Table of Contents

1 INTRODUCTION

A farmer's market is a physical retail market place intended to sell foods directly by farmers to consumers. The Online Farmers Market is the web based android shopping system for fruits, vegetables and flowers which are grown by farmers. All the products which are grown in the fields are directly available here which are fresh from the fields. This application helps the farmers to sell their goods to customers directly.

By doing this we can reduce the margin of the middle men so that the price of the products will be cheaper compared to the local offline market. The main motto here is to get the customers the fresh goods from the farmers directly the customers. This application helps the registered users to login themselves and shop for the particular product. The user can order multiple products at one time.

In this project provides the rich user interface for the customers. The Customers can order the goods for less price. The users can order multiple products at once so that they can receive all the products which they want at one shot. The user needs to be registered before login and after login they can view the products and the products which they need can be ordered.

The system provides the various payment options which the users can use at the time of ordering. The application is designed into two modules first is for customers who is willing to buy the products. Second is the admin who will maintain the product details, user details and other shipment information.

All the products available from our application will be undergoing through the quality check once the products pass the quality check will be delivered to the customers.

1.1 OBJECTIVES

Some of the objectives are listed below

- The main aim of this product is to get the exact price to the farmers for goods which they produce.
- This product eliminates the middle man in the selling business by this the farmers can get the actual amount which he is giving to consumers.
- The application provides the rich user interface for the users so that they can access the application with limited knowledge on mobiles.
- The user information which we are taking from the users is completely safe and we will not be giving the details to anyone.

1.2 SCOPE

This system has been implemented to any of the farmer or in one system multiple farmers join together and get the different products together and they can sell here. This helps to get the best price for the goods they produce. In the traditional way the markets will not operate 24*7. By using this application for selling the goods they can get the market which is 24*7.

The farmers are directly selling their products to the particular customers. The products which are delivered are the once which has undergone quality checked so that the customers can get the quality products.

The rich user interface helps the customer as well as the Farmers to easily use the application with minimum mobile knowledge. The easy access of the food items will be available in your fingertips easily with our application.

2 HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Hardware Requirements

Processor:	Pentium 4 or above
RAM:	4GB or more
Hard Disk:	500 GB or more
Monitor:	15” CRT, or LCD monitor
Key Board:	Normal or Multimedia
Mouse:	Compatible mouse

2.2 Software Requirements

Operating system:	Windows 7 or above.
Front End:	Java
Back End:	DB SQLite
IDE:	Android Studio.

3 SYSTEM REQUIREMENT SPECIFICATIONS

3.1 FUNCTIONAL REQUIREMENTS

A functional requirement in system engineering and software development is a document that specifies the functions that a system or component must perform.

3.2 MODULES DESCRIPTION

3.2.1 USER

3.2.1.1 Register Module

The registration function shall allow users to create secure accounts. The account will take the username, password, address, city, postal code.

This provides security to the account member by setting up an account that is password protected. This also offers convenience so the user only has to enter the information listed above once and then it is stored in the account.

3.2.1.2 Login Module

The account login functionality allows registered users to enter their user name and password. Once verified, users will be able to access account history, view products, place order, cancel order and update their account information.

This provides a method by which the users can access the restricted operations.

3.2.1.3 Order history

The customer can view the order details, like previous order date, time and the date of delivery of ordered products. These details be displayed for the orders with the status pending and delivered. For orders which are canceled those details will not be displayed for the customer.

3.2.1.4 View products

The user can view the products added by the admin and he can select that particular product to proceed with ordering. Once the order has been selected the system will as the customer to give quantity as an input after that the total price will be calculated and it will be displayed. The user can see the product details like the price of the product, the image of the product.

3.2.1.5 Place order

Here the user can place order, in the previous module the user will add some products into the cart the products which are added into the cart were displayed hear the user can see the products list hear and the quantity of the products. He can delete if any products are not necessary means after conforming the order list then he can place the order.

3.2.1.6 Cancel order

The cancel order functionality is to help the customers to cancel their particular order in order of their absence or if they placed the order by mistake. This functionality allows the user to select on the particular product which they want to cancel and select the cancel button.

3.2.2 Admin (Farmer)

3.2.2.1 Register Module

The registration function shall allow users to create secure accounts. The account will takes the, username, password, address, city, postal code and other credentials in order to register with the farmer's market system.

This provides security to the account member by setting up an account that is password protected. This also offers convenience so the user only has to enter the information listed above once and then it is stored in the account.

3.2.2.2 Login Module

The account login functionality allows registered users to enter their user name and password. Once verified, users will be able to access account history, view products, place order, cancel order and update their account information.

This provides a method by which the users can access the restricted operations.

3.2.2.3 Order details

The admin can see the all order details of every user. The admin can see the information like customer details the previous orders the customers, address and the contact information of the customer. Once the product has been delivered then that particular admins orders will be displayed.

3.2.2.4 Delivered

Once the order has been received by the customer the admin can change the status from processing to deliver. The option of changing the status is allowed to only admin. Once the status has been changed the customer can see the status which will be reflected in the customer order details.

3.3 NON-FUNCTIONAL REQUIREMENTS

3.3.1 Usability requirements

The system should have an attractive, user friendly and interactive graphical user interface and it should be easy to use even with the person with least knowledge of computers.

3.3.2 Reliability requirements

The Performance and response rate of the system should remain constant even as the number of concurrent users or data levels increase. Architecture used to build the system should be flexible enough to allow integration with other systems if need be in the future.

3.3.3 Security requirements

This system must be highly secured in the login part. This is because some of the privileges are only allowed for the admin level.

3.3.4 Portability requirements

The system needs to be portable on all major platforms. This system should not be restricted by any specific technology such as database and operating system.

