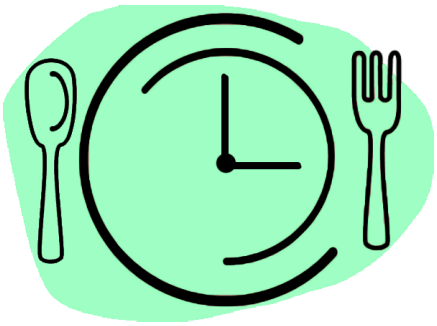

بسم الله الرحمن الرحيم

BSc Project
CS Department
UQU-CS-2015S-16
30 Nov 2015

Smart Restaurant Application

using beacon Technology



Supervised by: Dr.Aeshah alsiamy

Authors:

Ashwaq Al-Subhi (43200353)

Khulood Al-Malki (43200369)

Rehab Al-Megeireshi (43200954)

Sarah Saud Badri (43201123)

Samah Al-Thagafi (43200582)

Dept. of Computer science
Faculty of Computer and Information Systems
Umm Al-Qura University, KSA

Contact Information :

This project report is submitted to the Department of Computer and Information System at Umm Al-Qura University in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science**.**

Team Name :

Ideas Are Us

Team Logo :

Credit Hours: 4

Author(s):

leader : sarah saud badri

swerah-1414@hotmail.com

Ashwaq Al-Subhi
masaky1413@hotmail.com

KhuloodAl-malki
nanoo1432@hotmail.com

Rehab Al-megeireshi
ms-rgm12@hotmail.com

SamahAl-Thagafi
eroca8spe@hotmail.com

University supervisor:
Dr.Aeshah alsiamy
aasiyami@uqu.edu.sa
Computer Science

Dept. of Computer Science
Faculty of Computer and Information Systems
Umm Al Qura University Fax:

Internet: http://uqu.edu.sa

025270000 Kingdom of Saudi Arabia

Intellectual Property Right Declaration

This is to declare that the work under the supervision of Dr.Aeshah alsiamy having the title "Smart Restaurant Application using beacon technology" carried out in partial fulfillment of the requirements of Bachelor of Science in Computer Science, is the sole property of the Umm Al Qura University and the respective supervisor and is protected under the intellectual property right laws and conventions. It can only be considered/ used for purposes like extension for further enhancement, product development, adoption for commercial/organizational usage, etc., with the permission of the University and respective supervisor.

This above statement applies to all students and faculty members.

Date: _____

Author(s):

Name: Signature: _____

Name: Signature: _____

Name: Signature: _____

Name: Signature: _____

Name: Signature: _____

Supervisor(s):

Name: Dr.Aeshah alsiamy Signature: _____

Anti-Plagiarism Declaration

This is to declare that the above publication produced under the supervision of Dr.Aeshah alsiamy having the title "Smart Restaurant Application using beacon technology" is the sole contribution of the author(s) and no part hereof has been reproduced illegally (cut and paste) which can be considered as Plagiarism. All referenced parts have been used to argue the idea and have been cited properly. I/We will be responsible and liable for any consequence if violation of this declaration is proven.

Date: _____

Author(s):

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

ACKNOWLEDGMENTS

Thank to Allah for giving us the strength to complete a part of this project. This effort is dedicated to the Department of Computer Science, many thanks and gratitude for the help going through this experience, also to the great supervisor Dr.Aeshah alsiamy for guidance on this report, which will help to complete the project.

ABSTRACT

This project demonstrates a system that solves or minimizes the problem of waiting in a long queue at popular restaurants by developing a smart application.

Few techniques have been found that could be used in the system to attract the customer attention such as ibeacon technology, which can be used in the application to provide customers with a useful information of the restaurant, such as the restaurant status and sends offers information to customers when he is around the restaurant.

The system can be developed with android operating system using java programming language and create a database with SQL for the information entry.

GLOSSARY

iBeacons	Small devices, often battery-powered transmitters enable to learn what is nearby and allow mobile application to listen for signals from beacons.
Admin	The person who have the ability and the right to manage and manipulate the application database**.**
Cashier	Who have limited right in accessing the database.
Customers	Who can makes order using the application.
Estimote Beacon	Type of beacon related to estimate company which is a small wireless sensor that can be associated to any location or object.
Bluetooth	Technology with short range wireless communications, which mean to exchange data using radio transmissions.
External entities	Entity outside and interacts with the system.
Functional requirements	Functional requirements characterize what your system should do.
Non-functional requirements	Non-functional requirements characterize how your system works.
Estimote SDK	Which allows android application to interact with beacons.
The incremental model	A method for development a software where the model will be designed, implemented and tested incrementally.

TABLE OF CONTENTS

TABLE OF TABLES

Customers

Admin

TABLE OF FIGURES

Chapter 1 – INTRODUCTION

1.1 Purpose of the Project

1.2 Purpose of this Document

1.3 Motivation:

1.4 Project Statement

1.5 Project Contributions

1.6 Project Objectives

1.7 Project Scope

1.8 History of Beacons

1.8.1 Background of Estimote Beacons

1.9 Background of Bluetooth

1.10 Background of android

1.11 Feasibility Study of proposed system

1.12 Existing System

1.12.1 Latio

1.12.2 Mook Group

1.12.3 Bookatable

1.12.4 Frunk

1.12.5 Problems in the existing system

1.13 Overview of this Document

1.1 Purpose of the Project

One of the most important purposes of technology is to solve problems that we face in our life and to optimize our lifestyle. Also, it helps to save effort and time by applying new and smart technologies. Therefore, this project will apply the beacon sensor technology in a restaurant in order to help both the restaurant employee and customers through performing some tasks automatically.

iBeacons are small devices, often battery-powered transmitters that enable to identify what is nearby and allow Mobile Application (running on both iOS and Android devices) to listen for signals from beacons in the physical world and react accordingly. In essence, the underlying communication technology is the Bluetooth Low Energy[1].

What encouraged us to think of technologizing smart restaurant is the problem of waiting in long queues at popular restaurants. During peak hours, customers have to stand in queues for up to half an hour before being seated. Upon seating, you will be given the menu and it takes another 15 minutes to take your orders.

The application is designed to solve these problems and also will include different feature such as:

- Allow customers to see the menu and take their order before entering the restaurant or while waiting for an available table or waiting in the queue.
- Allow customers to keep up with special offers and process orders faster, if the order is being ordered via the application.
- Help customers to get an idea if the restaurant is full or not.

In the future, the smart restaurant application will be applied on more operating system in order to provide all means of comfort for restaurant customers.

1.2 Purpose of this Document

This document will present the development of the project. It will describe the idea on how to develop the system. It contains the whole information that is required for the system, presents the related system, and demonstrates the problems and the solutions. Also, it describes all the necessary steps to achieve such a successful project.

1.3 Motivation

Every idea has incentives that push it forward to make it concrete and applicable not just pent in thought. So, the idea starts off by noticing the huge spread of restaurants including cafes and cafeteria etc, and how people primarily depend on them in their food lifestyle. Accordingly, some problems occur like congestion, long queues and delayed orders, which can lead the customer to be nervous.

This affects the reputation of the restaurant, which can lead to reduce profits and closure of restaurants in the worst cases.

Therefore, building a smart application associated with beacon technology, to facilitate buying from restaurants, help the owner to manage the restaurants and to maintain customer's satisfaction. Moreover, smartphone devices serve as a key techniques for easing people daily life. So, the intention is to build a mobile application as a means of communication with customers and make the process of ordering and booking from a restaurant more convenient.

1.4 Project Statement

The project statement solves the regular problem of every restaurants that faces every customer, mainly lining up for long time.

This project will build an application that can be installed on android phones.

The application will bring menus to the customers. Also, it will notify them about special offers, restaurant state (full or not) and the available tables.

It will also show welcome messages in the customer phone when he enters the restaurant.

1.5 Project Contributions

Smart restaurant application will enable customer to make order, also, to display offers which dispatched by beacon sensor, and it will show a welcome message when entering the restaurant. By using beacon the restaurant can know if the customer is around or not for making confirmation or cancelation of his order. If the customer is around, total cost will be displayed, otherwise the order will be cancelled and rejected message will be sent as well. It also, allows the customer to know restaurant status .

1.6 Project Objectives

The objective of this project is to build a useful and flexible smart phone application.

These following points discuss the main goals and the functions that will be provided by the proposed application:

1. Easily used by the customers, because most people use smart phones and it will be very easy to just download the application.
2. To help customer by giving information about the restaurant status (full or available).
3. To display a welcome message that is shown in the customer's smart phone when entering the restaurant range.
4. To send messages for special offers to any one nearby the restaurant.
5. To display the menu for the customer in order to choose the order before entering the restaurant or while waiting for the available table.
6. To detect the customer's location and find if the customer is around the restaurant or not, if the customer is in the same area the order will be processed, if not it will be canceled.

7. To reduce messing up, before confirming the order the total cost will be calculated automatically and sent to the customer.
8. To show the number of available tables in the restaurant.
9. It is optional to create account in the application for protecting the privacy issue.

We suggest to apply this technology in our university cafeterias, to facilitate the work and to increase the number of students or staff who can benefit from these functions and technologies.

1.7 Project Scope:

The application can be used in restaurants "Sit-in or Take-Away", cafes and small cafeterias. The people who can use the application could be the restaurant owner in order to review bills. The system administrator can store the names of dishes in the menu and display them for the customer, update the welcome messages, number of tables and special offers. The cashier who can receive the customer order and close it . also, use the application for billing process. The application can store the customer information in the database, and can show these functions. Customers also can use the application to display the menu, place an order and benefit from application tasks.

Some techniques will be used to develop the restaurant system. For example : developing a database with SQL for saving information , using some programming language such as java in performing some task and developing a customer side part using eclipse. Also, we need Estimote beacons for dispatching specific information..

1.8 History of beacon:

Apple developed iBeacons in 2013 to help the retail industry by Facilitate payments and allowing offers on-site[2].In May 2014, many type of iBeacons hardware can be sold from at least $5 to more than $30 for one device.

IBeacon technology is still in the beginning, however; here there is a general idea about what is being discovered in this field:

McDonald's fast food restaurant:

McDonalds use iBeacons devices to send a special offers to the customers in its branches.

In the Home:

Beacons are used in homes, for example: turning the lights on when any ones enters a room or turning them off when exiting a room by connecting a phone via Bluetooth.

Smart Building Technologies:

Simple task like open the garage door automatically, controlling and monitoring lighting, security and safety are possible done using beacons.

Transportation:

Using beacons in a major public transportation areas such as for monitor the road, scheduling, ticket payment and make the traffic flow easier.

Healthcare & hospitals:

Using Beacons with a combination of other devices such as sensors to improve healthcare and also to use it in pharmacy, clinics and hospitals.

Tourism and museums:

You can see the beacons using in significant historic town centers and museums to create interactive round.

In Fitness:

Using Beacons in Fitness to offer some possibilities to encourage people to have more active live, such as using it in the gym equipment and make the training programs more interacting.

In Education:

The beacon was deployed to find education solutions such as interactive programs for children education, make the attendance automatically, indoor guiding mapping, and make the scheduling more feasible and convenient.

This will applied in many universities in US, UK, Canada, South Korea, Australia, Singapore, and Switzerland.

Smart cars:

The car will learn and know exactly what the people like when they are driving, and then make itself correspondence to who's driving or sitting in the passenger seat. The beacon in the car will be able to interact with the world around, while the passenger just have to sit back and enjoy the trip.

Beacons in Africa:

There is a team that working on project called "Beacons in Africa", this is one of the most perfect and helpful projects, this project will use the beacons technology to assist people in Africa such as track and analysis the disease in medical asset to improve healthcare, ensure elimination of common diseases, make the treatment monitor more feasible[2].

1.8.1 Background of Estimote Beacons :

What are Estimote Beacons?

Estimote Beacon is a small wireless sensors that can be associate to any location or object[3]. This sensor can broadcast small signals (radio) which any smart phone can receive and interpret, opening location service. Also, this beacon can be consider as small computer which has a strong 32 bit ARM (Advanced RISC Machine) RISC (Reduced Instruction Set Computer processor Cortex) with 256 KB flash memory , battery that can be powered up to three years, Bluetooth smart module and also new beacon has a motion and temperature sensors.

IBeacon identifier:

Estimote Beacons broadcast small packets of data or information that the programmer of beacon can specified, this data also containing their iBeacon ID and information about the strength of signal. So the Smartphone application can understand which beacon it listens and how far it is.

Every iBeacon ID is contain 20 bytes long and is divided into three parts:

- **UUID** (16 bytes) (Universally Unique Identifier)
- **major number** (2 bytes)(randomized , configure each restaurant to use a different “major” value for example: major 01 for restaurant branch number one and major 02 for restaurant branch number tow and so on)
- **minor number** (2 bytes)(randomized , placed the minor values according to the location inside the restaurant configure each of them to use a different “minor” value for example: minor 01 for the beacon that placed beside the door and minor 02 for the one that placed near the cashier and so on).

Those values are hierarchical.

There is a default UUID for All Beacons:

B9407F30-F5F8-466E-AFF9-25556B57FE6D

UUID, major, and minor values of specific beacons can be changed, however two Estimote Beacons cannot have the same ID.

Broadcasting Power and beacon’s range:

Broadcasting Power depends on the power which the beacon transmit or propagate its signal in its range. In Estimote Beacons, you can change it with Estimote SDK, Cloud, or app. The range have a value between -30 dBm and +4 dBm (Decibel-milliwatts), expect up to 40-50 m.

Broadcasting Power strongly impacts the range of signals. The more power, the longer the range is.

RSSI stands for (Received Signal Strength Indicator), which is used to measure or estimate approximate distance between the device and the beacon using another value specified by the iBeacon standard, called the measured power. The Measured Power show what's the predict RSSI at a distance of one meter to the beacon. Combined with RSSI which allows to measure the distance between the beacon and the device.

iBeacon specify four proximity zones for evaluating distance from current location to the beacon:

- Immediate (close to the beacon).
- Near (about 1-3 meter from the beacon).
- Far (away or the signal is vibrating too much to calculate the estimated value better).
- Unknown.

How to determine location?

If there is more than one beacon in the range, Smart phone can get more than one signal of beacon's BLE (Bluetooth Low Energy) at the same time. Accordingly, the Smartphone application will receive the locations from different beacons that is in the range and do some calculation for the distances.

1.9 Background of Bluetooth:

Bluetooth is a technology with short range wireless communications, which mean to exchange data using radio transmissions. It can be used between different set of devices like mobile phones and laptops. Bluetooth wireless technology is most widely supported, and secure wireless standard on the market today. The number of Bluetooth products on the market is increasing quickly[4].

Beacons is a small object transmitting distance information to smart phones and powered by Bluetooth.

Bluetooth brings some features to the beacon experience:

- Intelligent wireless connection to manage interactions.
- Extremely power efficient connection which mean that doesn't drain the phone battery.

1.10 Background of android:

Apple's iBeacon system uses BLE, it can connect only with iOS devices. But number of Android users more than iOS users by two to one.

IPhones and Android devices listen for signals of ID number broadcasted by beacons. Listening happens in iPhone within the operating system. It have a mobile application store a list of all beacons ID numbers that have been activated and will linked them with their location. If the device matches the ID of beacon with the ID in the mobile app, iPhone will then open the relevant application even if it is closed when they are within the beacon's range.

Android phone doesn't have a beacon system like apple level. If android users want to interact with beacons, they need to have an application running on their phone at least in the background to scan Bluetooth signals.

IBeacon will work better with Android devices, especially when we talk about battery life. By broadcasting the ID number (which is unique to a specific beacon at a location), beacons announce their presence. Mobile devices can then listen to these signals and position themselves relative to the beacon, which in-turn adds to battery drain. But Android allows to scan all beacon signals in the background, while Apple devices restrict background search to a pre-specified set of iBeacon identifiers alone[5].

1.11 Feasibility Study of proposed system:

In order to find out the main problem that restaurants have, we try to contact many restaurants to study the most and incurable problem that would really affect their sales, such as Al Baik and Domino's pizza restaurants. They actually talked about the long queue problem and how they try to reduce it by many ways, but they can't solve this problem. Therefore we told them about the beacon technology and how it will be used to solve this problem in an attractive way, they were really impressed about this technique. This meeting encouraged us to apply this technology to improve these restaurant's performance and reduce the current problem .

1.12 Existing System

With the rapid development of science and technology, every day cutting edge techniques and technologies. Each technique contributes to simplifying and facilitating human life. Accordingly, companies, restaurants, libraries… etc are racing to use these techniques to be the best and pioneers in their different field. Among the technologies that have appeared in recent years is beacon sensor technology.

In this project, the focus is on using beacon sensor and a smartphone technology in restaurants to help the employees and the customers in finishing some tasks automatically.

Many restaurant systems had been developed in the last few years that use the same idea or the same technique, but each one had met some objective criteria. So the literature study on previous system will give more reference experience and will give us greater chance for developing a better applications.

Being familiar with the previous system carefully and discuss the advantages and disadvantages of them, might help in improving the idea and combining some features of these systems in the proposed application in a different form.

1.12.1 Latio:

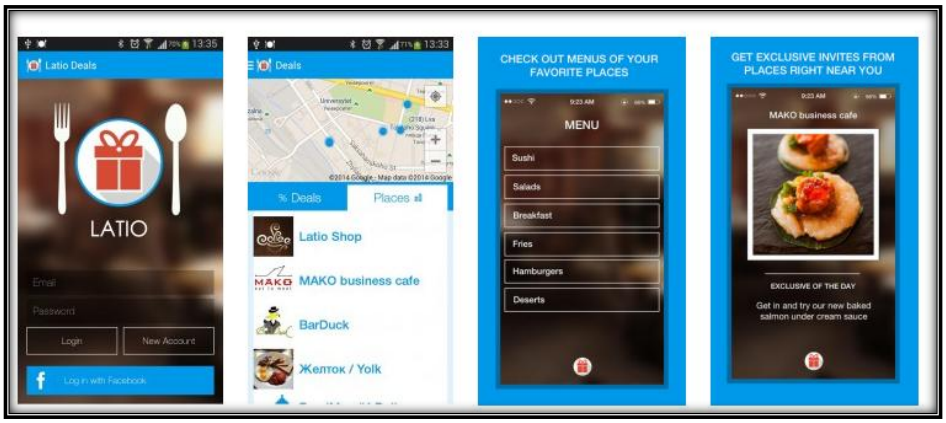


Figure 1: LATIO application

Latio, is a known restaurants application [6]. This application has been developed and implemented based on iBeacon technology. The application let restaurant and coffee house visitors to get relevant information on the place, as well as take part in various quests.

Users benefit by receiving invitations and detailed information from restaurants when passing by, as well as displaying digital menus with prices and photos near or inside restaurants.

User receives an invitation within the radius of 50 to 100 m (160 – 320 feet). It comes as a Push notification from the application. The Ukrainian startup promotes restaurants to use special offers for the users who receive such invitations.

From this principal idea, it has been decided to apply the same concept of displaying menus to the customers before they are even at the restaurants, but the difference is that the system will not accept the order before detecting the position of the customer if nearby range of the beacon or not. The system also will show a welcome message in the customer's smart phone when entering the restaurant range.

1. 12.2 Mook Group:

Figure 2:MOOK application

Managers at Frankfurt's upscale restaurants proud themselves on knowing the names of most of their regular customers [7].

Soon, they will cognize the diner's favorite table, exactly how long they spend in the restaurant, what dish they order most–and they'll even know when their customer is not in their table like they go to bathroom.

Restaurateur Christian Mook, is the owner of 5 high-end restaurants known as the Mook Group, is experiment an application using iBeacon technology, which tracks customers in the restaurants. If it works in one of the restaurants, Mook will roll it out to the other four restaurants.

The application calculates the time that guests spend in the restaurant and uses a ranking system to reward them for their allegiance. Mook says it later hopes to control indoor location, items and the amount customers pay all with a keenness toward offering new services and improving existing ones.

Some of the Mook Group feature violate the customer privacy. Therefore, this application will track the number of customer inside the restaurant only in order to find out if the restaurant is full or not.

1. 12.3 Bookatable:

Bookatable is a Europe's largest online restaurant reservation platform [8]. Bookatable, is currently testing beacon technology at 119 restaurants in London. In this application, the beacons will be used to send offers and menus to potential diners that have the Bookatable application on their smartphone as they pass by the participating locations.

This proposed application will send offers to people who pass by the restaurant and they have the smart restaurant application in addition to bring menus to the customers before they are seated at their table. Beacons will dispatched its distance to calculate location .

1.12.4 Frunk :

Frunk application is a new startup in Singapore [9]. Frunk re-think of a problem of long queue at different popular restaurants during peak hours.

Frunk enable the customers to view the menu and make decisions about their order while still in the queue.

The moment customers are seated at the tables, all they need to do is to tap their phones to the sticker on the table and the order will be sent immediately to the merchant. Orders will not be able to send unless customers are seated at their table at the restaurant.

Figure 3: Frunk application

In Frunk , iBeacons are used to determine the exact table number, and when verification is successful the orders will be sent .

In this project, some features have been added to the application that enable the customers to view the menu while still in the queue. However, beacons are used to send the distance to customers application to know if he/she is around the restaurant to confirm the order after do some calculation instead of tapping the sticker to make sure the customer serious about the order.

1.12.5 Problems in the existing system

Mook Group:

This application may cause the customers uncomfortable because it violated their privacy.

Bookatable:

In this application there is a risk that customers may not come to pick their orders, so this may cause the restaurant some loses because there is nothing guarantees the customer will come.

Also the application didn't use any other way to reduce the loses like using beacon to detect if the customer around or not.

Frunk:

In this application, the order maybe late, because the customer have to seat first then tap on the sticker then the order will be sent.

1.13 Overview of this Document

This report is structured as the following :

Chapter 1 provides an introduction and background about the project and present a general overview of the related systems.

Chapter 2 defines the functional and non-functional requirement specification and describe the proposed system solutions with alternatives.

Chapter 3 presents the architectural strategies and mention the hardware and software environment and future plans of the project.

Chapter 4 discusses the system design.

Chapter 5 shows its implementation and validation.

Chapter 2-SYSTEM ANALYSIS

2.1 Data Analysis

2.1.1 Data flow diagrams

2.1.2 System requirements

2.1.2.1 project stakeholders

2.1.2.2 Functional and data requirements

2.1.2.3 Non-functional requirements

2.1.2.3.1 Look and feel requirements

2.1.2.3.2 Usability requirements

2.1.2.3.3 Access control requirements

2.1.2.3.4 Performance requirement

2.1.2.3.5 Maintainability

2.1.2.3.6 Feasibility

2.1.2.3.7 Availability

2.1.2.3.8 Accessibility

2.1.2.3.9 Extensibility

2.1.2.3.10 Authorization

2.1.3 Actors use cases

2.1.4 Proposed Solutions and Alternative Solutions

2.1 Data Analysis

For any application or project you should specify requirements to make it easier later for the designer to build his project fully and properly.

The system requirements and its specification will be discussed in this chapter.

2.1.1 Data flow diagrams

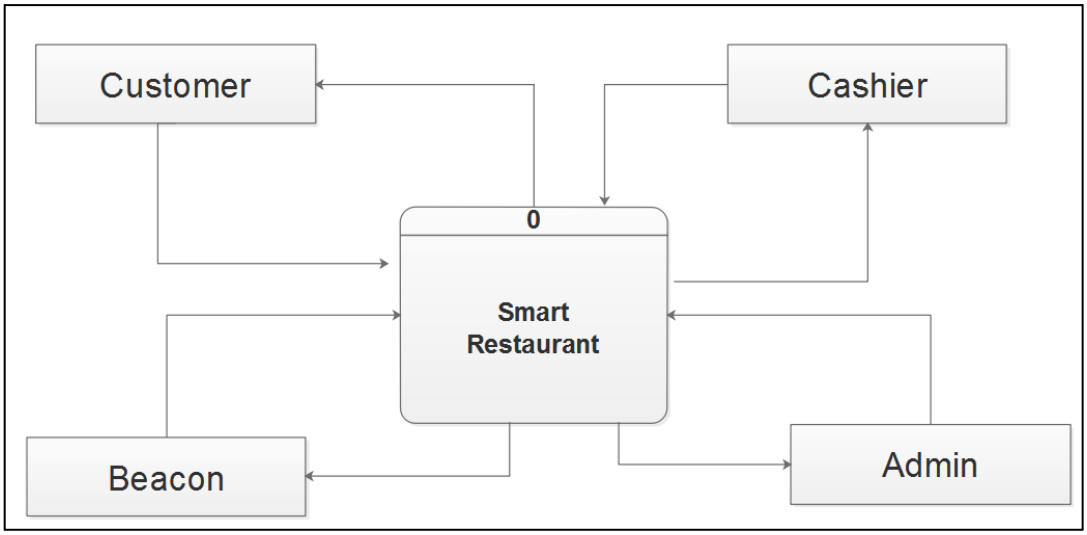


Figure 4 :Simple Context diagram

The diagram shows an overview of the whole system. The system will interact with four entities. These entities are admin, customer, cashier and beacon , each entity plays different role and has different tasks as it will be presented in the following diagrams

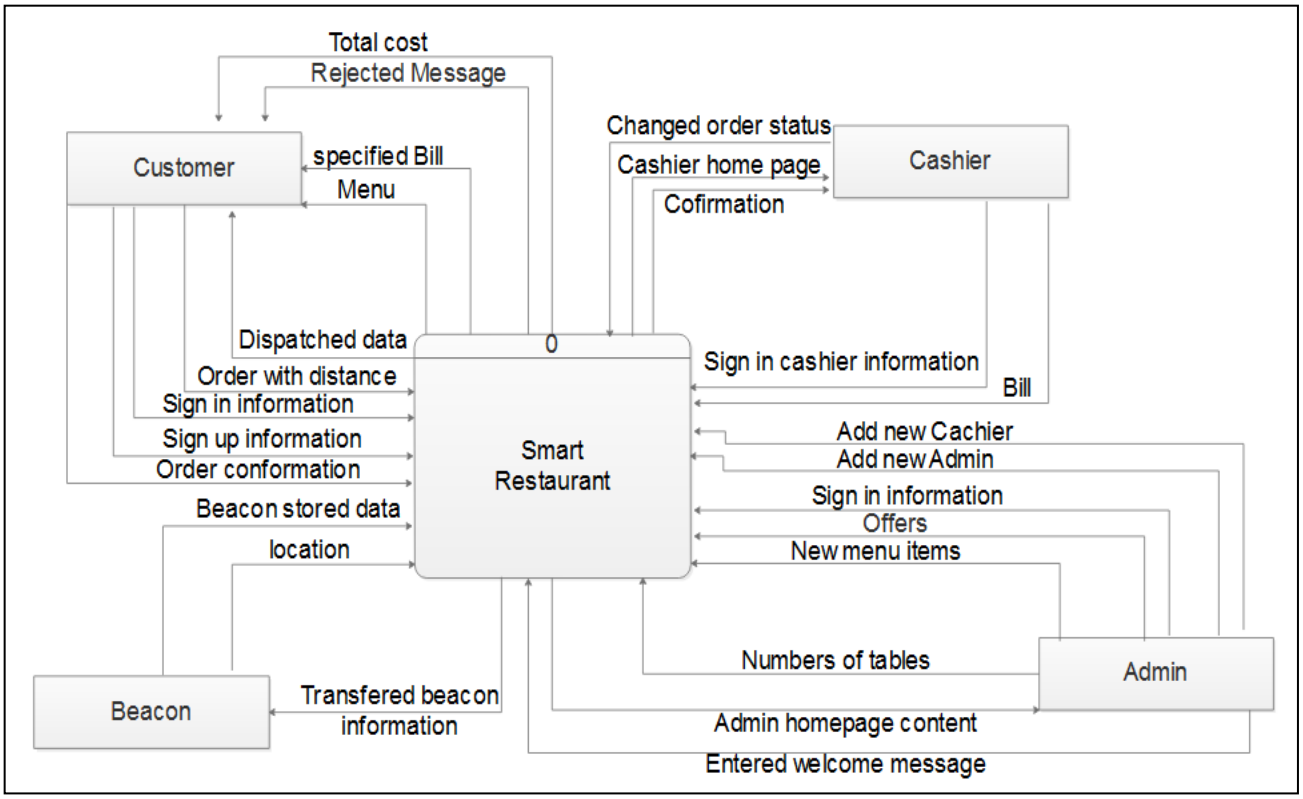


Figure 4.1:Context diagram Details

These interactions between the system and these external entities are shown in the diagram as flow in and flow out **. The detailed descriptions of these interaction will be discussed bellow..**

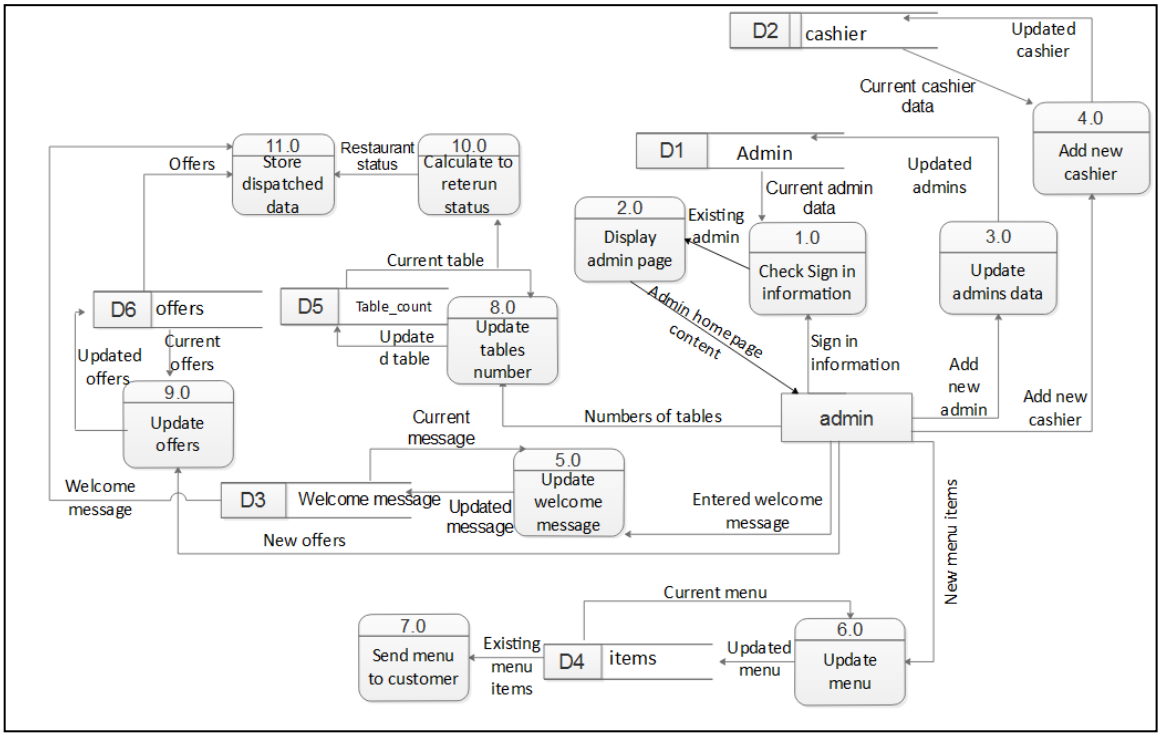


Figure 5: Admin DFD

This diagram displayed the admin part, which includes all his interaction with the system as bellow :

- The admin will sign in, his information should be checked by the system to verify if he allowed to enter the database and manipulate it. If the admin information is already in the database, home page content will be shown .
- New admin information can be added into **admin** table by the main admin, then the database will be updated .
- New cashier information can be added into **cashier** table by the main admin, then the database will be updated .
- Welcome message will updated by the admin, forwarded to beacon and dispatched later to the customer mobile.
- New offers for restaurant are inserted to the data store that is called offer, after that they are send to the beacon database .
- Tables that already in the restaurant are inserted, data store that called table_count will be updated with the update tables number, after that they are sent to the beacon .
- Menu items are inserted into the items table, that will be displayed in customer application.

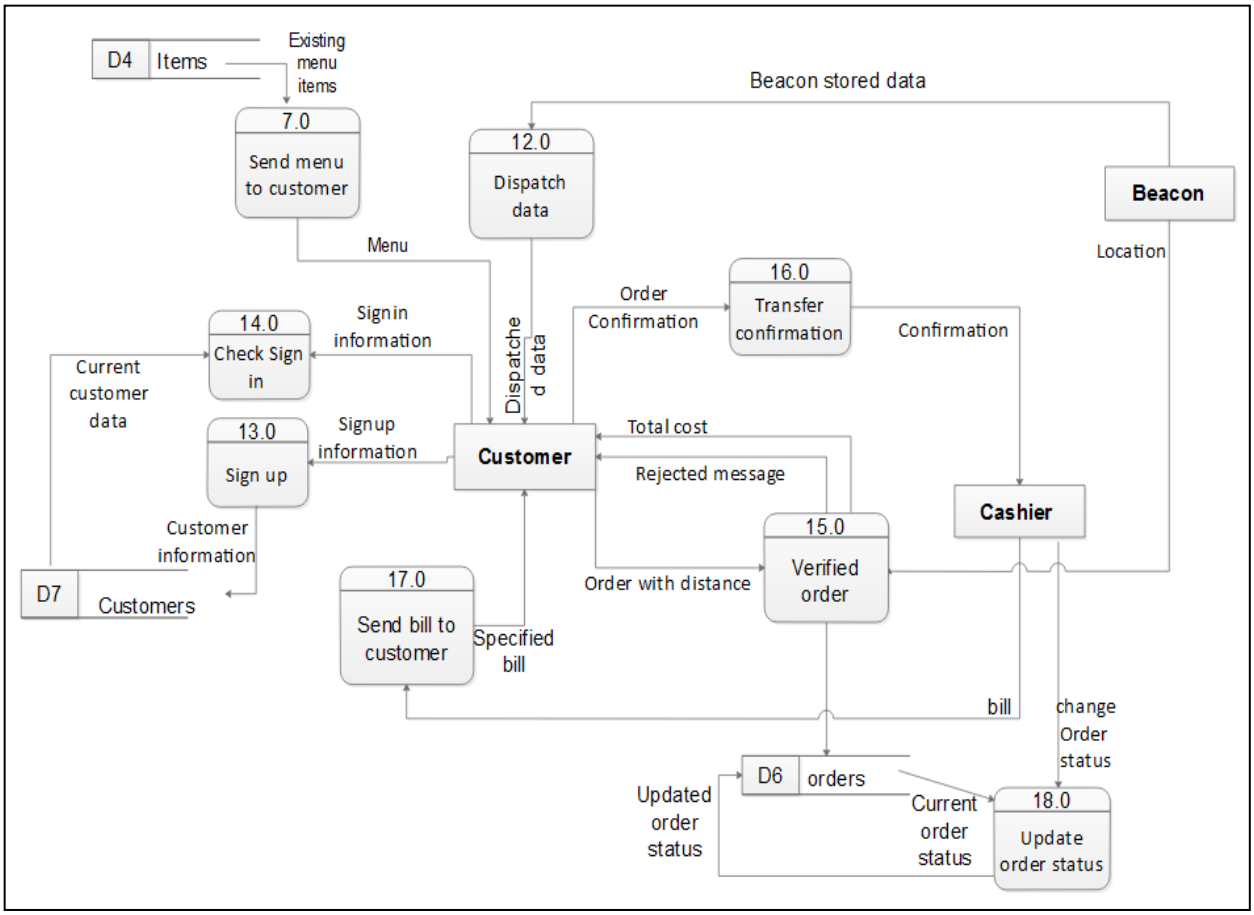


Figure 6: Customer DFD

This diagram displays the customer part and his tasks as following :

- Customer can sign up (optionally), then sign in to display the home screen, he can access the home page without signing in .
- The application will check the beacon signals, the customer will received beacon dispatched data if he is in the beacon range (dispatched data includes : The welcome message , special offers , the restaurant status and beacon location).
    - The welcome message and the special offers notification will be received and shown to the customer.
    - The customer can open the restaurant status screen to see the available tables number.
- The customer can open the menu screen, choose items from the menu and send his order with distance.
- The beacon location which dispatched by beacon will be received by the customer application , the distance between the customer location and beacon location will be calculated. If the customer in the restaurant range, the total cost will be received . Otherwise, the rejected message will be received.
- The verified orders will be stored in data store called orders.
- The customer will receive the total cost if he/she agreed to complete, then the confirmation will be sent and bill will be received .

