Food Sparker

Restaurant Order System

Final Project

December 9, 2018

Version 2.2

Team Name:

BugCreator

Team Members:

You Fu

Di Weng

Ziyu Zhou

Xinyue Cai

Zhengyang Xu

Table of Contents

Chapter 1 Introduction

1.1. Project Source and Significance

This project is to build an online client/server android Restaurant application. The restaurant wants to use internet technology to run its business and specifically use mobile devices where customers can order food online and pick up within half-hour.

1.2. Main tasks and objectives of the project

A front page menu shows the items and the buttons for selection. The customer is provided with “Order” button to start selection and a “Submit” button to submit the order. Once the submit button is pressed, an order is generated and placed on the customer OrderList.

The total cost of order is the cost of each item multiply by quantity plus 5% Tax and 25% profit.

An order is received by the kitchen thread and is added to customer OrderList. This thread manages the order in an orderly manner, as order comes in, is added to the back of the customer order list. The order has the following information: customer-id, customer name, items and quantity for each item, and status. You need to design the Order and OrderList, and define the status of an order during submitting, receiving, preparing, packaging, notification and delivery to front cashier.

Create “Inventory.txt” file to manage the menu items. An inventoryList is generated/updated every hour by reading 50 data entries for each item from the inventory file. The chef selects the top order from customer OrderList and verifies the availability of order items with the inventoryList. If any item in inventoryList is flagged “NotAvailable”, it means that the item is not in inventory and the food order: a) can only be prepared partially or b) cannot be prepared if all the items on the order are flagged “NotAvailable”. Depending on the status of the order, a notification is sent to customer that either “order can only be “PartiallyAvailable”, showing all the items which can be prepared. OR “cannot be prepared, inventory NotAvailable”. NOTE: There are additional thinking required here. i.e.: customer sends notification back either a) wants the partial order to be packaged, or b) wants to cancel the order. And further more if (a) is true then the order should not be put on the back of the customer order list, and should be handled immediately.

Chapter 2 System Requirements Analysis

2.1. System use case analysis

FoodSparker App is a platform that provides a dynamic food ordering Android application between the customer and the restaurant. Customers can select desired food from the Menu page while the kitchen side can track the real-time availability of each item. After receiving the order from the customer, kitchen will check the availability of each item. If the order cannot be fulfilled, the kitchen will send a confirmation message through server, telling the customer about the changing details. When the order is ready, the customer will pick up the order.

2.1.1. System Actor Analysis

The main users of this system can be divided into customers and the kitchen.

- Customer

Customer is able to register and login into the system. Then he/she can make orders through the Menu page and submit orders through the Cart page. When the order cannot be fulfilled, the customer will get a system message letting him/her choose to accept the change or decline. After the order is confirmed, customer will receive order status notifications, including order processing, order preparing, order packing, and order ready to pick up.

- Kitchen

Kitchen is able to check real-time item availability. According to the real-time item availability, kitchen can decide whether each item in the order can be fulfilled or not. If the order can only be partially fulfilled, kitchen can pass a message through server telling customers about the order change. Once the change is confirmed by the customer, the kitchen will prepare the order in accordance with the confirmed change. Kitchen will send notifications to the customer about the order status, letting the customer know about phases of the order.

2.1.2. Use case Diagram

The system is mainly divided into a client and a server. The client receives the user operation response and transmits the data to the server. After receiving the user action, the server processes the corresponding transaction according to the process. The system use case diagram is shown in Figure 2-1.