4 SYSTEM ANALYSIS

4.1 EXISTING SYSTEM

The complete Farmers market process is done manually. The farmers will not get the proper price for the goods which they have produced. This leads to the disappointment of the farmer. The completed selling of the goods is done through the middle man. The middle man gets the maximum profits compared to the farmer.

4.1.1 LIMITATIONS OF EXISTING SYSTEM

- Work is done manually by all the faculties in maintaining database.
- Time consuming.
- If the farmers are not getting the desired price then also they are forced to sell their products because of the debt they are in while growing the crops.
- Farmers are not getting the proper profits for their products.

4.2 PROPOSED SYSTEM

- In this system the complete process is online.
- With the help of our app the customer can get their products delivered to their home.
- Farmers can also sell their products easily for the proper market price
- The application allows the user

4.2.1 ADVANTAGES OF PROPOSED SYSTEM

- Rich user interface provides the users both admin and customer to interact easily with the application.
- Customer can order the products without any limit. The delivery charges are collected through cash after delivery.
- Customer can easily order and cancel the order if that product is not needed for him.
- The admin is the one responsible delivering the order once the product has been delivered then the user can search for that particular order and then confirm that the product has been delivered.
- Customer shopping experience will be increased drastically because the time take for the getting the product is easy and they don’t have to check the goods before buying like in offline market.

5 SYSTEM DESIGN

System Design is a modelling process. It can be defined as a transaction from user's view to programmers or the database view. The purpose of design phase is to plan a solution for problem specified by the requirements. System design, specification of those modules and how they interact with each other to produce the results. The goal of the design process is to produce a model or representation of a system can be used later to build that system. The produced module is called design of system

The term design is used in two ways:

- Used as noun
- Represents the result of the design process

5.1 CONTEXT DESIGN / LEVEL 0 DIAGRAM

The context diagram shows the system under consideration as a single high-level process and then shows the relationship that system as with other external entities. It is also called as Level 0 diagram. It defines the boundary between the system, or part of the system and its environment, showing the entities that interact with it.

5.2 ZERO LEVEL DFD

FIG1.1 Context diagram of the system

5.3 USE-CASE DIAGRAM

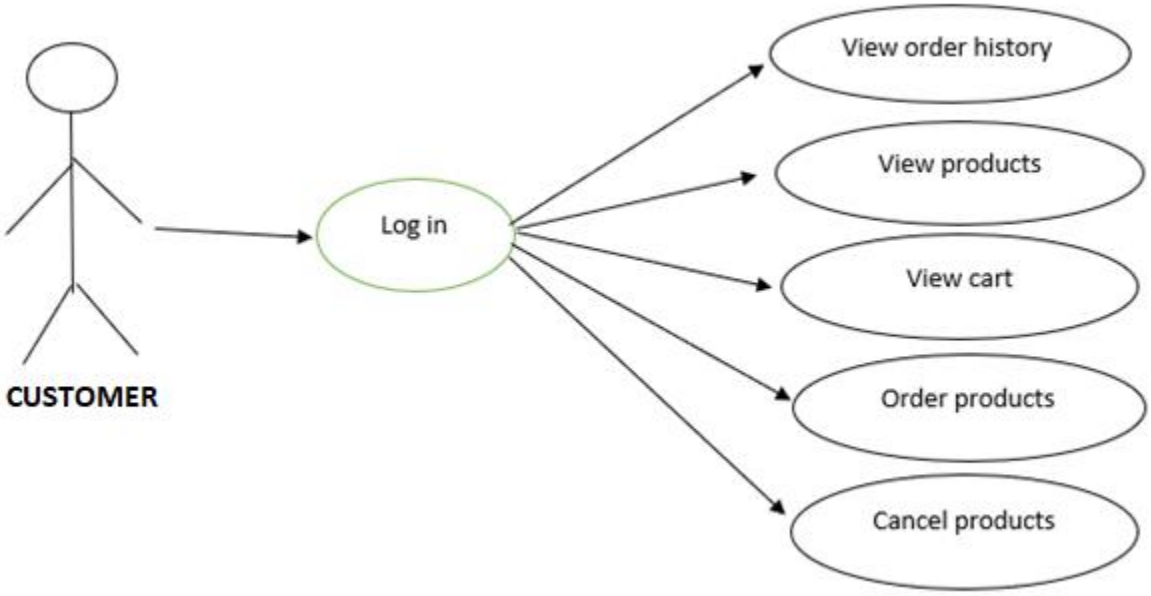


FIG 1.1 Use Case Diagram of User

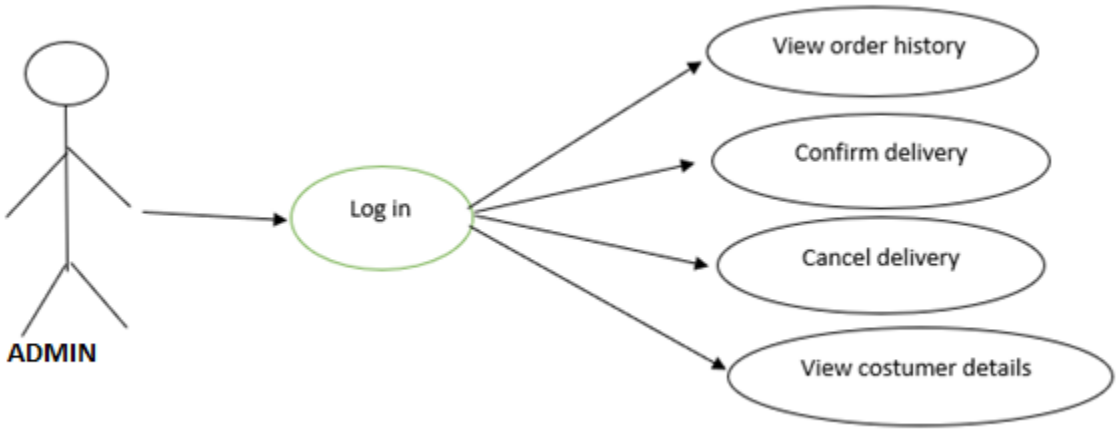


FIG 1.2 Use Case Diagram of Clerk

5.4 ENTITY-RELATIONSHIP DIAGRAM

The E-R data model is based on the perception of a real world, which consists of a set of basis objects called entities and relationships. This entity relationship modelling, along with sophisticated case tools can provide an effective and timely means of defining, controlling and definition of information needs. This diagram models the data and depicts how the business views its structure. An entity is defined as an item of interest to the enterprise. These must have occurrences, which can be uniquely identified from each other.