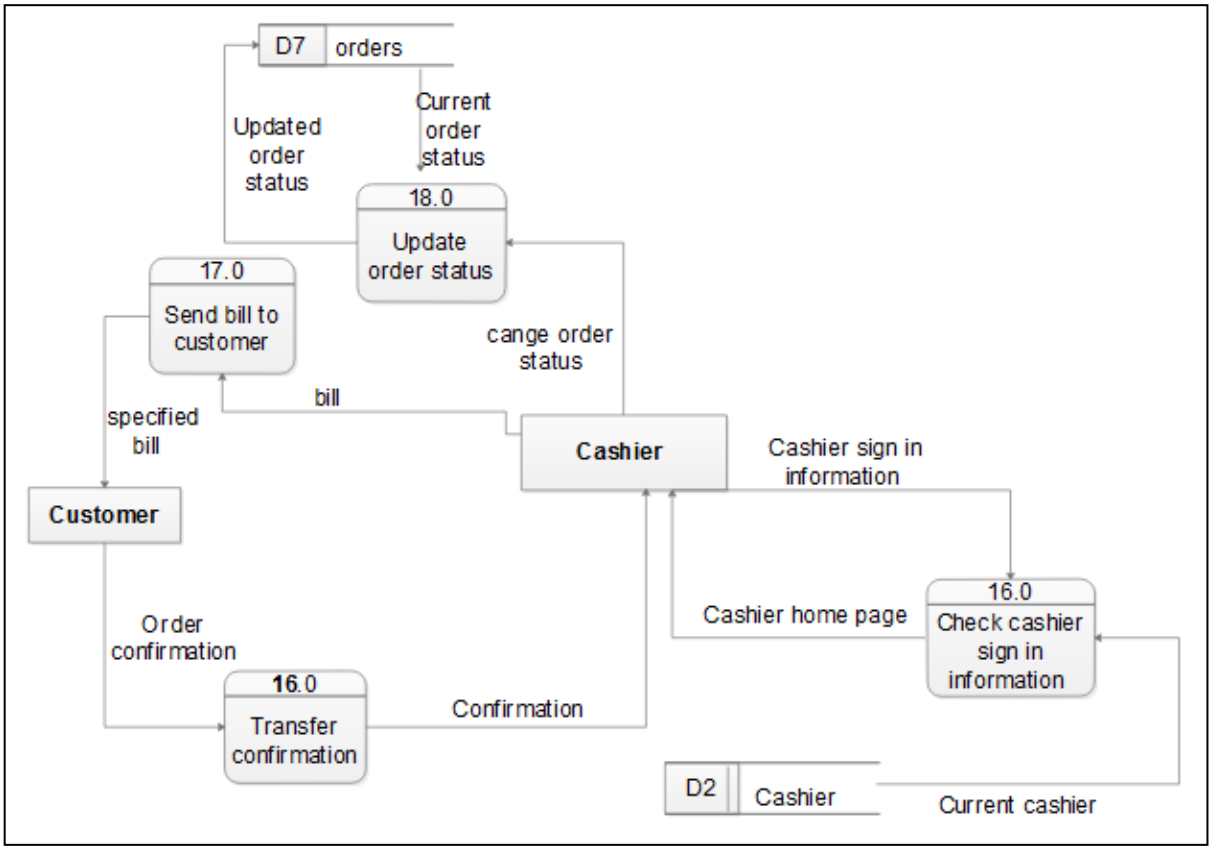


Figure 7 :Cashier DFD

This diagram shows how the cashier interact with the system :

- Cashier can sign in (cashier information will be stored in cashier data store by the main admin) then the information should be checked by the system to know if he is registered or not.
- If the information is verified the home screen will be shown.
- After changing the orders status to close the attached table status will change to available.
- Depending on previous point the restaurant status will be changed to full or not.
- customer can confirm the order.
- After receiving the conformation from the customer, the bill will be sent .
- After order completing, cashier will change the status of the order to 'close'.

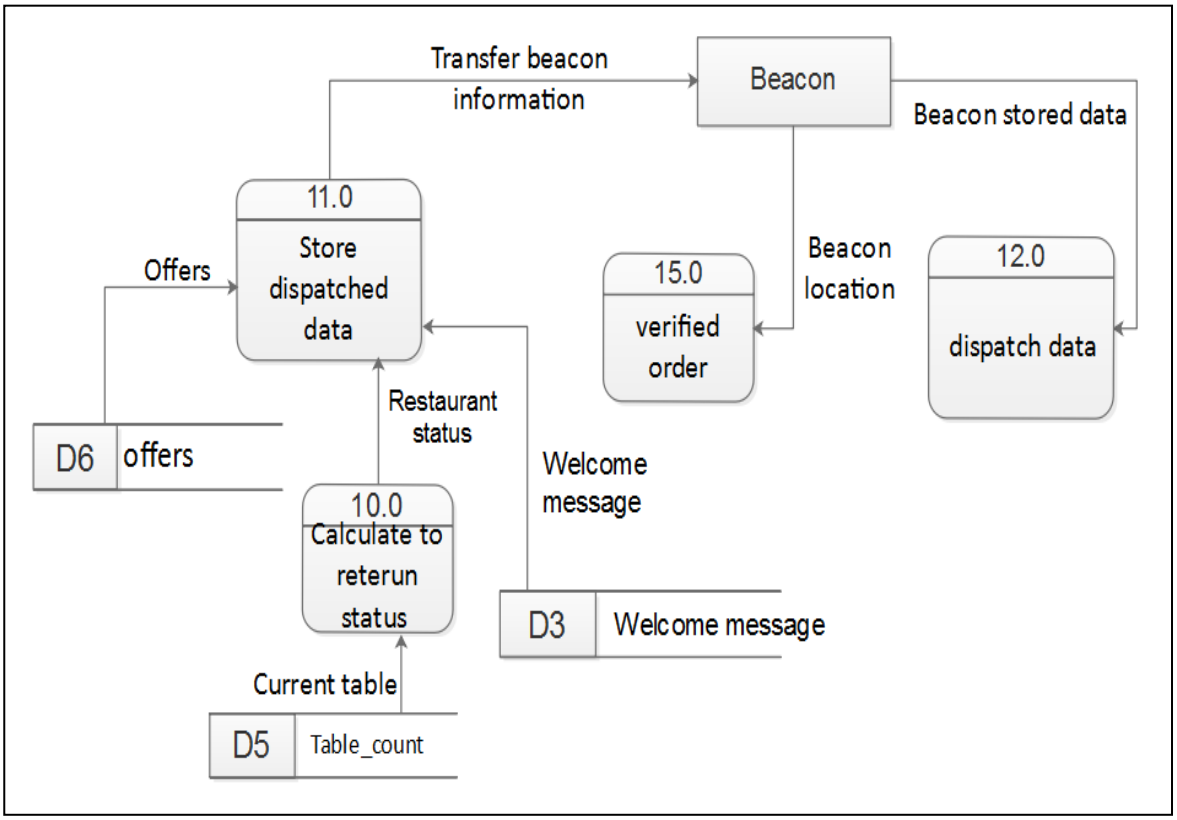


Figure 8 :Beacon DFD

This diagram shows the beacon interaction as following :

- Information of offers, welcome message and restaurant status will be updated .
- The beacon will send its location to customer application which calculate the distance between customer location and beacon location.
- Information that is stored in data stores :welcome message ,offers, table_count will be dispatched to customer.

2.1.2 System requirements

2.1.2.1 project stakeholders

- Restaurants: places where the application can be applied .
- Customers: who can contact directly with the application to show menu and place his order with distance and receive dispatched data from beacon, includes : The welcome message , the special offers notification ,current restaurant status and beacon location.
- Staff : people who work in the restaurant.
- Owner : who buy the application and support his restaurant with it .
- Administer : who has the right to enter into the database and update it, which responsible to manipulate tables, menu items, add new admin or cashier, welcome message and special offers.
- Cashier: who is responsible for close the order**.**

2.1.2.2 Functional and data requirements

The functional requirements determine the processes and tasks which the system and actors must perform. For smart restaurant it will be divided into five parts: the system, the admin, the customer, the cashier and the beacon .

For the system:

1- The system should be able to store menu of the restaurant.
2- The system should be able to store data of costumers and update it automatically.
3- The system should be able to calculate each customer order total cost and send it to him automatically.
4- The system should update each table state continuously and display available one to the customer.
5- The system should enable any customer to register or sign in to the system.
6- The system should be able to receive customer's orders.
7- The system should be able to receive customer's confirmation after ordering and calculating cost.
8- The system should be able to generate bill for customer.

For the admin:

1- The admin can add new admin or new cashier.
2- The admin can update table number.
3- The admin can add, update and delete menu.
4- The admin can add, update and delete menu items.
5- The admin can add, update and delete special offers.
6- The admin can update welcome message.

For the customer:

1- The customer should be able to sign in or sign up.
2- The customer should be able to make an order.

3- The customer should be able to see the available table.
4- The customer should be able to see the menu.
5- The customer should be able to receive special offers.
6- The customer should be able to receive restaurant status.
7- The customer should be able to send his distance with order.
8- The customer should be able to receive the total cost.

For the beacon:

1- The beacon is able to send its location to the customer application.
2- The beacon is able to dispatch welcome message for customers .
3- The beacon is able to dispatch offers to customers.
4- The beacon is able to dispatch restaurant status to customers.

For the cashier:

1- The cashier should be able to update order status.
2- The cashier should be able to receive customer order .
3- The cashier should be able to calculate order total cost.

Data requirements:

In smart restaurant system, a few information will be needed about the customer or the environment in order to determine the customer locations to indicate if he inside the restaurant range to accept his order .The determination of the customer distance is done by the beacon and customer application.

2.1.2.2 Non-functional requirements

2.1.2.2.1 Look and feel requirements

- Interfaces will be clear and simple.
- Application will have quick responses and smooth movement between different application interfaces.
- Icons in all interfaces will be clear and in suitable quality.
- The font color and size should be suitable and clear.
- The color which will be used for the background will be suitable.

2.1.2.3.2 Usability requirements

- The icon names will be easy to understand by different users, so they can learn how to use application by themselves .
- The application will be used by the user without the need for prior instruction.

- The application will be easy for naive user, because it will not include complex things in its different interfaces and the words that used in the application straight-forward.
- The application use for order almost same way as traditional restaurant applications, user supposed to know how to order.
- The main page will display the Smart Restaurant application main options, the user can use the main properties easily and directly .

2.1.2.3.3 Access control

- The admin will use specific username and password to access the database.
- No one can access the application database, just the admin can access it and modify the data.

2.1.2.3.4 Performance requirement

- The loading of distinct interfaces will not take a lot of time to load " just few seconds".
- The application will not use a lot of interfaces to make the movement between them faster.
- The application will be interactive and has Fast responses for different interactions.

2.1.2.3.5 Maintainability

- The system will have the ability to perform a successful repair action within a given time in case of failure occur. For example: if beacon be out of date then it can replaced with new one easily, also data will transferred to beacon as usual from existing database in restaurant system.

- Depending on the incremental method that will be used, proposed system will not affected by changing any requirement.

2.1.2.3.6 Feasibility

The feasibility of the application is done by using the open sources such as java and MYSQL.

2.1.2.3.7 Availability

The application will be free, the customer will be enable to download it and install it.

2.1.2.3.8 Accessibility

Anyone can install, access the application and use its features as they like.

2.1.2.3.9 Extensibility

Right now, this application will apply on one restaurant as a use case, and as future plan other restaurant branches or other restaurant can apply it as what happened in Frunk application when they add other restaurants menus to the application [9].

2.1.2.3.10 Authorization

The system gives the access authorization to only specific actors which means that no one from outside the system can access the system and not any actor who can access the system can make any change in the database , for example: adding a cashier or change data in the database of the system, only the admin has this permission by entering his information to system database. For the user the system just allows him to make some action and show some interface according to his role .

2.1.3 Actors use cases:

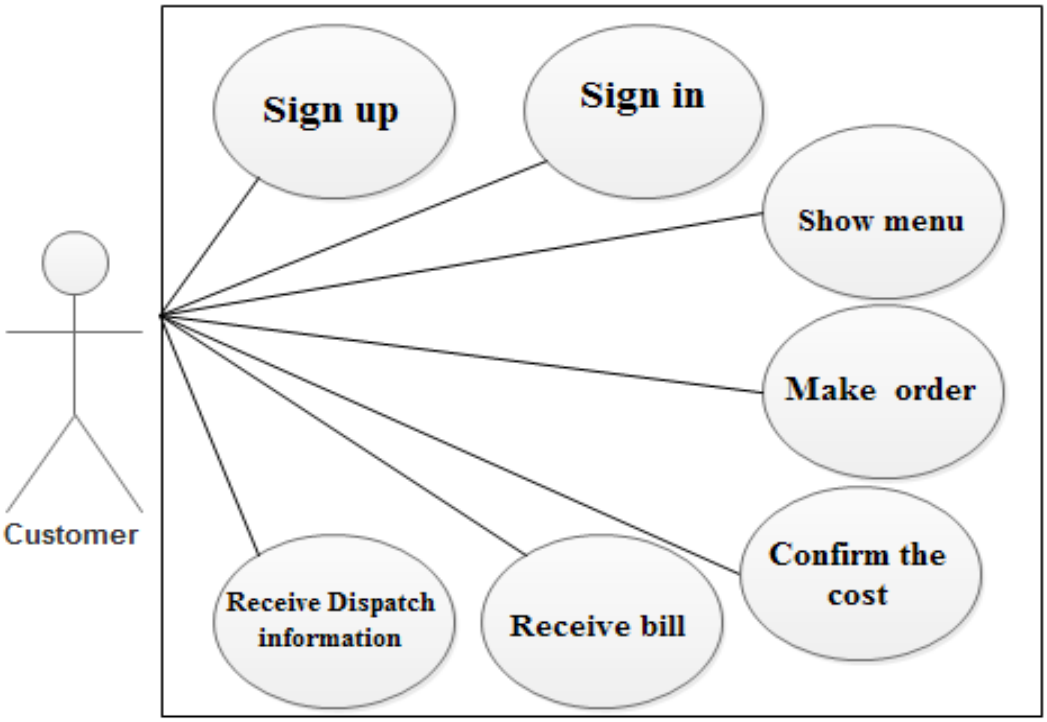


Figure 9:Customer use case

Figure 9.1 : Sign up

Table 1 :Sign up

Use case name : Sign up **Unique customerID**
Actor(s): The customer**.**
Description: Enable customer to sign up into restaurant application.
Steps Performed (main Path): 1- Open the application.
2-Choose sign up screen. 3-Fill the essential information in the form. 4-Click sign up button. 5-The customer information will successfully be stored in the database.
Preconditions: The customer must have the restaurant application.
Assumptions: The customer knows how to sign up.

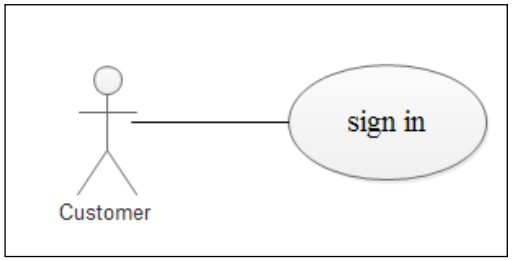


Figure 9.2 : Sign in

Table 2 :Sign in

Use case name : Sign in **Unique customerID**
Actor(s): Customer.
Description: Enable customer to sign in through restaurant application.
Steps Performed (main Path): 1 - Open the application.
2 –Click sign in button. 3- Enter his information. 5- Confirmation of successfully sign in is then displayed.
Preconditions: 1-The customer must have the restaurant application and already open it. 2-The customer should have an account in the application.
Assumptions: The customer knows how to sign in.

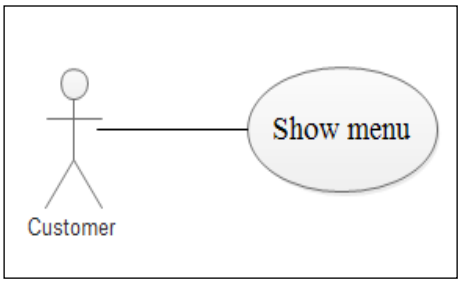


Figure 9.3 :Show menu

Table 3: Show menu

Use case name: Show menu. **Unique menuID**
Actor(s): Customer.
Description: Enable the customer to show menu that was stored in database.
Steps Performed (main Path) 1- Open the application. 2-Choose the order screen. 3- The customer can choose items from the menu.
Preconditions: The customer must have restaurant application and already open it.
Assumption: The customer has the ability to show menu.

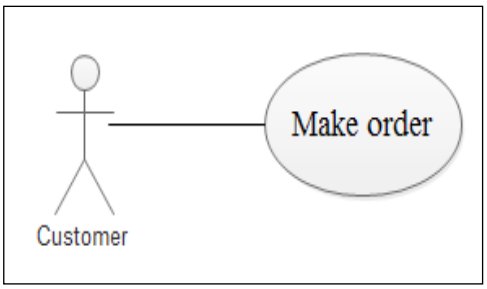


Figure 9.4 : make order

Table 4 :Make order

Use case name : Make order **Unique orderID**
Actor(s): Customer.
Description: Enable the customer to make his order from menu .
Steps Performed (main Path) 1- Opens the application. 2-Signs in or sign up (it's optional). 3-Choose order screen. 4- Choose some items from the menu with quantity . 5-Choose the order type (take-away, sit-in) 6- Click ok button.
Preconditions: The customer must have restaurant application and already open it.
Assumption: The customer has ability to make his order.

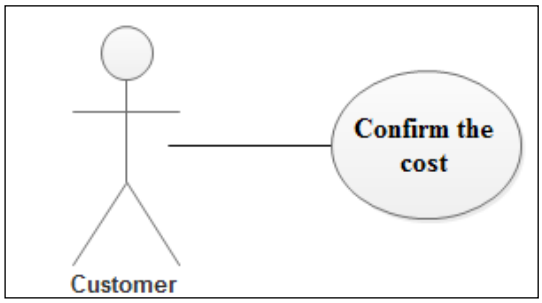


Figure 9.5 : Confirm the cost

Table 5: Confirm the cost

Use case name : Confirm the cost. **Unique billID**
Actor(s): The customer.
Description: Allow the customer to confirm the cost.
Steps Performed (main Path) 1- Order has done by customer. 2-Total cost is received by the customer if he inside the restaurant range . 3-Customer confirm the cost .
Preconditions: 1-The customer must have the restaurant application and already open it. 2-The customer already makes order and it's verified.
Assumption: The customer has the ability confirm the cost.

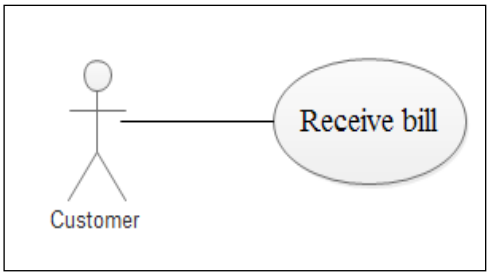


Figure 9.6 : Receive bill

Table 6 :Receive bill

Use case name: Receive bill. **Unique billID**
Actor(s): Customer.
Description: Enable the customer to receive bill for his order.
Steps Performed (main Path) 1- Open the application. 2-Order was made by customer and order has verified . 3- The customer receives the bill.
Preconditions: 1-The customer must have the restaurant application and already open it. 2-The customer already makes order and it's verified.
Assumption: The customer has the ability to receive bill from application.

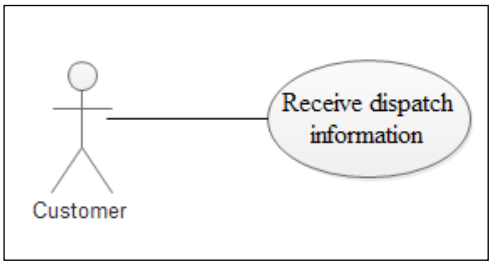


Figure 9.7: Receive dispatch information

Table 7:Receive dispatch information

Use case name: Receive dispatch information. **Unique beaconID**
Actor(s): Customer.
Description: Enable the customer to receive the dispatch Information.
Steps Performed (main Path): 1-The customer should install the application 2-Opens the application and Bluetooth . 3-The customer should be in the beacon range. 4- Receives dispatch information.
Preconditions: 1- The customer must have restaurant application and already open it. 2- The customer has to be in beacon range.
Assumption: The customer has ability to receive dispatch information.

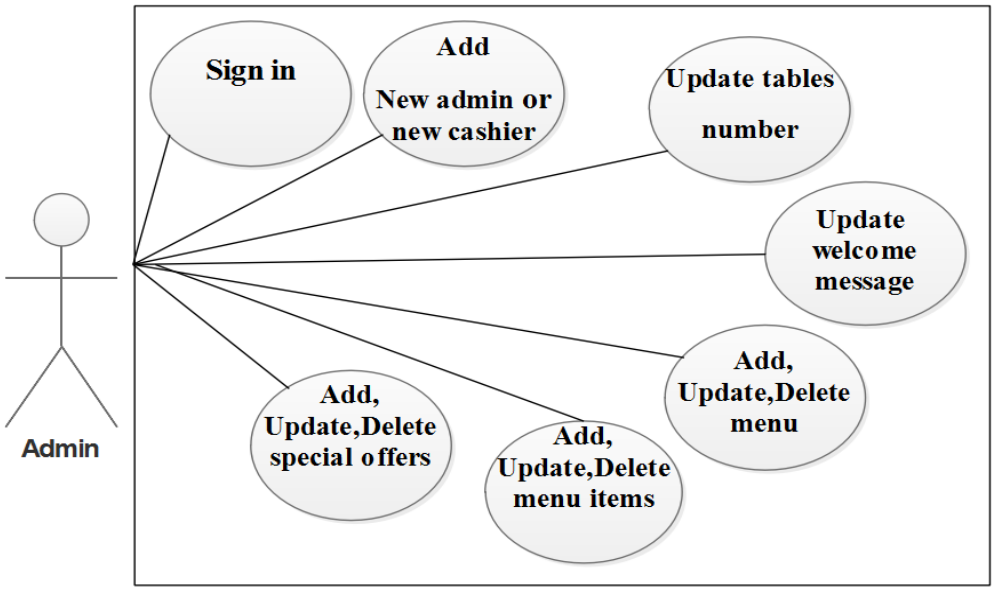


Figure 10:Admin use case

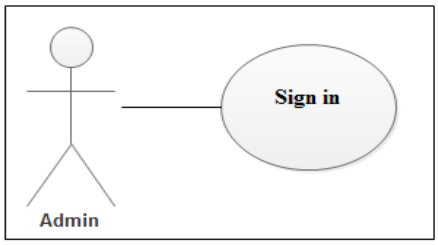


Figure 10.1 : Sign in

Table 8 :Sign in

Use case name : Sign in **unique adminID**
Actor(s): The administrator.
Description: Enable the administrator to sign in to the database.
Steps Performed (main Path): 1 - Open the website .
2 -Click login button. 3- Enter his information. 4- Confirmation of successfully sign in is sent.
Preconditions: 1- The administrator must open the website . 2- The administrator should have an account in the database.
Assumptions: The administrator knows how to sign in

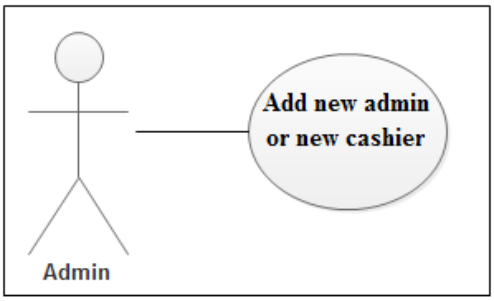


Figure 10.2 : Add new admin or new cashier

Table 9: Add new admin or new cashier

Use case name : Add new admin or new cashier **Unique admin ID**
Actor(s): The administrator
Description: Allow the administrator to add new admin or new cashier into the restaurant database.
Steps Performed (main Path):
1- The administrator signs in. 2- The administrator clicks on admin or cashier option. 3- The administrator enters new admin or new cashier information (ID, Name and other information) then click add button. 4- A successful adding confirmation page will be sent to the administrator.
Preconditions: The administrator should be already signed in.
Assumptions: The administrator has an account and permission to add new admin or new cashier.

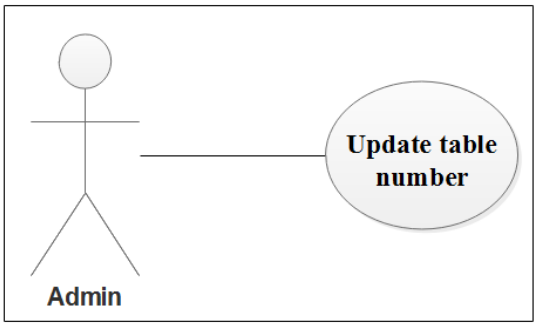


Figure 10.3 : Update tables number

Table 10: Update tables number

Use case name : Update tables number **Unique Table No**
Actor(s): Administrator
Description: Allow administrator to add table into restaurant database
Steps Performed (main Path):
1- Administrator will sign in. 2- Administrator clicks on update tables button. 3- Administrator enters table count (number of tables) then click save button. 4-Successful updating then confirmation page is sent to administrator.
Preconditions: Administrator should be signed in.
Assumptions: Administrator has an account and permission to add table.

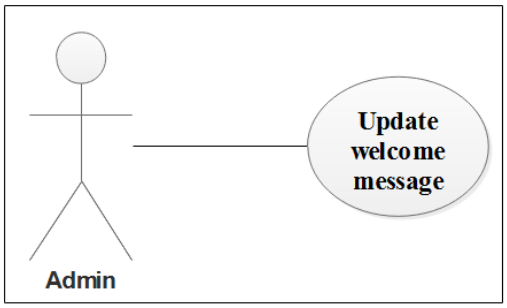


Figure 10.4 : Update welcome message

Table11:Update welcome message

Use case name: Update welcome message . **Unique Mssg Title**
Actor(s): The administrator
Description: Allow the administrator to update welcome message from restaurant database**.**
Steps Performed (main Path): **1-** The administrator sign in. **2-** The administrator clicks on welcome message option. 3- The administrator enters welcome message description then click save button. **4-** A successful adding then send a confirmation page to the administrator.
Preconditions: The administrator has already signed in.
Assumption: The administrator has an account and permission to update welcome message.

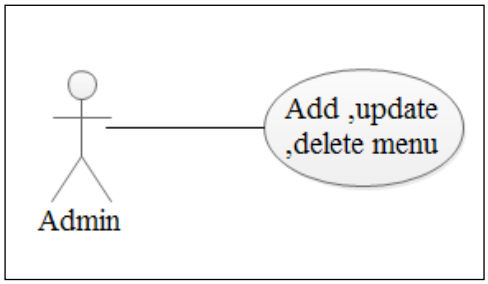


Figure 10.5: Add ,update ,delete menu

Table 12: Add ,update ,delete menu

Use case name : Add ,update ,delete menu **Unique menuID**
Actor(s): The administrator
Description: Allow the administrator to add ,update ,delete menu into restaurant database
Steps Performed (main Path):
1- The administrator signs in. 2- The administrator clicks on item option. 3- The administrator enters the items information (itemName, price and the category) 4- a successful adding or updating or deleting will send confirmation message to the administrator.
Preconditions: The administrator has already sign in.
Assumptions: The administrator has an account and permission to add, update and delete menu .

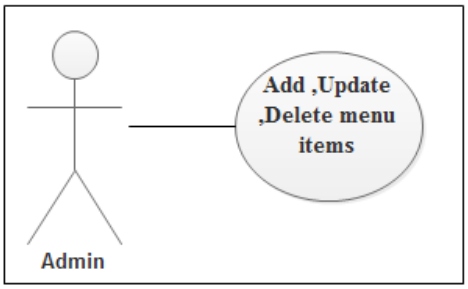


Figure 10.6: Add ,update ,delete menu items

Table 13: Add ,update ,delete menu items

Use case name : Add ,update ,delete menu item **Unique menuItemID**
Actor(s): The administrator
Description: Allow the administrator to add ,update ,delete menu item into restaurant database
Steps Performed (main Path):
1- The administrator signs in. 2- The administrator clicks on menu items option . 3- The administrator enters menu information (Item ID, itemName, price and other information) then click add new button. 4- a successful adding or updating or deleting will send confirmation message to the administrator.
Preconditions: The administrator has already sign in.
Assumptions: The administrator has an account and permission to add, update and delete menu item.

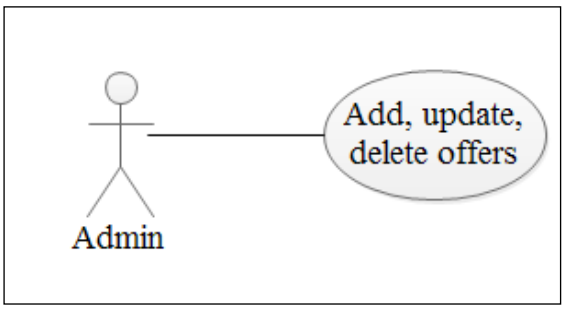


Figure 10.7: Add ,update ,delete offers

Table 14: Add ,update ,delete offers

Use case name: Add, update, delete offers. **Unique offersID**
Actor(s): The administrator
Description: Allow the administrator to add offers into restaurant database .
Steps Performed (main Path): 1- The administrator signs in. 2- The administrator clicks on offers option . 3- The administrator enters offers title and description and clicks add new button. 4- For update or delete option a list of offers will display with delete and update icons. 5- A successful adding or updating or deleting will send a confirmation message to the administrator.
Preconditions: The administrator has already sign in.
Assumption: The administrator has an account and permission to add ,update ,delete offers.

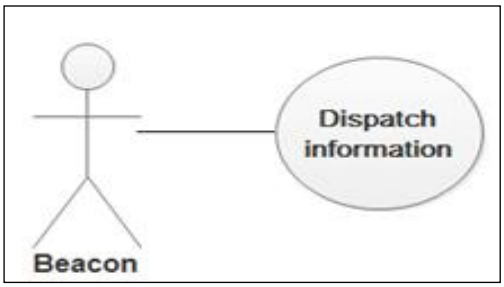


Figure 11 :Beacon use case

Table 15:Dispatch information

Use case name : Dispatch information **Unique beaconID**
Actor(s): Beacon
Description: Dispatch information by the beacon to the customer.
Steps Performed (main Path): Beacon dispatches stored information and its location to customer application.
Preconditions: 1-Customer must have restaurant application. 2-Open Bluetooth. 3-Customer has to be in beacon range.
Assumptions: Beacon has the ability to dispatch stored information and its location.

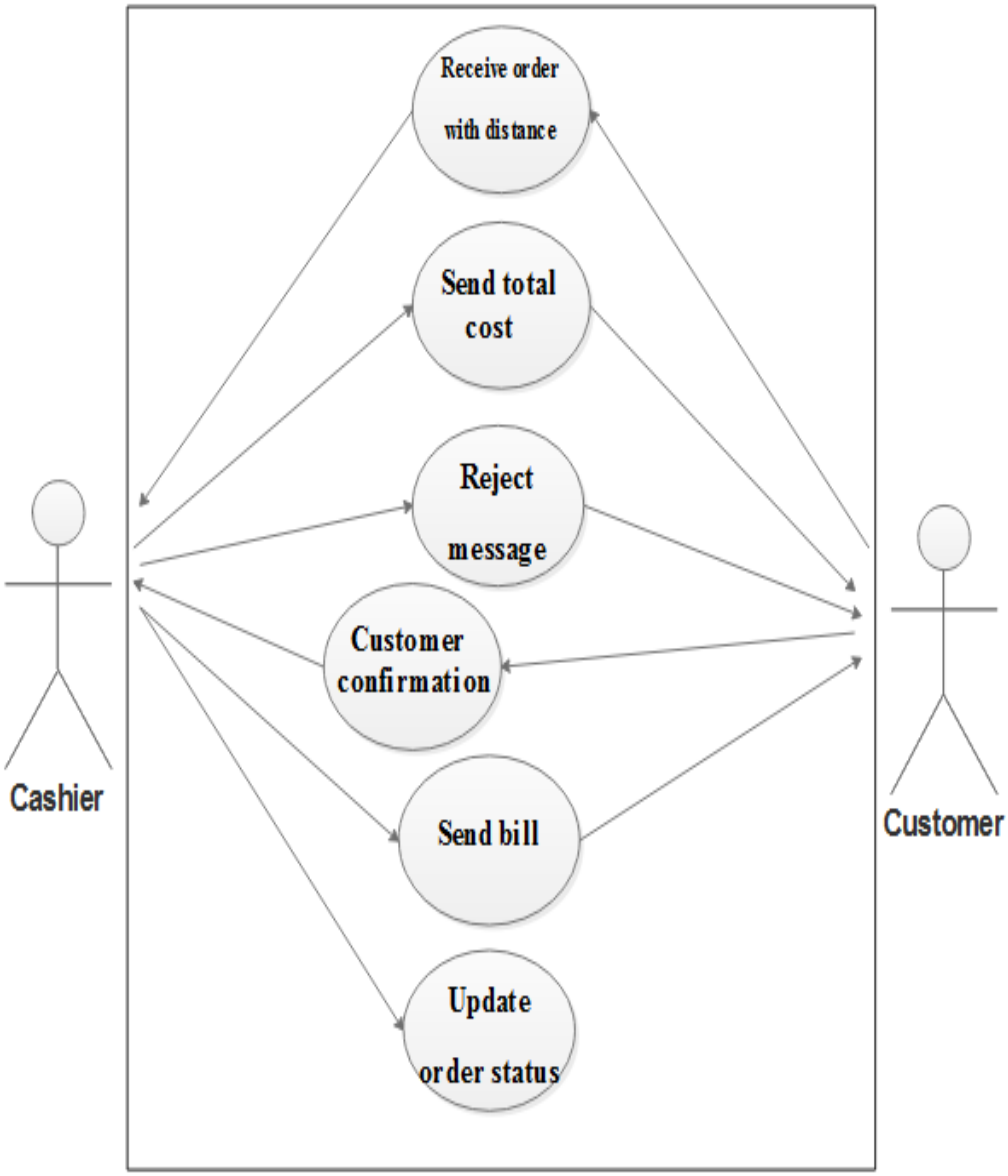


Figure 12: Cashier use case

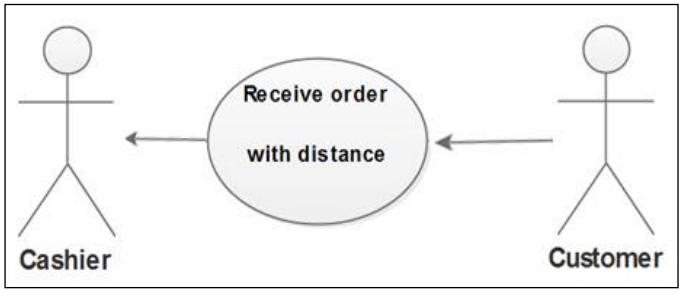


Figure 12.1: Receive order with distance

Table 16 :Receive order with distance

Use case name: Receive order with distance **Unique OrderID**
Actor(s): Cashier and customer .
Description: Enable the cashier to receive order with distance from customer application.
Steps Performed (main Path) 1. Order has done by customer. 2- Cashier receive the order with distance from customer application.
Preconditions: 1- Cashier open the home page and sign in . 2- Customer must have restaurant application. 3-Customer already makes his order.
Assumption: Cashier has ability to receive customer's order with distance.

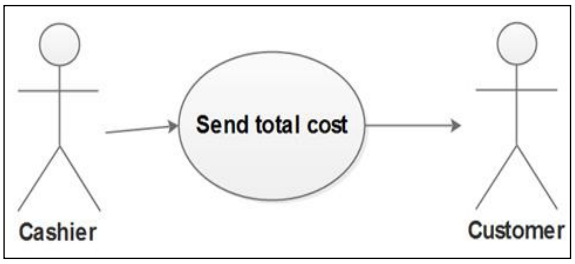


Figure 12.2: Send total cost

Table 17: Send total cost

Use case name: Send total cost **Unique orderID**
Actor(s): Cashier and customer .
Description: Enable the cashier to send the total cost to customer .
Steps Performed (main Path) 1. Order has done by customer. 2- Calculate customer location automatically. 3-Total cost is send to customer .
Preconditions: 1- Cashier should have an account and he sign in. 2- Customer must have restaurant application. 3-Customer has already made his order and he is in the restaurant range.
Assumption: Cashier has ability to send total cost to customer (automatically) .

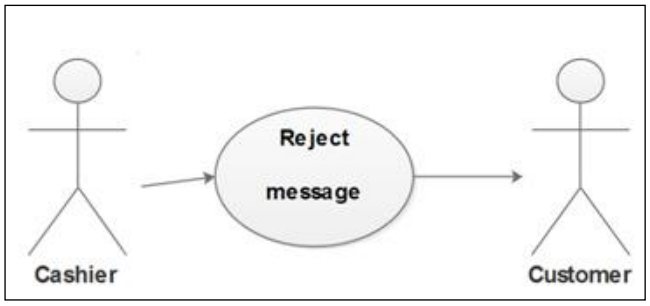


Figure 12.3: Reject message

Table 18:Reject message

Use case name: Reject message **Unique Order ID**
Actor(s): Cashier and customer .
Description: Enable the cashier to send reject message to customer .
Steps Performed (main Path) 1-Order has done by customer. 2- Customer is not in restaurant range. 3- Rejected message send to customer and his order will be cancelled automatically.
Preconditions: 1- Cashier should have an account and he sign in. 2- Customer must have restaurant application. 3-Customer has already made his order but he is out of the restaurant range.
Assumption: Cashier has ability to send reject message .

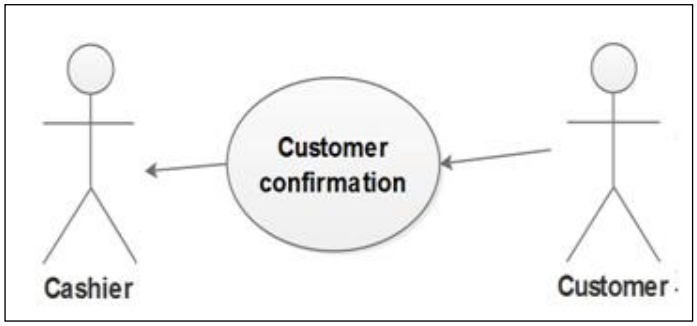


Figure 12.4: Customer confirmation

Table 19: Customer confirmation

Use case name: Customer confirmation **Unique OrderID**
Actor(s): Cashier and customer.
Description: Enable the cashier to receive customer confirmation.
Steps Performed (main Path) 1. Order has done by customer. 2-Calculate customer location automatically. 3-Total cost is received by customer. 4- Customer sends his confirmation.
Preconditions: 1- Cashier should have an account and he signs in. 2- Customer must have restaurant application. 3-Customer already makes his order and verified automatically.
Assumption: Cashier has ability to receive customer confirmation.

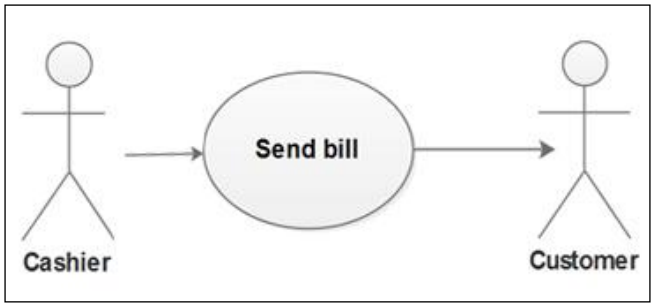


Figure 12.5: Send bill

Table 20: Send bill

Use case name: Send bill **Unique billD**
Actor(s): Cashier and customer .
Description: Enable the cashier to send bill to customer.
Steps Performed (main Path) 1. Order has done by customer. 2- Calculate customer location automatically. 3-Total cost is received by customer. 4- Customer sends his confirmation . 6- Bill will be send .
Preconditions: 1- Cashier should have an account and he signs in . 2- Customer must have restaurant application. 3-Customer has already made his order and send his confirmation.
Assumption: Cashier has ability to send bill to customer(automatically).

