91
 92         /**
 93          * The method gets a meal details and add it
     to the meals data base
 94          *
 95          * @param foodName - the name of the food on
     the meal
 96          * @param idUser   - the id of the user that
     add the meal
 97          * @param MealTime - the time of the meal
 98          * @param Calories - the calories of the
    food on the meal
 99          */
100
101         public void AddMeal(final String foodName,
```

```
101 final String idUser, final String MealTime, final
    String Calories) {
102             DateFormat dateFormat = new
    SimpleDateFormat("yyyy/MM/dd");
103             Date date = new Date();
104
105             SQLiteDatabase dataB = dbMeals.
    getWritableDatabase();
106             ContentValues values = new ContentValues
    ();
107             values.put("IdUser", idUser);
108             values.put("MealTime", MealTime);
109             values.put("nameOfFood", foodName);
110             values.put("Date", dateFormat.format(
    date));
111             values.put("Calories", Calories);
112             dataB.insert("Meals1", null, values);
113             Log.i("values", values.toString());
114
115         }
116
117         /**
118          * The method returns all the users on the
    database
119          *
120          * @return Cursor of the users
121          */
122         public String getUsers() {
123             Log.i("dbUsers", "IN GET USRERS");
124             Cursor cursor = dbUsers.allusers();
125             Log.i("dbUsers", cursor.getString(0));
126             return cursor.getString(0);
127         }
128
129         /**
130          * The method gets a user id and check if
    the user exist
131          *
132          * @param id - the user's id
133          * @return true if the user exist on the
    database , false if it not exist
```

```
134          */
135         public boolean ifUserExist(String id) {
136             Log.i("id input ", id);
137             Cursor cursor = dbUsers.allusers();
138             Log.i("firstposition", cursor.getString(
    0));
139             if (cursor.getString(0).equals(id))
140                 return true;
141             else {
142                 while (cursor.moveToNext()) {
143                     Log.i("firstposition", cursor.
    getString(0));
144 
145                     if (cursor.getString(0).equals(
    id))
146                         return true;
147                 }
148             }
149             return false;
150         }
151 
152     }
153 
154     /**
155      * The View Model class
156      * links between the view and the model
157      */
158 
159     class ViewModel {
160 
161         WebView webView;
162         Model model;
163         ExecutorService pool;
164 
165         public ViewModel(WebView webView, Model
    model) {
166             this.webView = webView;
167             this.model = model;
168             pool = Executors.newFixedThreadPool(4);
169         }
170 
```

```
171         /**
172          * The method links between the view and the
    model
173          * The method gets a category name and sends
    it to the 'listOfFood' method
174          * on the model in another thread
175          *
176          * @param Category - category of food
177          */
178         @android.webkit.JavascriptInterface
179         public void getFood(final String Category) {
180             final ArrayList food = model.listOfFood(
    Category);
181             // calling geyUser on model in another
    thread
182             pool.submit(new Runnable() {
183                 public void run() {
184
185
186                     runOnUiThread(new Runnable() {
187                         public void run() {
188                             webView.
    evaluateJavascript("printFood('" + food + "')", null
    );
189                         }
190
191                     });
192
193                 }
194
195             });
196         }
197
198         /**
199          * The method links between the view and the
    model
200          * The method gets a user details and sends
    them to the 'AddUser' method on the model
201          * in another thread
202          *
203          * @param id     - the user's id
```

```
204             * @param name   - the user's name
205             * @param weight - the user's weight
206             * @param height - the user's height
207             * @param gender - the users gender
208             * @param age    - the users age
209             */
210            @android.webkit.JavascriptInterface
211            public void putUsers(final String id, final
    String name, final String weight, final String
    height, final String gender, final String age) {
212                // calling geyUser on model in another
    thread
213                pool.submit(new Runnable() {
214                    public void run() {
215                        Log.i("mvvm", "view model");
216
217                        runOnUiThread(new Runnable() {
218                            public void run() {
219                                model.AddUser(id, name,
    weight, height, gender, age);
220                                Toast.makeText(
    getApplicationContext(), "You have successfully
    registered", Toast.LENGTH_SHORT).show();
221                            }
222
223                        });
224
225                    }
226
227                });
228            }
229
230            /**
231             * The method links between the view and the
     model
232             * The method gets a meal details and sends
    them to the 'calculateCalArray' method
233             * on the model in another thread
234             *
235             * @param selctedFood - the selected food
236             * @param idUser      - user's id
```