5.4.1 E-R DIAGRAM

Entity relationship diagram displays the relationships of entity set stored in a database. In this application we have 2 tables they are Rescue details and User details. Rescue details consist of 8 attributes and user details consist of 4 attributes.

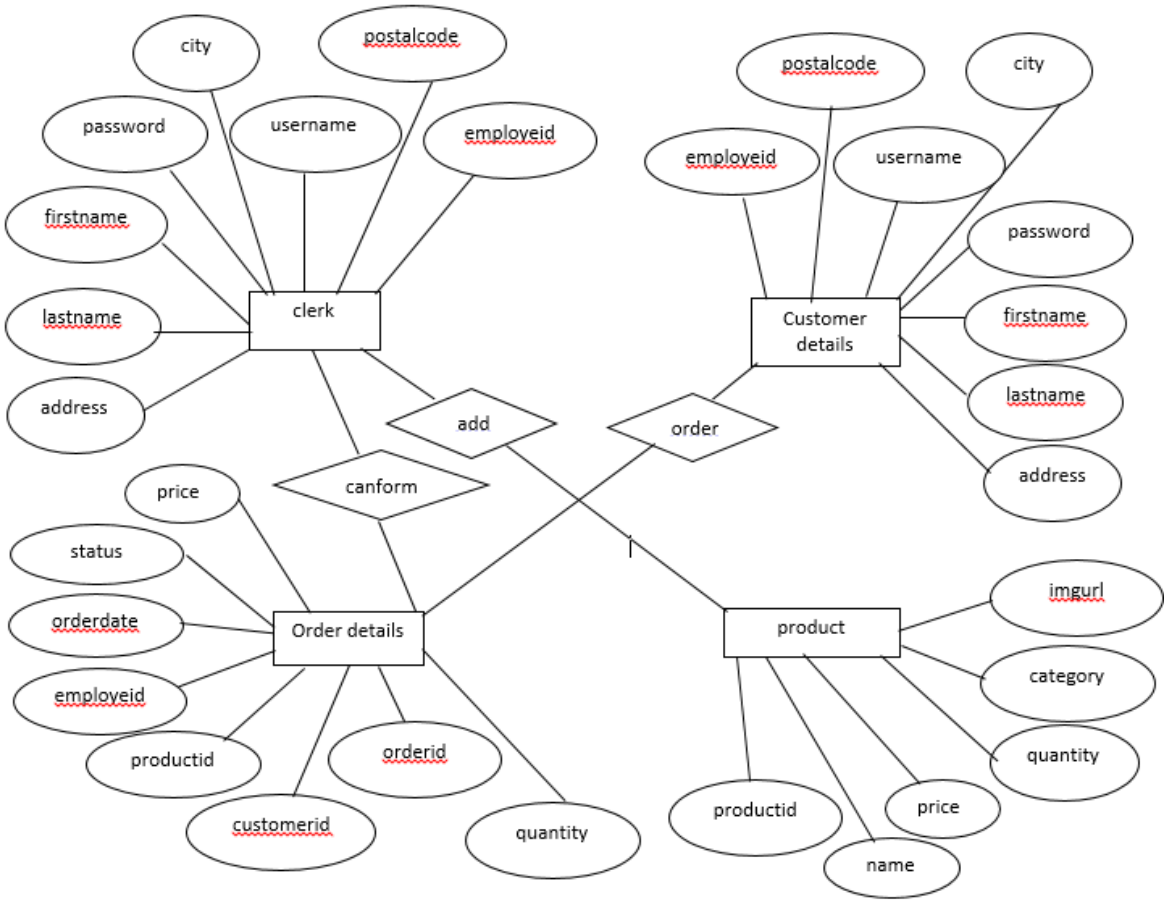


FIG 1.5 ER Diagram of farmers market

6 IMPLEMENTATION

AddProducts.java

```
package com.Farmers.Market;
import android.content.ContentValues;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import java.util.function.DoubleUnaryOperator;
import static com.Farmers.Market.DatabaseManager.PRODUCT;
public class AddProducts extends AppCompatActivity {
   Button addproduct;
   EditText id,name,price,quantity,category,url;
   DatabaseManager mydb;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_add_products);
      addproduct = (Button)findViewById(R.id.btnAddProduct);
      id = (EditText)findViewById(R.id.txtid);
      name = (EditText)findViewById(R.id.txtName);
      price = (EditText)findViewById(R.id.txtPrice);
      quantity = (EditText)findViewById(R.id.txtQuantity);
      category = (EditText)findViewById(R.id.txtCategory);
      url = (EditText)findViewById(R.id.imgUrl);
      mydb = new DatabaseManager(this);
      addproduct.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            if (id.equals("") || name.equals("") || price.equals("") || quantity.equals("") || category.equals("") ||
url.equals("")) {
               Toast.makeText(getBaseContext(), "Please Fill in all your Details",
Toast.LENGTH_SHORT).show();
            }else
            {
               boolean isinserted =
mydb.insertProductData(name.toString(),Double.parseDouble(price.getText().toString()),Integer.parseInt(
quantity.getText().toString()),category.toString(),url.toString());
               if(isinserted){
                  Toast.makeText(AddProducts.this, "Data Inserted", Toast.LENGTH_LONG).show();
               }else{
```

```
                Toast.makeText(AddProducts.this, "Data Not Inserted", Toast.LENGTH_LONG).show();
            }
        }
    }
  });
}
}
```