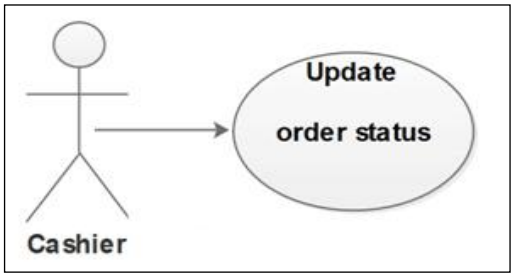


Figure 12.6: Update order status

Table 21 :Update order status

Use case name : Update order status **Unique orderID**
Actor(s): Cashier.
Description: Allow cashier to close the order status.
Steps Performed (main Path):
1- Cashier signs in. 2- Cashier can update order status by closing the order . 3- Successful change and send confirmation.
Preconditions: Cashier should have an account and he signs in.
Assumptions: Cashier has an account and permission to update order status.

2.1.4 Proposed Solutions and Alternative Solutions

Proposed Solutions:

- The project uses beacon sensor to help it to determine the customer locations and make sure that the customer is inside the restaurant range which will extensively reduce the restaurant losses, for example if the customer make the order and he is not in the restaurant range which means that the customer could not actually want the order, he just deceptive. In order to this the using of beacon to determine the customer distance with making the order will reduce the restaurant losses. Also, will reduce the congestion inside the restaurant by telling the customer if the restaurant is full or not.
- Beacon will be used in the system to encourage customer to visit the restaurant and to see the available offers.
- When the customer enters the beacon range and he has already installed the application. Any new offer will be shown as a notifications in the customer smart phone background. The customer will open the application to be able to see the restaurant status and the available table. The customer can see the menu and make his order then the cashier will receive his order with distance to process it.
  Total cost will be sent to the customer if he is in the restaurant range otherwise, rejected message will be sent and the order will be canceled .
- In this system, distance between customer and beacon will be needed. This information will be collected when the phone receive the beacon location and do some calculation to determine the customer location.

The customer need to install the application and open Bluetooth and enters the restaurant range then it will receive all restaurant information. This will encourage customers to visit and to try by themselves. It also give the restaurant more powerful control.

Alternative solution:

- The customer can make his order by the application or he can order directly from the restaurant without the application
- Offer and restaurant status can be sent from the application rather than beacon.

Chapter 3 - DESIGN CONSIDERATIONS

3.1 Design Constraints

3.1.1 Hardware environment

3.1.2 Software environment

3.2Architectural Strategies

3.2.1 Reuse of existing software components

3.2.2 Project management strategies

3.2.3 Development method

3.2.4 Future enhancements/plans

3.1 Design Constraints

3.1.1 Hardware environment:

- Android Smart phone will be used.
- 2 Personal computers hardware :

  Windows 7 platform

  Processor: Intel core i5,i7 CPU 2.40GHz

  System type: 32-bit(x86) .

  RAM: 3 GB
- Beacon **.**

3.1.2 Software environment:

Operating System:

Operating system is a special type of software, which manage the hardware and other resources. Windows platform will be used for the two personal computers in order to host the admin and the cashier application .The customer application will run on a smart phone with android operating system.

Database system :

- SQL Server will be used to build the data base and connect it with the android application.

Beacon library(altbeacon.beacon) :

This library allows Android devices to use beacons like iOS devices do. An application can request to get notifications when beacons appear or disappear. AltBeacon configured to detect a wide variety of beacons and easily configured to work with the most popular beacon types on the market. Any device with Android 4.3+ and a Bluetooth Low Energy can detect beacons with this library.

Eclipse:

Eclipse is a platform that has been designed from the ground up for building integrated web and application development tooling.

Eclipse provides a common user interface (UI) model for working with tools. It is designed to run on multiple operating systems while providing robust integration with each underlying OS. Plug-ins can program to the Eclipse portable APIs and run unchanged on any of the supported operating systems.

Other Software:

- **Edraw MAX (version 7.9)**

It a tool for drawing flowcharts such as : data flow diagram, Class diagram, Sequence diagram, use cases.. etc

- **Microsoft office PowerPoint (version 2010)**

Presentation slide will be made by this tool.

- **JustInMind (version 6.8)**

It is a tool that is used to design the prototype of the application.

- **SmartSheet website**

It is used to build the project management gantt diagram.

- **Notepad++**

It's used to write PHP code .

3.2 Architectural Strategies

3.2.1 Reuse of existing software components

Beacon programming is an open source code. The code will be used for beacon programming and to connect the application database to the beacon database. The main use of it that any changes in the application database can be dispatched to the customer by beacon.

3.2.2 Project management strategies

In this part, a good management and motivation among different task in the project will be presented

Table 22: project management

Initiating	• This phase includes discussing different ideas with the supervisor and choosing one of them. • It also includes writing a complete project proposal and delivering it on time to keep the project on.
Planning	• Studying the proposed idea and writing the literature review. • Setting up the time and all activities (schedule the project). • Writing the general system idea and compare it with the previous system. • Studying the advantage and disadvantage of all previous systems in order to build what is missing in these systems.
Analysis	• Defining different system requirements. • Drawing system models. • Writing different solutions (proposed and alterative solutions) of the system problem.
Design	• Defining the software and the hardware environment. • Searching for other existing components which can be used in the application. • Find solutions for the system problem such as: 1- Waiting in a long queue at popular restaurants, the solution will allow the customer to place his order while he is within the queue. 2- Detecting the customer's location in order to check if the customer is around the restaurant to confirm his order. 3- Showing the available table number in the restaurant. 4- Knowing the restaurant status (full or not).

Details tasks:

Appendix A

1.2.3 Development method

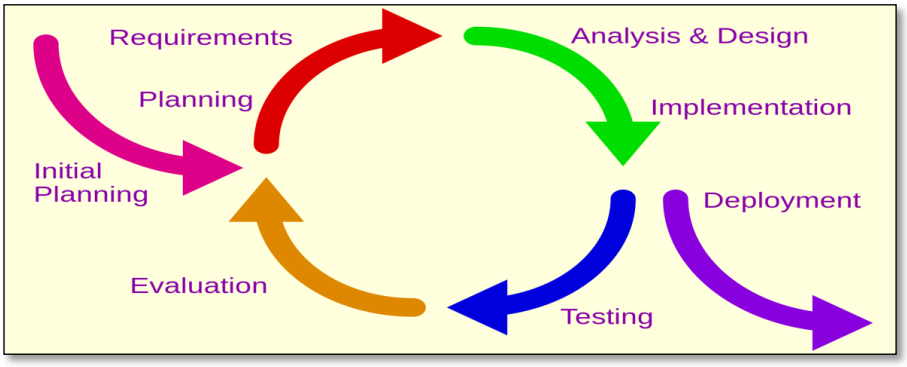


Figure 13: Incremental process

The incremental process model has been chosen for developing this project for many reasons [10]:

1) new technology is being used:

Using beacon sensor in restaurant area is rare, almost this project starting from scratch, in some steps developers phase a new questions or new information which affects other steps, so they need the ability to change and test for every step.

2) Major requirements has been defined :

Functional and non-functional requirements are defined clearly, but still there is a need to make some changes over time in some details which effect some requirements, but in general the most of them has been specified clearly although new technology will be used.

Advantages of using incremental model:

Incremental model in general provides an easy way to build the system, debug and test it, which help developers to make a good product in short time. In future, when an initial version is released, a feedback will be collected from customers and it used to make changes that meet their requirements since this model is flexible. Future enhancement can be applied as new features in each release.

Phases of incremental model:

1. **Planning:**

In this phase, way and method need to be decided then applied to have an optimal project, developer need to discuss different issues with the supervisor in order to draw clear goals.

2. **Analysis :**

In this phase, developers need to focus in details of what to do, and how it should be done, mean while keep searching for information about beacon sensor and how it can integrate with restaurant

application. Also, they need to think on how the system will work, study what was exist in this area and try to prevent repeating any mistake. Finally, in order to have a good project, they need to conduct a good analysis.

3. **Design:**

In this phase, they should try to have a good design for the application by working on the collecting requirements.

4. **Implementation.**
5. **Testing.**
6. **Evaluation. (details in chapter 5)**

3.2.4 Future enhancements/plans

In future, the application can be improved to be more efficient, applicable and provide more service to facilitate the customer life. Examples of these improvements include:

1- Using the customer information for delivering his order.
2- Improving the security protection in order to allow the customer to pay by using Visa Card.
3- Allowing the admin to offer discount based on the number of visits (how many time the customer use his account in placing his order).
4- Using the beacon in sending different information such as the history of restaurant or sending jokes to the customer in the restaurants.
5- Applying the application in many restaurants or in a restaurant with many branches.
6- Implementing an IOS version of the system.

Chapter 4- SYSTEM DESIGN

4.1 System Architecture and Program Flow

4.1.1 Major modules

4.1.2 Sub modules

4.2 Detailed System Design

4.2.1 Class diagram

4.2.2 Sequence diagram

4.2.3 Activity diagram

4.2.4 Data base

4.2.5 Interface description

4.1 System Architecture and Program Flow

4.1.1 Major modules

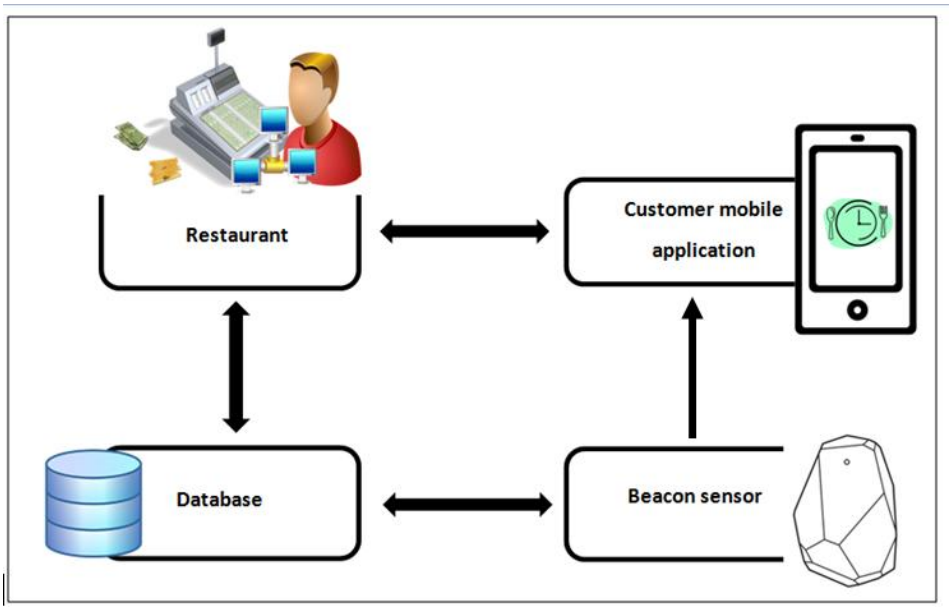


Figure 14:System components

The system consists of four components: beacon sensor, database, restaurant and customer mobile application.

Beacon sensors are used to dispatch information in the area that they exist in. It also will be used in the smart restaurant system for measuring the distance between it and the customer, then use these information in confirming the order. These information also can help the customer to see restaurant state and special offers.

Database will be used in the smart restaurant's system for storing all the information that is related to the restaurant such as menu, offers, tables, order, cashier, customer and welcome messages.

Some of these information need to be updated continuously such as restaurant state, available tables and special offers.

The customer mobile application will enable the user to interact with the system, it helps both the customers and employees to save their time and effort.

In the smart restaurant application, the customers can have the application downloaded on their own devices (tablets or smart phones), and then optionally can sign up. Data will be stored in the customer's table. After that, when the customer is around the restaurant a stored welcome message will be dispatched to the customer application by beacon. The application can also receive the offers from the beacon. The beacon also

will send its location in order to calculate the distance between it and the customer. If the customer is within the restaurant's range, the total cost will be sent, otherwise a reject message will be sent. If the customer agrees on the total cost, a confirmation will be sent back and save the information in the orders table, then the staff will start their procedure. The order Id will send it to customer .Then the bill will be sent to the customer with an ID. After the order is completed customer can take it away or finished his meal in the restaurant ,then the cashier will receive the payment and close the order.

The restaurant components is divided into two parts: the admin and the cashier. The admin can add a new admin or cashier to the system and save his information in the admin and the cashier table respectively. Also, the admin has the authority to update table number. The same process can be done on other data elements such as menu dishes and offers. The cashier part of the application will receives the customer's order and calculates the total cost(automatically).When customer within the restaurant range or a reject message if the customer outside the restaurant's range. When the customer agrees on the cost, the actual bill with detailed information will be sent back to the customer. When the customer pays for the order, the cashier will close the order, and the related table will be available.

4.1.2 Sub modules

The system has many sub modules that work together to perform the required functionality.

1- Special offers:

This service will be used to inform the customer about the latest offers provided by the restaurant.

When the customer is around the restaurant, the beacon will dispatch the offers information from offers table to the customer's application. The admin will be responsible for updating the offers information in the offers table.

2- Restaurant status:

This service will allow the customer to know if the restaurant is full or not and also help it to provide a good service to their customers.

Restaurant status had been taken from table_count table in the database and dispatched to the customer's device by the beacon.

3-Available tables:

Instead of checking around the restaurant for an available table, the customer can check by the application.

The number of available tables are calculated using the stored tables number in table_count table . The customer will be informed through the application about the available table numbers .

4-Sign up for customer:

It will be optionally in the customer's application to sign up and save his data in the customer's table or to make an order without asking for personal information.

First, the customer can sign up in the application and the information registered will be stored in the customer's table, so next time, the customer can just sign in to the system by entering the username and password. the system will check it automatically in order to verify it.

5- Admin sign in:

When a new admin joins the administration, the main admin will add information to the admin's table. So later, he can sign in and access the system. Every time he accesses the system, he must enter the data and the system will check it automatically for validation, then the admin page will be displayed.

6- Cashier sign in :

When the cashier can accesses the system by entering the email and password, and the system will check automatically, if the information is correct then the cashier home page will be displayed.

4.2 Detailed System Design

4.2.1 Class diagram

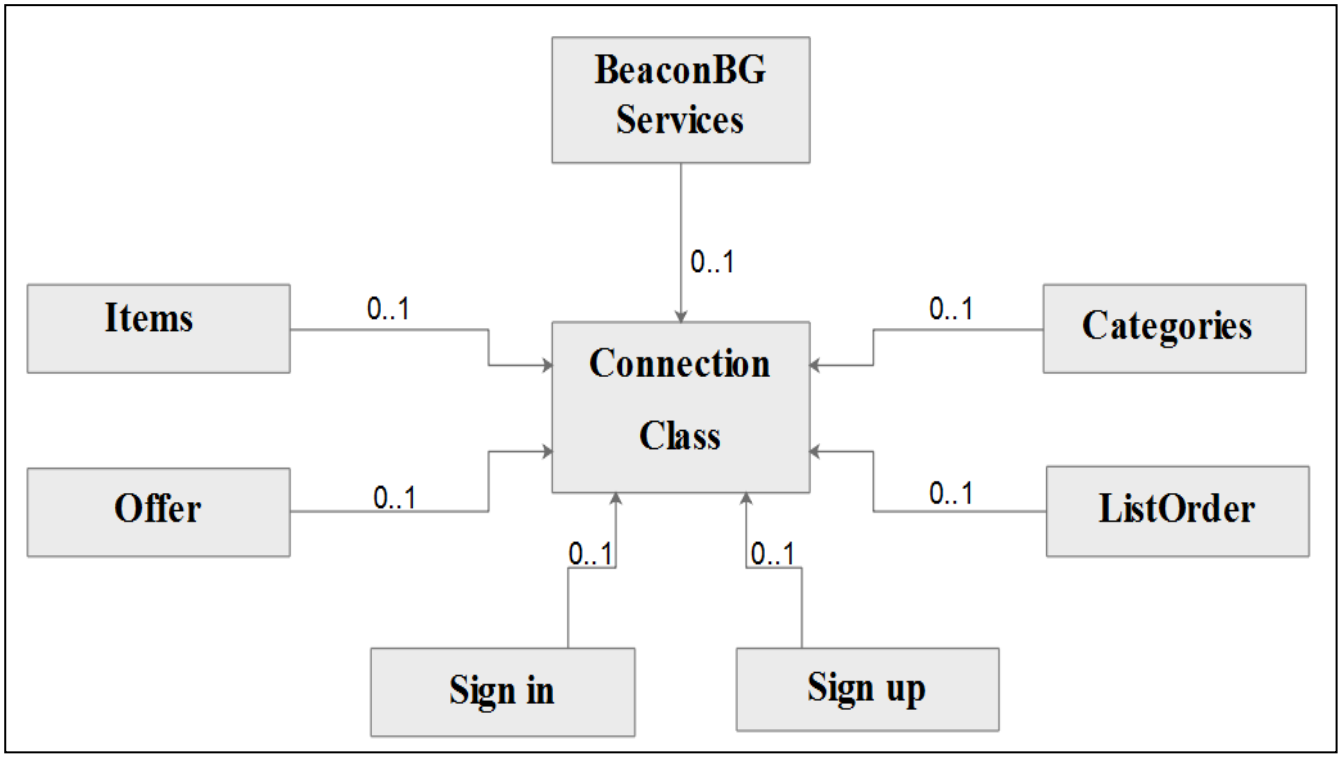


Figure 15: Simple Class diagram

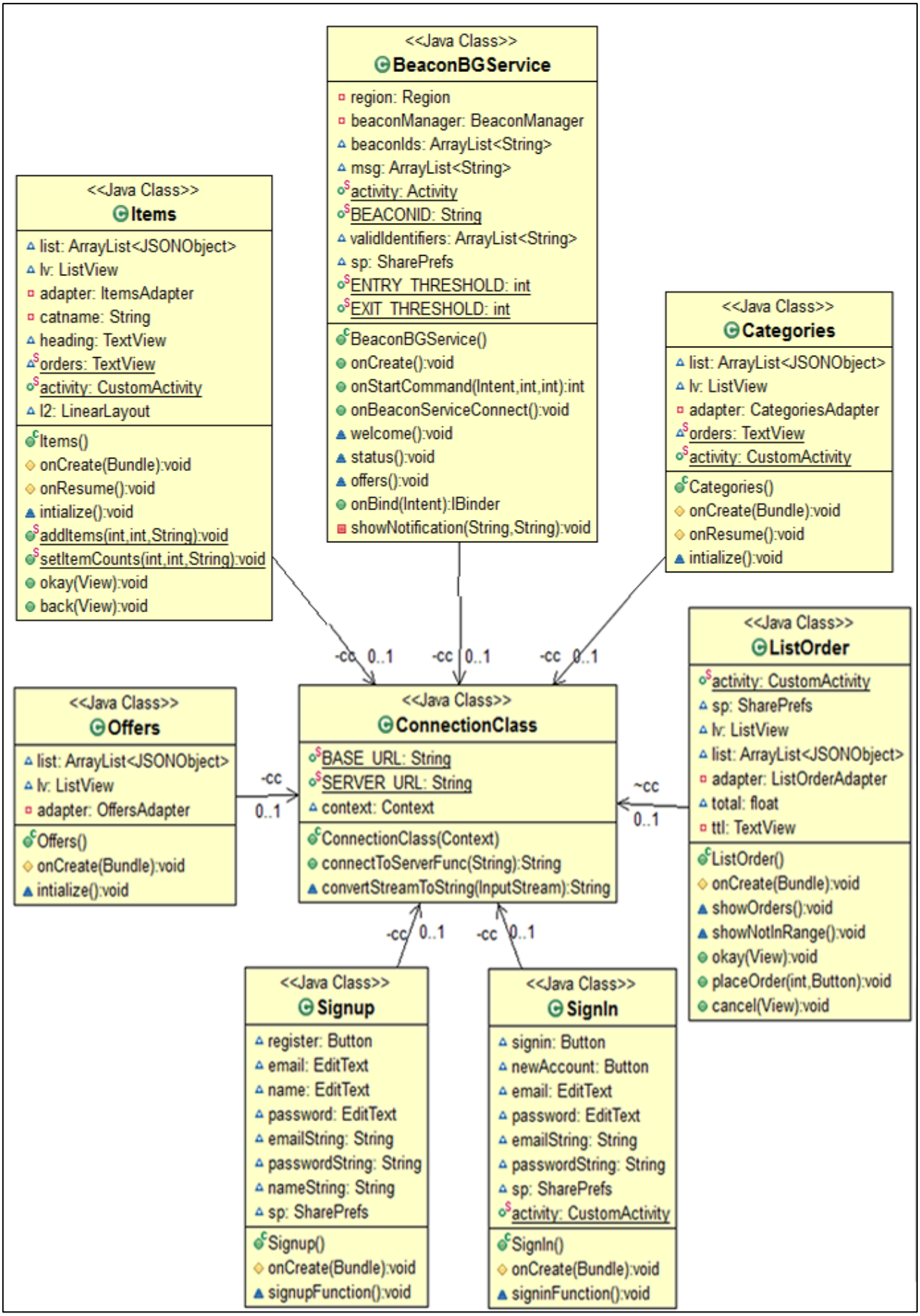


Figure 15.1: Class diagram details

4.2.2 Sequence diagram

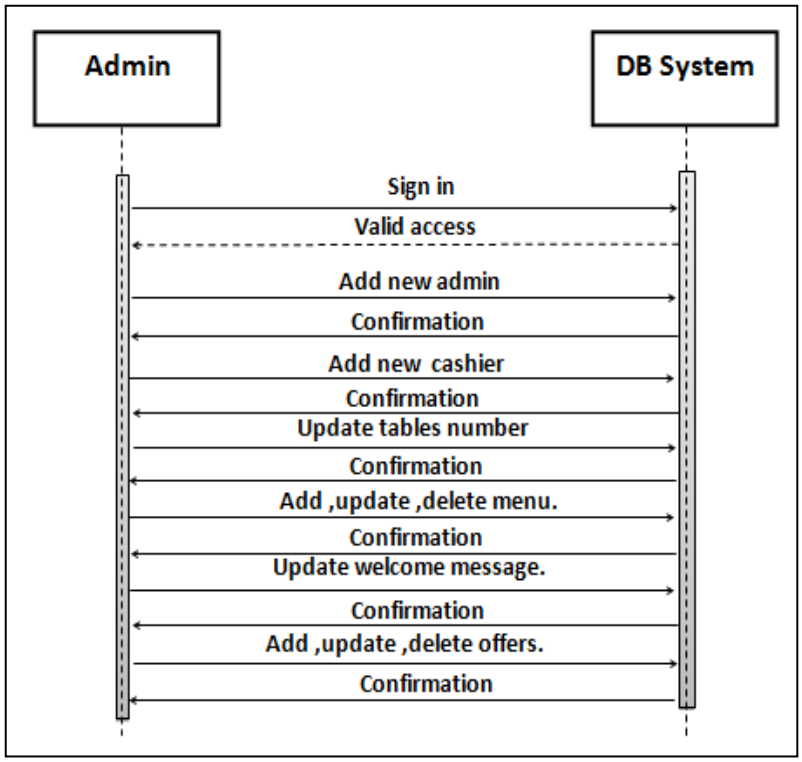


Figure 16: Admin sequence diagram

Table 23 : Admin Description

Name	Operation	Description
Admin	Sign in()	The admin will sign in to the system using the account created.
DB system	Valid access ()	The system will check the validity of the admin sign in information.
DB system	Admin homepage ()	After successful login, the homepage of the admin will be displayed.
Admin	Add new admin ()	The admin can add a new admin to the system and give authorization to control some of the system tasks.

DB system	Confirmation ()	After adding a new admin to the system, a confirmation will be sent if the process is successful.
Admin	Add new cashier ()	The main admin can add a new cashier to the system and give some authorization to control some system tasks.
DB system	Confirmation ()	After adding a new cashier to the system, a confirmation will be sent if the process is successful.
Admin	Update table number ()	The main admin has the responsibility to add number of tables of the restaurant.
DB system	Confirmation ()	After adding and updating or even deleting a table from the system a confirmation will be sent if the process is successful.
Admin	Add, update, delete menu ()	The main admin have a authority to add and update or even delete the menu and also the menu items in the system.
DB system	Confirmation ()	After adding and updating or even deleting a menu from the system the confirmation will send if the process is successful.
Admin	Add, update, delete offers ()	The main admin has the authority to add and update or even delete the offers that are being sent to the customer's application via beacon.

DB system	Confirmation ()	After adding and updating or even deleting an offer from the system, a confirmation will be sent if the process is successful.
Admin	Update welcome message ()	The main admin has the authority to update message which will be sent by the beacon to the customer's application.
DB system	Confirmation ()	After adding and updating or even deleting a welcome message from the system, a confirmation will be sent if the process is successful.

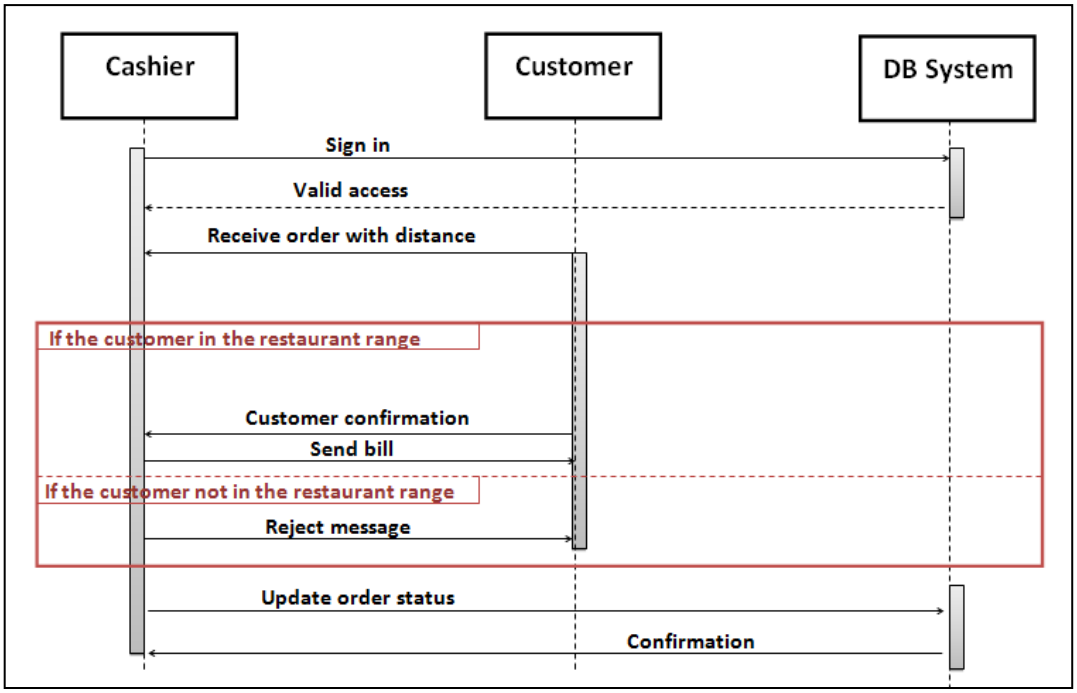


Figure 17: Cashier sequence diagram

Table 24: Cashier Description

Name	Operation	Description
Cashier	Sign in ()	The cashier will sign in to the system with an account created earlier.
DB System	Valid access ()	The system will check the validity of the cashier's sign in information.
Customer	Receive order with distance ()	The customer sends an order and the cashier will receive it with the distance.
If the customer location in restaurant range:		
Customer	Customer confirmation ()	The customer sends a confirmation to the cashier .
Cashier	Send bill ()	The cashier side will then send the bill.
If the customer location not in restaurant range:		
Cashier	Reject Message ()	The system will send a rejected message to customer if the distance shows that the customer's location is not in range.
Cashier	Update order status ()	The cashier will update the order status according to customer's confirmation.
DB System	Confirmation ()	After updating the order status in the system, a confirmation will be sent if the process is successful.

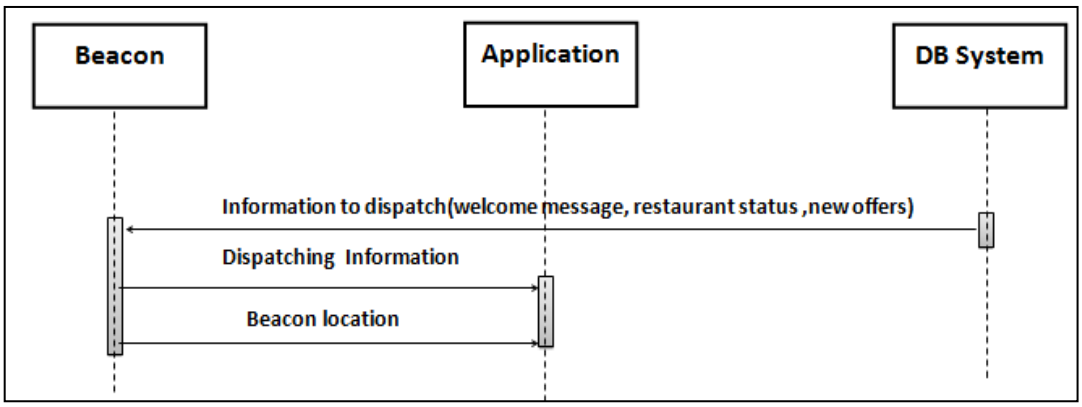


Figure 18: Beacon sequence diagram

Table 25 :Beacon description

Name	Operation	Description
DB System	Information to dispatch(welcome message, restaurant status, new offers) ()	The DB System will send the information to the beacon data table such as welcome messages, restaurant status and new offers.
Beacon	Dispatching information ()	The beacon will dispatch this information with its ID and location to the customer's applications.
Beacon	Beacon location ()	The beacon will send its ID and location to the customer's application, which will do some calculations to measure the distance between the customer and the beacon to determine the actual customer's location.

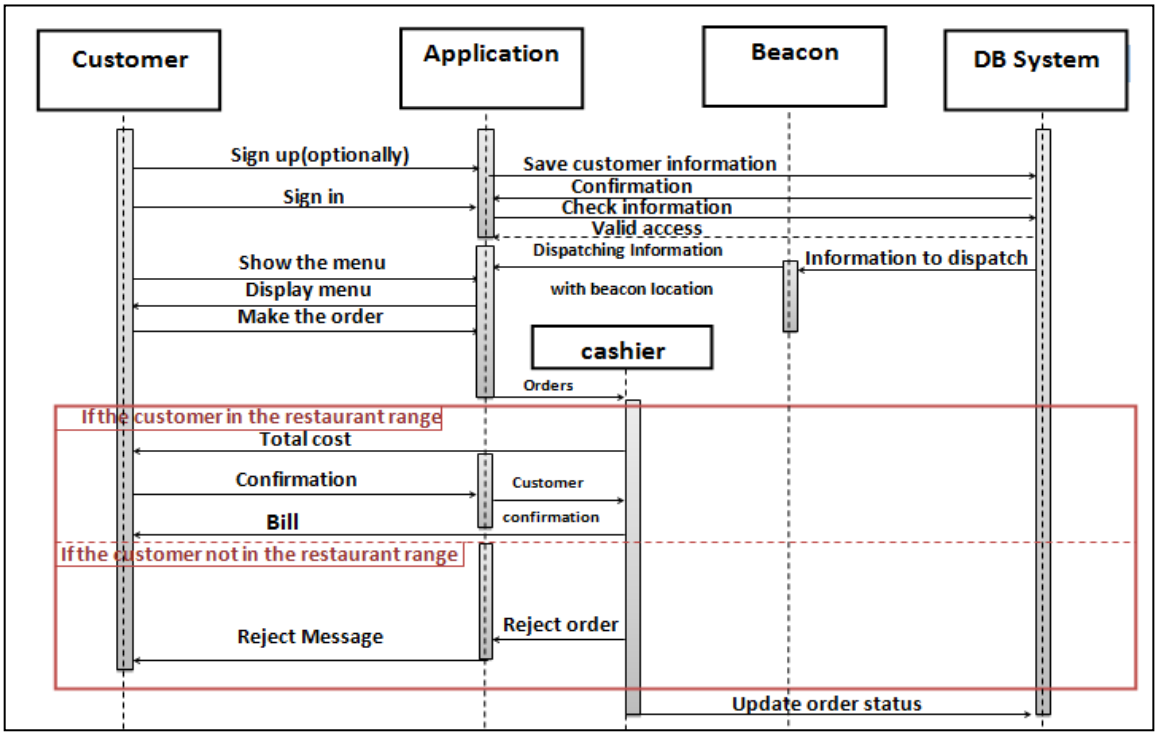


Figure 19: Customer sequence diagram

Table 25 :Customer description

Name	Operation	Description
Customer	Sign up (optionally)()	The customer can sign up and enter his information.
Application	Save customer information ()	The information entered will be saved in the DB.
DB System	Confirmation ()	After the customer signs up in the system, a confirmation will be sent if the process is successful.
Customer	Sign in ()	The customer can sign in to the application with the

		account created.
Application	Check information ()	The system will check the validity of the customer login information.
DB System	Valid access ()	The customer will get the authority to access the application.
DB System	Information to dispatch ()	Information transmits from system database to beacon database.
Beacon	Dispatching Information with beacon location ()	Information with beacon location will be dispatched by the beacon to the customer's application. Customer's application will do some calculations to measure the distance between the customer and the beacon to determine the actual customer's location, after that, the customer application will be able to check the customer's location if it's within the restaurant range or not.
Customer	Show the menu ()	The customer can choose to see the menu page.
Application	Display menu ()	The application shows the menu items to the customer.
Customer	Make the order ()	After choosing the order items, the customer can send an order to the system.
Application	Orders	The cashier will receive the customer's order details from the application .
If the location is in the restaurant range:		
Cashier	Send total cost ()	The cashier's side will calculate the total cost for all

		the customer's order items and sent it to the customer.
Customer	Confirmation ()	The customer has to decide to complete or cancel the confirmation and send it to the cashier's application.
Application	Customer confirmation ()	The application asks for the customer's confirmation to generate the bill.
Cashier	Bill	The customer will receive the bill from the cashier .
If the location is not in the restaurant range:		
Cashier	Reject order ()	If the customer is not around, the system will reject the order automatically.
Customer	Reject message()	The customer will receive a rejected message "not being around".
Cashier	Update order status()	The cashier will update the order status and saves it in the DB system.

4.2.3 Activity diagram

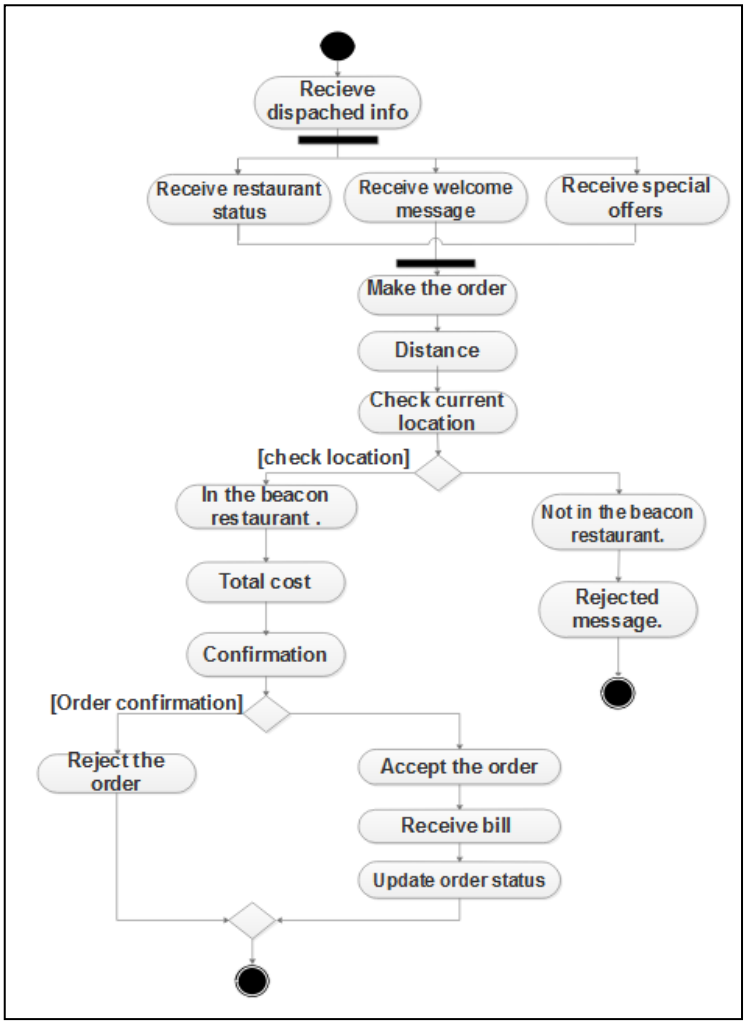


Figure 20: Activity diagram

This diagram showing a brief description about the most important operations in the system.

- First, the customer will receive the dispatched information when he enters the restaurant range :
    - Receive welcome message.
    - Receive new special offers.
    - Receive current restaurant status.
- The customer make the order by choosing items from the menu by clicking "+" symbol .
- The system will check the customer location using the distance between the beacon and the customer to make sure that the customer in the restaurant range.
    - <u>If the customer in the restaurant range that means customer is around</u>
        - The total cost will be sent to the customer.
        - The customer confirms the order (Accept it or Reject it).
        - If the order accepted, the customer will receive the bill from the cashier side which contain the detail information about his order.

- Then the cashier will update the order status and save the order information.

o If the customer is not in the restaurant range that means the customer is not around.

o A rejected message will be received .