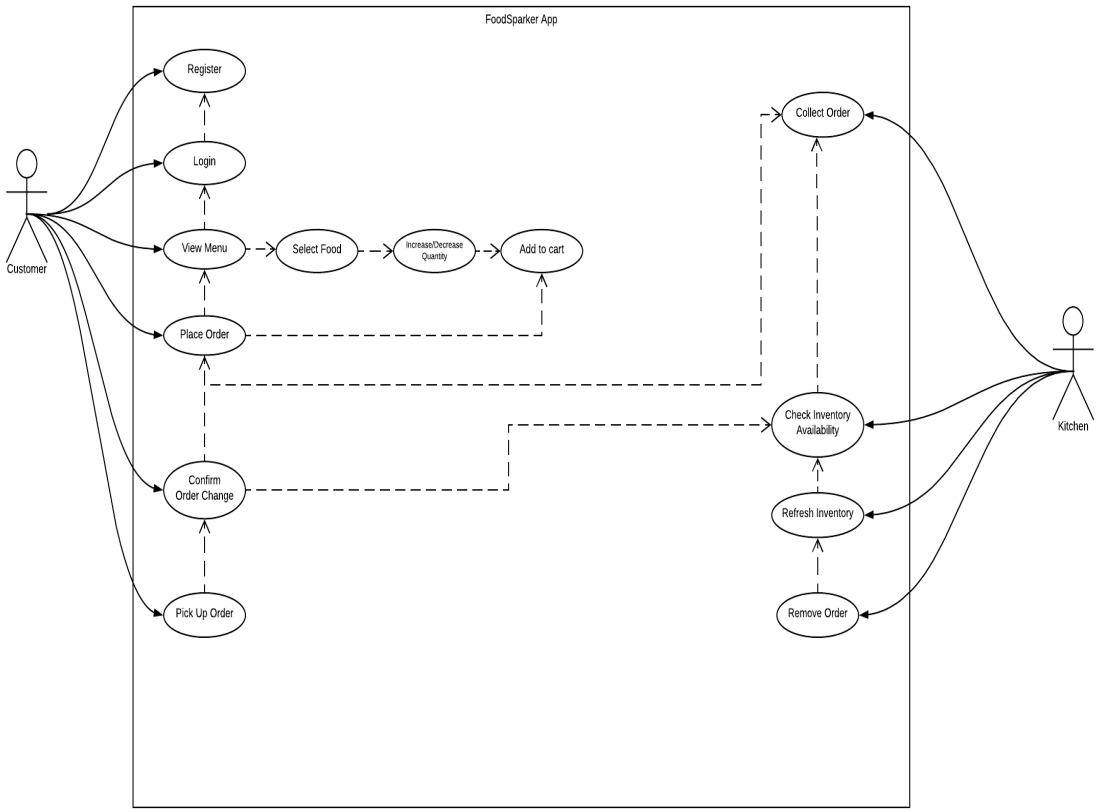


Figure 2-1 System use case diagram

2.1.3. Use case Demonstration

- Use case – Customer
    - Registration

    Goal: Allow new customers to register for the application

    Prerequisite: None.

    Description:

    1) Open FoodSparker App
    2) Type in username and password in the corresponding fields and press the *Register* button.
    3) The system will show notification if the registration is successful or not.

    - Login

    Goal: allow the existing customer to log into the system

    Prerequisite: Customer registration

    Description:

1) Open FoodSparker App
2) Type in username and password in the corresponding fields and press the *Login* button.
3) The system will go to the Menu page if the customer logins successfully.

- View Menu

Goal: allows the customer to view menu of the restaurant, to select food, to increase/decrease item quantities and to add to cart

Prerequisite: Customer Login

Description:

1) Customer picks the desired item and presses the Add button, which is on the right side of the item to increase its quantity, or presses the Minus button to decrease its quantity
2) After deciding the quantity of desired items, customer presses the *Add to Cart* button to go into checkout process.

- Place Order

Goal: allows the customer to place order after adding items into the cart

Prerequisite: View Menu, select food and add to cart

Description:

1) Customer is able to submit the order from the Order Summary screen by pressing the *Submit* button.

- Confirm Order Change

Goal: Customer can select accept or decline the order change after receiving order update message from the kitchen

Prerequisite: Place order

Description:

1) When the order cannot be fully fulfilled, customer will receive a message from the kitchen asking him/her whether to accept the change
2) If customer chooses to accept by clicking the *Accept* button, kitchen will fulfill the updated order; if the customer chooses to decline, the order is canceled.

- Pick Up Order

Goal: Let customer pick up the prepared order

Prerequisite: Place order and accept order change if applicable

Description:

1) When the order is well prepared, customer gets notification and can pick up the order within 30 minutes.

- Use case – Kitchen
  - Collect Order

Goals: To show all the orders customer placed and deal with them.

Prerequisite: Server send order information to kitchen client.

Description:

1) Customers place order from client;
2) Server send information;
3) Kitchen get information;

  - Check Inventory Availability

Goals: Send confirmation request to server and do the validation if the stock is enough to cover the order.

Description:

1) Kitchen send confirmation request;
2) Server verify the inventory then send the result;
3) Kitchen get result;

  - Refresh Inventory

Goals: Refresh the inventory of food after the order is completed.

Description: After the confirmation of kitchen and customer (cancel/partially available/available), the inventory of food changes.

  - Remove Order

Goals: To remove completed order from order list

Description: By clicking the confirm button, kitchen can remove the order from the recycler view.

Chapter 3 System Design

The system design of this project mainly includes five parts: architecture design, system workflow design, function module design, code framework design and data persistence design.

3.1. Architecture design

In order to meet the requirements of fast response, easy operation and easy maintenance, the system follows the design principles of software development and adopts the client-server (C/S) architecture. By properly assigning tasks to the client and server, the communication overhead of the system is reduced, and the development is easy and easy to operate.

As shown in Figure 3-1, in the system of Client/Server structure, the application is divided into two parts: client and server. The user's program is mainly on the client side, the client part is proprietary to each user, and the server side mainly provides data management, data sharing, data and system maintenance, and concurrency control, etc., and multiple users share their information and functions.

In this system, the client part is responsible for executing the foreground function, implementing the user interface and simple data processing functions, and is responsible for handling interactions with the application server, such as managing user interfaces, data processing, and report requests; and the server side performs background services. Responsible for processing application logic, specifically accepting requests from client applications, then interacting with files based on application logic and passing the results of the file interactions to the client application.

Figure 3-1 C/S two-tier architecture

3.2. System Workflow Design

The system passes the data access server through the client interface written by Android, and the processing result is returned to the client for display.