```
237             * @param MealTime     - the meal's time
238             */
239            @android.webkit.JavascriptInterface
240            public void AddMeals(final String[]
    selctedFood, final String idUser, final String
    MealTime) {
241                for (int i = 0; i < selctedFood.length;
    i++) {
242                    final String calc = model.
    calculateCalArray(selctedFood[i]);
243                    Log.i("select food ", selctedFood[i]
    );
244                    Log.i("id user  ", idUser);
245                    Log.i("meal time ", MealTime);
246                    Log.i("calc ", calc);
247                    model.AddMeal(selctedFood[i], idUser
    , MealTime, calc);
248                }
249                pool.submit(new Runnable() {
250                    final String calories = dbMeals.
    caloriesAllMealsPerDay(idUser);
251                    final String AllMeals = dbMeals.
    allMealsPerUserTime(idUser);
252
253                    public void run() {
254
255                        runOnUiThread(new Runnable() {
256                            public void run() {
257                                Toast.makeText(
    getApplicationContext(), "The meal successfully
    added!", Toast.LENGTH_SHORT).show();
258                                if (Integer.parseInt(
    calories) > 1500) {
259                                    Toast.makeText(
    getApplicationContext(), "NOTICE! you have passed
    the recommended calories", Toast.LENGTH_SHORT).show(
    );
260                                }
261                                webView.
    evaluateJavascript("MealAdded('" + "Meal Added!" +
    "+" + calories + "+" + AllMeals + "')", null);
```

```
262                             }
263
264                         });
265
266                     }
267
268                 });
269             }
270
271             /**
272              * The method links between the view and the
    model
273              * The method gets a user id and sends it to
    the 'allMealsPerUserTime' method
274              * on the meals data base in another thread
275              *
276              * @param idUser - user's id
277              */
278             @android.webkit.JavascriptInterface
279             public void AllMealsPerUser(final String
    idUser) {
280                 pool.submit(new Runnable() {
281                     final String allMeals = dbMeals.
    allMealsPerUserTime(idUser);
282
283                     public void run() {
284                         runOnUiThread(new Runnable() {
285                             public void run() {
286                                 webView.
    evaluateJavascript("GoBack2('" + allMeals + "')",
    null);
287                             }
288
289                         });
290
291                     }
292
293                 });
294             }
295
296             /**
```

```
297          * The method links between the view and the
     model
298          * The method gets a user id and sends it to
     the 'caloriesAllMealsPerDay' method
299          * on the meals data base in another thread
300          *
301          * @param idUser - user's id
302          */
303         @android.webkit.JavascriptInterface
304         public void CalcAll(final String idUser) {
305             pool.submit(new Runnable() {
306                 final String calories = dbMeals.
    caloriesAllMealsPerDay(idUser);
307
308                 public void run() {
309                     runOnUiThread(new Runnable() {
310                         public void run() {
311                             webView.
    evaluateJavascript("getAllCalories('" + calories +
    "')", null);
312                         }
313
314                     });
315
316                 }
317
318             });
319         }
320
321         /**
322          * The method checks if the user exist in
    the Data Base when the user
323          * tries to register
324          *
325          * @param id - user's id
326          */
327         @android.webkit.JavascriptInterface
328         public void registerUserExist(final String
    id) {
329             final boolean exist = model.ifUserExist(
    id);
```

```
330             Log.i("userexist", String.valueOf(exist)
    );
331             runOnUiThread(new Runnable() {
332                 public void run() {
333                     if (exist == true) {
334                         Toast.makeText(
    getApplicationContext(), "The ID is already exist,
    please try again", Toast.LENGTH_SHORT).show();
335                         webView.evaluateJavascript("
    AddUser('" + "true" + "')", null);
336                     } else {
337                         webView.evaluateJavascript("
    AddUser('" + "false" + "')", null);
338                     }
339                 }
340
341             });
342         }
343
344         /**
345          * The method get the user inputs and check
    validation
346          *
347          * @param id     - user's ID
348          * @param weight - user's weight
349          * @param height - user's height
350          */
351         @android.webkit.JavascriptInterface
352         public void validInputs(final String id,
    final String weight, final String height) {
353             boolean isValid = true;
354             if (Integer.parseInt(id) > 999999999 ||
    Integer.parseInt(id) < 100000000) {
355                 isValid = false;
356                 Log.i("id not valid ", id);
357                 Toast.makeText(getApplicationContext
    (), "The ID is not valid , please enter 9 digits",
    Toast.LENGTH_SHORT).show();
358             }
359             if (Integer.parseInt(weight) > 250 ||
    Integer.parseInt(weight) < 40) {
```