4.2.4 Database

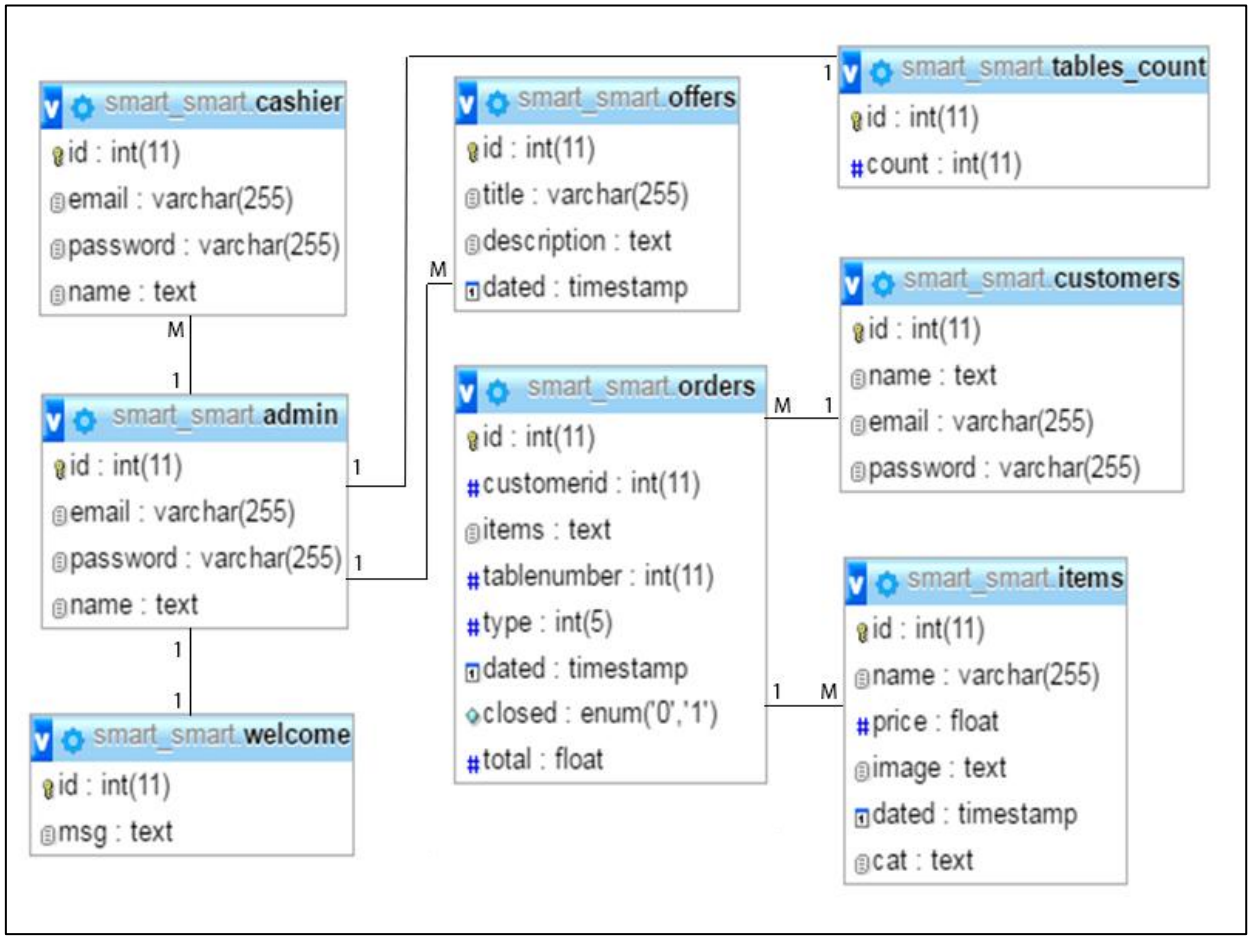
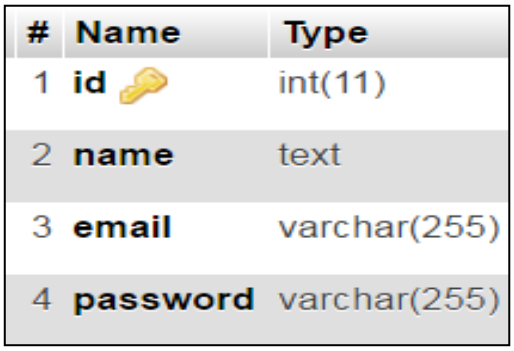


Figure 21: Database Relationship

Customer:



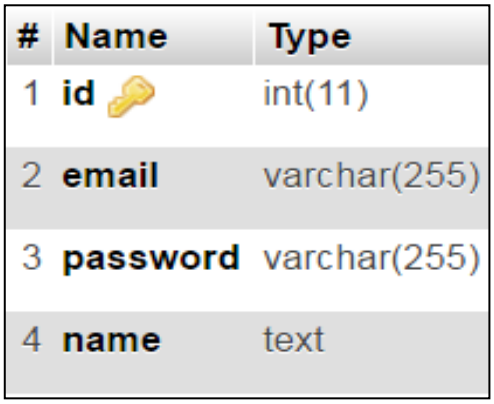
#	Name	Type
1	id	int(11)
2	name	text
3	email	varchar(255)
4	password	varchar(255)

Figure 22: Customer Table

Table 27 : Customer Table details

Name	Type	Key	Description
Customer_Id	Int	Primary key	The id of the customer.
Customer_name	text	-	The name of the customer.
Password	Varchar	-	The password to enable the customer to access the account created earlier in the application.
Email	Varchar	-	The customer's email address.

Admin:



#	Name	Type
1	id	int(11)
2	email	varchar(255)
3	password	varchar(255)
4	name	text

Figure 23:Admin Table

Table 28 : Admin Table details

Name	Type	Key	Description
Admin_Id	Int	Primary key	The Id of the admin.
name	text	-	The name of the admin.
Password	Varchar	-	The password to access the System.
Email	Varchar	-	The email of the admin that used to access the database.

Cashier :

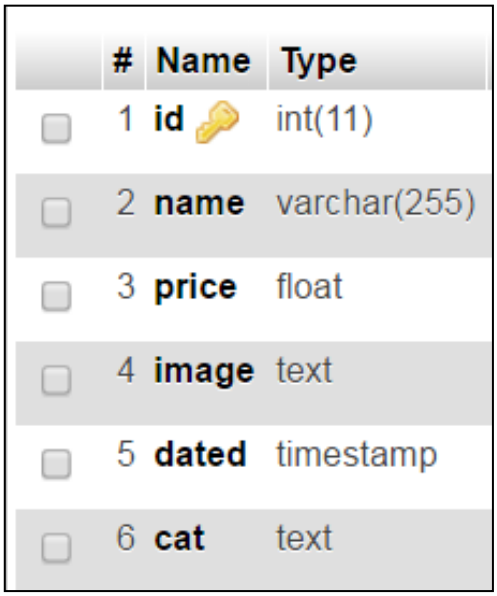
#	Name	Type
1	id	int(11)
2	email	varchar(255)
3	password	varchar(255)
4	name	text

Figure 24:Cashier Table

Table29 : Cashier Table details

Name	Type	Key	Description
Cashier_Id	int	Primary key	ID of the cashier in the database.
Cashier_name	text	-	Name of the cashier.
Password	Varchar	-	The password is used to access the system.
email	Varchar	-	The email of the cashier used to access the system.

Items:



	#	Name	Type
☐	1	id	int(11)
☐	2	name	varchar(255)
☐	3	price	float
☐	4	image	text
☐	5	dated	timestamp
☐	6	cat	text

Figure 25:Menu items Table

Table 30 : Items Table details

Name	Type	Key	Description
Item_Id	Int	Primary key	The Id of the item.
Item_Name	Varchar	-	The name of the item.
price	Float	-	The cost of the item.
Image	Text	-	Image of the item .
dated	Timestamp	-	Date of the day when item added.
Cat	Text	-	Category name

Offer:

#	Name	Type
1	id	int(11)
2	title	varchar(255)
3	description	text
4	dated	timestamp

Figure 26:Offer Table

Table 31: Offer Table details

Name	Type	Key	Description
Id	Int	Primary key	Id of the offer.
Title	Varchar	-	The title of the offer.
Description	Text	-	The description (content) of the offer.
Date	Timestamp	-	Date when offer added .

Orders:

#	Name	Type
1	id	int(11)
2	customerid	int(11)
3	items	text
4	tablenumber	int(11)
5	type	int(5)
6	dated	timestamp
7	closed	enum('0', '1')
8	total	float

Figure 27:Orders Table

Table 32 :Order Table details

Name	Type	Key	Description
Order_Id	Int	Primary key	The id of the order.
Customer_Id	Int	-	The id of the customer who request the order.
items	Text	-	The items of order .
tablenumber	Int	-	The table number in the restaurant.
Type	int	-	The type of the order : 1=Take-away,2=Sit-In
Dated	Timestamp	-.	The date of order .
Closed	Enum(0,1)	-	completed order =1 uncompleted order=0
Total	Float	-	Total cost of the order.

Table_count:

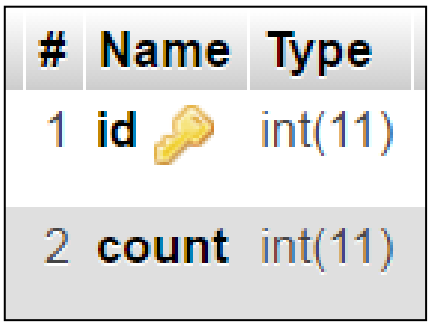


Figure 28:Table_count Table

Table 33: Table _count details

Name	Type	Key	Description
Id	Int	Primary key.	The id of table
Count	Int	-	The number s of tables .

Welcome:

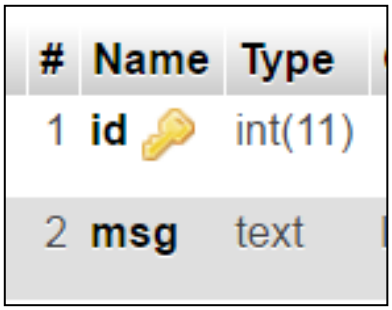


Figure 29:Welcome message Table

Table 34: Welcome Table details

Name	Type	Key	Description
id	Int	Primary key.	The id of the message.
Msg	Text	-	The description (content) of the message.

4.2.5 Interface description

Application interfaces (prototype):

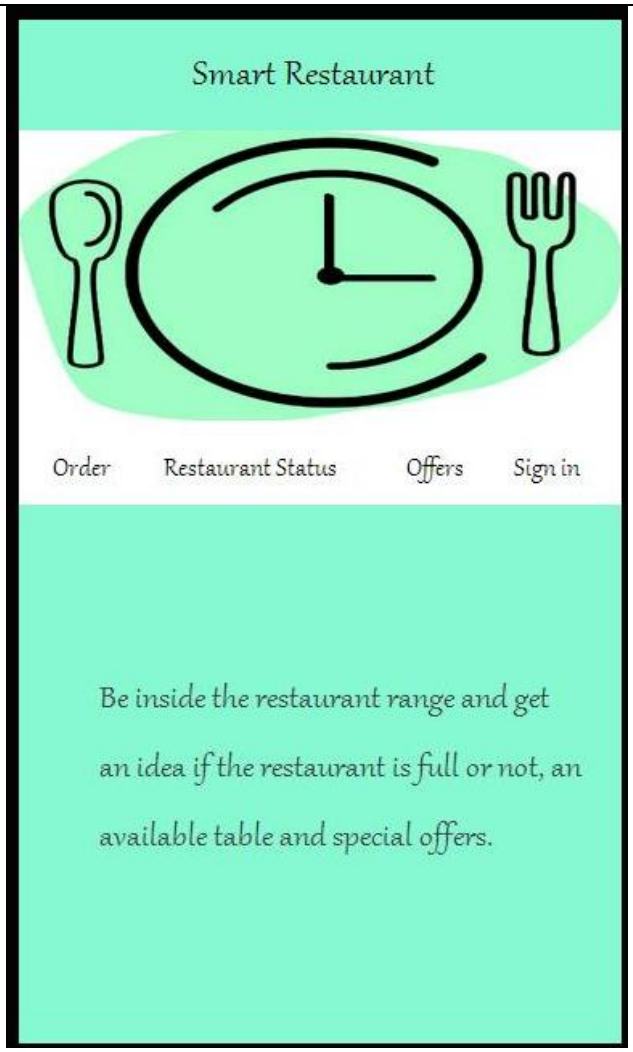


Figure 30 :Smart Restaurant main interface

Figure 31 :Sign in interface

The main page will display a small description about the Smart Restaurant application and the main four options :**Order**: to display the menu to make the order.

Restaurant status: to check if the restaurant is full or not.

Offers: to know about special offers.

Sign in: to register the customer information (optionally).

When the customer choose sign in option :

Sign in page will be displayed. Fill the two label with an email and password, then click on the sign in button.

If the customer did not have an account, the customer can choose the sign up option to register the information.

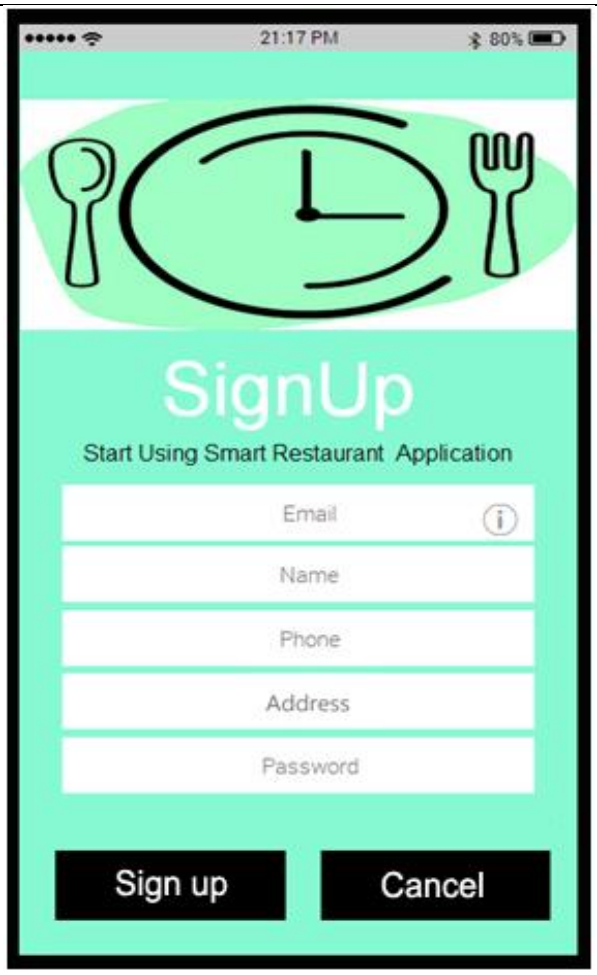


Figure 32 :Sign up interface

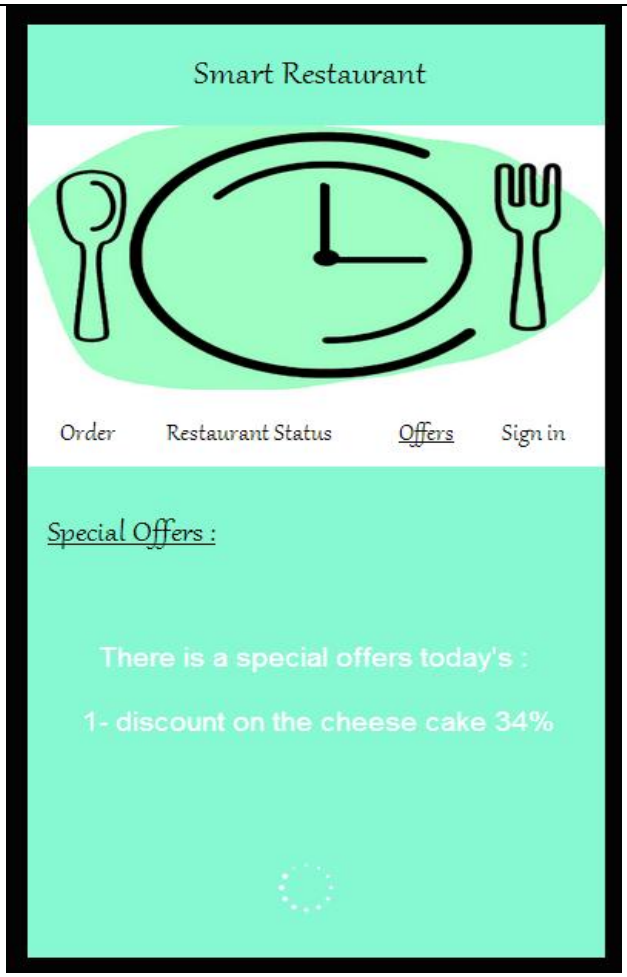


Figure 33 :Special offers interface

If the customer has no account, sign up option is displayed.

The customer has to fill the labels with email, name, phone, address and password then click on sign up button to register the information.

Signing up is optional, it will be helpful in future plans for delivery services.

When customer choose offers option :

List of special offers will be displayed

(discount ,free meal ..etc).

This offer will be updated continuously.

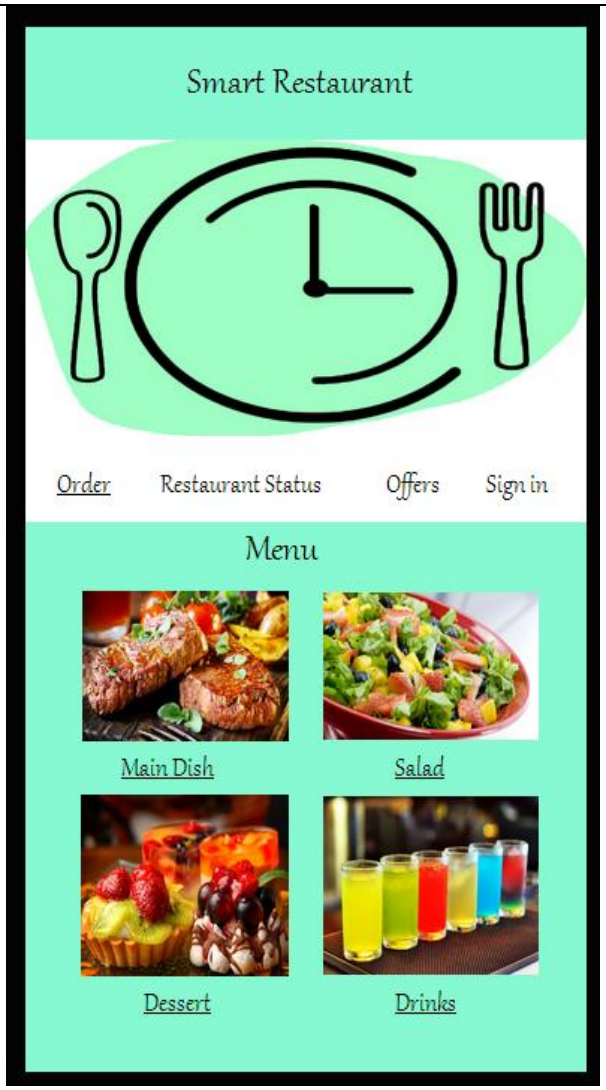


Figure 34 :Categories interface

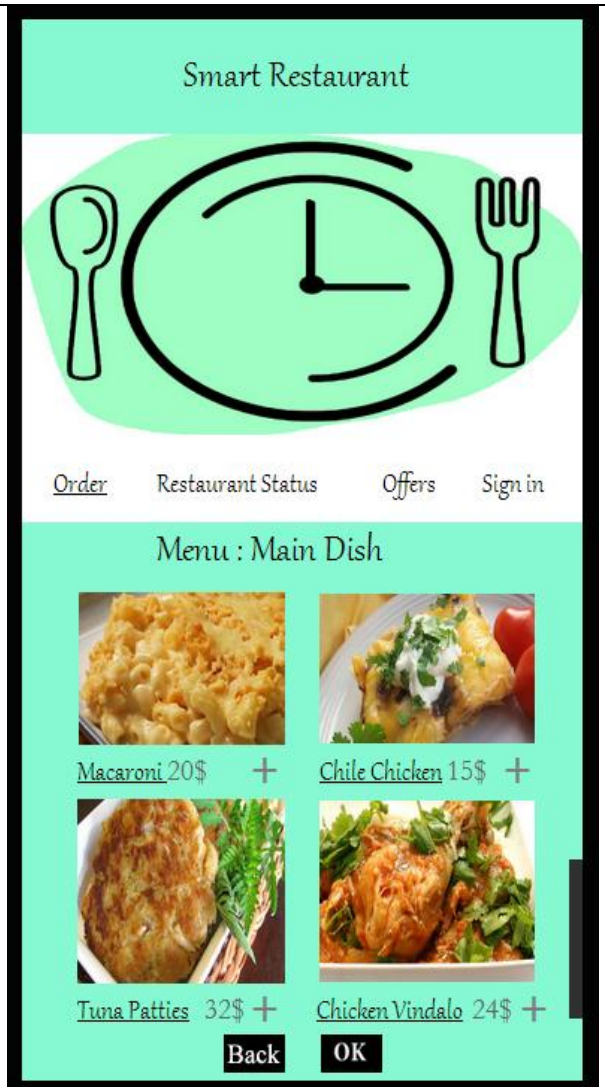


Figure 35 :Items interface

When the customer chooses order option:

List of menu categories will be displayed, the customer can choose any category to display the dishes with its prices.

When the customer chooses the main dish category, its dishes will be displayed with their own price.

To make the order, the customer can click on the add symbol (+) that falls under each dish then click ok, also he/she can go back to the main category by clicking the “back” button to choose another dishes.

In the same way the customer also can choose :

Salad, dessert and drinks from the main category.

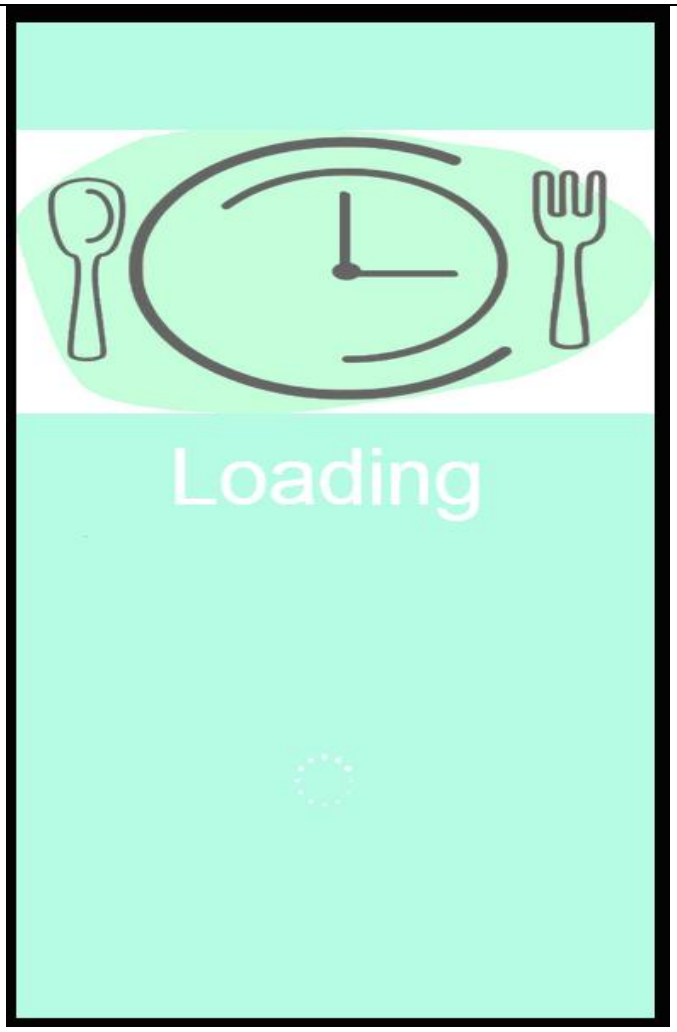


Figure 36 :Checking interface

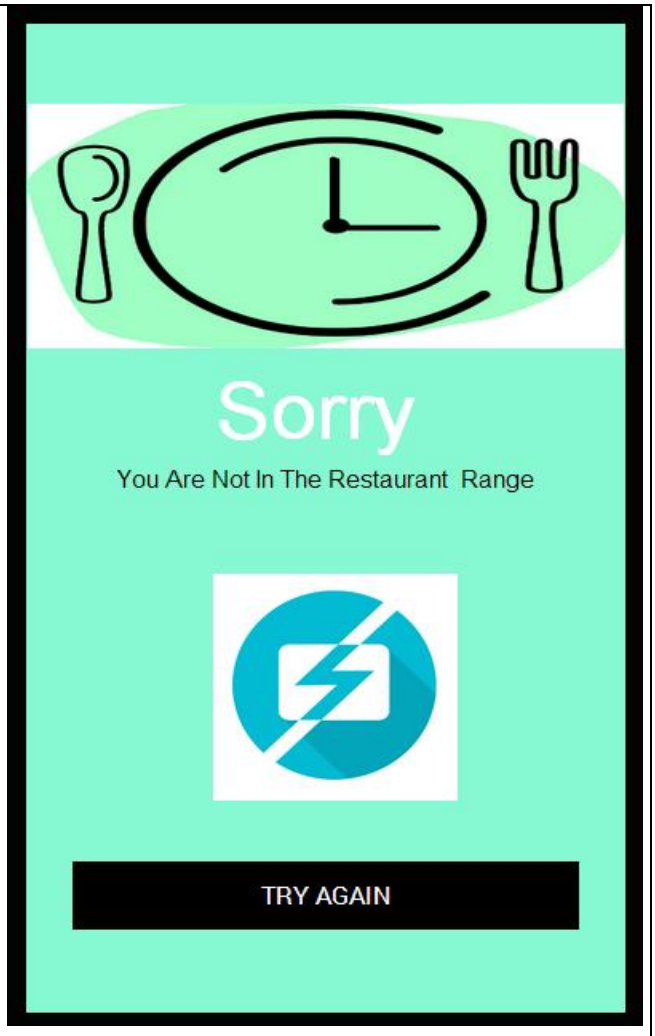


Figure 37 :Reject message interface

The system will check if the customer is around the restaurant or not (using the beacon to calculate the distance of the customer).	After checking the customer distance, this error page will be displayed if not in the restaurant's range.

Figure 38 :order with total interface

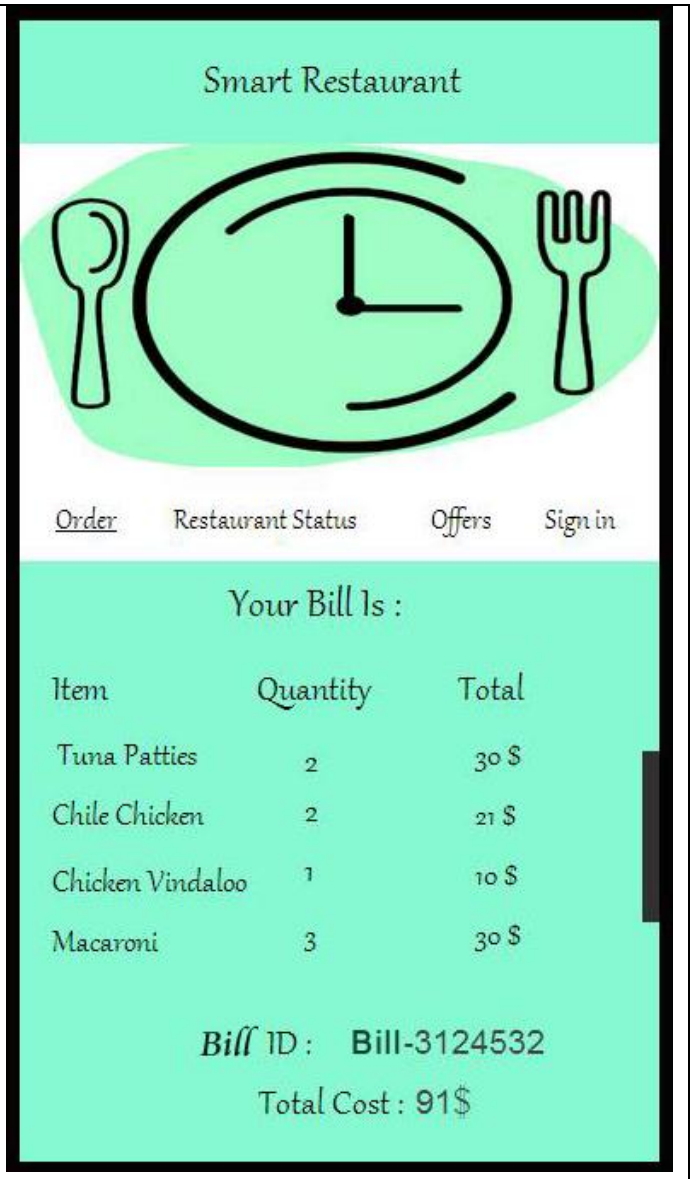


Figure 39 :bill interface

If the customer is in the restaurant's range, the whole order will be displayed with total cost.

Then, the customer can confirm the order or cancel it (customer can't edit the order after confirmation).

After the customer accepts the order, the bill will be generated with the bill id and the total cost, then it will be sent to the customer.

Figure 40 :Restaurant status interface1

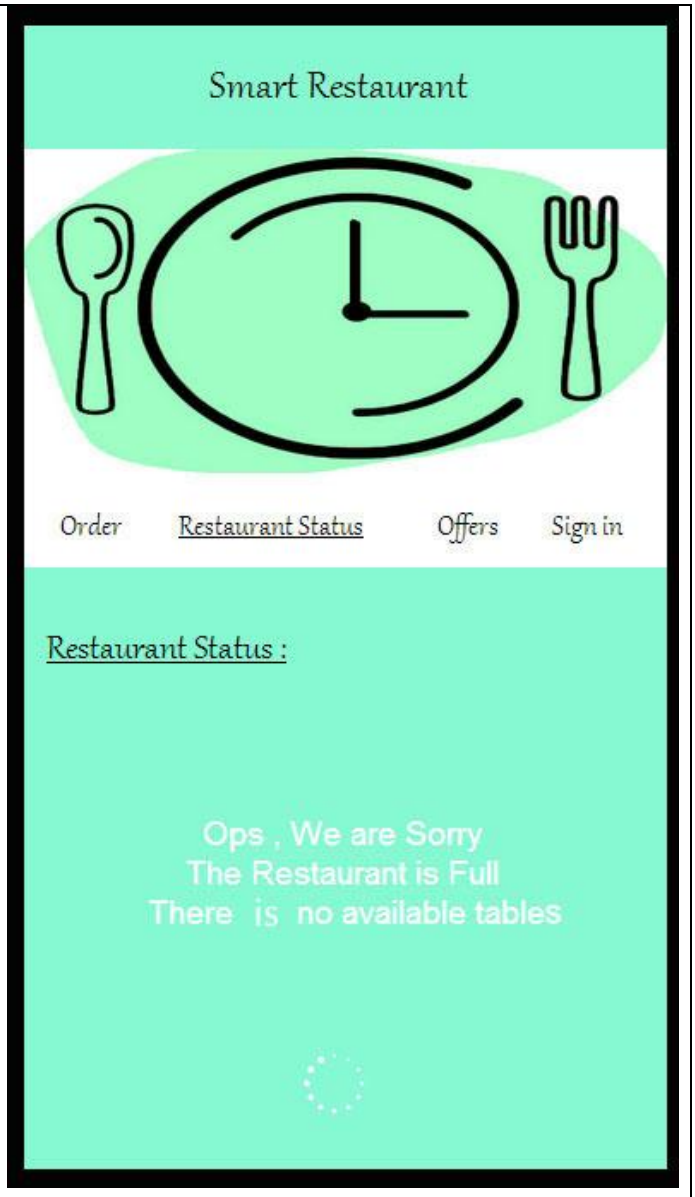


Figure 41 :Restaurant status interface2

When the customer clicks on the restaurant status option, a message will be displayed to tell the customer if the restaurant is full or not.

This page will be displayed when the restaurant has available tables, it also mentions the number of these tables.

This page will be displayed when the restaurant is full and there is no available tables.

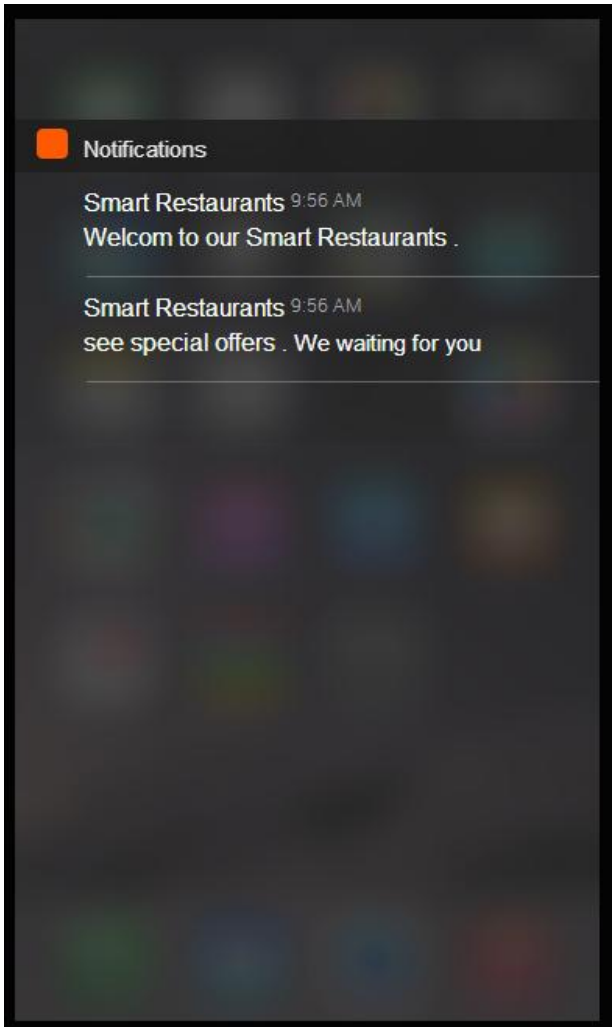


Figure 42 :Notification interface

This interface represent the notification of the smart restaurant when the customer receives the offers and welcome message .

Admin interfaces:

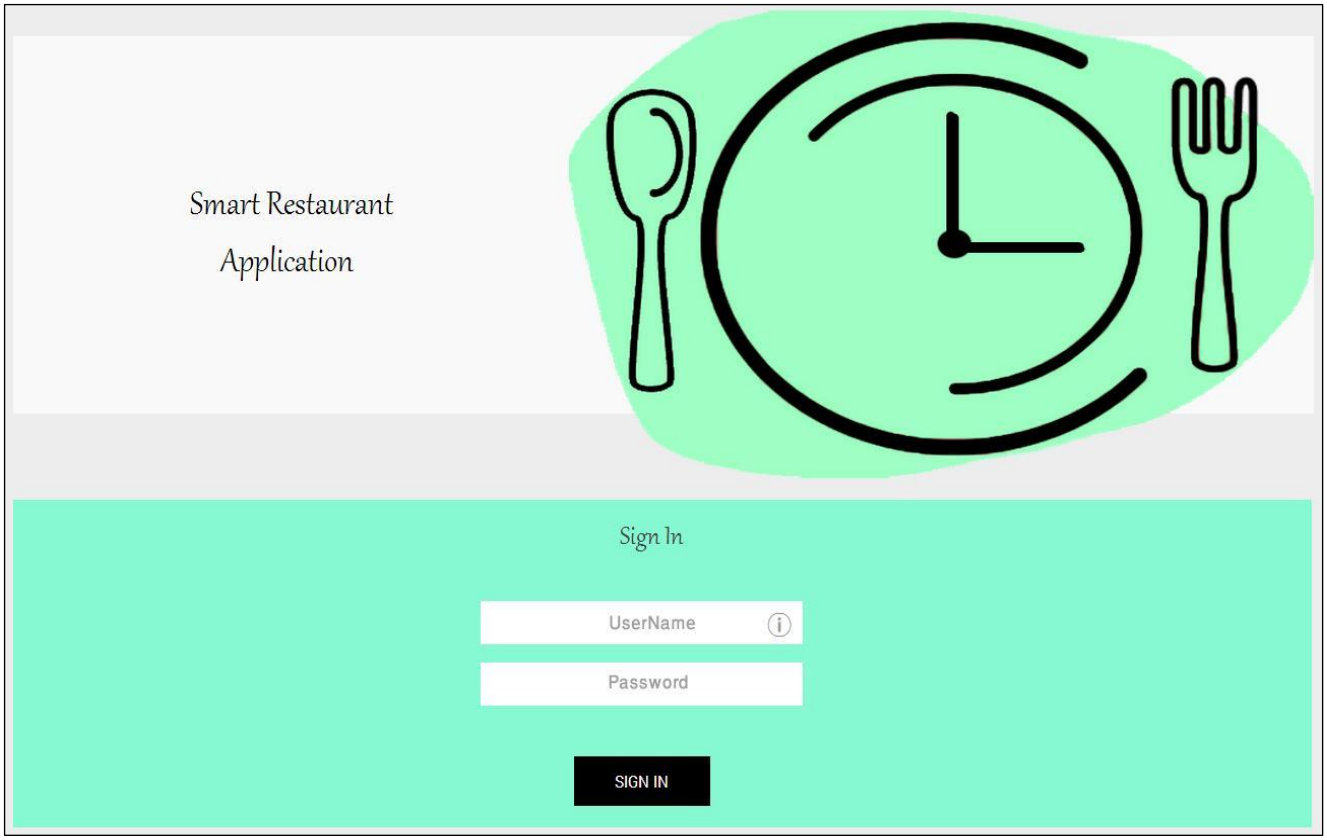


Figure 43:Admin sign in screen

This screen will be displayed to the admin:

Admin can sign in with the username and password to access the database and modify it.

Figure 44:Admin page

After signing in, the admin can access the database to modify it:

- Offers: adds new offer, updates or deletes from the database.
- Welcome message: adds, updates or deletes the welcome messages from the database.
- Menu: adds, updates or deletes the list of the menu items.
- Table: adds, updates and deletes table number and the chair number.
- Adds, updates and deletes admin.
- Adds, updates and deletes cashier.