3.2.1. Customer Interface Process

The main process of the customer Android interface is shown in Figure 3-2:

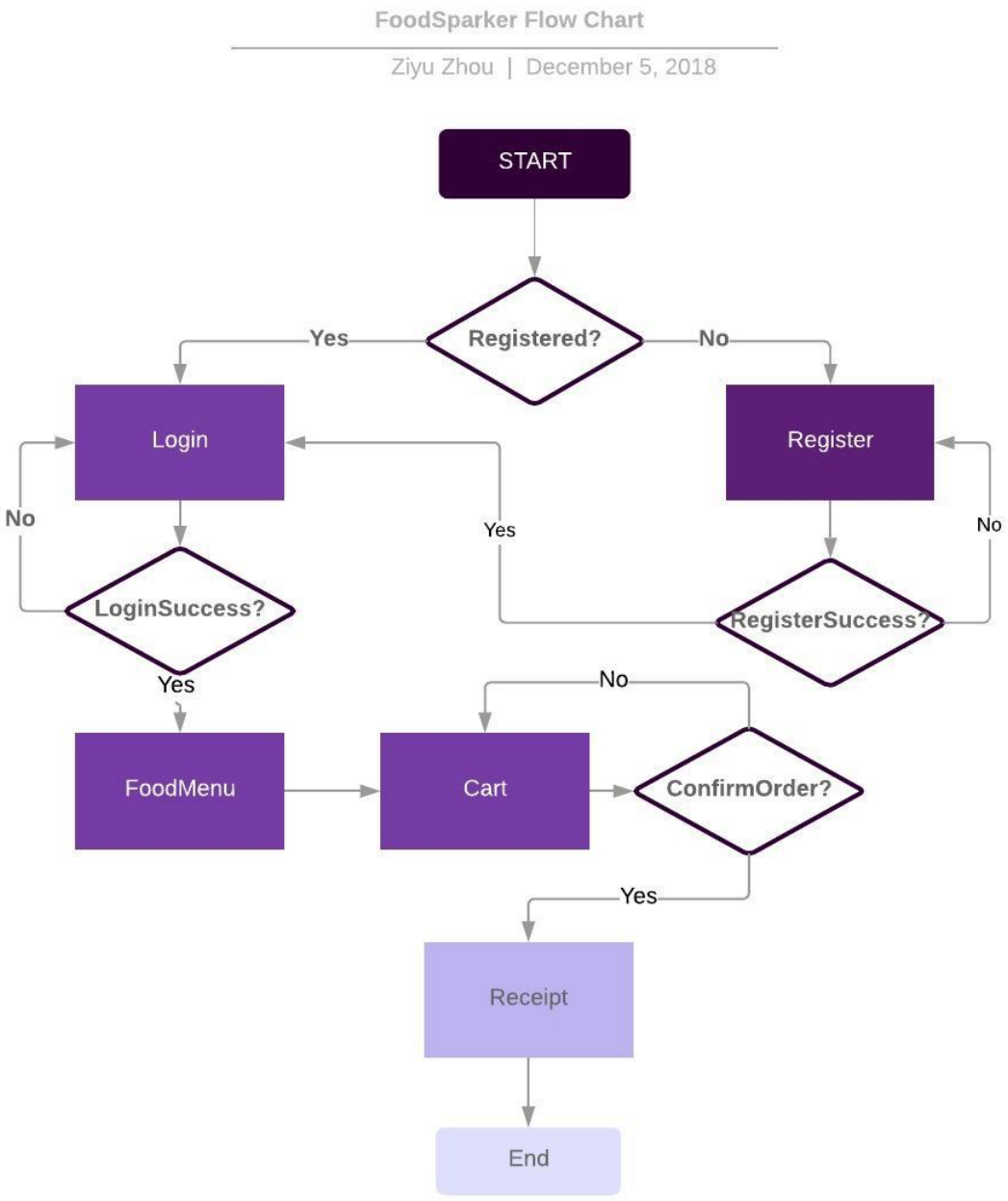


Figure 3-2 Interface structure

When you first open the application you have to register first. After the registration you can log in with the username and password, after that you can view the menu. Customers are able to add specific quantity of food to cart and then place order. The orders are passed to kitchen and then wait for the confirmation from kitchen. If the required quantity is available, the order is completed. If the order is partially available, the kitchen would send a message and check if customers still want the food. After the confirmation from customer and the showing of receipts, the whole work flow is completed.

3.2.2. Kitchen Interface Process

The main process of the kitchen Android interface is shown in Figure 3-3:

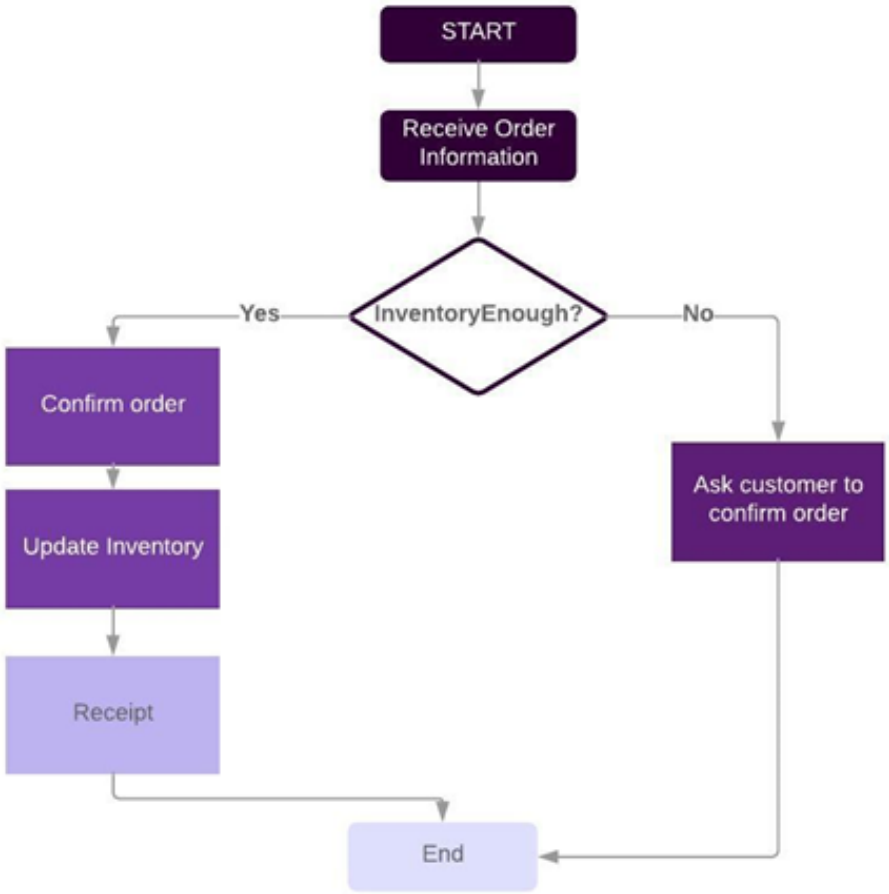


Figure 3-3 Kitchen Interface structure

3.3. Data Persistence Design

The persistence operation of the result data of this project is mainly realized by writing it to the .txt file and saving it to the specified location of the local hard disk.

- Customer.txt：store user information.
- Inventory.txt: store dish item information.

Chapter 4 System Implementation

4.1. Related technology implementation

4.1.1. Java Multithreading

In the project, use the Java.lang.Thread class or the Java.lang.Runnable interface to define, instantiate, and start new threads. In Java, each thread has a call stack, and even if no new threads are created in the program, the threads are running in the background. A Java application always runs from the main() method, which runs inside a thread and is called the main thread. Once a new thread is created, a new call stack [8] is generated.

In this project, both the Android client and the server implement multi-threading and thread-safe maintenance.

In the C/S structure, since some functions of the background program need to take a certain amount of time, in order to not affect the operation of other functions during this period, when the background server is running, a new thread is used for its execution.

The code implementation is shown in Figure 4-1:

```
// each time a client is connected, the ServerThread start to serve the client.
new Thread(new ServerThread(socket)).start();
```

```
public class AutoUpdateInventoryService implements Runnable{

    @Override
    public void run() {
        while(true) {
            try {
                Thread.sleep(1000*60*60);
                InventoryUtil.autoUpdateIventory();
            }
            catch(InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figure 4-1 Multi-threaded implementation code diagram

4.1.2. Android timing execution task implementation

In order to periodically send a request to the server to obtain item inventory data and update the UI display content, this project uses Handler's postDelayed(Runnable, long) method to implement Android scheduled task execution. The code is implemented as shown in Figure 4-2:

```
handler = new Handler();
Runnable task = () → {

        try{
            final MsgTransfer MSG = RequestItemStock.requestItemStock(socket);
            //      ArrayList<Item> inventoryList = RequestItemStock.requestItemStock(socket);

            String cmd = MSG.getCmd();
            if(cmd.equals("updateStock"))
            {
                ArrayList<Item> inventoryList = (ArrayList<Item>) MSG.getData();
                displayStock(inventoryList);
            }

            else {
                Customer customer = (Customer)MSG.getData();
                customerList.add(customer);

                orderAdapter = new OrderAdapter( context: KitchenActivity.this,customerList);
                mRecyclerView.setAdapter(orderAdapter);
                orderAdapter.buttonSetOnclick((view, position) → {
                        try{
                            MSG.setCmd("Confirm");
                            customerList.remove(position);
                            orderAdapter.notifyItemRemoved(position);
                        }catch (Exception e){
                            e.printStackTrace();
                        }