```
360                     isValid = false;
361                     Log.i("weight not valid ", weight);
362                     Toast.makeText(getApplicationContext
    (), "Your weight is not valid!please enter weight
    between 40 - 250", Toast.LENGTH_SHORT).show();
363                 }
364                 if (Integer.parseInt(height) > 220 ||
    Integer.parseInt(height) < 120) {
365                     isValid = false;
366                     Log.i("height is not valid", height)
    ;
367                     Toast.makeText(getApplicationContext
    (), "Your height is not valid!please enter height
    between 120-220", Toast.LENGTH_SHORT).show();
368                 }
369                 if (isValid == true) {
370                     Log.i("all valid", id + "," + weight
     + "," + height);
371                 }
372                 final boolean isValid2 = isValid;
373                 runOnUiThread(new Runnable() {
374                     public void run() {
375                         if (isValid2 == true) {
376                             webView.evaluateJavascript("
    IfvalidUser('" + "true" + "')", null);
377                         } else {
378                             webView.evaluateJavascript("
    IfvalidUser('" + "false" + "')", null);
379                         }
380                     }
381
382                 });
383
384             }
385
386             /**
387              * The method links between the view and the
     model
388              * The method gets a user id and sends it to
     the 'ifUserExist' method on the model
389              * in another thread
```

```
390             *
391             * @param id - user's id
392             */
393
394            @android.webkit.JavascriptInterface
395            public void fetchUsers(final String id) {
396                Log.i("mvvm", "inside the fetchusers");
397                // calling geyUser on model in another
    thread
398                pool.submit(new Runnable() {
399                    final String calories = dbMeals.
    caloriesAllMealsPerDay(id);
400                    final String allMeals = dbMeals.
    allMealsPerUserTime(id);
401
402                    public void run() {
403                        Log.i("mvvm", "after the
    function inside the run");
404
405                        final boolean exist = model.
    ifUserExist(id);
406                        runOnUiThread(new Runnable() {
407                            public void run() {
408                                if (exist == true) {
409                                    Toast.makeText(
    getApplicationContext(), "Welcome!", Toast.
    LENGTH_SHORT).show();
410                                    if (Integer.parseInt
    (calories) > 1500) {
411                                        Toast.makeText(
    getApplicationContext(), "NOTICE! you have passed
    the recommended calories", Toast.LENGTH_SHORT).show(
    );
412                                    }
413                                    webView.
    evaluateJavascript("checkIfUserExist('" + "true" +
    "+" + calories + "+" + allMeals + "')", null);
414                                } else {
415                                    Toast.makeText(
    getApplicationContext(), "Please Register!", Toast.
    LENGTH_SHORT).show();
```

```
416                                     webView.
    evaluateJavascript("checkIfUserExist('" + "false" +
    "')", null);
417                                 }
418                             }
419
420                         });
421
422                     }
423
424                 });
425             }
426         }
427
428         /**
429          * This method initialize the activity
430          *
431          * @param savedInstanceState
432          */
433
434         @SuppressLint("SetJavaScriptEnabled")
435         @Override
436         protected void onCreate(Bundle
    savedInstanceState) {
437             super.onCreate(savedInstanceState);
438             // creating model
439             Model model = new Model();
440             dbUsers = new UsersDB(this);
441             dbFood = new FoodDB(this);
442             dbMeals = new MealsDB(this);
443             // creating the webView object
444             WebView webView = new WebView(this);
445             setContentView(webView);
446             webView.getSettings().setJavaScriptEnabled(
    true);
447             webView.setWebChromeClient(new
    WebChromeClient());
448             webView.loadUrl("file:///android_asset/login
    .html");
449
450             // creating the viewModel
```

```
451             ViewModel vm = new ViewModel(webView, model)
    ;
452
453             // attaching viewModel object to web view
454             webView.addJavascriptInterface(vm, "vm");
455
456         }
457
458
459 }
```