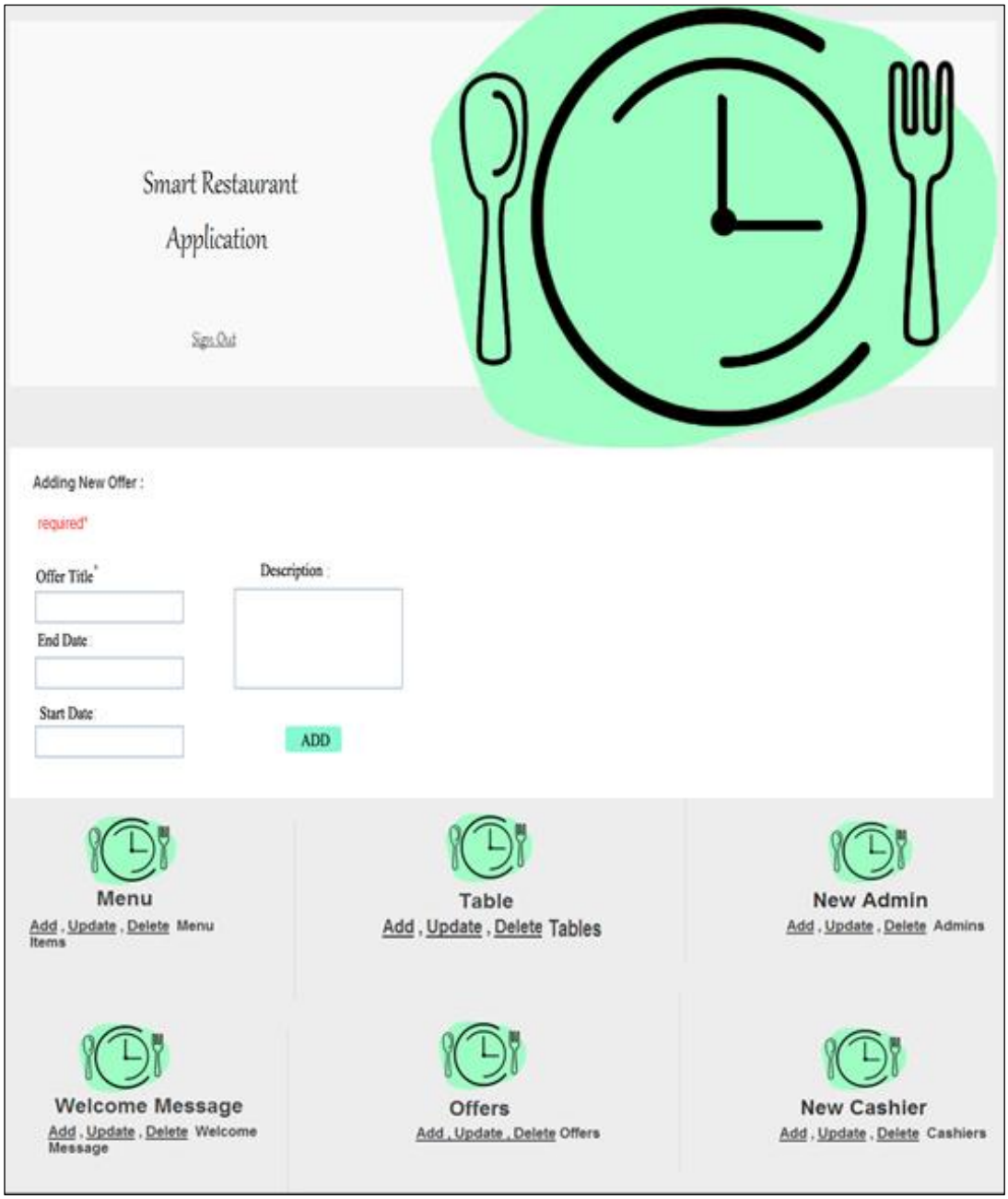


Figure 45:Add offer for admin page

If the admin chooses to add a new offers, this screen will be displayed.

Admin can add offer title, start date, end date for this offer and a small description of it, then he will click "ADD" button to record the information in the database.

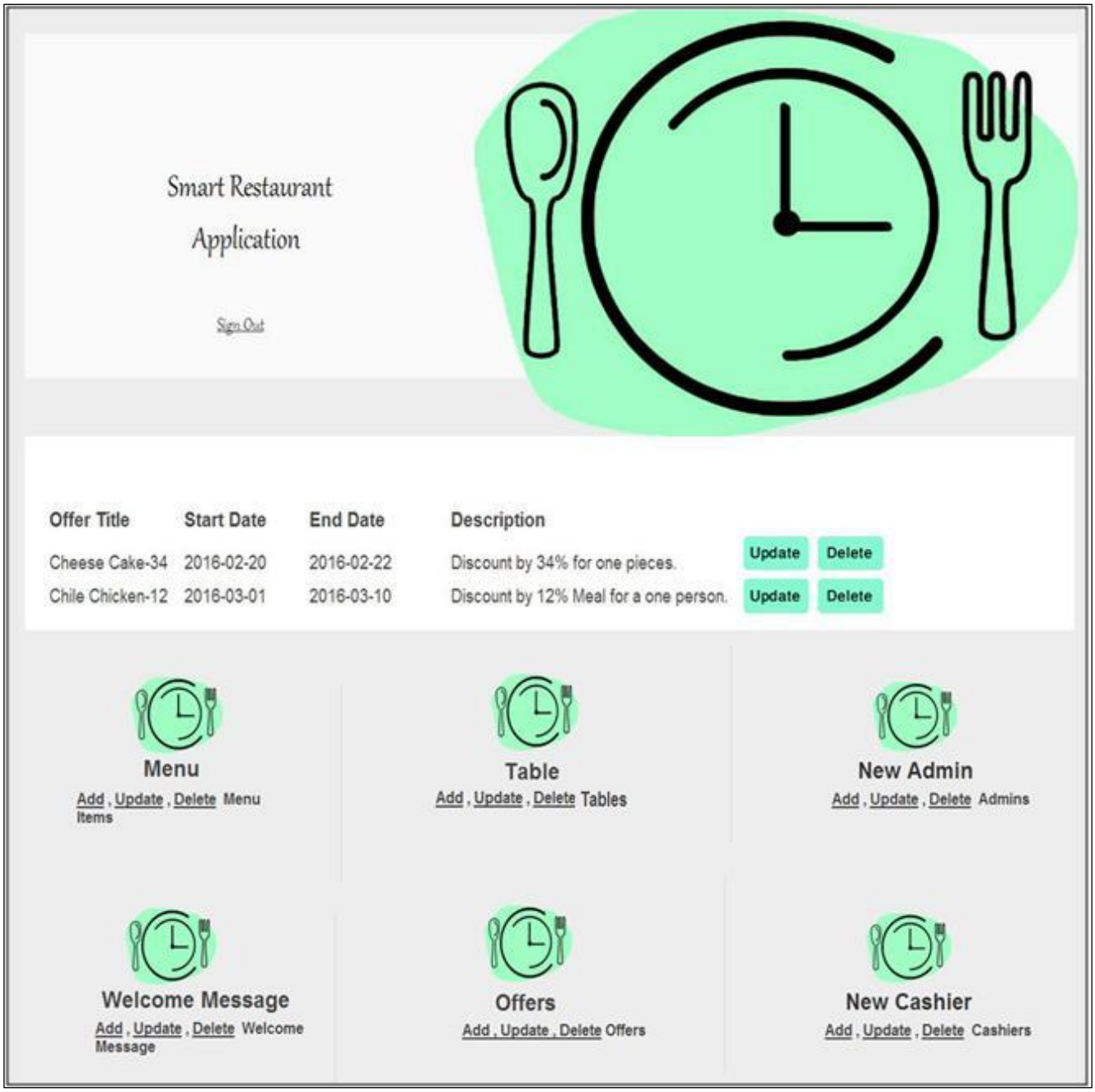


Figure 46:Update and delete offer for admin page

The offers information will displayed with UPDATE and DELETE button for each offer.

The admin can update or delete offers by clicking on these button.

Cashier interfaces:

Figure 47:Cashier sign in page

This screen will be displayed to the cashier:

The cashier can sign in with the username and password to access the database and modify it.

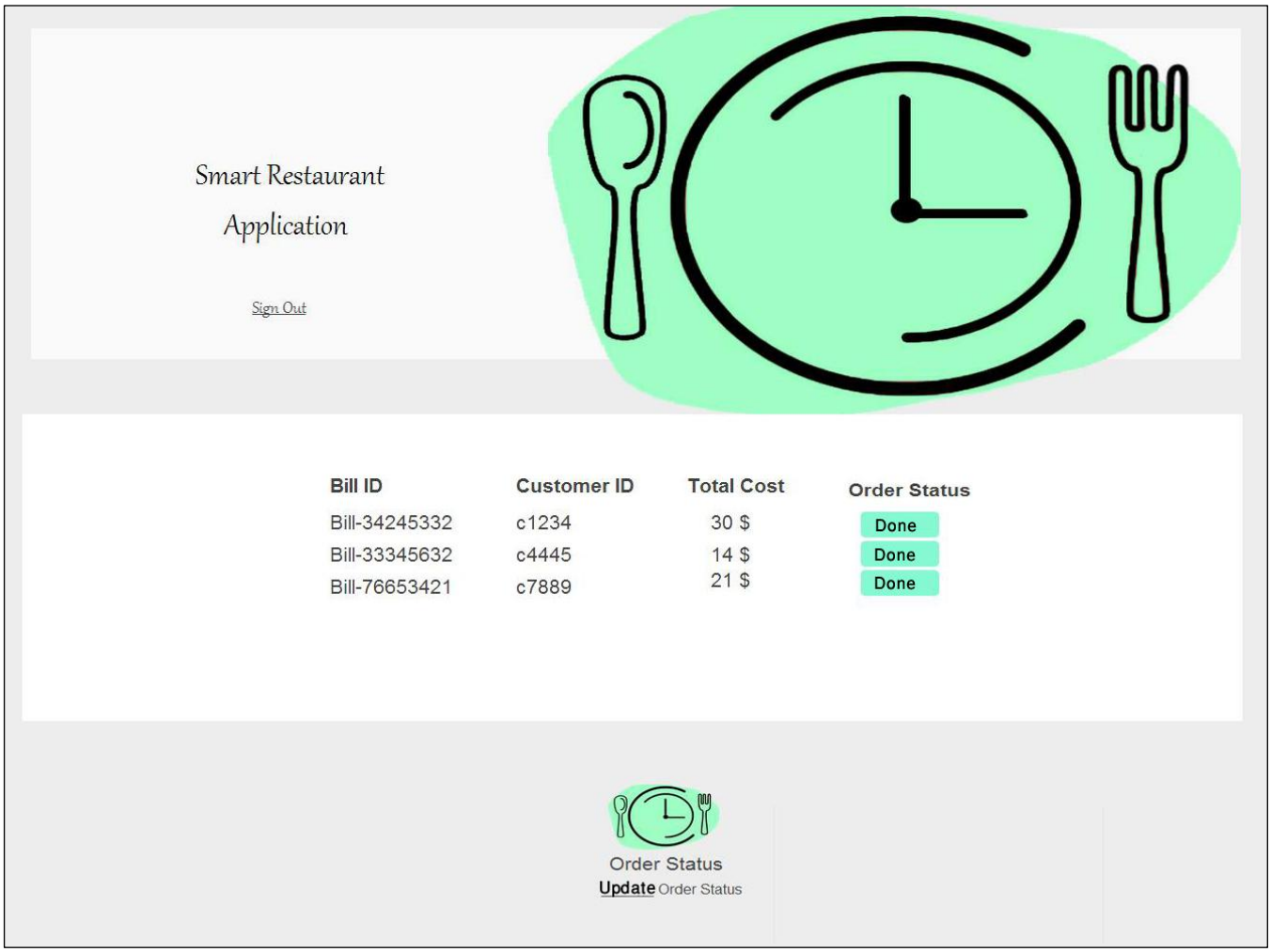


Figure 48:Cashier page

After the cashier sign in ,this screen will be displayed.

The cashier will receive the order of the customer, then update the order status.

Chapter 5 – Implementation and Validation

5.1 Implementation Phases

5.1.1 Designing Server Side

5.1.1.1 Admin Side

5.1.1.2 Cashier Side

5.1.2 Designing Interfaces

5.1.2.1 Customer Interfaces

5.1.3 Connecting Part

5.1.3 .1 Connecting Server Side With Database

5.1.3 .2 Connecting Database With Beacon

5.1.3 .3 Connecting Customer Application With Database

5.2 Project Test And evaluation

5.2.1 Running And Testing

5.2.2Testing Tools And Environment

5.4 Challenges

5.5 Conclusion

Introduction:

Now, it is time to confirm the theoretical side, which outlined in the previous chapters, to the practical side in order to make this project. By implementing the application environment, it allows having a complete managing of the restaurant and the ordering process.

The designing part is about designing the interfaces of admin, cashier and customer application in order to meet the feel and the look requirements. The programming part is about programming the necessary software that enables the server side, the database and the customer application to connect with the beacon sensor to get the main idea for this project.

As what has been discussed before, the project contains 3 main parts, which are the server side, the customer application and beacons.

The server side refers to the admin and the cashier who have the authorization to access the database and manipulate the data. Customer application that meet the customer needs, for example make an order and receive the updated offers. The last one is dispatch data to customers by the beacon.

5.1 Implementation phase

5.1.1 Designing Server Side

This part will discuss and show the main steps of creating the server side that include the admin and the cashier part, it also explains how the database tables are implemented for the application . MySql , php, notepad++ and xampp have been used as tools for implementing this part.

For the server side, html and JavaScript had been used to build the pages and the Css was used for styling. Firstly, the database "smart_smart" has been created by using phpMyAdmin with all tables and fields that we need. The basic step in the implementation is to link the program with the database and retrieve the data when needed. The following figure shows all the tables that have been created in the database .

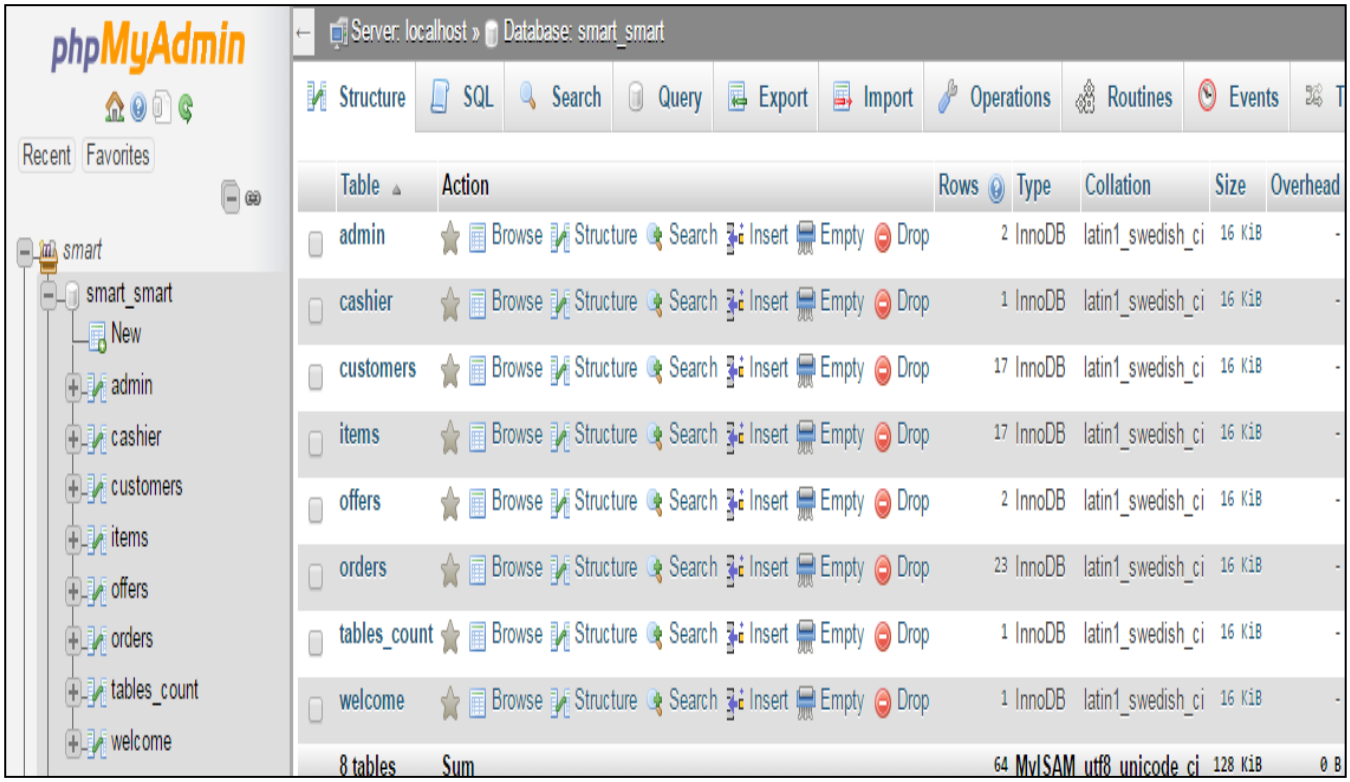


Figure 49:smart_smart DB

The following figure shows the main page at the server side (http://ideasareus.website/).

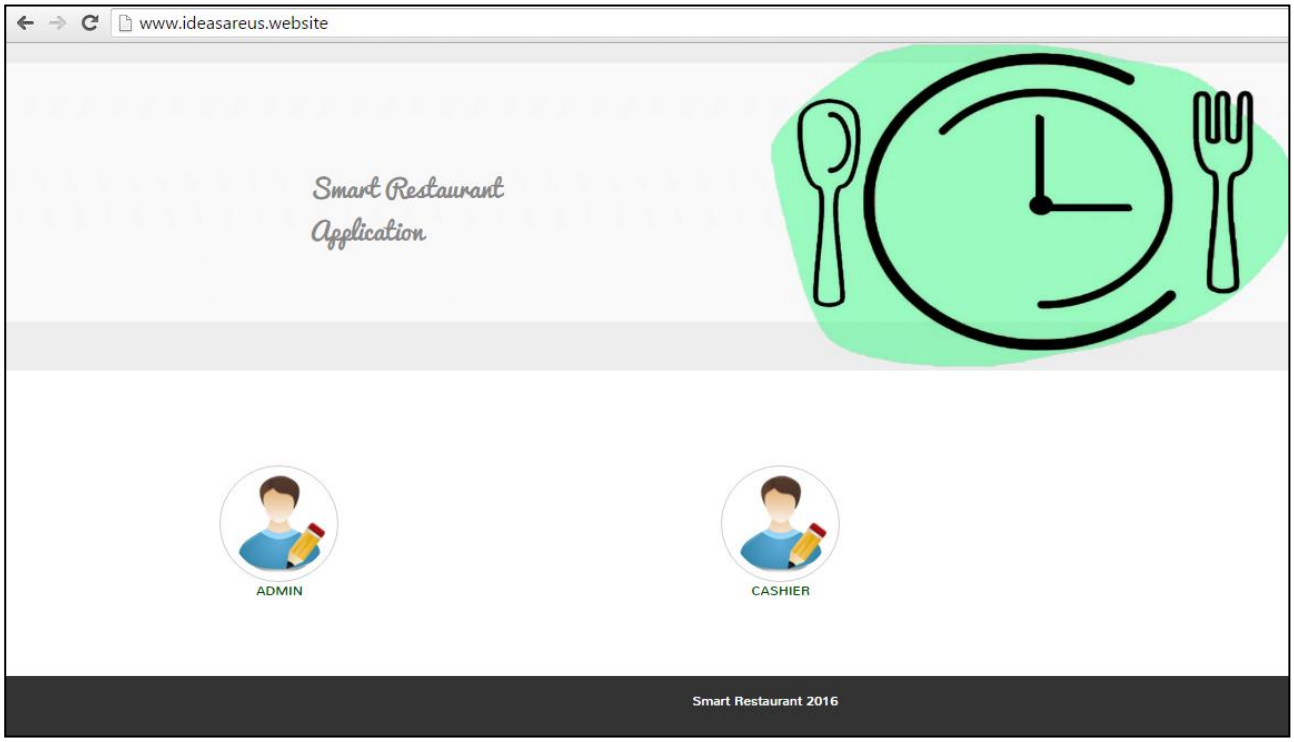


Figure 50:Ideasareus website

The page is for the admin and the cashier to manage their work. For the security purpose no one else has the authority to enter it.

5.1.1.1 Admin Side

All the admin information is saved in the Admin table in these columns (id ,email ,password and name)

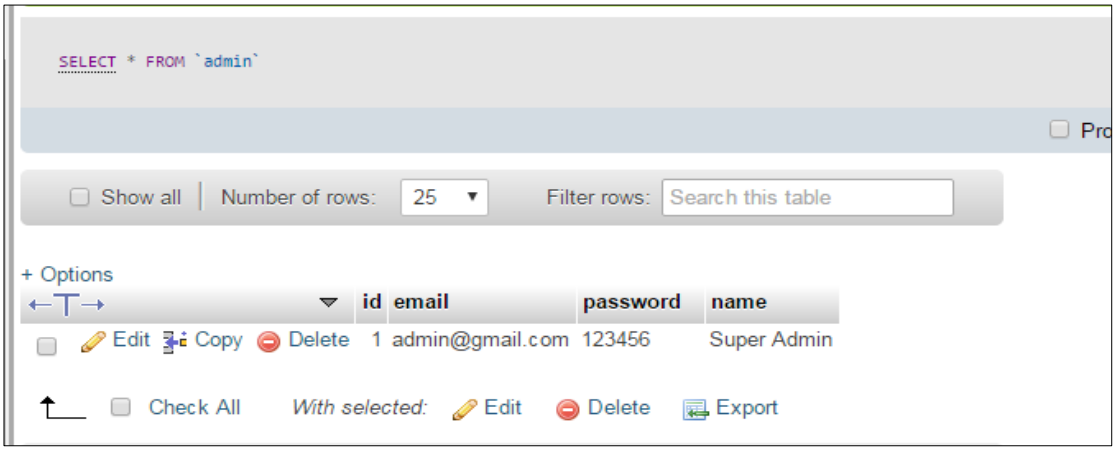


Figure 51:Admin table in DB

Admin page :

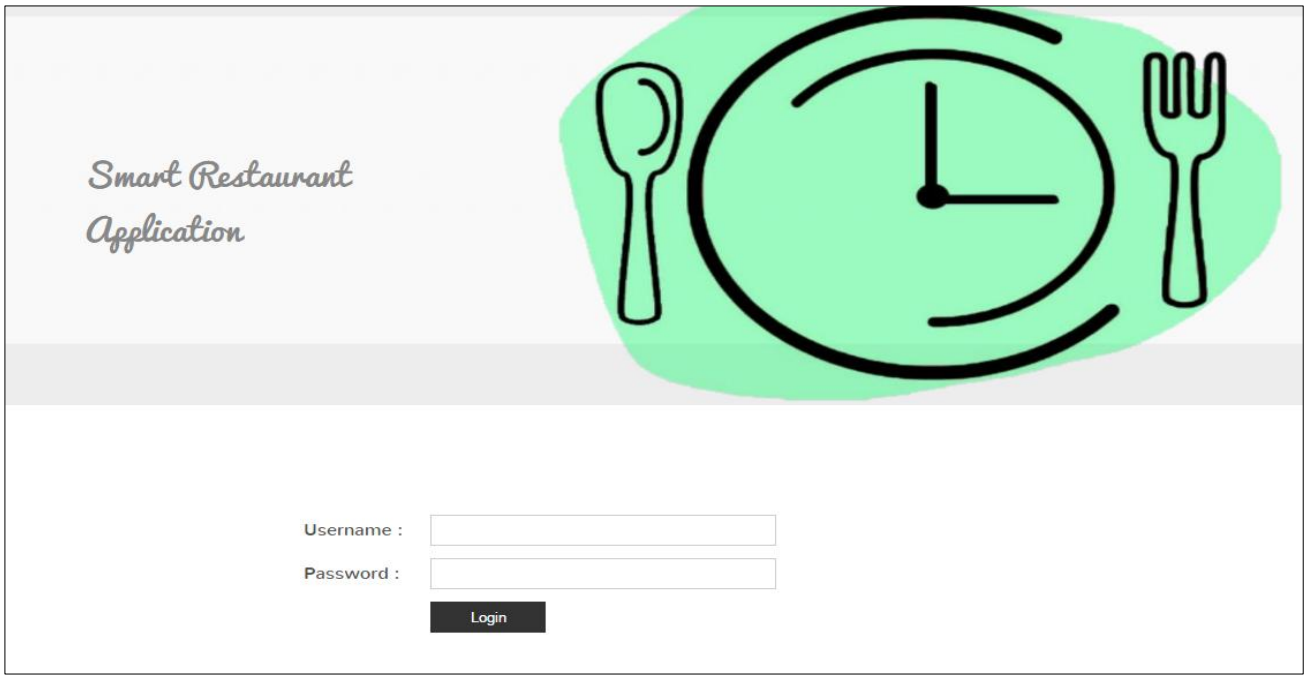


Figure 52:Admin sign in page

The admin login page was created by Using notepad++, and the two fields for username and password had been added to the page using html tags. When the admin sign in, the website will validate his information by checking the saved data from the admin table in the database . The main admin has been added to the admin table in the database (later on he can change the password to improve the website security). After sign in his homepage will be shown :

Figure 53:Admin home page

This page is for the admin to manage the whole system by connecting to the database and modify it. It contains six options .

- **Item option** : For adding new food items ,update or delete it .
- **Table option** : For adding information of the restaurant tables.
- **Welcome message option** :For updating a new welcome message to send it to the customer as notifications.
- **Offer option** : For adding new offers or delete old one from the offers table.
- **Cashier option** : For adding new cashier to the system or remove others .
- **Admin option:** For adding new admin to the system or remove others.

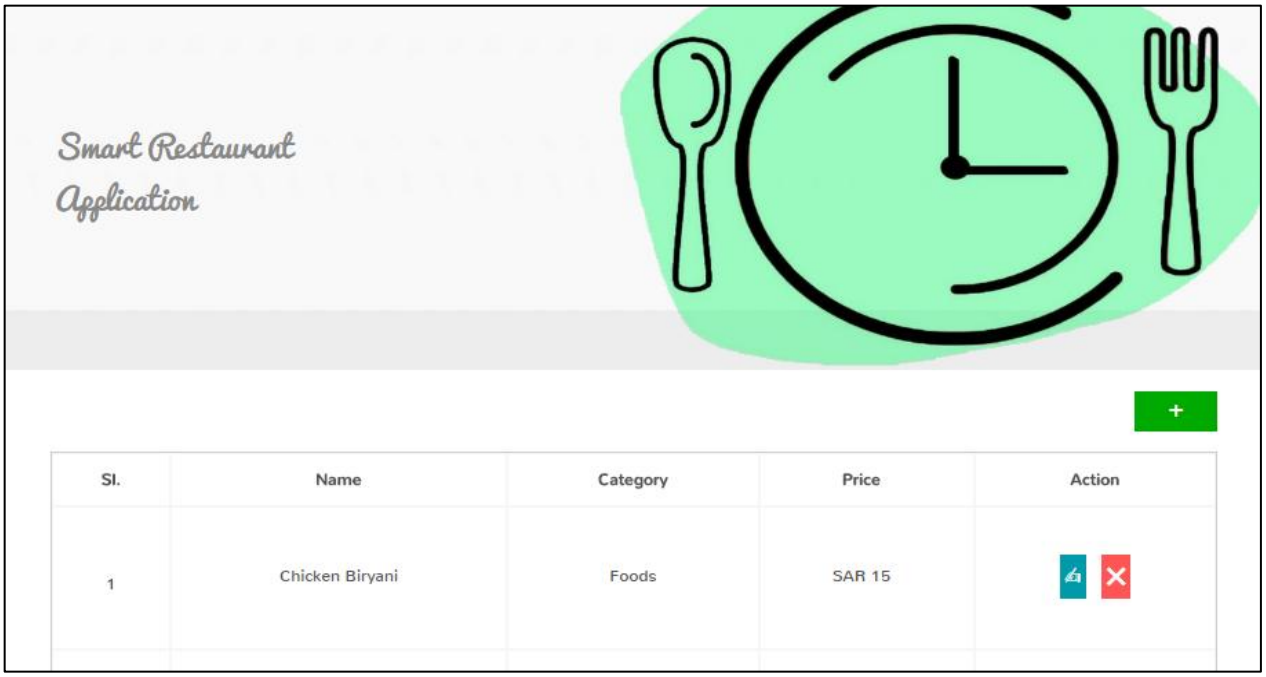


Figure 54:Add items page

When the admin choose the item option, the previous page will be shown. He can add items name, price in the restaurant menu. Also, he can delete or update any item by manipulating action column.

Figure 55:Update tables number page

This page will be uploaded when the admin choose the table icon. The page information is to provide customers with the restaurant available table. The admin here can put the number of all tables in the restaurants, and the available ones is shown to the customer by doing some calculation.

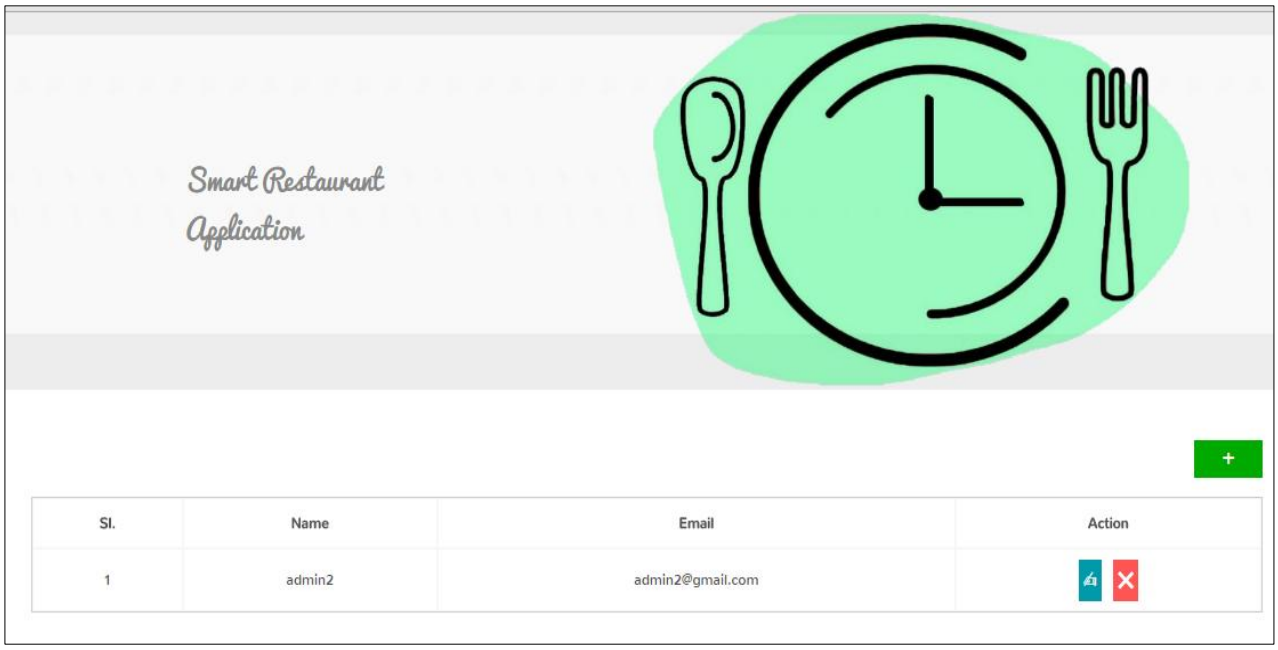


Figure 56:Update admins page

From this page, the main admin can add a new admin with his information like an email and a name. He can delete or update existing information using action column. This process can be done by choosing the admin icon.

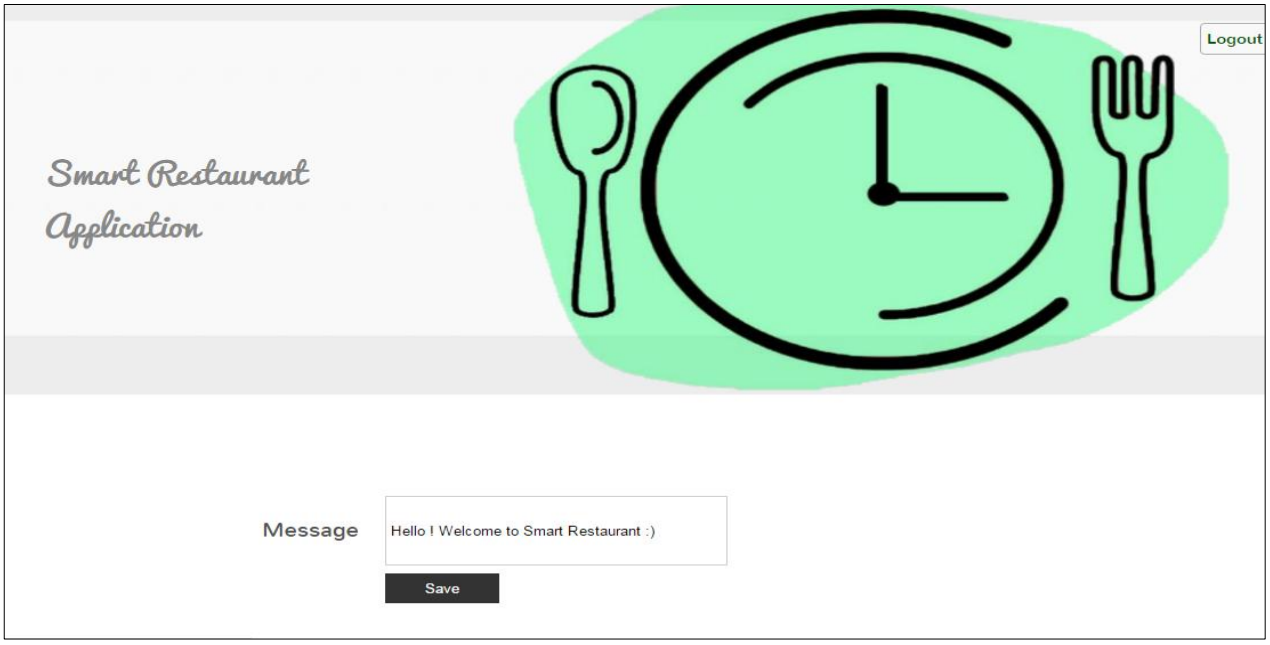


Figure 57:Welcome message page

The welcome message option allow the admin to update the welcome message that is sending as a notification when the customer enters the restaurant range.

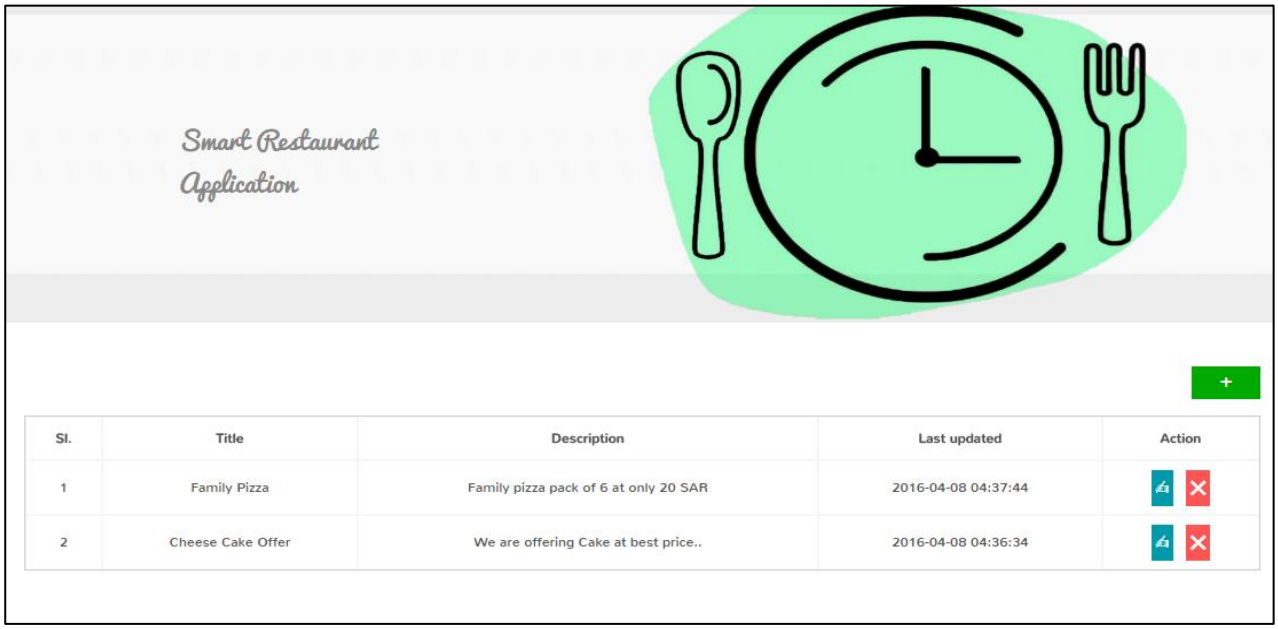


Figure 58:Offer page

Offer page allows the admin to add offers with title and description. He can edit the offers using action column. This page is uploaded by choosing the offer icon.

5.1.1.2 Cashier Side

Cashier table contain these columns: id ,email ,password and name to save the cashier information.

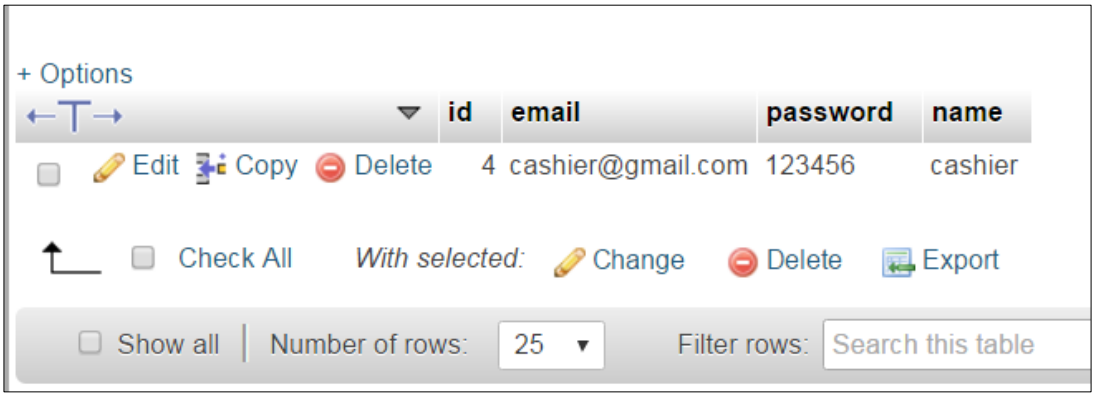


Figure 59:Cashier table

After he signs in, his page will be uploaded . This page shows the order details like the name of the customer if he logs in, the items he ordered, the order detail such as takeaway or sit in the restaurant with the total price. The cashier can close the order after the customer receive it .