                    });
            }

            handler.postDelayed( r: this, delayMillis: 1000);
        }
        catch (Exception e){
            e.printStackTrace();
        }
```

Figure 4-2 Android timing execution task implementation code diagram

4.2. Interface GUI implementation

The GUI design of this project is implemented using Android XML components, using multiple layouts and controls.

- GUI – Customer side
  - Login Screen: background set
    - EditText: To allow customer to type in username and password
    - ImageButton: one is for login button and the other is for register; when clicked, page turns into menu screen
    - TextView: when registered successfully, a message shows telling the customer registration is successfully
  - Menu Screen: menu set as background

- FloatingActionButton: to allow customer to increase/decrease the amount of desired item
- TextView: show the chosen quantity for each item
- ImageButton: for submitting orders and turns to cart screen when pressed

▪ Car Screen:

- TextView: show selected quantities for each item, subtotal, taxes and total price
- ImageButton: to confirm the order

4.2.1 Layout settings

This project mainly uses ConstraintLayout and custom style layout manager to manage the components.

- Customer side
  - ▪ Login Screen: background set
    - EditText: To allow customer to type in username and password
    - ImageButton: one is for login button and the other is for register
    - TextView: shows a message telling the customer registration is successfully

  - ▪ Menu Screen: menu set as background
    - FloatingActionButton: to increase/decrease the amount of desired item
    - TextView: show the chosen quantity for each item
    - ImageButton: for submitting orders and turns to cart screen when pressed

  - ▪ Car Screen:
    - TextView: show selected quantities, subtotal, taxes and total price
    - ImageButton: to confirm the order

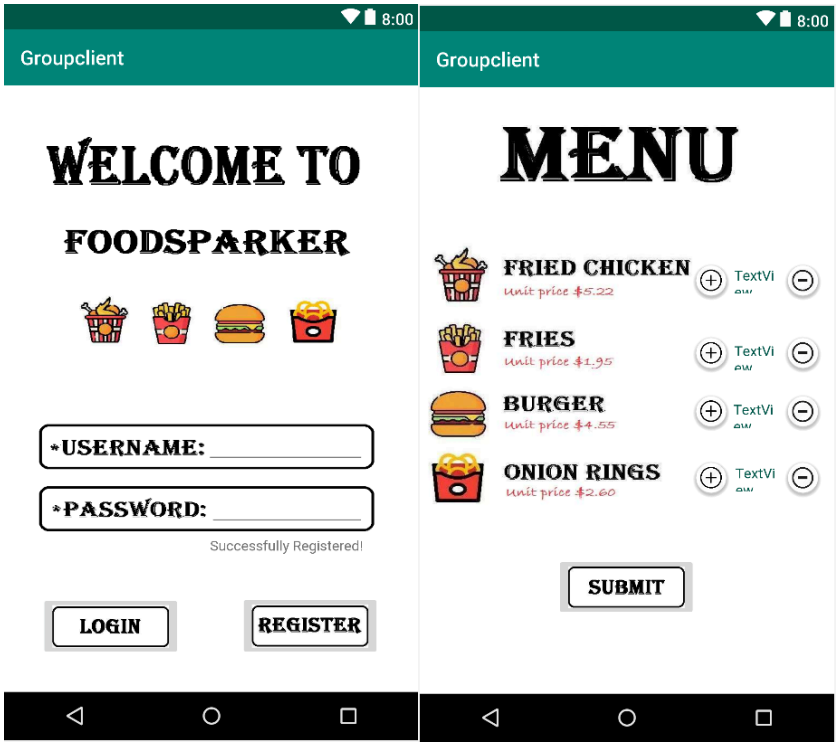
- Kitchen side
  - ▪ Activity_cook.xml
    - TextView BurgerStock: display the stock of burgers.
    - TextView ChickenStock: display the stock of chicken.
    - TextView FrenchFriesStock: display the stock of French fries.
    - TextView OnionRingStock: display the stock of onion rings.
    - View HorizontalDevider: to devide the inventory and the orders.
    - RecyclerView recyclerView: to display the order list.

  - ▪ Item.xml

- TextView CustomerName: display the username of the customer.
- TextView BurgerQ: display the quantity of burgers in this order.
- TextView ChickenQ: display the the quantity of chicken in this order.
- TextView FrenchFriesQ: display the quantity of French fries in this order.
- TextView OnionRingQ: display the quantity of onion rings in this order.
- ImageButton buttonConfirm: to confirm the order and refresh the inventory.

4.2.2 Achieving effect

According to the above control method and layout settings, use Android technology to implement the client interface of the system. The effect is shown in Figure 4-3.



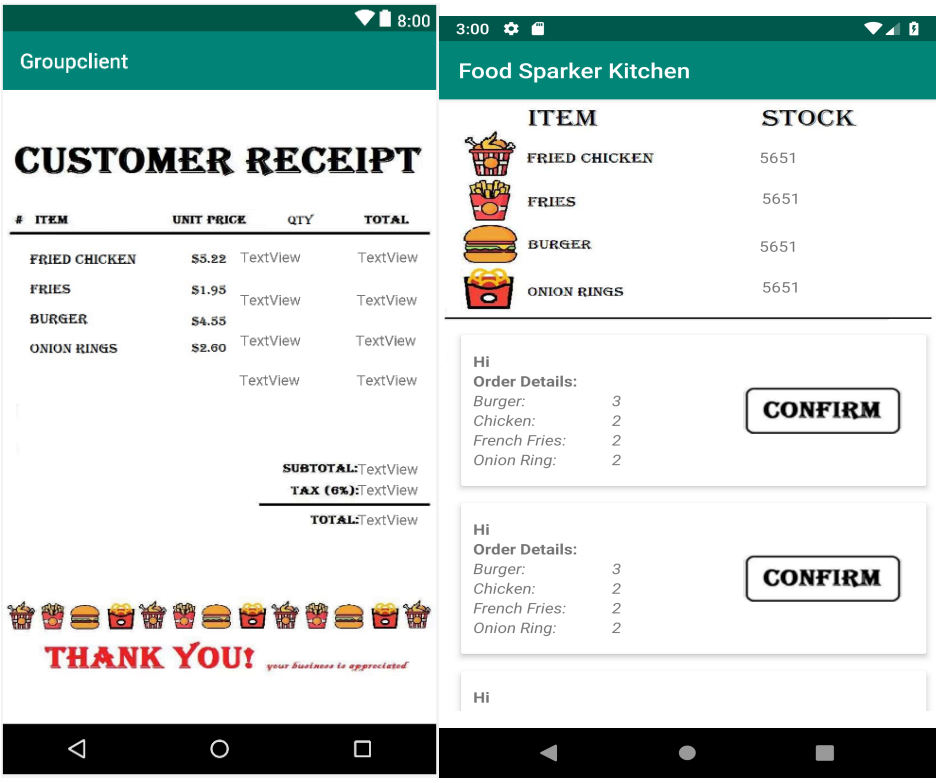


Figure 4-3 GUI implementation effect

4.3. Main function realization

This project is to build an online client/server android Restaurant application. The restaurant wants to use internet technology to run its business and specifically use mobile devices where customers can order food online and pick up within half-hour.

This system mainly implements the following functions:

- Socket network connection