DatabaseManager.java

```
package com.Farmers.Market;
import java.util.ArrayList;
import java.util.List;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class DatabaseManager extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "com.rafaeltimbo.shoppingbasket";
   private static final int DATABASE_VERSION = 17;
   public static final String tables[]={"tbl_customer","tbl_clerk", "tbl_product", "tbl_order"};
   public static final Integer CUSTOMER = 0;
   public static final Integer CLERK = 1;
   public static final Integer PRODUCT = 2;
   public static final Integer ORDER = 3;
   private static final String tableCreatorString[] =
        {"CREATE TABLE " + DatabaseManager.tables[CUSTOMER] + " (customerId INTEGER
PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT," +
            " firstname TEXT, lastname TEXT," +
            " address TEXT, city TEXT, postalCode TEXT);",
            "CREATE TABLE " + DatabaseManager.tables[CLERK] + " (employeeId INTEGER
PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT," +
                " firstname TEXT, lastname TEXT, " +
                " address TEXT, city TEXT, postalCode TEXT);",
            "CREATE TABLE " + DatabaseManager.tables[PRODUCT] + " (productId INTEGER
PRIMARY KEY AUTOINCREMENT, name TEXT, price TEXT," +
                " quantity INTEGER, category TEXT, imageUrl TEXT);",
            "CREATE TABLE " + DatabaseManager.tables[ORDER] + " (orderId INTEGER
PRIMARY KEY AUTOINCREMENT, " +
                "customerId INTEGER KEY REFERENCES tbl_customer(customerId), " +
                "productId INTEGER KEY REFERENCES tbl_product(productId), " +
                "employeeId INTEGER KEY REFERENCES tbl_clerk(employeeId), " +
                "orderDate DATE, status TEXT," +
                "price FLOAT, quantity INTEGER );"
            ,};
```

```
//class constructor
public DatabaseManager(Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
//initialize database table names and DDL statements
public void dbInitialize( )
{
}
// Create tables
@Override
public void onCreate(SQLiteDatabase db) {
   // Drop existing tables
   for (String table : tables) db.execSQL("DROP TABLE IF EXISTS " + table);
   //create them
   for (int i=0;i<tables.length;i++)
      db.execSQL(tableCreatorString[i]);
}
public void recreateOneTable(int i) {
   SQLiteDatabase db = getWritableDatabase();
   db.execSQL("DROP TABLE IF EXISTS " + tables[i]);
   db.execSQL(tableCreatorString[i]);
   db.close();
}
//create the database
public void createDatabase(Context context)
{
   SQLiteDatabase mDatabase;
   mDatabase = context.openOrCreateDatabase(
         DATABASE_NAME,
         SQLiteDatabase.CREATE_IF_NECESSARY,
         null);

}
//delete the database
public void deleteDatabase(Context context)
{
   context.deleteDatabase(DATABASE_NAME);
}
// Upgrading database
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   // Drop existing tables
   for (String table : tables) db.execSQL("DROP TABLE IF EXISTS " + table);
```

```
    // Create tables again
    onCreate(db);
}
//////////////////////
// Database operations
//////////////////////
// Add a new record
long addRecord(ContentValues values, String tableName, String fields[],String record[]) {
    SQLiteDatabase db = this.getWritableDatabase();
    for (int i=1;i<record.length;i++)
        values.put(fields[i], record[i]);
    // Insert the row
    long id = db.insert(tableName, null, values);
    db.close(); //close database connection
    return id;
}
// Read all records
public List getTable(String tableName) {
    List table = new ArrayList(); //to store all rows
    // Select all records
    String selectQuery = "SELECT  * FROM " + tableName;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);
    ArrayList row=new ArrayList(); //to store one row
    //scroll over rows and store each row in an array list object
    if (cursor.moveToFirst())
    {
        do
        { // for each row
            for (int i = 0; i < cursor.getColumnCount(); i++) {
                row.add(cursor.getString(i));
            }
            //if (!table.contains(row)) {
                table.add(row); //add row to the list
            //}
        } while (cursor.moveToNext());
    }
    cursor.close();
    // return table as a list
    return table;
}
public List queryTable(String tableName, String queryWhere, String[] selectionArgs) {
    List table = new ArrayList(); //to store all rows
    // Select all records
```

```
    String selectQuery = "SELECT  * FROM " + tableName + " WHERE " + queryWhere;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, selectionArgs);
    ArrayList row=new ArrayList(); //to store one row
    //scroll over rows and store each row in an array list object
    if (cursor.moveToFirst())
    {
       do
       { // for each row
          for (int i = 0; i < cursor.getColumnCount(); i++) {
             row.add(cursor.getString(i));
          }
          //if (!table.contains(row)) {
          table.add(row); //add row to the list
          //}
       } while (cursor.moveToNext());
    }
    cursor.close();
    // return table as a list
    return table;
}
// Update a record
public int updateRecord(ContentValues values, String tableName, String fields[],String record[]) {
    SQLiteDatabase db = this.getWritableDatabase();
    for (int i=1;i<record.length;i++)
       values.put(fields[i], record[i]);
    // updating row with given id = record[0]
    return db.update(tableName, values, fields[0] + " = ?",
          new String[] { record[0] });
}
// Delete a record with a given id
public void deleteRecord(String tableName, String idName, String id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(tableName, idName + " = ?",
          new String[] { id });
    db.close();
}
//inserting product data
public boolean insertProductData(String name, Double price, int quantity, String category, String url) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    //contentValues.put("productId", id);
    contentValues.put("name", name);
    contentValues.put("price", price);
```

```
        contentValues.put("quantity",quantity);
        contentValues.put("category", category);
        contentValues.put("imageUrl", url);
        long result = db.insert("tbl_product", null, contentValues);
        if (result == -1)
            return false;
        else
            return true;
    }
}
```