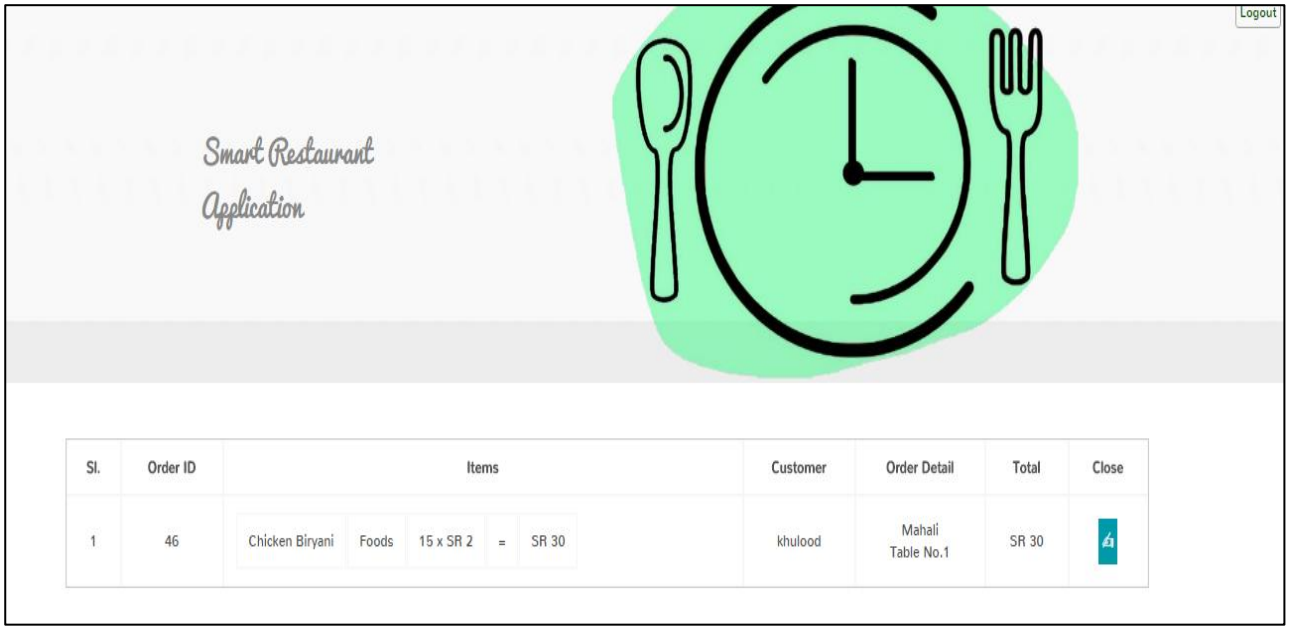


Figure 60:Cashier page

In the application, there are two techniques that are being used to perform this part functions: JSON and PHP pages.

JSON:

JavaScript Object Notation it is readable for structuring data. It is used especially to transmit data between a server and web program. is a way for storing and transporting data. It is language independent and it is easy to understand. JSON format is just text. By any programming language the code for reading and generating JSON data can be written.

PHP pages :

All pages are normal php pages, using forms ,buttons and paragraphs . Also select ,update and delete statements are used to manage the tables of the database. However, cashier-home page contains some important codes , which should be mentioned here :

The code bellow is for close order when the button is clicked, the close column in the order table will change from 0 to 1, which means the order is done and the table that associated with that order is empty now.

```
if($_REQUEST['close'] == "1")
{
    $query = "UPDATE orders SET closed='1' WHERE id='$_REQUEST[id]'";
    mysql_query($query);
}
```

In order to calculate the order price, **Json** object has to be decoded, then the items details as id and quantity can be retrieved .

Also it responsible for displaying the total in the cashier-home as :

15 x SR 2	=	SR 30

5.1.2 Designing Interfaces

In this part interfaces for the customer application will be built, to communicate with the restaurant application and finish his order quickly rather than waste his time.

In this application there are many interfaces like sign in interface, offers interface ,order interface …etc. For building interfaces eclipse has been used.

To start coding the android application, ADT (Android development tool) has to be downloaded and setup.

The following packages has to be installed:

- Android 4.4.2(API 19)
- Android support library
- Android support repository
- Android Auto API simulators

5.1.2.1 Customer Interfaces:

Sign up:

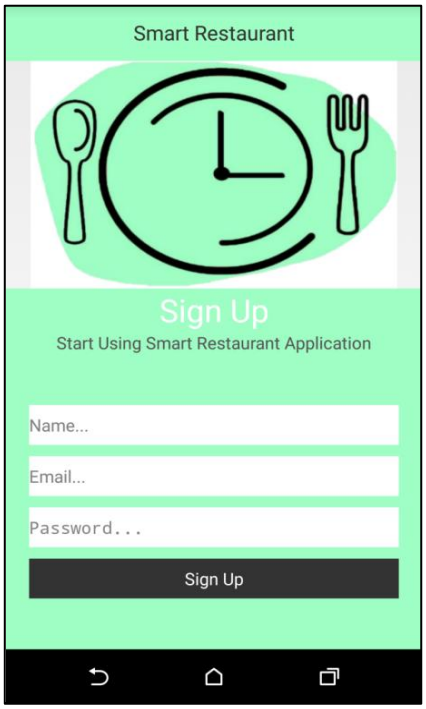


Figure 61:Sign up screen

Sign up java class has buttons and edit text "text fields".

The connection class objects is created to make a connection with the database in order to store a new customer information in the customer table in the database.

Also object of sharedPref was created .

sharedPref object is a key, value pair data can be saved and retrieved from Shared preferences, data will be continual even though user closes the application.

Also getSharedPreferences() method can be used to get values from Shared preferences,

an editor to edit and save the changes will be needed in shared preferences.

setContentView is an important function. It is always called with a layout class "xml class" to define the UI. It was almost used in each java class in the smart restaurant application. Signup.xml is the layout resource that define the UI of sign up part.

Each element added to XML files have an ID as following : `android:id="@+id/editText1"`

This ID is used to declare the element in the java class, using **findViewById** view.

When register button is clicked, **SetOnClickListener** is used to call **signupFunction** .

signupFunction():

When the customer enters enter the information in the fields in signup interface ,these information will be collected using **getText()**, then stored in variables.

If customer leaves any field empty, toast contain error message will be shown.

This link send data to database but first they have to pass by controller php file , where signup function "described before" insert them to customer table using insert statement.

doInBackground returns the result of **connectTotheServer()** which returns a string of **json data**. If this condition `if(Integer.parseInt(jobj.getString("id")) > 0 )` is checked then the intent will be used to move from this interface to category interface so the customer can make an order. By this activity the sign in activity will be skipped, so the customer doesn't have to sign in again. Toast contains a message that confirms the registration will be shown.

SignIn:

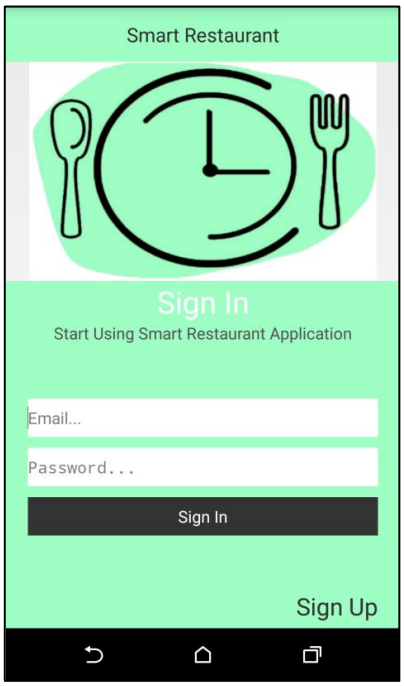


Figure 62:Sign in screen

Sign in class has almost the same code as sign up class. The difference is the following :

```
if(sp.isLoggedIn())
{Intent intent = new Intent(SignIn.this,Categories.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(intent);
    this.finish();}
```

If the customer clicked sign in, the function **isLoggedIn** from **SharedPrefs** class is called up to check if the customer's ID is already exist, if not a toast with error message will be shown, otherwise he will sign in with his name.

Categories class:

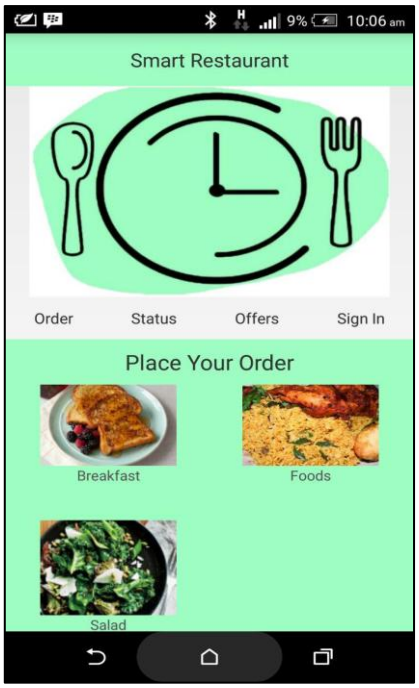


Figure 63:Categories screen

ListView displays the elements in list form, so the customer can scroll to see all the elements in the list. It is the same as other interface elements returned using **findViewById. getView** function was used as a link between listView in Categories class and rows in Category_row. Inflater has been created, category_row.xml file will be inflated to give a View. Inflater is always used to deal with listView. ListView displays the elements in list form. It has multiple views and those views must be in another xml layout .

```
public class Categories extends CustomActivity{

        private ConnectionClass cc;
        ArrayList<JSONObject> list = new ArrayList<JSONObject>();
        ListView lv;
        private CategoriesAdapter adapter;
        static TextView orders;
        public static CustomActivity activity;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            activity    = this;
            setContentView(R.layout.categories);
            lv                  = (ListView)findViewById(R.id.lv);
```

In **untialize()** function **doInBackground()** was used to retrieve data from the database and add these data to the array list, then it can be used in **CategoryAdapter** class to add them to the ListView .

CategoriesAdapter:

To start manipulating the application screen using another xml layout, the help of **LayoutInflater** class must be used. Inflate means reading the XML file and taking it as input, then creating actual objects that correspond to it. After that, it will make the object visible within an application screen.

```
try
{
            holder.parent1.setVisibility(View.VISIBLE);
            final JSONObject jobj    =list.get(tmpPosition);

            holder.name1.setText(jobj.getString("cat"));

            holder.parent1.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View arg0) {
                    try
                    {
                        Intent intent = new Intent(mContext,Items.class);
                        intent.putExtra("data", jobj.toString());
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        mContext.startActivity(intent);
```

Here, the first linear layout code **visibility** will set to visible. First, the category name will be set to **name1** textView. When the first category linear layout is clicked, the customer will move from this interface to the items interface to make an order.

Items and ItemsAdapter have approximately the same code. The differences will be discussed next .

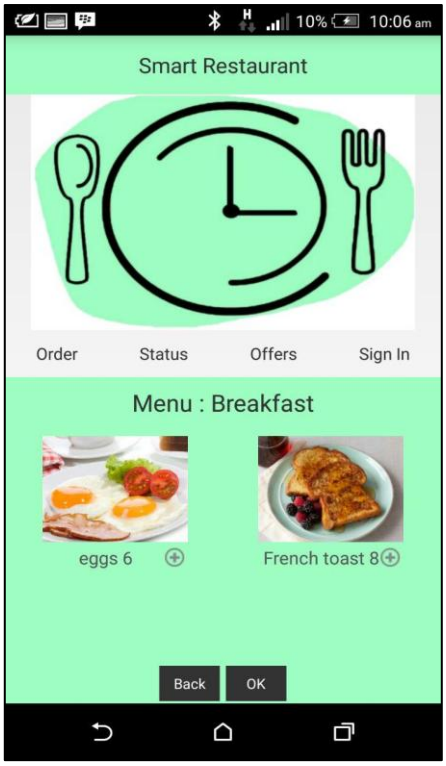


Figure 64:Items screen

```
public static void setItemCounts(int idval,final int qty,final String name)
{
    try
    {
        Items.activity.runOnUiThread(new Runnable() {

            @Override
            public void run() {
                Toast.makeText(Items.activity, name+" : "+qty, 80).show();
            }
```

runOnUiThread: is used to update elements in the UI, here orders**. Toast** will show the number of the items in the order after retrieving it from addItem **hashMap .**

```
public static void addItems(int idval,final int qty,final String name)
{

    MainActivity.addItem.put(idval,qty);
    setItemCounts(idval,qty,name);
    Context context = null ;
    CharSequence text = "Items Successfully added ";
    int duration = Toast.LENGTH_SHORT;

    final Toast toast = Toast.makeText(Items.activity, text, duration);

    toast.show();
}
```

The previous function is to add each item ID, name, and quantity to addItem **hashMap**. This function is used in **ItemsAdapter** class to add items with each click on item button.

In ItemsAdapter class:

When any item is clicked, the quantity will be updated and 1 will be added to that item .

ListOrder:

```
if(MainActivity.addItem.get(idval) > 0) // if items quantity lager than 0
{

    list.add(MainActivity.itemDetails.get(idval));
    float v = MainActivity.addItem.get(idval) * Float.parseFloat(list.get(j).getString("price"));
    j++;
    total += v ;//total calculation
}
```

If the item is in the addItem hashMap, then its json object where all its details are stored will be add to the array list, and the total for that item will be calculated (total means if there is more than one item of the same kind). The total will be calculated by multiplying the quantity and price of that item .

When the ok button is clicked, the following dialog will be shown:

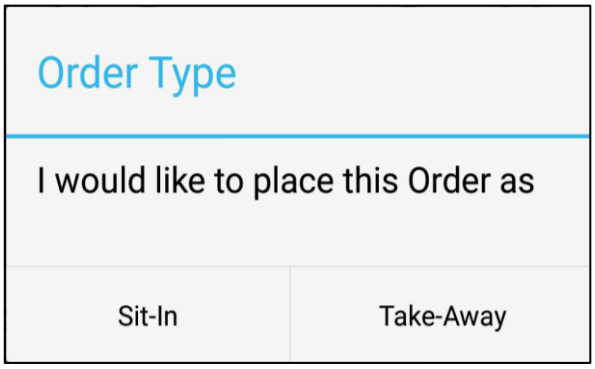


Figure 65:Order Type

This dialog will appear to enable the customer to choose the type of order. This dialog has two options: one for sit_in orders and the other to take away orders. After clicking one of them, the number of the chosen type will transfer to placeOrder function (1=Take-Away,2=Sit-In).

When customer click cancel button, then the order will be canceled. That means the addItems hashMap will be deleted, and the customer will be transferred again to the categories layout.

```
public void placeOrder(final int type,final Button v)
{
    v.setEnabled(false);
    JSONArray jarray = new JSONArray();

    for(int i=0;i<MainActivity.idlist.size();i++)
    {
        try
        {
            JSONObject jobj  = new JSONObject();
            int id = MainActivity.idlist.get(i);
            if(MainActivity.addItem.get(id) > 0)//put items of the order into Json array
            {
                jobj.put("id",id);
                jobj.put("qty",MainActivity.addItem.get(id));

                jarray.put(jobj);
            }
        }catch(Exception e){}
    }
    final String items = Uri.encode(jarray.toString());//change Json array into proper form to send it to the database
```

placeOrder function is for collecting the items in the order and for preparing them to be sent to the database. If the quantity of that item is greater than 0 ,then that item ID and requested quantity is added to json object. After that, all json objects of all the requested items are added to the json array. Then, that json array is encoded to be ready to sent to the orders table in the database.

```
protected void onPostExecute(String result) {
        try
        {
            JSONObject jobj = new JSONObject(result);

            if(jobj.getString("msg").equals("1"))//msg came from placeOrder1 as response after check the order validation
            {
                MainActivity.addItem.clear();
                Intent intent = new Intent(ListOrder.this,AfterOrder.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                intent.putExtra("data", jobj.getString("data"));
                startActivity(intent);
                ListOrder.this.finish();
            }

            else if(jobj.getString("msg").equals("2"))
            {
                Toast.makeText(ListOrder.this, "There is no available table", 2000).show();
            }
```

After the order is send to the database, **placeOrder1()** function in **controller.php** file checks the order (described later) and returns a json object contain a' msg' which is the number "1 or 2 " and a 'data' which contains a message that describes the order situation .

If 'msg' ==1, then the order is completed and the customer will be transfered from ListOrder interface to AfterOrder interface. Also 'data' will be printed in AfterOrder interface.

'data ' may be one of the following two, depending on the type of the order :

- **Thank you for Ordering**
  Your Order ID : $orderid
  Table number : $t
- **Thank you for Ordering**
  Your Order ID : $orderid
  Please wait and we will call you once the food is prepared :).

If 'msg'==2, which means the order of type **sit_in** and there is no available table.

ListOrderAdapter is just like categotiesAdapter and itemsAdapter classes that just displaying the order items in listView.

AfterOrder class:

AfterOrder has nothing new just a text view to show the received message from ListOrder class. The message shown after the order is completed .

Figure 66:Order ID(Take-Away)

Figure 67:Order ID(Sit-In)

Offer and Restaurant Status interface :

Figure 68:Offer screen

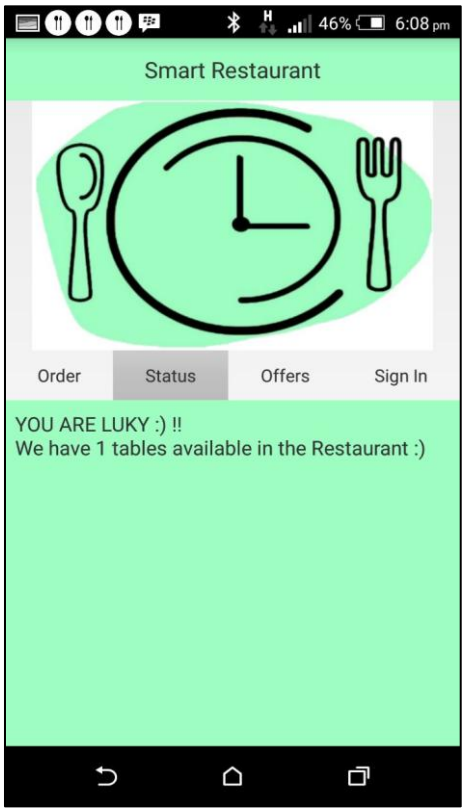


Figure 69:Status screen

These screens just show the restaurant status and the offers that the restaurant have.

5.1.3 Connecting Part

Connectivity is very important to complete the project and achieve the target goal. There are three parts for connecting: connecting the server side with the database, connecting the database with the beacon, connecting the customer application with the database. This was a big challenge because of connecting different parts using new technology. This was a new experience. Many obstacles were faced, but at the end we learned many new things in programming and computer science.

5.1.3 .1 Connecting the Server Side With the Database

To connect the server side admin and the cashier pages with database **config .php** class will be created by using notepad++. This class mentions the name of the database.

More details of this code:

```php
<?php
error_reporting(E_ERROR);
mysql_connect("localhost","smart_smart","");
mysql_select_db("smart_smart");


?>
```

5.1.3 .2 Connecting the Database With the Beacon

Before starting, this part explains how the beacon was used in smart restaurant application with eclipse. The open source library "altbeacon.beacon" and codes and tutorial were downloaded and used [11].Now let's go through the Beacon region and how to build it. With the beacon region, there are many options to initialize it : UUID + major + minor, UUID + major, UUID alone, or none of them . In case of UUID + major + minor, you can specify all these value to make the region more specific to one beacon. With UUID + major to make the region to one place, or less specific requiring a UUID, or none of them which means "all Beacons".

Steps to build the application interact with beacons :

First step, use these classes from "altbeacon.beacon" library:

```
import org.altbeacon.beacon.Beacon;
import org.altbeacon.beacon.BeaconConsumer;
import org.altbeacon.beacon.BeaconManager;
import org.altbeacon.beacon.BeaconParser;
import org.altbeacon.beacon.Identifier;
import org.altbeacon.beacon.MonitorNotifier;
import org.altbeacon.beacon.RangeNotifier;
import org.altbeacon.beacon.Region;
```

The most important class is **BeaconManager** which is actually the gate to most interactions with Beacons.

Below, the picture shows the initialization of the **BeaconManager** and other important classes such as region .

```
public class BeaconBGService extends Service implements BeaconConsumer{


   private Region region;
   private BeaconManager beaconManager;
   ArrayList<String> beaconIds = new ArrayList<String>();
   public static Activity activity;
   public static String BEACONID = "0xB9407F30F5F8466EAFF925556B57FE6D";// the id of our beacon called blueberry
   public static int ENTRY_THRESHOLD = 1;
   public static int EXIT_THRESHOLD = 3;
```

The ID of the beacon has been found from estimote cloud website .

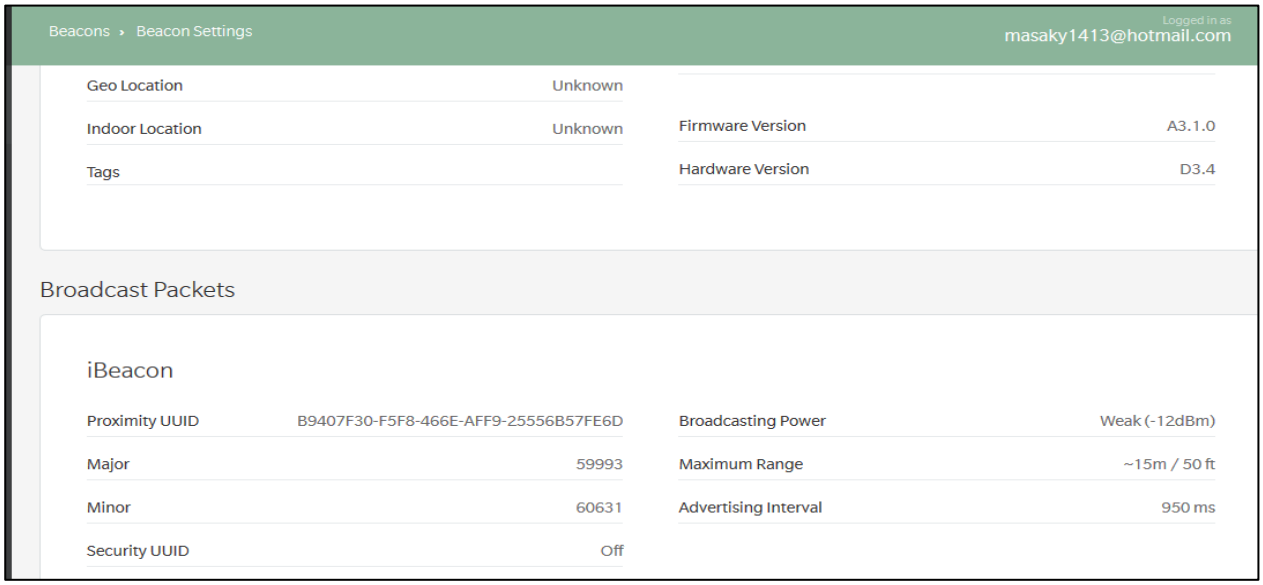


Figure 70:Beacon information

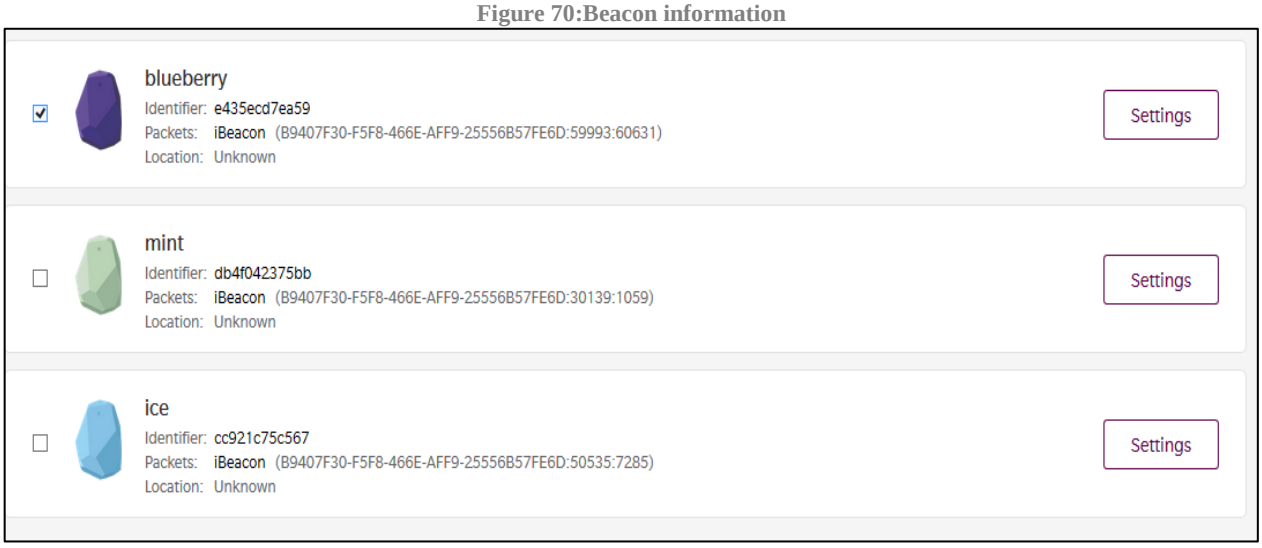


Figure 71:Beacon type

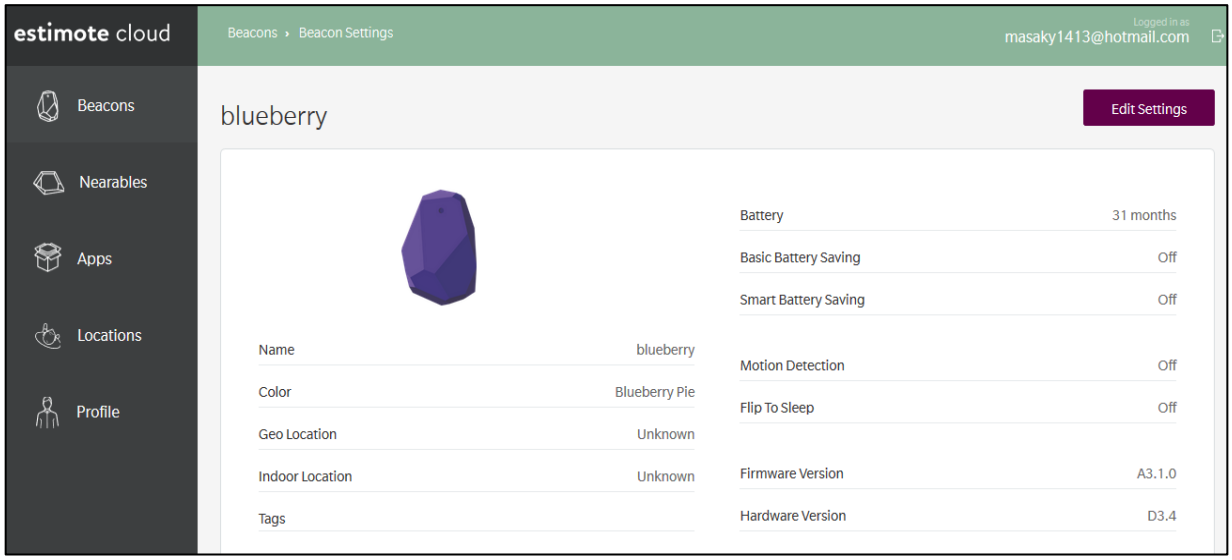


Figure 72:Blueberry Beacon information

And then there is code that will check the condition of whether the customer is in range or not:

```
public void didRangeBeaconsInRegion(final Collection<Beacon> beacons, Region region) {
   for(int i=0;i<beacons.size();i++){
      final Beacon beacon = beacons.iterator().next();
       String idVal = beacon.getId1()+"";

        if(validIdentifiers.contains(idVal.toUpperCase())){
            float distance = (float)beacon.getDistance();
            if(sp.getLastDistance() > ENTRY_THRESHOLD && distance < ENTRY_THRESHOLD)
            { // assuming the consumer is entering the restaurant range
                    sp.setLastDistance((float)beacon.getDistance());

                    if(activity == null)
                    {
                        Intent intent = new Intent(getApplicationContext(),MainActivity.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        intent.putExtra("welcome", true);
                        startActivity(intent);
                    }

                    welcome();
                    offers();
                    status();

                }
```

Here, the picture shows if the customer is in the restaurant range. The beacon will dispatch its location and will get this distance and compare it with the user's location. Also with the use of the ENTRY_THRESHOLD variable, if the condition is true, it will call the methods **welcome(), offers(), status()** that will send the notifications to the customer application on the Smartphone.

```
}
else if(distance >= EXIT_THRESHOLD && sp.getLastDistance() < EXIT_THRESHOLD)
{
    sp.setLastDistance((float)beacon.getDistance());

    showNotification("Thank you","Will look forward to serving you again :)");
}
```

Here are the same actions, but if the customer exits the restaurant range, it will just show the notifications with a simple static massage "Thank you, "Will look forward to serving you again :)"

Now is the time to show a real action like the welcome message, special offers, and status of the restaurant in a notification. The code below is for a welcome message. The function **getHelloMsg** is a function in php controller file. It will retrieve the data from the database and put it in the call of function **showNotification**, in the same way the offers() and status() methods.

```
    void welcome()
    {
    cc  = new ConnectionClass(this);
    new AsyncTask<Void, Void, Void>()
    {
    private String helloMsg;

    @Override
    protected Void doInBackground(Void... params) {
helloMsg= cc.connectToServerFunc("f=getHelloMsg");// f is the name of the function that return getHelloMsg function
                return null;
            }

    protected void onPostExecute(Void result) {
                if(helloMsg != null)
                    showNotification("Welcome",helloMsg);
                else
                    showNotification("Welcome","Welcome to Smart Restaurant");
            };
            }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
    }
```

showNotification is a helper method. It will send and show the notification on the customer's Smartphone according to its use in many other function like welcome(), offers() and status() functions.

Notifications Part :

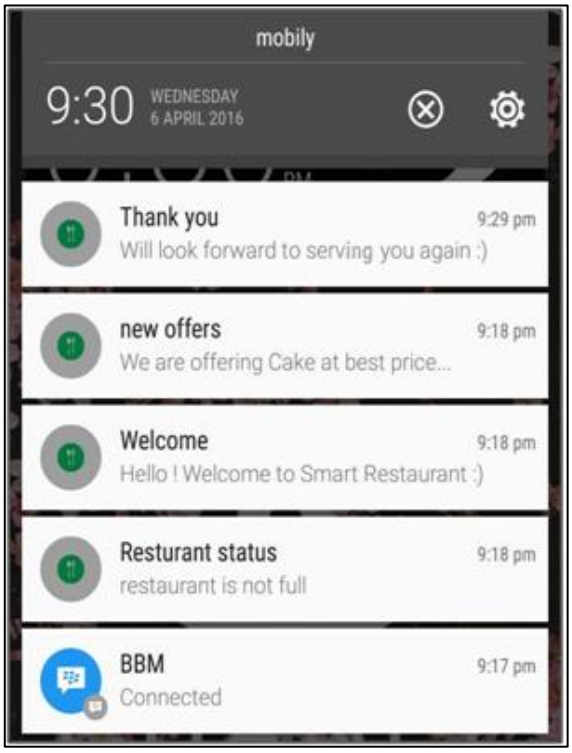


Figure 73:Notification

Notification is an open source code [13].

Beacon Part :

When the beacon activity was initialized, the important functions which is include in ANDROID LIFE-CYCLE activity will start such as:

- onResume(): which means when Begin Interaction, it will start the beacon service which identifies in the BeaconBGService class.
- onDestroy(): which means final call received before the activity is destroyed and this call would be **BeaconBGService.activity**

```
public class CustomActivity extends Activity{

    @Override
    protected void onResume() {
        super.onResume();

        BeaconBGService.activity = this;
        Intent intent = new Intent(this,BeaconBGService.class);
        startService(intent);
    }


    @Override
    protected void onDestroy() {
        super.onDestroy();
        BeaconBGService.activity = null;
    }
```

Connecting the customer order with the beacon:

Here is the most important part in the project. This code actually checks if the customer is inside the restaurant's range or not when placing the order. It will compare the customer distance with the beacon and if the condition is true it will call the function **showOrders().** Otherwise, it will call the function **showNotInRange().**

```
        if(sp.getLastDistance() <= BeaconBGService.ENTRY_THRESHOLD)
        {
            showOrders();
        }
        else
        {
            showNotInRange();
        }
    }
```

showNotInRange() :

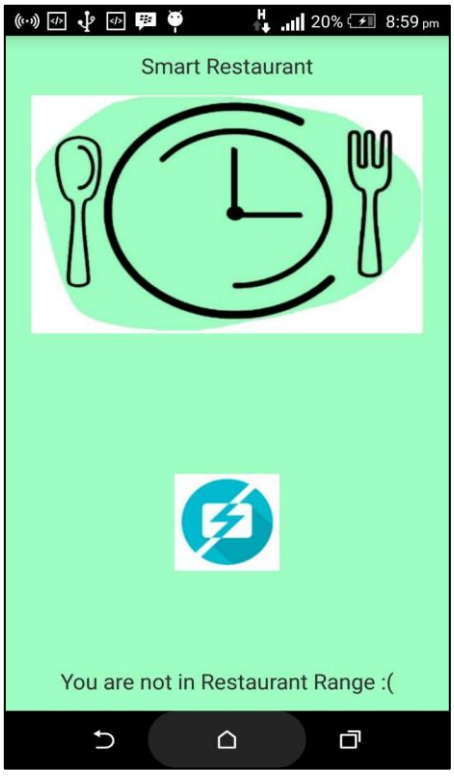


Figure 74:Rejected message screen

5.1.3 .3 Connecting the Customer Application With the Database

The server pages and the database tables has been uploaded on the server with a domain named "ideasareus.website" .

An eclipse class named **ConnectionClass** was created, whose main purpose is to connect to the database for all the project classes. There is no need for repeating the code.

The following lines of code from **ConnectionClass** contain the link of the server where the database is uploaded [14].

```
public static String BASE_URL="http://ideasareus.website/";
public static String SERVER_URL=BASE_URL+"controller.php";
```

Controller php :

Controller.php file contain many functions. These functions are used as a connection between the codes of different classes in a smart restaurant application and the database tables . The functions are responsible for manipulating and retrieving the data from the database.

The controller contains the following functions:

- placeOrder1().
- login().
- signup().
- getOffers().
- getCategories().
- getItems().
- getHelloMsg().
- getoffersMsg().
- appFreeTable().
- getFreeTable1().(notification status)
- getFreeTable2().(restaurant status in application)

Here is the description of each function and its work:

signup():

```
function signup()
{
    $name       = $_REQUEST['name'];
    $email      = $_REQUEST['email'];
    $password   = $_REQUEST['password'];


    $query      = "SELECT * FROM customers WHERE email='$email'";
    $res        = mysql_query($query);
    $num        = mysql_num_rows($res);
    if($num)
    {
        echo "Email taken";
        exit();
    }
```

The ***signup*** function inserts name, email and password which is sent by the application to the customer's table. It also selects from **customers table** using email to check if the customer is already has an account in the application

After inserting the data to customer table, select statement has been used to retrieve inserted data from database as json object that contains all the data of that customer. This object can be used in the application to keep the customer signed in.

login():

Log function uses the email and password which is sent by the application and select from customers table, using them to retrieve json object which contains all the data of that customer. This object can be used in the application when it is needed.

getOffers():

```
function getOffers()
{
    $array      = array();
    $query      = "SELECT * FROM offers ORDER BY dated DESC";
    $res        = mysql_query($query);
    while($data     = mysql_fetch_assoc($res))
    {
        $array[count($array)]    = $data;
    }

    echo json_encode($array);
}
```

getOffers function is used to retrieve all the offers in **offers table** and to print them in offer interface in smart restaurant application .