```
private static final int PORT = 8888;
serverSocket = new ServerSocket(PORT);
setSocket(new Socket( host: "10.41.142.189", port: 8888));
```

- User registration

```
public class RegisterService {

    public static boolean register(Customer customer) {
        String filename = "/Users/curlyfu/eclipse-workspace/RestaurantServer/Customer.txt";
        String thisLine = "";

        try(BufferedReader br = new BufferedReader(new FileReader(filename));){

            while((thisLine = br.readLine()) != null) {
                String[] customer_info = thisLine.split(",");
                if(customer_info[0].equals(customer.getUsername())) {
                    br.close();
                    return false;
                }
            }

            BufferedWriter bw = new BufferedWriter(new FileWriter(filename, true));
            String customer_toString = customer.getUsername() + "," + customer.getPassword();
            bw.write(customer_toString);
            bw.newLine();

            br.close();
            bw.flush();
            bw.close();

            return true;

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

- User login

```
public class LoginService {

    public static boolean login(Customer customer) {
        String filename = "/Users/curlyfu/eclipse-workspace/RestaurantServer/Customer.txt";
        String thisLine = "";

        try(BufferedReader br = new BufferedReader(new FileReader(filename));){
            while((thisLine = br.readLine()) != null) {
                String[] customer_info = thisLine.split(",");

//              for(String str : customer_info) {
//                  System.out.println(str);
//              }

                if(customer_info[0].equals(customer.getUsername()) && customer_info[1].equals(customer.getPassword())
                    br.close();
                    return true;
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

- Manage item inventory

```
public static synchronized void autoUpdateIventory() {
    ArrayList<Item> inventoryList = readInventoryFromFile();

    for(Item item : inventoryList) {
        int current_inventory = item.getStock() + 50;
        item.setStock(current_inventory);
    }
    writeInventoryToFile(inventoryList);
}

public static void sendIventoryToKitchen() {

}
```

```
public static synchronized void writeInventoryToFile(ArrayList<Item> inventoryList) {

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(filename));

        for(Item item : inventoryList) {
            String item_toString = item.getName() + "," + item.getCost() + "," + item.getStock();
            bw.write(item_toString);
            bw.newLine();
        }

        bw.flush();
        bw.close();
    }catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public static synchronized void updateInventoryByOrder(Order order){
    ArrayList<Item> inventoryList = readInventoryFromFile();
    Map<Item, Integer> item_quantity = order.getItem_QuantityMap();
    Iterator<Item> it = item_quantity.keySet().iterator();

    while(it.hasNext()) {
        Item item_order = it.next();

        for(Item item_inventory : inventoryList) {
            if(item_order.getName().equals(item_inventory.getName())) {
                int current_inventory = item_inventory.getStock() - item_order.getStock();
                item_inventory.setStock(current_inventory);
            }
        }
    }

    writeInventoryToFile(inventoryList);
}

public static ArrayList<Item> readInventoryFromFile() {
    ArrayList<Item> inventoryList = new ArrayList<Item>();
    String thisLine = "";

    try(BufferedReader br = new BufferedReader(new FileReader(filename));){
        while((thisLine = br.readLine()) != null) {
            String[] item_inventory = thisLine.split(",");

            Item item = new Item();
            item.setName(item_inventory[0]);
            item.setCost(Double.parseDouble(item_inventory[1]));
            item.setStock(Integer.parseInt(item_inventory[2]));

            inventoryList.add(item);
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return inventoryList;
}
```

- Verify order availiability

```
else if (cmd.equals("Confirm")){

    // get client socket
    Socket csocket = null;
    String IP = MSG.getIP();
    int num = RestaurantServer.socketList.size();
    for (int index = 0; index < num; index++) {
        Socket mSocket = RestaurantServer.socketList.get(index);
        String ip = mSocket.getInetAddress().getHostAddress();

        if (ip.equals(IP)) {
            csocket = mSocket;
        }
    }

    //inventory status in order
    //if inventory has enough item, add 1
    ArrayList<Integer> inventoryStatus = new ArrayList<>();

    //check from inventory
    //get data from inventory
    ArrayList<Item> inventoryList = InventoryUtil.readInventoryFromFile();

    //read data from socket
    Customer customer = (Customer) MSG.getData();
    ArrayList<Order> orderList = customer.getOrderList();
    //get orders in orderList
    for(Order order : orderList){

        //determine if each item in an order is available
        //get item from order
        Map<Item,Integer> item_q = order.getItem_QuantityMap();
        Iterator<Item> it = item_q.keySet().iterator();
        while(it.hasNext()) {
            Item item_order = it.next();
            //get item from inventory
            for(Item item_inventory : inventoryList) {
                //find item
                if(item_order.getName().equals(item_inventory.getName())) {
                    if(item_q.get(item_order) <= item_inventory.getStock()) {
                        inventoryStatus.add(1);
                    }
                    else{
                        inventoryStatus.add(0);
                    }
                }
            }
        }
```

```
//determine the order status
for(Integer status: inventoryStatus){
    //all 1, no 0 for status, whole order can be completed
    if(inventoryStatus.contains(1) && !inventoryStatus.contains(0)){
        MSG.setResult("Order has been accepted.");
        MSG.setFlag(true);
        orderList.remove(order);
        customer.setArrayList(orderList);
        MSG.setData(customer);
            MSG.setCmd("finish_confirm");
        //update Inventory
        InventoryUtil.updateInventoryByOrder(order);
        SendInventoryToKitchen.sendInventoryToKitchen(socket);