LoginActivity.java

```
package com.Farmers.Market;
import android.content.Intent;
import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class LoginActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        final Button signupBtn = findViewById(R.id.signupBtn);
        final Button loginBtn = findViewById(R.id.loginBtn);
        final EditText usernameET = findViewById(R.id.usernameLoginScreen);
        final EditText passwordET = findViewById(R.id.passwordLoginScreen);
        User.setDb(getApplicationContext());
        Order.setDb(getBaseContext());
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
            String usernameInput = usernameET.getText().toString();
            String passwordInput = passwordET.getText().toString();
            if (login(usernameInput, passwordInput, false)) {
                // load initial product settings
                Product.setDb(getBaseContext());
                Intent listOrders = new Intent(getBaseContext(), ListOrdersActivity.class);
                startActivity(listOrders);
            }
            }
        });
```

```
        signupBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent signUp = new Intent(getBaseContext(), SignupActivity.class);
                startActivity(signUp);
            }
        });
    }
    private boolean login(String usernameInput, String passwordInput, boolean isClerk) {
        User savedUser;
        try {
            if (isClerk) savedUser = User.queryClerkByUsername(usernameInput);
            else         savedUser = User.queryCustomerByUsername(usernameInput);
            if (usernameInput.equals(savedUser.username)) {
                if (passwordInput.equals(savedUser.password)) {
                    savePreferences(savedUser.username, isClerk);
                    return true;
                } else {
                    Toast.makeText(LoginActivity.this, "invalid password", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(LoginActivity.this, "invalid username", Toast.LENGTH_SHORT).show();
            }
            return false;
        } catch ( NullPointerException e) {
            if (isClerk) {
                Toast.makeText(LoginActivity.this, "user does not exist", Toast.LENGTH_SHORT).show();
            } else return login(usernameInput, passwordInput, true);
            return false;
        }
    }
    private void savePreferences(String username, Boolean isClerk) {
        SharedPreferences preference = getSharedPreferences("Login", MODE_PRIVATE);
        SharedPreferences.Editor editor = preference.edit();
        editor.putString("username", username);
        editor.putBoolean("isClerk", isClerk);
        editor.commit();
    }
}
```

Order.java

```
package com.Farmers.Market;
import android.content.ContentValues;
import android.content.Context;
```

```
import android.util.Log;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
public class Order {
    public String id;
    public String customerId;
    public String employeeId;
    public String orderDate;
    public Status status;
    public String productId;
    public Double price;
    public Integer quantity;
    private static DatabaseManager db;
    private static final String tbl_order_fields[] = {
        "orderId", "customerId", "productId", "employeeId", "orderDate", "status", "price", "quantity"
    };
    private static final int size = tbl_order_fields.length;
    public Order(String id, String customerId, String productId, String employeeId, String orderDate,
String status, Double price, Integer quantity){
        this.id = id;
        this.customerId = customerId;
        this.productId = productId;
        this.employeeId = employeeId;
        this.orderDate = orderDate;
        this.status = Status.fromString(status);
        this.price = price;
        this.quantity = quantity;
    }
    public Order(Product p, String username) {
        Log.d("rafaeltimbo.timbo.Order", username);
        Product product = Product.queryProductByName(p.name);
        User user = User.queryCustomerByUsername(username);
        this.id = "0";
        this.customerId = user.id;
        this.productId = product.id;
        this.employeeId = "0"; // arbitrary employee id
        //SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:MM");
        Date now = Calendar.getInstance().getTime();
        String date = now.toString();
        this.orderDate = date;
        this.status = Status.Processing;
```

```
        this.price = product.price;
        this.quantity = product.quantity;
    }
    public static void setDb(Context context) {
        db = new DatabaseManager(context);
        //db.deleteDatabase(context);
        //db.createDatabase(context);
        //initOrders();
    }
    @Override
    public String toString() {
        User customer = User.queryCustomerById(this.customerId);
        return String.format(Locale.CANADA,
                "Order %s by %s: $%.2f (%s)",
                this.id, customer.username, this.price * this.quantity, this.status);
    }
    public String toStringLog() {
        return String.format("id: %s, customerId: %s, productId: %s, employeeId: %s, orderDate: %s, status:
%s, quantity: %s, price: %s",
            this.id, this.customerId, this.productId, this.employeeId, this.orderDate, this.status, this.quantity,
this.price);
    }
    private String[] getRecord() {
        return new String[] { this.id, this.customerId, this.productId, this.employeeId, this.orderDate,
                        this.status.toString(), this.price.toString(), this.quantity.toString()
        };
    }
    public Order addToDatabase() {
        Log.d("rafaeltimbo.timbo", "creating order " + db.toString() + this.toString());
        Long id = db.addRecord(new ContentValues(),
DatabaseManager.tables[DatabaseManager.ORDER], tbl_order_fields, this.getRecord() );
        Log.d("rafaeltimbo.timbo", "created order " + id.toString());
        this.id = id.toString();
        return this;
    }
    public void cancel() {
        db.deleteRecord(DatabaseManager.tables[DatabaseManager.ORDER], tbl_order_fields[0], this.id );
    }
    public void deliver() {
        this.status = Status.Delivery;
        updateDatabase();
    }
    public void updateDatabase() {
```

```
        db.updateRecord(new ContentValues(), DatabaseManager.tables[DatabaseManager.ORDER],
tbl_order_fields,  this.getRecord() );
    }
    public static Order queryById(String id) {
        List orderList = db.queryTable(DatabaseManager.tables[DatabaseManager.ORDER], "orderId = ?",
new String[] { id });
        orderList = convertDbResults(orderList);
        return (Order) orderList.get(0);
    }
    public static ArrayList<Order> fetchOrders(String customerId) {
        List orderList = db.queryTable(DatabaseManager.tables[DatabaseManager.ORDER], "customerId =
?", new String[] { customerId });

        Log.d("rafaeltimbo.timbo.customerId", orderList.toString());
        return convertDbResults(orderList);
    }
    public static ArrayList<Order> fetchOrders() {
        List orderList = db.getTable(DatabaseManager.tables[DatabaseManager.ORDER]);
        Log.d("rafaeltimbo.timbo.fetchOrder", orderList.size() + ", " + orderList.toString());
        return convertDbResults(orderList);
    }
    private static ArrayList<Order> convertDbResults(List orderList) {
        ArrayList<Order> orderArrayList = new ArrayList<Order>();
        for (int i = 0; i < orderList.size() ; i++) {
            List l = (List) orderList.get(i);
            orderArrayList.add(new Order(
                l.get(Order.size * i).toString(),
                l.get(Order.size * i + 1).toString(),
                l.get(Order.size * i + 2).toString(),
                l.get(Order.size * i + 3).toString(),
                l.get(Order.size * i + 4).toString(),
                l.get(Order.size * i + 5).toString(),
                Double.valueOf(l.get(Order.size * i + 6).toString()),
                Integer.valueOf(l.get(Order.size * i + 7).toString())
            ));
        }
        return orderArrayList;
    }
}
```