This function uses select to retrieve all offers that saved in offers table and orders them by their dates. Then it adds them to array and converts it into json object . This object can be used in the application to show the offers to the customer .

getHelloMsg() and getoffersMsg():

These two functions have the same work and approximately the same code. They just used select statement to retrieve messages from 'welcome' and 'offers' tables, and then print them. Each of them is used in the notification message.

getCategories():

getCategories function uses select statement to retrieve all rows from items table , groups them and orders them by cat (category) column, then adds them to array and converts it into json object . This object can be used in the application to show the categories in category listView rows .

jobj.getString("cat") is used in the application to retrieve category name from json object and is use it where it is needed.

getItems():

```
function getItems()
{
    $catName    = $_REQUEST['cat'];
    $itemsList  = array();
    $query      = "SELECT * from items WHERE cat='$catName' GROUP BY name ORDER BY name ASC";
    $res        = mysql_query($query) or die(mysql_error());
    while($data = mysql_fetch_assoc($res))
    {
        $data['image']          = str_replace("images/items/","",$data['image']);
        $data['image']          = SERVER_URL."images/items/".rawurlencode($data['image']);
        $data['price_with_currency']    = "SR"+$data['price'];
        $itemsList[count($itemsList)]   = $data;
    }
    echo json_encode($itemsList);
}
```

getItems() function uses category name 'cat' which is sent by the application and selects from items table, to retrieve json objects which contain all the data of the items of that category. This object can be used in the application to show the items in item listView rows.

placeOrder1():

```
   function placeOrder1()
   {$type      = $_REQUEST['type'];    // 1 = Take-Away   2= Sit-In
       $items      = $_REQUEST['items'];
       $customerid = intval($_REQUEST['uid']);
       $jsonDecode = json_decode($items,true);
        $total=0;
       if(!count($jsonDecode))return;
for($i=0;$i<count($jsonDecode);$i++){
       $id=$jsonDecode[$i]['id'];
       $qty=$jsonDecode[$i]['qty'];

       $query1         = "SELECT * FROM items where id='$id'";
   $res1       = mysql_query($query1);
       $data1      = mysql_fetch_assoc($res1);
       $price=$data1['price'];
    $total+=$price*$qty;}


       $t = json_decode($this->appFreeTable(),true);
       $t = intval($t['num']);
       if(!$t && $type == "2")
       {$element['msg']    = "2";
           echo json_encode($element);
           exit();}
```

```
if($type == "2")
    $query      = "INSERT INTO `orders` (`customerid`, `items`, `tablenumber`, `type`,`total`)
VALUES ('$customerid', '$items', '$t', '$type','$total')";
else
    $query      = "INSERT INTO `orders` (`customerid`, `items`,`type`,`tablenumber`,`total`)
VALUES ('$customerid', '$items', '$type','-1','$total')";

$element    = array();
$exe        = mysql_query($query) or die(mysql_error());
$orderid    = mysql_insert_id();
if($orderid > 0)
{$element['msg']    = "1";


if($type == "2")
        $element['data']    = "<b>Thank you for Ordering<b><br>Your Order ID :
    $orderid<br>Table number : $t";
    else
        $element['data']    = "<b>Thank you for Ordering<b><br>Your Order ID :
    $orderid<br>Please wait and we will call you once the food is prepared :)";
}
echo json_encode($element);
exit();}
```

After receiving the type of order and the items of that order, the order items json array is decode and the available number from **appFreeTable()** is retrieved. If the table number does not exist and the order type is 2, which mean sit_in, then $element['msg'] will be set to 2 and it will be used to print this ' Please wait and we will call you once the food is prepared ' in customer application. Otherwise, the order is inserted into orders table. If ordered >0 which means the order is added to the database successfully, then $element['msg'] will be set to 1 and $element['data'] value is assigned to it according to the type of the order. Finally $element encodes the json object so it can be used in the application where it is needed.

appFreeTable():

This function returns the available table number so it can be used in placing a sit_in order. First ,select statement is used to return the table number of the order that still in the restaurant from order table and use explode to break it into an array called $t. After that all restaurant table count is returned from **table_count** table. And put all these numbers in an array "using loop" called $t2Array.

"The **array_diff()** function compares two or more arrays values, and returns the differences."

After that implode and explode are used to arrange the numbers in $freeTables array . Finally, the first available table number will be returned . This number is add to json object and is used where needed .

getFreeTable1():

getFreeTable1() uses a table number that is returned from appFreeTable() to check if there is any available table in the restaurant. If there is no available table, then "Restaurant is full " message will be shown as a notification . Otherwise, "Restaurant is not full " message will be shown. This function is used in show status notification .

More Code details in appendix B

5.2 Project Test and evaluation

When the project has been finished, it is time to test it. Testing is the last process. We want to find bugs or validate the program to meet the customer requirements, also to be sure that the application works as expected.

5.2.1 Testing Tools And Environment

There are important setup which should be kept in mind and used it before the test can be done :

Hardware: The program will run in any Smart phones that use Android OS(+4.4.2).

Operating system: Android OS

Other tools: Bluetooth and beacon sensor and wifi.

5.2.2 Running and Testing

It is expected when you reach this testing stage, the application is ready and programming the main parts is finished. The requirements of a Smart restaurant project will be used to test it, if it works as expected.

Testing steps:

1- Download smart restaurant application.
2- Enter the beacon range, or be outside the range.
3- Start using the application.
4- Try to make an order.

Requirements evaluation:

This part will evaluate how the application meet the required function.

Table 35:Test project

Num	functions	Test description	Expected result	Actual result	Action taken if any
Application					
1	Sign up	Enter email ,name and password to sign up in application .	Sign up will be done successfully	Sign up done successfully , and customer information store in database.	-
2	Sign in	Enter email and password to sign in in application .	Sign in will be done successfully	Sign in done successfully .	-
3	Display offers	Open offer interface in application and display offers .	All the available offers will be shown successfully.	offer interface opened and offers shown successfully.	-
4	Display table number	Open status interface in application and display table number.	The available table number will be shown successfully.	Status interface opened and table number shown successfully.	-
5	Display menu	Open order interface in application and display menu categories.	The menu categories will be displayed successfully.	Menu categories displayed successfully.	-
6	Display Items	Choose one the categories and display category items.	Each category item will be shown successfully.	Category items displayed successfully.	-
7	Make order	After clicking on many items and	Order will be done successfully	Items had been chosen	-

		Press ok button order will be done .		successfully and when ok button is clicked, order is done .	
8	Calculate total cost	Clicking ok button and display total .	Cost for all order items will be calculated successfully .	Ok button licked successfully and total was shown successfully.	-
9	Receive bill (in range)	Customer in beacon range ,click ok button and display bill.	After order making, bill will be received successfully.	Bill was shown successfully.	-
10	Reject message (not in range)	Customer not in beacon range ,press ok button and display reject message.	After order making , rejected message will be received successfully.	Reject message was shown successfully.	-
10	Notification (welcome)	Enter beacon range to receive welcome message notification.	Welcome message notification will be received successfully.	Welcome message notification was received successfully.	-
11	Notification (Offer)	Enter beacon range to receive offer notification.	Offer notification will be received successfully.	Offer notification was received successfully.	-
12	Notification (Status)	Enter beacon range to receive status notification.	Status notification will be received successfully.	Status notification was received successfully.	-
Admin Side					
13	Admin sign in	Enter email and password to sign	Sign in will be done and home	Sign in is done successfully and	-

		in.	page will opened successfully.	home page is opened - successfully.	
14	Add new admin and cashier	Enter email , name and password to add new admin and cashier .	New admin or cashier will be added successfully .	New admin or cashier information added and store in database successfully .	-
15	Update welcome message	Enter new welcome message in welcome message field .	Welcome message will be updated successfully .	Welcome message updated successfully .	-
16	Update table number	Enter new table number in table number field .	Table number will be updated successfully.	Table number was updated successfully.	-
17	Add items(delete,update)	Fill new item name, price, category and image in their proper fields in items page . Update new item name, price ,category and image in their proper fields in items page . Delete item by clicking delete button.	items will be added ,updated or deleted successfully.	New items information were added and store in database successfully. Existed items information were updated successfully . Existed items information were deleted successfully	-
18	Add offers(delete,update)	Fill new offer title and description in	offers will be added, updated or deleted	New offers information were added and	-

		their proper fields in offers page . Update new offer title and description in their proper fields in offers page . Delete offer by clicking delete button.	successfully.	stored in database successfully. Existed offers information were updated successfully . Existed offers information were deleted successfully	
Cashier side					
19	Close the order	Order details shown in the cashier home page ,cashier click close button and close order.	Order will be close successfully.	Order was closed and disappeared from cashier homepage	-

Test Smart Restaurant application in Zonah cafeteria :

Figure 75:Zonah menu

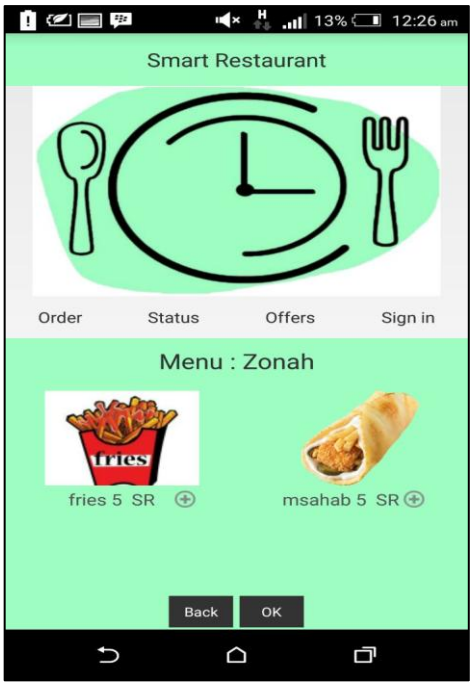


Figure 76:Zonah Items

The test was taken place at the Zonah cafeteria in the Umm Al Qura university. Two evaluators were chosen for evaluating the customer application on their smartphone, and the cashier side was evaluated by us.

Scenario 1(Evaluator 1):

At first, the evaluator downloads the application and opens Bluetooth. She opens the application and tries to make an order about one meters from the beacon, which means she is in the range. The order will be made and the total will be displayed.Also, they will receive the welcome message, restaurant status and offers. They can choose the order type. If they choose Sit-In , the available tables will be displayed in the application.

Table 36:Test application (evaluator1)

Step	evaluation action	Expected result	Result(pass/fail)
Functional Requirements			
1	Sign up/sign in	Customer can make a new account and sign in.	Pass
2	Make an order.	The order will be made.	Pass
3	The total display.	The total for the order will display on the screen.	Pass
4	Receive the welcome message, offers and status .	Get notification when the customer enters the range.	Pass
5	Choose order type.	Choose between Sit-In or Take-away.	Pass
6	Available table message display.	If Sit-In option is chosen, the available table will be displayed in the application.	Pass
Non- Functional Requirements			Good/bad /not bad
6	Interfaces will be clear and simple.	The look of interfaces is good. Not annoying and distracting	Good
7	The icon names will be easy to understand by different users	The terminology that is used is easy and simple.	Good
8	The application will be interactive and has fast responses for different interaction.	Fast responses.	Not bad
9	Easy for new users to learn to perform the task	Clear text icon and simple options.	Not bad

1- Do you have any suggestions for the development and improvement the program?

No

2- Can you tell us if there is any difficulty in the program?

No, there is not.

3- What are the things that you liked most in the program?

Easy to use .

4- Do you think it covers all the necessary aspects for the success of the application?

Yes, I think.

Evaluator execution:

Name	**Degree**	**Age**
Ghaddi Saadawi	Student	23

Scenario 2 (Evaluator 2):

At first, the evaluator downloads the application and opens Bluetooth. Customer opens the application about five meters from the beacon. If the customer tries to make an order, it won't made because the restaurant is not in range. She will receive this note on the screen: "You are not in Restaurant Range :(". The customers won't receive the welcome message ,restaurant status and offers.

Table 37:Test application (evaluator1)

Step	Test action	Expected result	Result(pass/ fail)
Functional Requirements			
1	Sign in /up	Customer can make a new account and sign in.	Pass
2	Try to make an order	Customer can't make order.	Pass
3	See message in the screen	You are not in Restaurant Range :(.	Pass
4	Customers won't receive the welcome message and offers.	Can't receive welcome message and offers.	Pass
Non- Functional Requirements			Good/Bad /Not bad
5	Interfaces will be clear and simple.	The look of interfaces is good. Not annoying and distracting	Good
6	The icon names will be easy to understand by different users	The terminology that used was easy and simple.	Good
7	The application will be interactive and has fast responses for different interaction	Fast responses	Bad
8	Easy for new users to learn to perform the task	Clear text icon and simple options.	Good

1- Do you have any suggestions for the development and improvement the program?
No

2- Can you tell us if there any difficulty in the program?
No

3- What are the most things that you like it in the program?

   Nice idea.

4- Do you think it covers all the necessary aspects for the success of the application?

   Approximately.

Evaluator execution:

Name	**Degree**	**Age**
Abrar Al-Harthi	Student	23

Cashier test:

Step	Test action	Expected result	Result(pass/ fail)
Functional Requirements			
1	Sign in	Cashier can sign in	Pass
2	Receive the order details	Cashier received the order details with ID	Pass
3	Close the order	Cashier can close the order	Pass
Non- Functional Requirements			Good/Bad /Not bad
4	Interfaces will be clear and simple.	The look of interfaces is good. Not annoying and distracting	Good
5	The icon names will be easy to understand by different users	The terminology that used was easy and simple.	Good
6	Easy for new users to learn to perform the task	Clear text icon and simple options.	Good

Evaluator execution:

Name	**Degree**	**Age**
Zonah Cashier	-	27

5.4 Challenges:

In implementation, many challenges have been faced during programming the application:

1- A lack of source code and tutorials for android application in connecting beacon with the database and with application. Most are for IOS.
2- The SDK for Estimote Beacons is complicated so beacon library was used instead of .

5.5 Conclusion:

Sensors and robots are the future of technology. We may wake up someday and find everything around us is controlled completely by sensors or done by robots. In this project, developers are just trying to use one sensor, called beacon sensor. Actually, it is easy to understand how to program it for programmers. It has a lot of applications, which programmers can use. A popular one is that it can be embedded in any place like library, university, mall and so many other places.

In this project, it was embedded in a restaurant to serve both customers and people working inside the restaurant. Data were collected and analyzed. Requirements were determined. An approach to the work was selected and a prototype of the project was decided.

Then, the practical side was started. Sometimes everything went right; other times it did not . Accordingly, there were some modifications was made to the previous phases because everything on paper is easy to do, but in reality challenges forced us to make changes.

The project has a future plan and enhancements which can be applied to existing ones.

At the end, this project was a good experience. It may be the basis of another creative application which makes life easier or solves a complex problem in this world.

Reference:

[1] Ibeaconinsider team ,2014. [Online]. Available : http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacons/.[Accessed :September 2015]

[2] Wikipedia ,' IBeacon', 2015. [Online]. Available:https://en.wikipedia.org/wiki/IBeacon .[Accessed: October 2015].

[3] The estimote team ,'Ibeacon ',2015 . [Online]. Avaliable: http://developer.estimote.com/ibeacon. [Accessed : October 2015].

[4] Bluetooth SIG team ,' Retail location based services ',2014.[Online].Available :https://www.bluetooth.com/marketing-branding/markets/retail-location-based-services.[Accessed: October 2015].

[5] Devika. Girish ,' How android is catching up with beacons ', 2015.[Online]. Avaliable: http://blog.beaconstac.com/2015/02/how-android-is-catching-up-with-beacons.[Accessed: October 2015].

[6] Mariia.Yemelianova,' Ibeacon restaurants ', 2014. [Online]. Avaliable: https://itechcraft.com/ibeacon_restaurants . [Accessed: September 2015].

[7] Natalia. Drozdiak, 'Will Apple's iBeacon Whet German Diners' App-etite? ',2014.**[Online].** Avaliable: http://blogs.wsj.com/digits/2014/07/28/will-apples-ibeacon-whet-german-diners-app-etite/.[Accessed: September 2015].

[8] ibeaconinsider team , ' Tests ibeacon in london in iBeacon', 2014.[Online] Avaliable: http://www.ibeacon.com/ bookatable-tests-ibeacon-in-london in iBeacon.[Accessed: September 2015].

[9] Jacky.Yap, ' Frunksingapore ibeacon ',2014.[Online]. Avaliable:https://vulcanpost.com/62231/frunksingapore-ibeacon. [Accessed: September 2015].

[10] Amir Ghahrai ,' Incremental model ', 2008 .[Online]. Avaliable: http://www.testingexcellence.com/incremental-model.[Accessed: September 2015].

[11] David G,' AltBeacon/android-beacon-library ',2015 .[Online]. Avaliable: https://github.com/AltBeacon/android-beacon-library. [Accessed:January 2016]

[12] Androidexample,' androidexample.com',2015.[Online]. Avaliable: http://androidexample.com/Download_Images_From_Web_And_Lazy_Load_In_ListView_-_Android_Example/index.php?view=article_discription&aid=112&aaid=134 [Accessed:Feb 2016]

[13] Androidexample,' androidexample.com/ ',2015.[Online]. Avaliable: http://androidexample.com/Create_Notification_Alert_Android_Example/index.php?view=article_discription&aid=102#. [Accessed:Feb 2016]

[14] Marc Gravell,' stackoverflow.com/ ',2016.[Online]. Avaliable: http://stackoverflow.com/questions/4457492/how-do-i-use-the-simple-http-client-in-android. [Accessed: Feb 2016]

Appendix A

Table 26: Details tasks

Roles	Responsibilities
Team leader: Sarah Saud Badri.	Who's responsibility is to divide the tasks between the group members, Manage and motivate: This includes managing the team activities in meeting, encouraging, helping in decision-making and also writing some parts of the report such as CHAPTER 1 introduction. ▪ Project Objectives. ▪ Background of Estimote beacons. ▪ The history of beacon. CHAPTER 2 ▪ Defining requirements and data requirements. ▪ Defining the use case for: 1. The admin and the customer sign in-up. 2. The admin adds updates and deletes special offers. 3. The cashier. CHAPTER 3 ▪ Project management strategies. CHAPTER 4 ▪ Class and sequence diagram with descriptions. ▪ Interface prototype. ▪ Activity diagram. CHAPTER 5 ▪ Work on android and report part.

Samah Aziz Althagafi	Writing some parts of the report include: CHAPTER 1 ▪ Background. ▪ The structure of the project. ▪ Background of Bluetooth. ▪ Background of android. CHAPTER 2 ▪ Use case: 1. The admin adds, updates and deletes customer. 2. Draw use cases. CHAPTER 3 ▪ Design constrains, hardware and software environment. CHAPTER 4 ▪ Interface description. CHAPTER 5 ▪ Work on design, database and report part.
Ashwaq Abdulmohsen Alsubhi	Writing some parts of the report include: CHAPTER 1 ▪ Project Statement. ▪ Some of previous system such as Bookatable, Frunk. CHAPTER 2 ▪ Non Functional requirements. ▪ Context diagram. ▪ DFD with description. ▪ Use case: 1. The customer receives special offers.

	2. The customer receives restaurant status. CHAPTER 4 ▪ Database tables and descriptions. ▪ Activity diagram description. CHAPTER 5 ▪ Work on android and database part.
Khulood Eidah Almalki	Writing some parts of the report include: CHAPTER 1 ▪ Introduction. ▪ Motivation. ▪ Project Scope. ▪ Some previous systems Latio, Mook Group. CHAPTER 2 ▪ Project stakeholders. ▪ Context diagram. ▪ DFD with description. ▪ Use case: 1. The admin adds table. 2. The admin adds, updates, deletes menu. 3. Tables Use case. CHAPTER 3 ▪ Reuse of existing software components. ▪ Future enhancements/plans. CHAPTER 5 Work on report and database part.

Rehab Ghazi Almejereshi	Writing some parts of the report include: CHAPTER 1 ▪ Project Contributions. CHAPTER 2 ▪ Functional requirements. ▪ Use case: 1. The customer makes order. 2. The customer displays the available table. 3. The customer displays the menu. 4. The beacon dispatch information. CHAPTER 3 ▪ Development method. CHAPTER 4 ▪ Major modules, Sub modules. CHAPTER 5 Work on report and database part.

Table 38: Gantt table for semester 1

Primary Column	Task description	Start	End	Duration
Initiating	choose the idea	01/09/15	07/09/15	6
	prepare for project proposal and delivered	07/09/15	02/10/15	24
Planning	Study the proposed idea and understand it	17/09/15	02/10/15	14
	Setting up the time and the activities	08/10/15	15/10/15	7
	Compare proposed system idea with the pervious system	21/10/15	25/10/15	4
Analysis	Defining The functional requirement	25/10/15	29/10/15	4
	Defining The non-functional requirement	25/10/15	30/10/15	5
	Draw Data flow diagrams	31/10/15	07/11/15	7
	use case daigram	01/11/15	02/11/15	1
	use case description	02/11/15	03/11/15	1
	Write the different solutions of the system problem with alterative	02/11/15	04/11/15	2
Design	Defining the software and hardware environment	05/11/15	06/11/15	1
	Search if we can Reuse other existing components	05/11/15	06/11/15	1
	Future enhancements/plans	06/11/15	08/11/15	2
	System Architecture	08/11/15	11/11/15	3
	Class diagram	11/11/15	13/11/15	2
	Sequence diagram	13/11/15	14/11/15	1
	Activity diagram	14/11/15	15/11/15	1
	Data base	15/11/15	18/11/15	3
	Interface description	19/11/15	22/11/15	3

Gantt chart (Semester 1):

Primary Column	Task description
Initiating	choose the idea
	prepare for project proposal and delivered
Planning	Study the proposed idea and understand it
	Setting up the time and the activities
	Compare proposed system idea with the pervious system
Analysis	Defining The functional requirement
	Defining The non-functional requirement
	Draw Data flow diagrams
	use case daigram
	use case description
	Write the different solutions of the system problem with alterative
Design	Defining the software and hardware environment
	Search if we can Reuse other existing components
	Future enhancements/plans
	System Architecture
	Class diagram
	Sequence diagram
	Activity diagram
	Data base
	Interface description

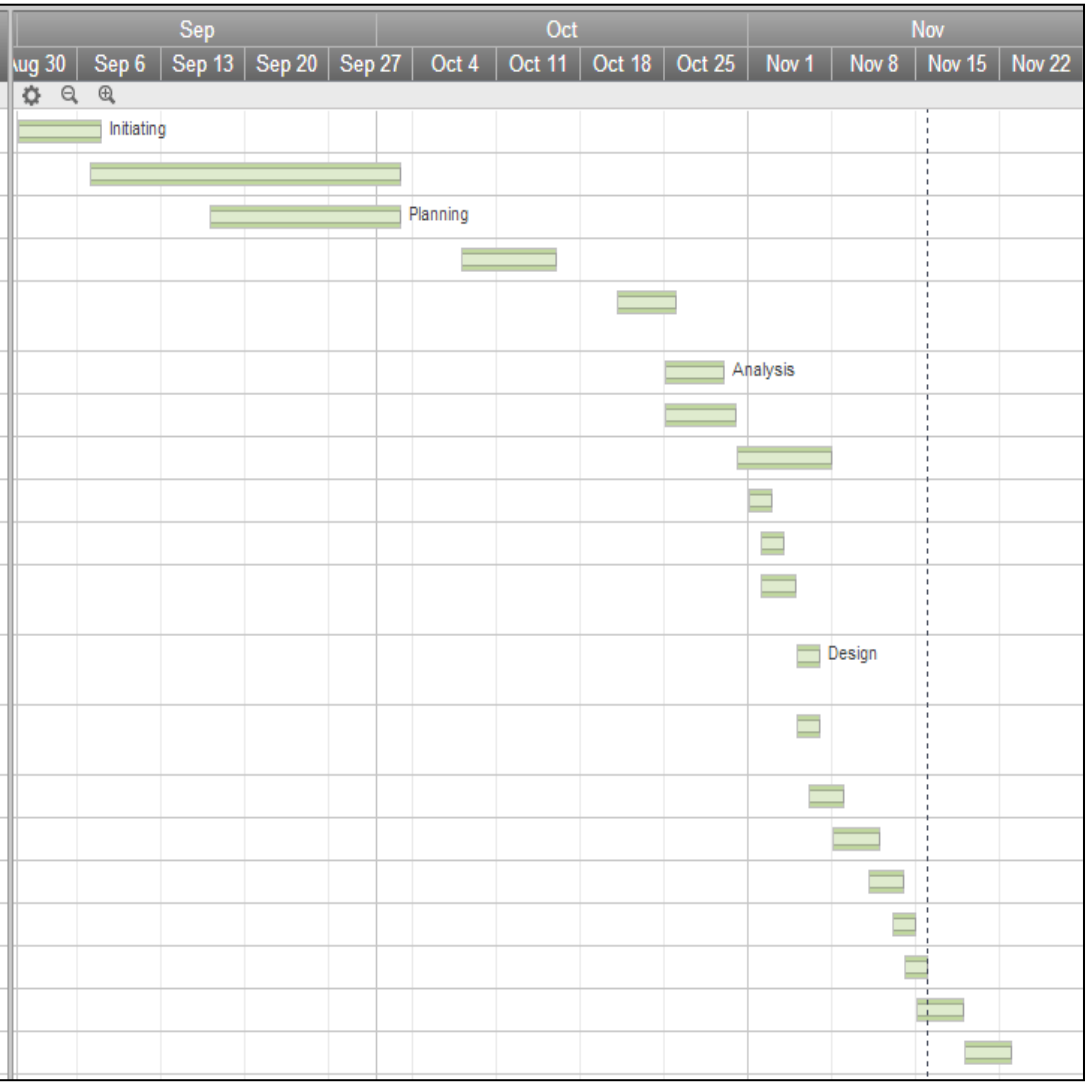


Figure 77: Gantt chart (semester 1)

Table 39: Gantt table for semester 2

Primary Column	Task Description	Start	End	Duration
Implementaion	Create DataBase	01/29/16	01/31/16	3
	Design admin side	02/01/16	02/05/16	5
	Host web and Connection	02/06/16	02/09/16	4
	Beacon Connection test	02/11/16	02/20/16	10
	Design application interface	02/15/16	02/21/16	7
	Sign in class	02/21/16	02/25/16	5
	Server Connection	02/25/16	02/29/16	3
	Sign up class	02/29/16	03/01/16	3
	Offers	03/29/16	03/02/16	4
	Status	03/03/16	03/06/16	4
	Notification	03/05/16	03/14/16	10
	Order part	03/17/16	04/01/16	13
	Calculate total cost	03/31/16	04/04/16	5
Test	Test and evaluation	04/10/16	04/19/16	9
	Editt Report	04/14/16	04/20/16	7

Gantt chart (Semester 2):

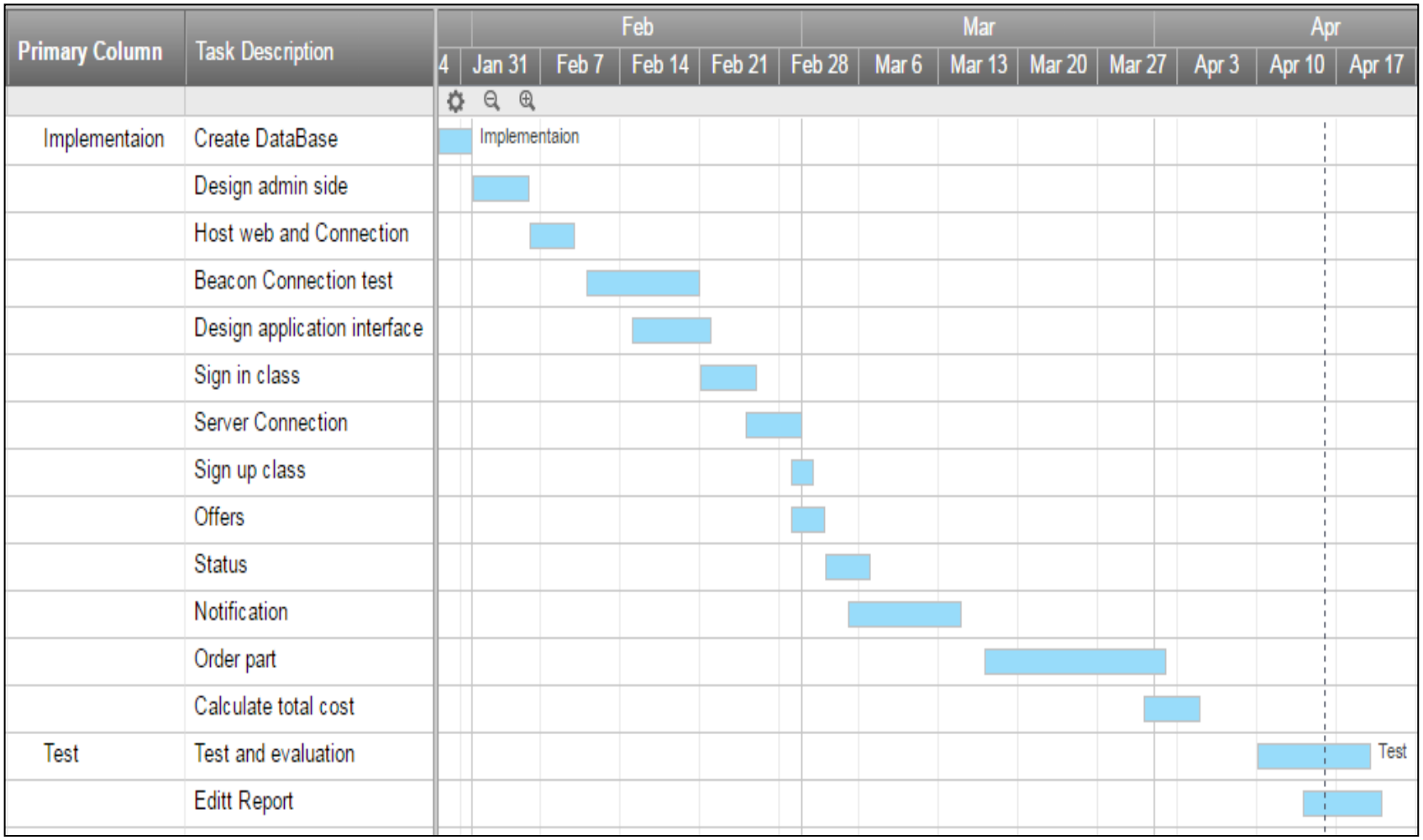


Figure 78:Gannt chart(semester 2)

Appendix B

Java Classes

Beacon class

```
1. package com.smart.restaurant;
2.
3.       import java.util.ArrayList;
4.       import java.util.Collection;
5.       import java.util.List;
6.       import org.altbeacon.beacon.Beacon;
7.       import org.altbeacon.beacon.BeaconConsumer;
8.       import org.altbeacon.beacon.BeaconManager;
9.       importorg.altbeacon.beacon.BeaconParser;
10.      importorg.altbeacon.beacon.Identifier;
11.      importorg.altbeacon.beacon.MonitorNotifier;
12.      import org.altbeacon.beacon.RangeNotifier;
13.      import org.altbeacon.beacon.Region;
14.      import android.app.Activity;
15.      import android.app.Notification;
16.      import android.app.NotificationManager;
17.      import android.app.PendingIntent;
18.      import android.app.Service;
19.      import android.content.Context;
20.      import android.content.Intent;
21.      import android.os.AsyncTask;
22.      import android.os.Bundle;
23.      import android.os.IBinder;
24.      import android.os.RemoteException;
25.      import android.text.TextUtils;
26.      import android.util.Log;
27.      import android.widget.Toast;
28.
29.       public class BeaconBGService extends Service implemes BeaconConsum
   er{
30.
31.
32.          private Region region;
33.          private BeaconManager beaconManager;
34.          public static Activity activity;
35.          ArrayList<String> beaconIds =new ArrayList<String>();
36.          ArrayList<String> msg = new ArrayList<String>();
37.
     public static String BEACONID="0xB9407F30F5F8466EAFF925556B57FE6D";//
   the id of our beacon called blueberry
38.          ArrayList<String> validIdentifiers = new ArrayList<String>();
39.          SharePrefs sp;
40.          private ConnectionClass cc;
41.          public static int ENTRY_THRESHOLD = 1;
42.          public static int EXIT_THRESHOLD = 3;
43.          @Override
44.          public void onCreate() {
45.           super.onCreate();
46.           region = new Region(BEACONID, null, null, null);
47.           beaconManager = BeaconManager.getInstanceForApplication(this);
48.      BeaconParser beaconParser    = new BeaconParser();
49.              beaconParser.setBeaconLayout("m:2-3=0215,i:4-19,i:20-
   21,i:22-23,p:24-24");
50.         beaconManager.getBeaconParsers().add(beaconParser);
51.              beaconManager.bind(this);
```

```
52.
53.
54.             validIdentifiers.clear();
55.             validIdentifiers.add("e2c56db5-dffb-48d2-b060-
   d0f5a71096e0".toUpperCase());
56.             validIdentifiers.add("2f234454-cf6d-4a0f-adf2-
   f4911ba9ffa6".toUpperCase());
57.             validIdentifiers.add("B9407F30-F5F8-466E-AFF9-
   25556B57FE6D".toUpperCase());
58.
59.             sp   = new SharePrefs(getApplicationContext());
60.
61.         }
62.
63.         @Override
64.     public int onStartCommand(Intent intent, int flags, int startId) {
65.             return START_STICKY;
66.         }
67.
68.         @Override
69.      public void onBeaconServiceConnect() {
70.       beaconManager.setRangeNotifier(new RangeNotifier() {
71.             @Override
72.
    public void didRangeBeaconsInRegion(final Collection<Beacon> beacons,
   Region region) {
73.           for(int i=0;i<beacons.size();i++){
74.           final Beacon beacon = beacons.iterator().next();
75.                   String idVal = beacon.getId1()+"";
76.
77.        if(validIdentifiers.contains(idVal.toUpperCase())){
78.          float distance = (float)beacon.getDistance();
79.
   if(sp.getLastDistance() > ENTRY_THRESHOLD && distance <ENTRY_THRESHOLD)
80.     { // assuming the consumer is entering the restaurant range
81.        sp.setLastDistance((float)beacon.getDistance());
82.
83.                   if(activity == null)
84.                    {
85.                      Intent intent =
86.     newIntent(getApplicationContext(),MainActivity.class);
87.           intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
88.           intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
89.           intent.putExtra("welcome", true);
90.           startActivity(intent);}
91.
92.                  welcome();
93.                  offers();
94.                  status();}
95.     else if(distance >= EXIT_THRESHOLD &&
   sp.getLastDistance() < EXIT_THRESHOLD)
96.                              {
97.       sp.setLastDistance((float)beacon.getDistance());
98.
99.         showNotification("Thank you","Will look forward to serve you
   again :)"); }
100.
101.    }
102.    }}}});
```

```
103.     try
104.      {
105.        beaconManager.startRangingBeaconsInRegion(region);
106.             } catch (RemoteException e) {
107.                 e.printStackTrace();
108.             }
109.     }
110.
111.         void welcome()
112.         {
113.         cc  = new ConnectionClass(this);
114.         new AsyncTask<Void, Void, Void>()
115.         {
116.         private String helloMsg;
117.
118.         @Override
119.         protected Void doInBackground(Void... params) {
120.      helloMsg= cc.connectToServerFunc("f=getHelloMsg");// f is the name
   of the function that return getHelloMsg function
121.             return null;}
122.
123.        protected void onPostExecute(Void result) {
124.               if(helloMsg != null)
125.                  showNotification("Welcome",helloMsg);
126.               else
127.                  showNotification("Welcome","Welcome to Smart
   Restaurant");};
128.         }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR); // this
   for parallel excution
129.         }
130.
131.         void status()
132.         {
133.             cc  = new ConnectionClass(this);
134.             new AsyncTask<Void, Void, Void>()
135.             {
136.             private String statusMsg;
137.
138.         @Override
139.         protected Void doInBackground(Void... params) {
140.     statusMsg = cc.connectToServerFunc("f=getFreeTable1");
141.                     return null;}
142.
143.        protected void onPostExecute(Void result) {
144.          if(statusMsg != null)
145.          showNotification("Resturant status ",statusMsg);
146.          else
147.          showNotification("Welcome","Welcome to Smart Restaurant");};
148.
149.             }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
150.         }
151.
152.
153.         void offers()
154.         {
155.         cc = new ConnectionClass(this); // this class that deals with
   internet to get the data
156.        new AsyncTask<Void, Void, Void>()
157.             {
158.        private String offersMsg;
```

```
159.
160.             @Override
161.         protected Void doInBackground(Void... params){
162.             offersMsg    = cc.connectToServerFunc("f=getoffersMsg");
163.                  return null;}
164.
165.         protected void onPostExecute(Void result) {
166.           if(offersMsg != null)
167.             showNotification("new offers",offersMsg);
168.            else
169.             showNotification("Welcome","Welcome to Smart Restaurant");
170.                  };
171.
172.              }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
173.          }
174.
175.          @Override
176.          public IBinder onBind(Intent arg0) {
177.              // TODO Auto-generated method stub
178.              return null;
179.          }
180.
181.
182.          private void showNotification(String title,String eventtext) {
183.              // Set the icon and time, (spoon) is the name of the icon
184.
    Notification notification = newNotification(R.drawable.spoon,eventtext,
   System.currentTimeMillis());
185.              PendingIntent contentIntent =
186.     PendingIntent.getActivity(this, 0,newIntent(), 0);
187.       // Set the title and its text to be displayed .
188.         notification.setLatestEventInfo(this, title, eventtext,
   contentIntent);
189.       NotificationManager notificationManager =
190.     (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE)
   ;
191.       notificationManager.notify(title, 0, notification);// Sendnotifica
   tion to customer smartphone.
192.          }
193.     }
```

Connection Class

```
1. package com.smart.restaurant;
2.       import java.io.BufferedReader;
3.       import java.io.InputStream;
4.       import java.io.InputStreamReader;
5.
6.       import org.apache.http.HttpResponse;
7.       import org.apache.http.client.HttpClient;
8.       import org.apache.http.client.methods.HttpPost;
9.       importorg.apache.http.impl.client.DefaultHttpClient;
10.
11.      import android.content.Context;
12.      import android.util.Log;
13.
14.      public class ConnectionClass {
15.          public static String BASE_URL="http://ideasareus.website/";
```

```
16.       public static String SERVER_URL=BASE_URL+"controller.php";
17.          Context context;
18.
19.       public ConnectionClass(Context c) {
20.              context  = c; }
21.
22.       public String connectToServerFunc(String u) {
23.           try
24.           {
25.             String urlLink=SERVER_URL+"?"+u;
26.             HttpClient client = new DefaultHttpClient();
27.             HttpPost http_get = new HttpPost(urlLink);
28.             HttpResponse responses;
29.             responses = client.execute(http_get);
30.                  if (responses != null)
31.                  {
32.          InputStream in = responses.getEntity().getContent();
33.          String a = convertStreamToString(in);
34.                  return a; }}
35.             catch (Exception e)
36.                {
37.                    e.printStackTrace();
38.                }
39.                return null;
40.            }
41.
42.            StringconvertStreamToString(InputStreamis) {
43.
44.
    BufferedReader reader =new BufferedReader(new InputStreamReader(is));
45.           StringBuilder sb = new StringBuilder();
46.           String line = null;
47.                try
48.                {
49.                while ((line = reader.readLine()) != null)
50.                    {
51.                        sb.append(line);
52.                    } }
53.                catch (Exception e){
54.                   }
55.                finally
56.                {
57.                    try
58.                    {is.close();}
59.                    catch (Exception e)
60.                    {
61.                    }}
62.                return sb.toString();
63.            }}
```