        MSG.setCmd("");

        ConfirmationService.sendMSGtoCustomer(MSG);
    }
    //all 0, no 1 for status, order should be canceled
    else if(inventoryStatus.contains(0) && !inventoryStatus.contains(1)){
        MSG.setResult("Order Canceled: no sufficient inventory.");
        MSG.setFlag(false);
        orderList.remove(order);
        customer.setArrayList(orderList);
        MSG.setData(customer);

        MSG.setCmd("");
        ConfirmationService.sendMSGtoCustomer(MSG);
    }
    //have 0 and 1 for status, order can be partially completed
    else{
        MSG.setResult("Order can be partially completed");

        //create new order
        int bq = 0;
        int cq = 0;
        int fq = 0;
        int oq = 0;

        ArrayList<Item> inventory =  InventoryUtil.readInventoryFromFile();
        for(Item iter : inventory) {
            if("Burger".equals(iter.getName())){
                bq = iter.getStock();
            }
            else if("Chickens".equals(iter.getName())){
                cq =  iter.getStock();
            }
            else if ("French Fries".equals(iter.getName())){
                fq =  iter.getStock();;
            }else if("Onion Rings".equals(iter.getName())){
                oq =  iter.getStock();
            }
        }
```

4.4. Data Persistence Module

The implementation of this module mainly calls the Java File I/O interface, writes the customer object data and the item object data to .txt, and stores the file to the specified address.

Chapter 5 Unit Testing

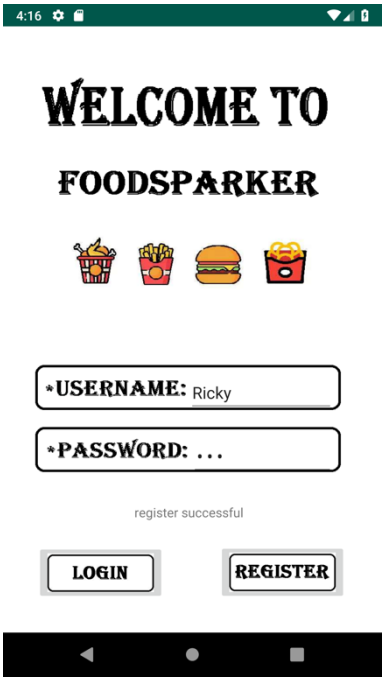
Unit testing is a level of software testing where individual components of a software are tested. The purpose of this process is to validate that each unit of the software performs as designed.

5.1. Functional testing

5.1.1 Customer-side functions

- Test customer registering function

If the customer is new to the app, the register function allows him/her to register an account with username and password. It will show “register successful” if the username has never been registered by other customers.

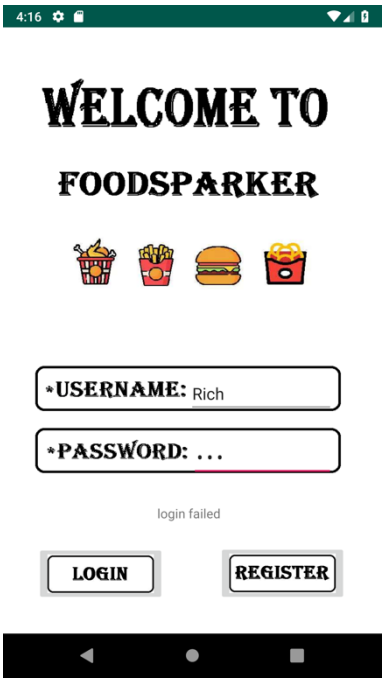


If the customer enters a username that has already been taken, the system will remind the customer that “register failed” and he/she has to re-register.

- Test customer logging in function