7 TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.1 UNIT TESTING:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results. The modules such are login, register, user home and admin home modules are carried out individually.

7.2 INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components. All the modules are integrated and then the testing has been carried out.

7.3 TEST CASES

Test case Id	Module Name	Test case description	Test Case Steps	Expected Result	Actual Result	Status
TC01	Register page	Verify the Register page is displayed or not.	1. Run the Application 2. Click on the sine up option.	on clicking the sine up link user should be navigated to register page	same as expected	Pass
TC02	Register page	Verify the Register page with the valid credentials	1. Run the Application 2. Click on the sine up option. 3. Enter the valid credentials 4. click on the register button after entering all the necessary fields	user should be registered successfully	same as expected	Pass
TC03	Register page	Verify the Register page with the invalid credentials	1. Run the Application 2. Click on the sine up option. 3. Enter the invalid credentials 4. click on the register button after entering all the necessary fields	system should not allow user to register with invalid credentials	same as expected	Pass
TC04	Register page	Verify the login page navigation from the registration page	1. Run the Application 2. Click on the sine up option. 3. click on the login link present in the registration page	system should not allow user to navigate to the login page	same as expected	Pass
TC05	login page	Verify the Login page is displayed or not.	1. Run the Application 2. Click on the login option.	on clicking the login link user should be navigated to login page	same as expected	Pass

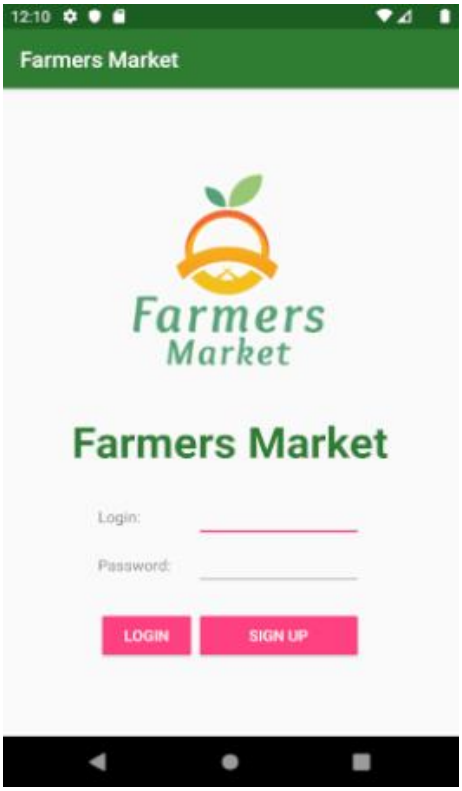
TC06	login page	Verify the Login page with the valid credentials	1. Run the Application 2. Click on the login option. 3. Enter the valid credentials 4. click on the login button after entering all the necessary fields	user should login successfully	same as expected	Pass
TC07	login page	Verify the Login page with the invalid credentials	1. Run the Application 2. Click on the login option in the home page. 3. Enter the invalid credentials 4. click on the login button after entering all the necessary fields	system should not allow user to login with invalid credentials	same as expected	Pass
TC08	login page	Verify the Login page navigation from the registration page	1. Run the Application 2. Click on the login option. 3. click on the register link present in the registration page	system should not allow user to navigate to the register page	same as expected	Pass
TC09	User Order history.	verify the user able to see his previous order details.	1. Run the Application 2. Click on the login option. 3. Enter the valid credentials.	User should be navigated to user order history.	same as expected	Pass
TC11	User Order history	verify the user not able to access the user home page after login with invalid credentials	1. Run the Application 2. Click on the login option. 3. Enter the invalid credentials 4. click on the login button after entering all the necessary fields	An error showing the user is not registered should be displayed	same as expected	Pass