Order class

```
1. package com.smart.restaurant;
2.        import java.util.ArrayList;
3.        import org.json.JSONArray;
4.        import org.json.JSONObject;
5.        import android.app.AlertDialog;
```

```
6.      import android.content.DialogInterface;
7.      import android.content.Intent;
8.      import android.net.Uri;
9.      import android.os.AsyncTask;
10.     import android.os.Bundle;
11.     import android.text.Html;
12.     import android.util.Log;
13.     import android.view.View;
14.     import android.widget.Button;
15.     import android.widget.ListView;
16.     import android.widget.TextView;
17.     import android.widget.Toast;
18.
19.     public class ListOrder extendsCustomActivity{
20.
21.         public static CustomActivity activity;
22.         SharePrefs sp;
23.         ListView lv;
24.         ArrayList<JSONObject> list = new ArrayList<JSONObject>();
25.         private ListOrderAdapter adapter;
26.         float total =0;
27.         private TextView ttl;
28.         ConnectionClass cc;
29.         @Override
30.         protected void onCreate(Bundle savedInstanceState) {
31.             super.onCreate(savedInstanceState);
32.     activity = this;
33.             sp          = new SharePrefs(this);
34.             cc          = new ConnectionClass(this);
35.
36.      if(sp.getLastDistance() <= BeaconBGService.ENTRY_THRESHOLD)
37.             {
38.                 showOrders();
39.             }
40.             else
41.             {
42.                 showNotInRange();
43.             }
44.         }
45.
46.         void showOrders()
47.         {
48.             setContentView(R.layout.show_orders);
49.
50.             lv  = (ListView)findViewById(R.id.lv);
51.             ttl = (TextView)findViewById(R.id.textView201);
52.             list.clear();
53.             int j =0;
54.             for(int i=0;i<MainActivity.idlist.size();i++)
55.             {
56.                 try
57.                 {
58.                    int idval=MainActivity.idlist.get(i);
59.                    if(MainActivity.addItem.get(idval) > 0) // if items
   quantity lager than 0
60.                     {
61.     list.add(MainActivity.itemDetails.get(idval));
62.     float v = MainActivity.addItem.get(idval) *Float.parseFloat(list.ge
   t(j).getString("price"));//total calculation
```

```
                    j++;
                    total += v; }
                   }catch(Exception e){
                       e.printStackTrace();
                   }
               }

               adapter = new ListOrderAdapter(this, list);
               lv.setAdapter(adapter);
               ttl.setText("Total SR"+total);
           }
           void showNotInRange()
           {
               setContentView(R.layout.not_in_range);
           }
          public void okay(final View v)//after pressing OK this dialog  will be shown
           {
       AlertDialog.Builder dialog = new AlertDialog.Builder(ListOrder.this
   );
            dialog.setTitle("Order Type");//dialog to choose order type
          dialog.setMessage("I would like to place this Order as");
          dialog.setPositiveButton("Take-
   Away", newDialogInterface.OnClickListener() {

       public void onClick(DialogInterface dialoginterface, int j) {
             placeOrder(1,((Button)v));
                       }
                   });
       dialog.setNegativeButton("Sit-In",
        new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialoginterface, int i) {
              placeOrder(2,((Button)v));
                   }
                   });
               dialog.show();
           }

        public void placeOrder(final int type,final Button v)
           {
               v.setEnabled(false);
               JSONArray jarray = new JSONArray();

               for(int i=0;i<MainActivity.idlist.size();i++)
               {
                   try
                   {
                       JSONObject jobj  = new JSONObject();
                       int id = MainActivity.idlist.get(i);
                       if(MainActivity.addItem.get(id) > 0)//put items of the
   order into Json array
                       {
                jobj.put("id",id);
                jobj.put("qty",MainActivity.addItem.get(id));
                jarray.put(jobj);}
                   }catch(Exception e){}
               }
          final String items = Uri.encode(jarray.toString());//change Json
   array into proper form to send it to the database


```

```
119.     new AsyncTask<Void, Void,String>(){
120.      @Override
121.      protected String doInBackground(Void... params) {
122.
   return cc.connectToServerFunc("f=placeOrder1&items="+items+"&uid="+sp.get
   Uid()+"&type="+type);//send  order to  database
123.     //order pass first by controller placeOrder1 function
124.                 }
125.
126.     protected void onPostExecute(String result) {
127.            try
128.            {
129.            JSONObject jobj = new JSONObject(result);
130.
131.                        if(jobj.getString("msg").equals("1"))//msg came from
   placeOrder1 as response after check the order validation
132.                 {
133.     MainActivity.addItem.clear();
134.        Intent intent =
135.     new Intent(ListOrder.this,AfterOrder.class);
136.         intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
137.         intent.putExtra("data", jobj.getString("data"));
138.         startActivity(intent);
139.         ListOrder.this.finish();}
140.
141.      else if(jobj.getString("msg").equals("2")){
142.        Toast.makeText(ListOrder.this, "There is no available
   table", 2000).show();}
143.
144.               }catch(Exception e){
145.                 e.printStackTrace();
146.               }
147.                         v.setText("Confirm");
148.                         v.setEnabled(true);
149.
150.                 };
151.
152.      }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
153.         }
154.
155.         public void cancel(View v)
156.         {
157.             MainActivity.addItem.clear();
158.     Intent intent = new Intent(this,Categories.class);
159.          intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
160.          startActivity(intent);
161.         }}
```

Offers Class

```
1. package com.smart.restaurant;
2. import java.util.ArrayList;
3. import org.json.JSONArray;
4. import org.json.JSONObject;
5. import android.os.AsyncTask;
6. import android.os.Bundle;
7. import android.widget.Adapter;
8. import android.widget.ListView;
```

```
9.  public class Offers extendsCustomActivity{

   private ConnectionClass cc;

10.      ArrayList<JSONObject> list= new ArrayList<JSONObject>();
11.      ListView lv;
12.      private OffersAdapteradapter;
13.      @Override
14.      protected voidonCreate(Bundle savedInstanceState) {
15.                 super.onCreate(savedInstanceState);
16.                 setContentView(R.layout.offers);
17.
18.                  lv  = (ListView)findViewById(R.id.lv);
19.
20.                  cc  = new ConnectionClass(this);
21.                  list.clear();
22.
23.                  adapter = new OffersAdapter(this, list);
24.                  lv.setAdapter(adapter);
25.                  intialize();
26.          }
27.
28.          void intialize()
29.          {
30.              new AsyncTask<Void, Void, Void>()
31.              {
32.             @Override
33.             protected Void doInBackground(Void... params) {
34.                  try
35.                   {
36.      String result = cc.connectToServerFunc("f=getOffers");
37.      JSONArray jarray = new JSONArray(result);
38.              for(int i=0;i<jarray.length();i++){
39.                   list.add(jarray.getJSONObject(i)); }
40.                     }catch(Exceptione){
41.                         e.printStackTrace();
42.                     }
43.                     return null;
44.                 }
45.                 protected voidonPostExecute(Void result) {
46.                    adapter.notifyDataSetChanged();
47.                 };
48.        }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
49.          }}
```

Status Class

```
1. package com.smart.restaurant;
2. import org.json.JSONObject;
3. import android.os.AsyncTask;
4. import android.os.Bundle;
5. import android.widget.TextView;
6.
7. public class Status extends CustomActivity{
8.      TextView tv;
9.      ConnectionClass cc;
10.       @Override
11.       protected void onCreate(Bundle savedInstanceState) {
12.             super.onCreate(savedInstanceState);
```

```
13.              setContentView(R.layout.status);
14.              cc      = new ConnectionClass(this);
15.         try
16.           {
17.          tv  = (TextView)findViewById(R.id.textView21);
18.              }catch(Exception e){
19.                  e.printStackTrace();}
20.
21.      new AsyncTask<Void, Void, String>()
22.              {
23.           @Override
24.           protected String doInBackground(Void... params) {
25.
   String result = cc.connectToServerFunc("f=getFreeTable2");
26.                      return result;}
27.
28.               protected void onPostExecute(String result) {
29.             try{
30.      JSONObject jobj = new JSONObject(result);
31.      int i   = Integer.parseInt(jobj.getString("count"));
32.                  if(i > 0 )
33.                  tv.setText("YOU ARE LUKY :) !!\nWe have "+i+" tables
   available in the Restaurant :)");
34.                   else
35.                  tv.setText("We are sorry :( All the tables are busy
   now.");
36.      }catch(Exception e){}
37.                  };
38.
39.       }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);}}
```

Item class

```
1. package com.smart.restaurant;
2.
3.       import java.util.ArrayList;
4.       import org.json.JSONArray;
5.       import org.json.JSONObject;
6.       import android.content.Context;
7.       import android.content.Intent;
8.       import android.os.AsyncTask;
9.       import android.os.Bundle;
10.      import android.view.View;
11.      import android.widget.LinearLayout;
12.      import android.widget.ListView;
13.      import android.widget.TextView;
14.      import android.widget.Toast;
15.
16.      public class Items extends CustomActivity{
17.          /*Items class:
18.           class for showing the items "after retrieve them from items
   table in database" as a listView.
19.           corresponding to items.xml layout.
20.               */
21.              private ConnectionClass cc;
22.              ArrayList<JSONObject> list = new ArrayList<JSONObject>();
```

```
23.
24.             ListView lv;
25.             private  ItemsAdapter adapter;
26.             private String catname;
27.             TextView heading;
28.             static TextView orders;
29.             public static CustomActivity activity;
30.             LinearLayout l2;
31.
32.
33.             @Override
34.     protected void onCreate(Bundle savedInstanceState) {
35.                 super.onCreate(savedInstanceState);
36.                 activity  = this;
37.                 setContentView(R.layout.items);
38.
39.        lv  = (ListView)findViewById(R.id.lv);
40.        l2  = (LinearLayout)findViewById(R.id.l2);
41.        heading = (TextView)findViewById(R.id.textView21);
42.        orders = (TextView)findViewById(R.id.textView1);
43.
44.                 cc    = new ConnectionClass(this);
45.                 list.clear();
46.
47.                 adapter = new ItemsAdapter(this, list);
48.                 lv.setAdapter(adapter);
49.
50.                 try
51.                 {
52.     String data = getIntent().getExtras().getString("data");
53.      JSONObject jobj = new JSONObject(data);
54.      catname          = jobj.getString("cat");
55.
56.                     heading.setText("Menu : "+catname);
57.                 }catch(Exception e){
58.                     e.printStackTrace();
59.                     heading.setText("Items");
60.                 }
61.
62.                 if(catname != null)
63.                     intialize();
64.         }
65.
66.
67.             @Override
68.             protected void onResume() { super.onResume();}
69.
70.          void intialize()
71.          {
72.              new AsyncTask<Void, Void, Void>()
73.              {
74.                  @Override
75.             protected Void doInBackground(Void... params) {
76.
77.                     try
78.                     {
79.     String result =cc.connectToServerFunc("f=getItems&cat
80.     "+catname);
81.        JSONArray jarray = new JSONArray(result);
82.            for(int i=0;i<jarray.length();i++)
```

```
83.             {
84.                 list.add(jarray.getJSONObject(i));
85.                 int idval =Integer.parseInt(list.get(i).ge
86.                 String("id"));
87.
88.             if(!MainActivity.idlist.contains(idval))
89.               MainActivity.idlist.add(idval);
90.             if(MainActivity.itemDetails.get(idval) == null)
91.               MainActivity.itemDetails.put(idval,list.get(
92.     ));
93.     if(MainActivity.addItem.get(idval) == null)
94.               MainActivity.addItem.put(idval, 0);
95.                         }
96.                     }catch(Exception e){
97.                         e.printStackTrace();
98.                     }
99.                     return null;
100.                }
101.
102.               protected void onPostExecute(Void result) {
103.                     adapter.notifyDataSetChanged();
104.                 };
105.
106.      }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
107.         }
108.
109.         public static void addItems(int idval,final int q
110.         y,final String name)
111.         {
112.
113.             MainActivity.addItem.put(idval,qty);
114.             setItemCounts(idval,qty,name);
115.             Context context = null ;
116.             CharSequence text = "Items Successfully added
117.             ";
118.             int duration = Toast.LENGTH_SHORT;
119.
120.             final Toast
121.             toast = Toast.makeText(Items.activity,
122.     text, duration);
123.            toast.show();
124.         }
125.
126.         public static void setItemCounts(int idval,final
127.         nt qty,final Stringname)
128.         {
129.             try
130.             {
131.                 Items.activity.runOnUiThread(new Runnable
132.           ) {
133.
134.     @Override
135.     public void run() {
136.     Toast.makeText(Items.activity, name+" :
137.     "+qty,80).show();
138.                     }
139.                 });
140.             }catch(Exception e){
141.                 e.printStackTrace();
```

```
142.             }
143.
144.             if(MainActivity.addItem.size() == 0)
145.             {
146.                 Items.activity.finish();
147.             }
148.         }
149.
150.         public void okay(View v)
151.         {
152.     Intent intent = new Intent(this,ListOrder.class);
153.     intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
154.             startActivity(intent);
155.         }
156.
157.         public void back(View v)
158.         {
159.             this.finish();
160.         }
161.     }
162.
```

Items_Adapter

```
1. package com.smart.restaurant;
2.
3.       import java.util.ArrayList;
4.       import org.json.JSONObject;
5.       import android.app.Activity;
6.       import android.content.Context;
7.       import android.view.LayoutInflater;
8.       import android.view.View;
9.       import android.view.ViewGroup;
10.      importandroid.widget.BaseAdapter;
11.      importandroid.widget.ImageView;
12.      importandroid.widget.LinearLayout;
13.      import android.widget.TextView;
14.
15.
16.
17.      public class ItemsAdapterextends BaseAdapter {
18.          /*
19.          ItemsAdapter :
20.      in this class we display the list of items into the listview ,by
   converting item_row.XML into a view and inflate it using LayoutInflater.
21.
22.          */
23.      private Activity mContext;
24.      ArrayList<JSONObject> list;
25.             View cell;
26.             int minus=0;
27.             LazyImageLoader imageLoader;
28.      public ItemsAdapter(Activity
29.      c,ArrayList<JSONObject> l) {
30.      mContext = c;
31.      list     = l;
32.      imageLoader =
```

```
33.      new LazyImageLoader(mContext);//reference [12]
34.             }
35.
36.             public int getCount() {
37.                int len = list.size()/2;
38.                if((list.size()%2) > 0)
39.                    len++;
40.                return len;
41.             }
42.
43.      public Object getItem(intposition) {
44.                return null;
45.             }
46.
47.             public longgetItemId(int position) {
48.                return 0;
49.             }
50.
51.             static class ViewHolder
52.             {
53.
54.                 TextView name1,name2;
55.      ImageView img1,img2;
56.                 LinearLayout parent1,parent2;
57.             }
58.
59.             public View getView(final int position, View
60.      convertView, ViewGroup parent) {
61.      LayoutInflater
62.      inflater =  (LayoutInflater)mContext.getSystemService
63.      Context.LAYOUT_INFLATER_SERVICE);
64.                   ViewHolder holder;
65.       if(convertView == null)
66.          {
67.      holder=new ViewHolder();
68.      convertView= inflater.inflate(R.layout.item_row,null);
69.
70.      holder.name1=
71.      (TextView)convertView.findViewById(R.id.title1);
72.      holder.img1=
73.      (ImageView)convertView.findViewById(R.id.imageView1);
74.      holder.parent1=
75.      (LinearLayout)convertView.findViewById(R.id.item1);
76.      holder.name2=
77.      (TextView)convertView.findViewById(R.id.title2);
78.      holder.img2=
79.      (ImageView)convertView.findViewById(R.id.imageView2);
80.      holder.parent2=
81.      (LinearLayout)convertView.findViewById(R.id.item2);
82.
83.             convertView.setTag(holder);
84.
85.                     }
86.           else
87.           {
88.             holder = (ViewHolder)convertView.getTag();
89.           }
90.
91.      final int tmpPosition = position * 2;
```

```
92.                     try
93.                     {
94.       holder.parent1.setVisibility(View.VISIBLE);
95.       final JSONObject jobj    =list.get(tmpPosition);
96.       imageLoader.DisplayImage(jobj.getString("image"),
97.       holder.img1);
98.       holder.name1.setText(jobj.getString("name")+"
99.      "+jobj.getString("price")+"  SR");
100.
101.     holder.name1.setOnClickListener(new
102.     View.OnClickListenr() {
103.          @Override
104.          public void onClick(View arg0) {
105.             try
106.             {
107.         int idval =Integer.parseInt(jobj.getString("id"));
108.     Items.addItems(idval,MainActivity.addItem.get(idval)+
109.     ,jobj.getString("name"));//change quantity by adding 1
110.     to previous quantity
111.         }catch(Exception e){
112.          e.printStackTrace();
113.          }
114.          }
115.                               });
116.
117.        } catch (Exception e) {
118.          e.printStackTrace();
119.          holder.parent1.setVisibility(View.INVISIBLE);
120.            }
121.
122.          try
123.          {
124.        holder.parent2.setVisibility(View.VISIBLE);
125.        final JSONObject jobj    =list.get(tmpPosition+1);
126.        imageLoader.DisplayImage(jobj.getString("image"),
127.        holder.img2);
128.        holder.name2.setText(jobj.getString("name")+"
129.        "+jobj.getString("price")+"  SR");
130.
131.        holder.name2.setOnClickListener(new View.OnClickLis
132.        ener() {
133.            @Override
134.            public void onClick(View arg0) {
135.             try
136.              {
137.         int idval =Integer.parseInt(jobj.getString("id"));
138.         Items.addItems(idval,MainActivity.addItem.get(idv
139.         l)+1,jobj.getString("name"));
140.            }catch(Exception e){
141.              e.printStackTrace();
142.                }
143.                }
144.                });
145.     } catch (Exception e) {
146.        e.printStackTrace();
147.        holder.parent2.setVisibility(View.INVISIBLE);
148.                    }
149.        return convertView;
150.                    }
151.     }
```

PHP classes

cashier-home.php

```
1. <?php
2. require_once("top.php");
3. if($_REQUEST['close'] == "1"){//if cashier click on
   close button this code will be executed
4.     $query = "UPDATE orders SET closed='1' WHERE id='$_REQUEST[id]'";
5.     mysql_query($query);
6. ?>
7. <html>
8. <head>
9.     <TITLE>Smart Restaurant ::Cashier</TITLE>
10.         <?php require_once("header.php");?>
11.     </head>
12.     <body>
13.     <div class="wrapper">
14.         <?php require_once("logo_menu.php");?>
15.         <p> </p>
16.         <div class="content-box">
17.     <table class='items-table' cellpadding="0" cellspacing="0">
18.                     <TR>
19.                         <TH>Sl.</TH>
20.                         <TH>Order ID</TH>
21.                         <TH>Items</TH>
22.                         <TH>Customer</TH>
23.                         <TH>Order Detail</TH>
24.                         <TH>Total</TH>
25.                         <TH>Close </TH>
26.                     </TR>
27.
28.     <?php
29.         $i   = 1;
30.         $query  = "SELECT * FROM orders WHERE closed='0' ORDER BY
   id ASC";
31.        $res     = mysql_query($query) or die(mysql_error());
32.             while($data = mysql_fetch_assoc($res))
33.                         {
34.                   $total   = 0;
35.                   $edit    = "<a href='cashier-
   home.php?id=$data[id]&close=1'
   class='edit' onclick='javascript:if(confirm(\"Close transaction ?\"))
   return true;else return false;'>&#9997;</a>";
36.                           $action ="<span>$edit </span>";
37.
38.             $itemTables = "<table border='0'>";
39.             $jsonDecod  =json_decode($data['items'],true);
40.                   for($k=0;$k<count($jsonDecod);$k++)
41.                       {
42.             $id = $jsonDecod[$k]['id'];
43.             $query  = "SELECT * FROM items WHERE id='$id' ORDER BY
   id ASC";
44.             $res2    = mysql_query($query);
45.                 while($data2 = mysql_fetch_assoc($res2))
46.                             {
```

```
        $prc = floatval($data2['price']) *intval($jsonDecod[$k]['qty']);
          $itemTables.="<tr>";
          $itemTables.="<td>".$data2['name']."</td>";
          $itemTables.="<td>".$data2['cat']."</td>";
          $itemTables.="<td>".$data2['price']." x ".CURRENCY."
  ".$jsonDecod[$k]['qty']."</td>";
          $itemTables.="<td>=</td>";
          $itemTables.="<td>".CURRENCY." ".$prc."</td>";
          $itemTables.="</tr>";//$total+=$prc;
                                    }
                                }

          $itemTables .="</table>";
          $orderDetail = "Take_Away";
            if($data['type'] == "2")
              {
           $orderDetail = "Sit_In<br>Table No.".$data['tablenumber'];
                                    }
     $total =  $data['total'];
     $query  = "SELECT * FROM customers WHERE id='$data[customerid]'";
            $res2    = mysql_query($query);
            $data2 = mysql_fetch_assoc($res2);
            $custom = $data2['name'];
                  if(trim($custom) == "")
                     $custom = "Uknown";
                      echo "<TR>
                      <TD>$i</TD>
                      <TD>$data[id]</TD>
                      <TD>$itemTables</TD>
                      <TD>$custom</TD>
                      <TD>$orderDetail</TD>
                      <td>".CURRENCY." ".$total."</td>
                      <TD>$action</TD>
                      </TR>";
                       $i++;}
                     ?>
                  </table>
          </div>
     </div>
     </body>
     <?php require_once("footer.php");?>
     </html>
```

Controller.php

```
<?php
define("SERVER_URL","http://108.167.189.112/~smart/");
$controller     = new Controller();
$function    = $_REQUEST['f'];
if($function != "")
    $controller->$function();

class Controller
{
         public function __construct()
         {
             require_once("config.php");
```

```
13.             }
14.
15.      function getAllItems()
16.          {
17.      $array       = array();
18.      $query       = "SELECT * FROM items ORDER BY name ASC";
19.      $res         = mysql_query($query);
20.         while($data      = mysql_fetch_assoc($res))
21.             {
22.                 $data['image']       =
   str_replace("images/items/","",$data['image']);
23.                 $data['image']       =
   SERVER_URL."images/items/".rawurlencode($data['image']);
24.                 $array[count($array)] = $data;
25.             }
26.
27.             $query       = "SELECT * FROM tables_count";
28.             $res         = mysql_query($query);
29.             $data        = mysql_fetch_assoc($res);
30.
31.             $element             = array();
32.             $element['data']         = $array;
33.             $element['tables']       = $data['count'];
34.             echo json_encode($element);
35.          }
36.
37.
38.       function allFreeTables()
39.          {
40.             $customerid = $_REQUEST['customerid'];
41.             $query       = "SELECT * FROM tables_count";
42.             $res         = mysql_query($query);
43.             $data        = mysql_fetch_assoc($res);
44.             $count       = $data['count'];
45.             $array       = array();
46.             for($i=1;$i<=intval($count);$i++)
47.             {
48.                 $query       = "SELECT * FROM orders WHERE processed
   ='0' AND `tablenumber`='$i' AND `customerid`!='$customerid'";
49.                 $res         = mysql_query($query);
50.                 $isBusy      = mysql_num_rows($res);
51.
52.                 if(!$isBusy)
53.                     $array[count($array)] = $i+"";
54.             }
55.
56.             echo implode(",",$array);
57.          }
58.
59.
60.      function placeOrder1()
61.      {
62.      $type = $_REQUEST['type'];// 1 = Take-Away  2= Sit-In
63.             $items       = $_REQUEST['items'];
64.             $customerid = intval($_REQUEST['uid']);
65.             $jsonDecode = json_decode($items,true);
66.              $total=0;
67.             if(!count($jsonDecode))return;
68.       for($i=0;$i<count($jsonDecode);$i++){
```

```
69.             $id=$jsonDecode[$i]['id'];
70.             $qty=$jsonDecode[$i]['qty'];
71.
72.         $query1 = "SELECT * FROM items where id='$id'";
73.         $res1       = mysql_query($query1);
74.         $data1      = mysql_fetch_assoc($res1);
75.         $price=$data1['price'];
76.         $total+=$price*$qty;}
77.
78.
79.             $t = json_decode($this->appFreeTable(),true);
80.             $t = intval($t['num']);
81.             if(!$t && $type == "2")
82.             {$element['msg']    = "2";
83.                 echo json_encode($element);
84.                 exit();}
85.
86.             if($type == "2")
87.                 $query      = "INSERT INTO `orders` (`customerid`,
    `items`, `tablenumber`, `type`,`total`) VALUES ('$customerid', '$items',
    '$t', '$type','$total')";
88.             else
89.                 $query      = "INSERT INTO `orders` (`customerid`,
    `items`,`type`,`tablenumber`,`total`) VALUES ('$customerid', '$items',
    '$type','-1','$total')";
90.
91.             $element    = array();
92.             $exe        = mysql_query($query) or die(mysql_error());
93.             $orderid    = mysql_insert_id();
94.             if($orderid > 0)
95.             {$element['msg']    = "1";
96.
97.
98.             if($type == "2")
99.              $element['data']    = "<b>Thank you for
    Ordering<b><br>Your Order ID : $orderid<br>Table number : $t";
100.            else
101.               $element['data']    = "<b>Thank you for
    Ordering<b><br>Your Order ID : $orderid<br>Please wait and we will call
    you once the food is prepared :)";
102.            }
103.            echo json_encode($element);
104.            exit();}
105.
106.        function getFreeTable()
107.        {
108.            echo $this->appFreeTable();
109.        }
110.
111.        function appFreeTable()
112.        {
113.            $query      = "SELECT GROUP_CONCAT(tablenumber) AS t FROM
    orders WHERE closed='0'";
114.            $res        = mysql_query($query);
115.            $data       = mysql_fetch_assoc($res);
116.
117.            $t      = explode(",",$data['t']);
118.
119.            $query      = "SELECT * FROM tables_count";
120.            $res        = mysql_query($query);
```

```
121.             $data       = mysql_fetch_assoc($res);
122.             $t2     = intval($data['count']);
123.             $t2Array    = array();
124.             for($i=1;$i<=$t2;$i++)
125.             {
126.                 $t2Array[count($t2Array)] = $i;
127.             }
128.             $freeTables = array_diff($t2Array,$t);
129.             $freeTables = implode(",",$freeTables);
130.             $freeTables = explode(",",$freeTables);
131.
132.             $t      = $freeTables[0];
133.
134.
135.             $element['num'] = $t."";
136.             $element['count'] = count($freeTables)."";
137.
138.             return json_encode($element);
139.         }
140.
141.
142.         function login()
143.         {
144.             $email      = $_REQUEST['email'];
145.             $password   = $_REQUEST['password'];
146.             $query      = "SELECT * FROM customers WHERE email='$email'
    AND password='$password'";
147.             $res        = mysql_query($query);
148.             $data       = mysql_fetch_assoc($res);
149.             echo json_encode($data);
150.         }
151.
152.
153.         function signup()
154.         {
155.             $name       = $_REQUEST['name'];
156.             $email      = $_REQUEST['email'];
157.             $password   = $_REQUEST['password'];
158.
159.
160.             $query      = "SELECT * FROM customers WHERE
    email='$email'";
161.             $res        = mysql_query($query);
162.             $num        = mysql_num_rows($res);
163.             if($num)
164.             {
165.                 echo "Email taken";
166.                 exit();
167.             }
168.
169.
170.             $query      = "INSERT INTO customers(name,email,password)
    VALUES('$name','$email','$password')";
171.             $exe        = mysql_query($query);
172.             if($exe)
173.             {
174.                 $query      = "SELECT * FROM customers WHERE
    email='$email'";
175.                 $res        = mysql_query($query);
```

```
176.                 $data       = mysql_fetch_assoc($res);
177.                 echo json_encode($data);
178.             }
179.
180.         }
181.
182.         function getOffers()
183.         {
184.             $array      = array();
185.             $query      = "SELECT * FROM offers ORDER BY dated DESC";
186.             $res        = mysql_query($query);
187.             while($data     = mysql_fetch_assoc($res))
188.             {
189.                 $array[count($array)]    = $data;
190.             }
191.
192.             echo json_encode($array);
193.         }
194.
195.
196.         function getCategories()
197.         {
198.             $catList    = array();
199.             $query  = "SELECT * from items GROUP BY cat ORDER BY cat
    ASC";
200.             $res    = mysql_query($query) or die(mysql_error());
201.             while($data = mysql_fetch_assoc($res))
202.             {
203.                 $data['image']          =
    str_replace("images/items/","",$data['image']);
204.                 $data['image']          =
    SERVER_URL."images/items/".rawurlencode($data['image']);
205.                 $catList[count($catList)]   = $data;
206.             }
207.             echo json_encode($catList);
208.         }
209.
210.     function getItems()
211.         {
212.     $catName    = $_REQUEST['cat'];
213.     $itemsList = array();
214.             $query      = "SELECT * from items WHERE cat='$catName'
    GROUP BY name ORDER BY name ASC";
215.             $res        = mysql_query($query) or die(mysql_error());
216.             while($data = mysql_fetch_assoc($res))
217.             {
218.                 $data['image']          =
    str_replace("images/items/","",$data['image']);
219.                 $data['image']          =
    SERVER_URL."images/items/".rawurlencode($data['image']);
220.                 $data['price_with_currency']    = "SR"+$data['price'];
221.                 $itemsList[count($itemsList)]   = $data;
222.             }
223.             echo json_encode($itemsList);
224.         }
225.
226.         function getHelloMsg()
227.         {
228.             $array      = array();
229.             $query      = "SELECT * FROM welcome";
```

```
230.             $res        = mysql_query($query);
231.             $data       = mysql_fetch_assoc($res);
232.             echo $data['msg'];
233.         }
234.
235.     function getoffersMsg()
236.         {
237.             $array      = array();
238.             $query      = "SELECT * FROM offers";
239.             $res        = mysql_query($query);
240.             $data       = mysql_fetch_assoc($res);
241.             echo $data['description'];
242.         }
243.
244.
245.     function statusMessege()
246.         {
247.             $customerid = $_REQUEST['customerid'];
248.             $query      = "SELECT * FROM tables_count";
249.             $res        = mysql_query($query);
250.             $data       = mysql_fetch_assoc($res);
251.             $count      = $data['count'];
252.             $array      = array();
253.             for($i=1;$i<=intval($count);$i++)
254.             {
255.                 $query      = "SELECT * FROM orders WHERE processed
    ='0' AND `tablenumber`='$i' AND `customerid`!='$customerid'";
256.                 $res        = mysql_query($query);
257.                 $isBusy     = mysql_num_rows($res);
258.
259.                 if(!$isBusy)
260.                     $array[count($array)] = $i+"";
261.             }
262.             if($count!=0)
263.             echo "restaurant not full";
264.
265.             else
266.                 echo "restaurant is full";
267.         }
268.
269.
270.     function getFreeTable1()
271.         {   $t = json_decode($this->appFreeTable(),true);
272.         $t = intval($t['num']);
273.         if(!$t){echo"Resturant is full";}
274.         else echo "restaurant is not full";
275.
276.         }
277.
278.     function appFreeTable2()
279.         {
280.             $query      = "SELECT *  FROM orders WHERE closed='0' and
    type ='2'";
281.             $res        = mysql_query($query);
282.             $t1     = mysql_num_rows($res);
283.
284.             $query      = "SELECT * FROM tables_count";
285.             $res        = mysql_query($query);
286.             $data       = mysql_fetch_assoc($res);
```

```
            $t2     = intval($data['count']);
        $t3=$t2 - $t1;

            $element['num'] = $t1."";
            $element['count'] = $t3."";

            return json_encode($element);
        }

    function getFreeTable2()
        {
            echo $this->appFreeTable2();
        }
    }
    ?>
```

Items.php

```
<?php
require_once("top.php");
?>
<html>
<head>
    <TITLE>Smart Restaurant :: Items</TITLE>
    <?php require_once("header.php");?>
</head>
<body>
     <div class="wrapper">
         <?php require_once("logo_menu.php");?>
         <p> </p>
     <div class="add-cont"><a href='item-oper.php'><label class="add"~ezentity_gt+ezentity_lt~/label></a></div>
         <div class="content-box">
         <table class='items-table' cellpadding="0" cellspacing="0">
                    <TR>
                        <TH>Sl.</TH>
                        <TH>Name</TH>
                        <TH>Category</TH>
                        <TH>Price</TH>
                        <TH>Image</TH>
                        <TH>Action</TH>
                    </TR>

     <?php
      $i   = 1;
      $query  = "SELECT * FROM items ORDER BY id DESC";
      $res    = mysql_query($query) or die(mysql_error());
        while($data = mysql_fetch_assoc($res))
                        {
     $img = "<img src='$data[image]' class='thumb-src'>";
     $edit    = "<a href='item-oper.php?id=$data[id]' class='edit'>&#9997;</a>";//pass id to item-oper page
                $delete = "<a href='javascript:void(0)' onclick='deleteData(\"$data[id]\",\"items\",\"items.php\")' ezonclick="true"  class='delete'>&#10006;</a>";
     //go to js function called deleteData
           $action = "<span>$edit   $delete</span>";
```

```
                                    echo "<TR>
                                            <TD>$i</TD>
                                            <TD>$data[name]</TD>
                                            <TD>$data[cat]</TD>
                                            <TD>SAR $data[price]</TD>
                                            <TD>$img</TD>
                                            <TD>$action</TD>
                                          </TR>";
                                    $i++;
                                }
            ?>
            </table>

        </div>
        </div>
        </body>
        <?php require_once("footer.php");?>
        </html>
```

Items-oper.php

```
<?php
require_once("top.php");

$str    = "Add New";
$alert  = "Item added successful";
$error3 = "Error in adding item";
$data   = $_POST;
$catList    = array();
$query = "SELECT DISTINCT cat from items ORDER BY cat ASC";
            $res    = mysql_query($query) or die(mysql_error());
            while($data = mysql_fetch_assoc($res))
            {
            $catList[count($catList)] = " ".$data['cat']." ";}
            $catList = implode(" , ",$catList);
            $catList = "Present Cat : ".$catList;

            if(intval($_REQUEST['id']) > 0)
            {
                $str    = "Save";
                $alert  = "Item updated successful";
                $error3 = "Error in update item";
                $query  = "SELECT * FROM items WHERE id='$_REQUEST[id]'";
                $res    = mysql_query($query) or die(mysql_error());
                $data   = mysql_fetch_assoc($res);
                $imgDi  = "<img src='$data[image]' class='thumb-src'>";}

            if($_POST['save']  == $str)
            {
                $name   = trim($_POST['name']);
                $price  = trim($_POST['price']);
                $cat    = trim($_POST['cat']);
                $image  = $_FILES['transaction-img'];
                if($name == "")
                {
```

```
35.                     $error  = "Name cannot be blank";
36.                 }
37.                 else if($cat == "")
38.                 {
39.                     $error  = "Category cannot be blank";
40.                 }
41.                 else if(floatval($price) == 0 )
42.                 {
43.                     $error  = "Price cannot be blank";
44.                 }
45.                 else
46.                 {
47.                     if($image['tmp_name'] != "")
48.                     {
49.                         $img    =
   IMG_ITEMS."_".time()."_".$image['name'];
50.                         $var    =
   move_uploaded_file($image['tmp_name'],$img);
51.                         if($var)
52.                         {
53.                             $updateImg  =" ,image='$img' ";
54.                         }
55.                         else
56.             print("<script>alert('Image uploading error');</script>");}
57.             if(intval($_REQUEST['id']) > 0)
58.               $query  = "UPDATE items SET name='$name',cat='$cat' ,
   price='$price' $updateImg WHERE id='$_REQUEST[id]'";
59.             else
60.                $query  = "INSERT INTO items(name,price,image,cat)
   VALUES('$name' , '$price' , '$img','$cat')";
61.
62.                     $res        = mysql_query($query);
63.                     if($res)
64.                     {
65.             print("<script>alert('$alert');location.replace('items.php');
   </script>");
66.             exit();}
67.                     else
68.                      $error  = $error3;
69.             }
70.             }
71.             ?>
72.             <html>
73.             <head>
74.                 <TITLE>Smart Restaurant :: Item</TITLE>
75.                 <?php require_once("header.php");?>
76.             </head>
77.
78.             <body>
79.             <div class="wrapper">
80.                 <?php require_once("logo_menu.php");?>
81.                 <h3 class="error"> <?=$error?></h3>
82.                 <p> </p>
83.                 <div class="content-box">
84.             <form method="POST" action="item-oper.php"
   enctype="multipart/form-data">
85.             <input type="hidden" value="<?=$_REQUEST['id']?>" name="id">
86.             <table align="center" class="login-tbl">
87.                             <TR>
88.             <TD><label class="data-lbl">Name</label></TD>
```

```
89.           <TD><input type='text' class='form-input' name="name"
   value='<?=$data['name']?>' autocomplete='false'></TD>
90.           </TR>
91.           <TR>
92.           <TD><label class="data-lbl">Category</label></TD>
93.           <TD><input type='text' class='form-input' name="cat"
   value='<?=$data['cat']?>' autocomplete='false'></TD>
94.           </TR>
95.           <TR>
96.           <TD colspan="2" align="center"><label style="font-
   size:10px;color:#888;text-align:center;"><?=$catList?></label></TD>
97.            </TR>
98.            <TR>
99.            <TD><label class="data-lbl">Price</label></TD>
100.           <TD><input type='text' class='float-val form-input'
    name="price" value='<?=$data['price']?>'></TD>
101.           </TR>
102.           <TR>
103.           <TD><label class="data-lbl">Image</label></TD>
104.           <TD>
105.           <label class="chkdate" id="f-name">No image is
   selected</label>
106.           <input type="file" class="form-control" id="transaction-img"
   name="transaction-img" style="opacity:0;margin-left:-230px;"
   accept="image/*">  <?=$imgDi?>
107.          <!--accept only img -->
108.           </TD>
109.           </TR>
110.           <TR>
111.           <TD> </TD>
112.           <TD><input type='submit' class='submit-input'
113.           value='<?=$str?>' name='save'></TD>
114.           </TR>
115.           </table>
116.           </form>
117.          </div>
118.          </div>
119.          </body>
120.          <?php require_once("footer.php");?>
121.          </html>
```

Config.php

This php to connect to DB with server

```
1. <?php
2. error_reporting(E_ERROR);
3. mysql_connect("localhost","smart","gx4aOshep6,K");
4. mysql_select_db("smart_smart");
5. ?>
```