If the correct username and password are entered, the customer will be directed to our menu page to add item to the cart.

If a wrong username or password is entered, the system will remind the customer “login failed” and he/she has to re-register to enter the main menu.

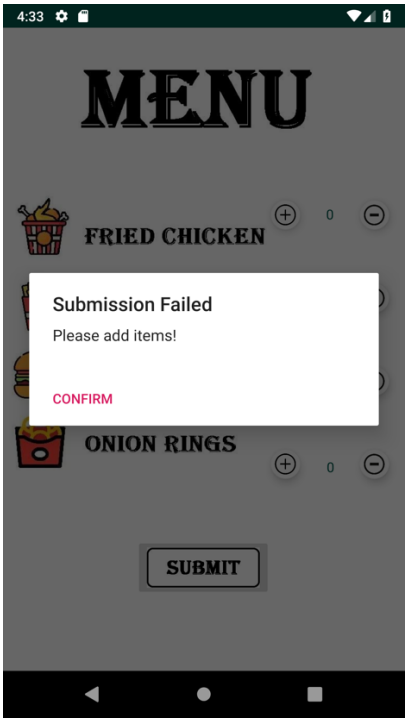


- Test adding/deleting food function

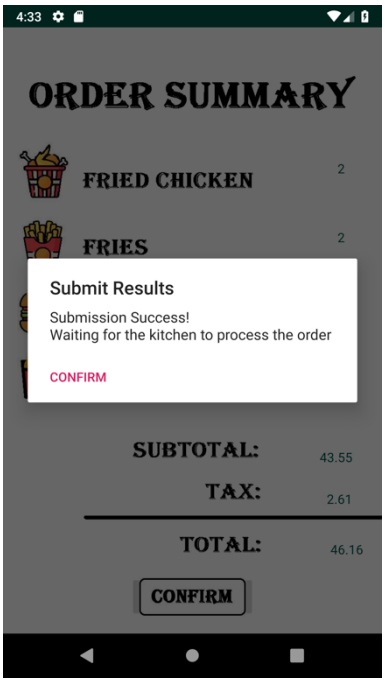
After successfully logging into the system, the customer is about to place an order using the add/delete food function. The subtotal price of the order will be calculated simultaneously.

- Test order submission function

If no item is selected and the submit button is pressed. A pop-up window will remind the customer to add item.

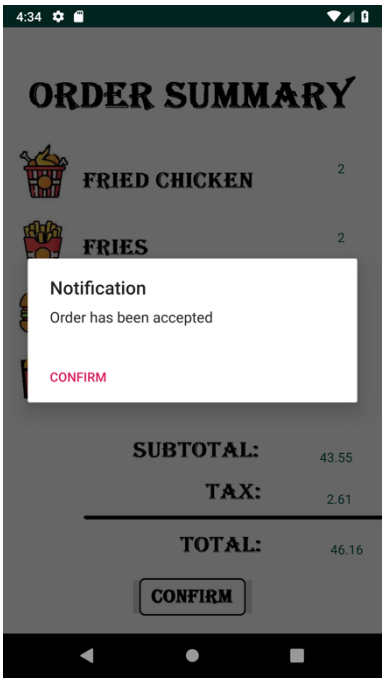


If the customer does put something in the cart and submit the order. A pop-up window will show that submission successful and the kitchen is processing the order.



- Test order confirmation function

After the kitchen-side confirm the order, the customer will receive a notification that the order has been accepted.

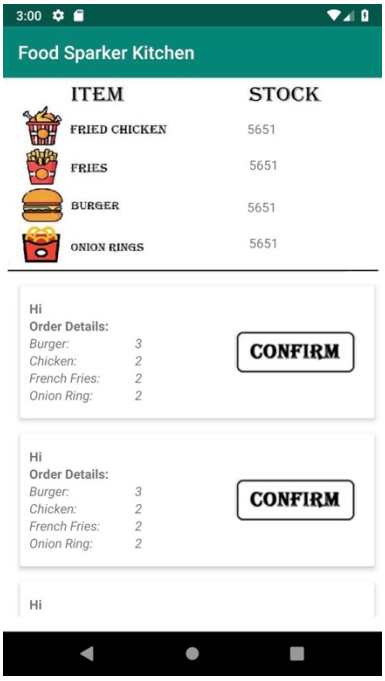


5.1.2. Kitchen-side functions

- Test inventory checking function

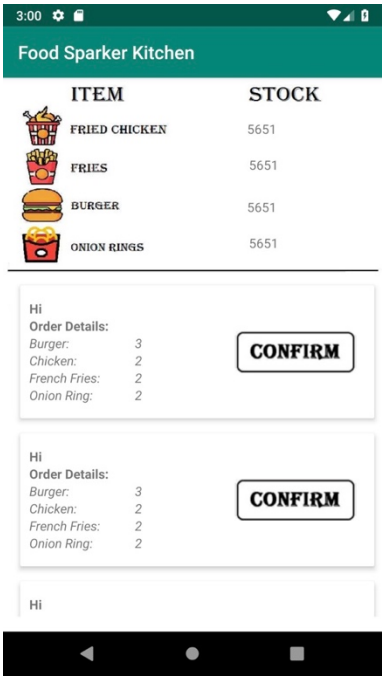
The kitchen side could monitor the inventory and customer order through their end.

From the testing view, the stock is increasing and the orders are sorted in lines by time.



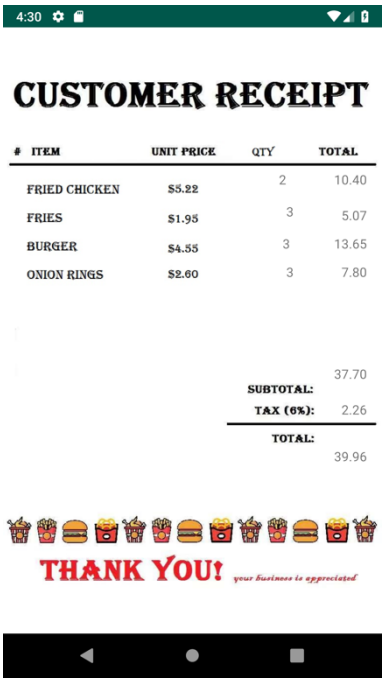
- Test order confirmation function

When the confirm button is pressed, the order disappears. The stock is deducted and the next order comes to the first.



- Test receipt printing function

Once the order is confirmed, the receipt of the order will be printed.



5.2. Non-functional testing

5.2.1. UI testing

The system's Graphical User Interface of the application we build involves the screens with the controls like menus, buttons, icons, and all types of dialogue boxes and windows. All of them work well in the testing.

5.2.2. Usability testing

Usability testing is the practice of testing how easy a design is to use on a group of representative users. In this process, we ask three of our friends to use our system to place an order and ask about their feedback. All of them feel it easy to use our app to place orders and they like the design of the user interface we created.