TC12	View products	Verify the user able to view the products.	1. Run the Application 2. Click on the view product option in the user order history page. 3. Enter the valid credentials	on clicking on submit the rescue details should be added and successful message should be displayed	same as expected	Pass
TC13	Order product	Verify the user able to order the product by selecting the product details	1. Run the Application 2. Click on the login. 3. Enter the valid credentials 4. Click on the login button after entering all the necessary fields 5. Enter the value for quantity. 6. click on the confirm order button	on clicking on submit the order details should be added and un successful message should be displayed	same as expected	Pass
TC14	Order product	verify the user is able to view the rescue history	1. Run the Application 2. Click on the login option in the home page. 3. Enter the valid credentials 4. Click on the login button after entering all the necessary fields 5. Click on the rescue history present on the user home page.	On clicking on the order button the user should be able to order the product	same as expected	Pass
TC15	Cancel order	Verify the user is able to cancel his order	1. Run the Application 2. Customer login should be successful 3. in view order details user able to select the product and able to cancel the product which status is pending	On clicking the cancel button the user order should be cancelled	same as expected	Pass

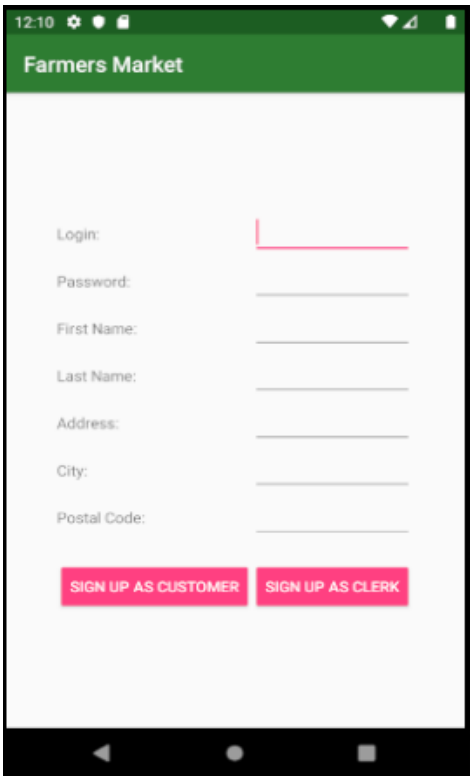
TC16	Admin order details	Verify the admin is able to view the order details of the all customer	1. Run the Application 2. Admin login should be successful 3. In home page the order details of all customer should be displayed	User should be able to see the order details	Same as expected	Pass
TC17	Admin change status	Verify the admin is able to change the status of the order once it has been delivered	1. Run the Application 2. Admin login should be successful 3. In home page the order details of all customer should be displayed. 4. select the order on clicking on the order. 5. user should be able to cancel the order by selecting the order details	User should be able to change the order status	Same as expected	Pass

8 SCREENSHOTS

Login page

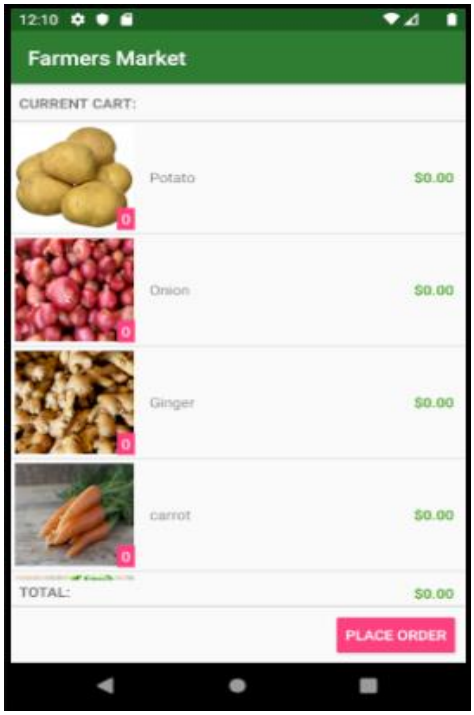


Register page



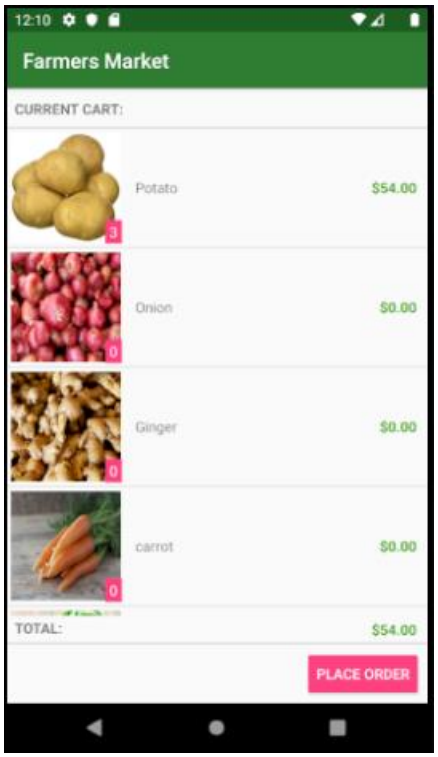
View order history

View products

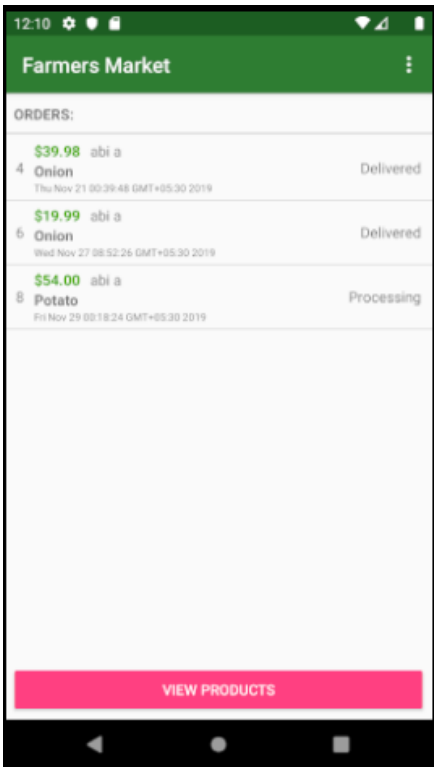


Add to cart

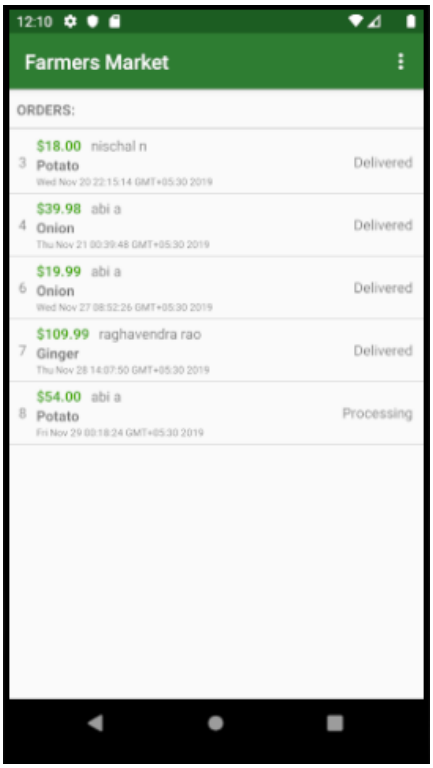
Place order



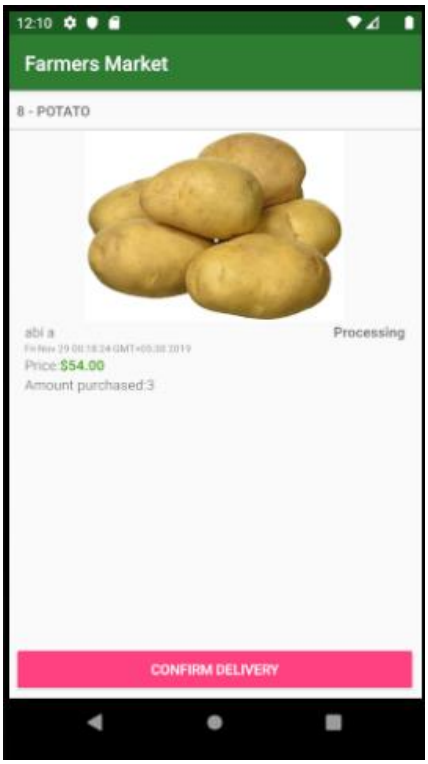
Order placed



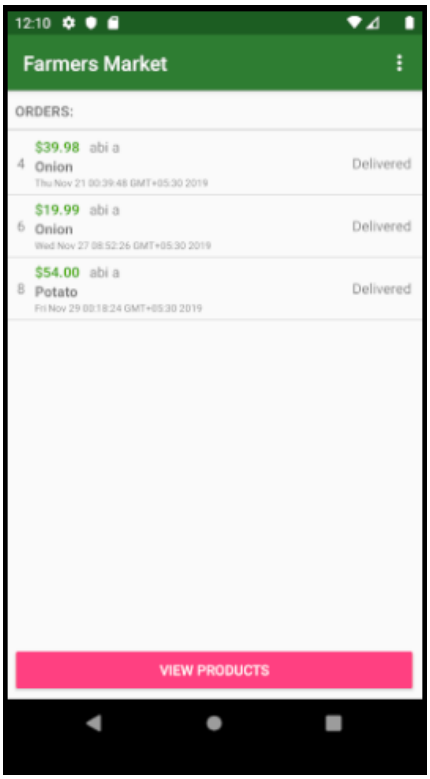
View order history by Admin



Confirm order by Admin



Delivered to customer



9 CONCLUSION

To conclude, the farmer's market online project works well with the limited number of products and the orders. When the traffic increases then we have implement the resource hungry modules so that the process can be implemented

- This product helps both the users.
- Admin can sell their products.
- Users can get the products easily.

10 FUTURE ENHANCEMENT

- Some of the features such as tracking order, payment gateway, can be implemented in the future.
- Mail notification for registering, order confirmation and tracking can be added.
- Guest user support can be added so that the guest can view the product similar to window shopping.
- User interface can be improved so that the look and feel of the application can be improved.
- Adding products and maintain the stock can be implemented so that the huge number of stock can be handled easily.
- Agriculture related products such as fertilizers and other necessary goods can be added so that the farmers can get the products.
- This project works with only the newer versions of android. For older versions can be implemented so that the users with older version of android can access the application.

11 BIBILOGRAPHY

- Introduction to Android: *http://developer.android.com/guide/index.html*.
- Android API: *http://developer.android.com/reference/packages.html*
- Java 6 API: *http://docs.oracle.com/javase/6/docs/api/*
- Android Fundamentals: *http://developer.android.com/guide/components/fundamentals.html*
- The Java Tutorials: *http://docs.oracle.com/javase/tutorial/*
- Android User Interfaces: *http://developer.android.com/guide/topics/ui/index.html*
- Layout: *http://developer.android.com/guide/topics/ui/declaring-layout.html*
- Common Tasks: *http://developer.android.com/guide/appendix/faq/commontasks.html*
- Google Maps: *http://code.google.com/android/add-ons/google-apis/maps-overview.html*
- Iconography: *http://developer.android.com/guide/practices/ui_guidelines/icon_design.html*
- Sample Source Code: *http://developer.android.com/resources/samples/get.html*
- Android Training: *http://developer.android.com/training/index.html*.
- Android Developer's Blog: *http://android-developers.blogspot.com/*
- Developer FAQ: *http://developer.android.com/resources/faq/*
- Developer Forums: *http://developer.android.com/resources/community-groups.html*
- Android Developer's Group: *http://groups.google.com/group/android-developers?lnk=*
- XDA-Developers Forums: http://forum.xda-developers.com/