

# ST. XAVIER’S COLLEGE - AUTONOMOUS, MUMBAI

# DEPARTMENT OF INFORMATION TECHNOLOGY

## COURSE CODE: ITS.6.PJ

## ONLINE FOOD DELIVERY & CAB SERVICES

### DONE BY:

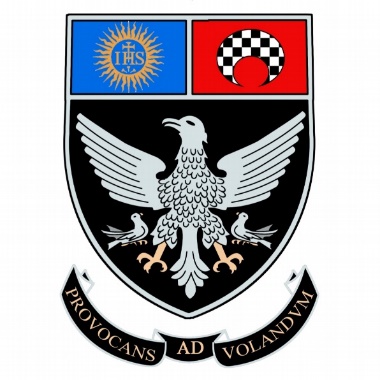
**NAME:** SHYAM PATEL

**UID:** 165074

### SUPERVISOR:

ASST. PROF. LYDIA FERNANDES

This Project is submitted to the department of **Information Technology** of **S****t. Xavier’s College - Autonomous, Mumbai** which is affiliated to **University of Mumbai**



**CERTIFICATE**

This is to certify that the project titled **‘Online Food & Cab Services’**, undertaken at the St. Xavier’s College – Autonomous, Mumbai by **Shyam Patel**. In partial fulfilment of the BSc. IT degree (Semester VI) examination has not been submitted for any other examination and does not form part of any other course undergone by the candidate. It is further certified that I have completed all the required phases of the project.

**Signature Signature Signature**

**[Internal Guide] [Internal Examiner] [External Examiner]**

**Prof. Roy Thomas**

**[HOD of IT Dept]**

DECLARATION

I, **Shyam Patel** [UID: 165074], hereby declare that this project report entitled: **Online Food & Cab** **Services** which is being submitted in fulfilment of the Bachelors of Science in Information Technology Examination conducted by St. Xavier’s College – Autonomous under Mumbai University is the result of the work carried out by me under the supervision of **Assist. Prof. Lydia Fernandes** of St. Xavier’s College – Autonomous, Mumbai.

This work has not been previously submitted to any other university for any examination. Wherever references have been made to previous work of others, it has been clearly indicated as such and included in the bibliography.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Shyam Patel

DEDICATION

This project proposal is dedicated with profound admiration and appreciation to God almighty for giving me strength and breathe and my beloved friends, family and professors for their moral support. Great appreciation also goes to my supervisor Prof. Lydia Fernandes and all my lecturers, classmates and Xavier’s family who made it a success through their constant support supervision, encouragement and moral support.

ACKNOWLEDGEMENT

Most importantly, I sincerely thank the Almighty God for giving me strength and breathe throughout the preparation of this project proposal. I also wish to extend my sincere and heartfelt gratitude to my loving family for the financial, moral and emotional support, my classmates and friends who helped in various ways for the successful completion of this project proposal.

Finally, it is a great pleasure for me to also acknowledge the assistance and support of all the people who helped me start and finish this project proposal successfully especially Prof. Lydia Fernandes who has been a great supervisor during this period. I would like to give my special thanks to Xavier’s fraternity for giving me enough knowledge and skills that made me to innovatively and successfully research and compile this project proposal.

**Thank you every one!**

ABSTRACT

A fast food restaurant also known as quick service restaurant (QSR) within the food service industry is a specific type of restaurant characterized both by its fast food cuisine and by minimal table service. Food served in fast food restaurants is offered from a limited menu, cooked in bulk in advance and kept hot, is finished and packaged for order and is usually available ready for pickup or to be delivered though seating may also be provided. The customers presently spend an average of 60 minutes per day going to the restaurant, selecting their meals and paying. Some restaurants have the provision of customers making a call to the restaurant in advance to order a meal to be ready for them for pick or to be delivered to them. Some of the customers don’t always get the selection they want because the restaurants run out of certain items or because there is no provision of ordering custom meals.

This project is aimed at developing a complete online ordering system for use in the food service industry which will allow the restaurants to quickly and easily manage an online menu which customer can browse and use to place orders with just a few clicks. The customers will have to choose whether they want the food to be delivered to them or it will be packaged for pick up and the payment method will be upon delivery or pick up. There will be a system administrator who will have the right to add and manage user accounts, a manager who will be managing product and orders and last but not least a meal deliverer who will be dealing specifically with pending deliveries. The customer will be in a position to view the products, register and place an order. There will be a confirmation receipt for each and every order made by the customer which can be printed.

The development of this system will be based on SDLC with Android Studio and Java as the programming languages while Firebase server as the database of the system. Java language is advantageous due to its easy to use and learn validation properties while Firebase Relational Database has better advanced features and properties, has good security, is open source and has cross platform operability. The advantages of using Android Studio and Java programming language in developing this system include:

* It is a stable open source language developed and maintained by a large group of Java developers which help in creating a support community and abundant extension library.
* It’s easy and quick to learn and use
* Can be run on many platforms thus easy for users to find hosting service
* It has built-in database connection modules which makes it easy to connect to the database.

On the other hand, the disadvantage of Android Studio and Java programming language is System’s Performance decreases due to the consumption of high RAM/CPU. Java takes more memory space than other native programming languages like C /C+ +. Java has limited options for latency critical tuning. Java’s ability for producing portable, architecturally code is desirable, the method used to create this code is inefficient.

TABLE OF CONTENTS

INDEX

|  |  |  |
| --- | --- | --- |
| Sr. No | Particulars | Page No. |
| 1) | Chapter 1.0: Introduction | 1 |
| 2) | Chapter 2.0: Methodology | 4 |
| 3) | Chapter 3.0: Overall Description | 6 |
| 4) | Chapter 4.0: Analysis and Design | 9 |
| 5) | Chapter 5.0: ER Diagram | 14 |
| 6) | Chapter 6.0: Planning | 19 |
| 7) | Chapter 7.0: Implementation and Testing | 21 |
| 8) | Chapter 8.0: Diagram and Visualisation | 32 |
| 9) | Chapter 9.0: Car Booking App | 37 |
| 10) | Chapter 10.0: Databases | 48 |
| 11) | Chapter 11.0: System Coding | 49 |
| 12) | Conclusion | 94 |
| 13) | Future Enhancement | 95 |
| 14) | Goals and Achievement | 96 |
| 15) | References | 97 |

CHAPTER 1.0: INTRODUCTION

Computers have become part of the life for accessing almost any kind of information. Life in the 21st century is full of technological advancement and in this technological age it is very difficult for any organization to survive without utilizing technology. The World Wide Web contributes greatly to the creation of an ever-increasing global information database. It could also be used as a mechanism to share information within an enterprise.

In today’s age of fast food and take-out, many restaurants have chosen to focus on quick preparation and speedy delivery of orders rather than offering a rich dining experience. Until very recently, all of these delivery orders were placed over the phone, but there are many disadvantages to this system, including the inconvenience of the customer needing to have a physical copy of the menu, lack of a visual confirmation that the order was placed correctly, and the necessity for the restaurant to have an employee answering the phone and taking orders.

What I propose is an online ordering system, which is a technique of ordering foods online applicable in any food delivery industry. The main advantage of my system is that it greatly simplifies the ordering process for both the customer and the restaurant. When the customer visits the ordering webpage, they are presented with an interactive and up-to-date menu, complete with all available options and dynamically adjusting prices based on the selected options. After making a selection, the item is then added to their order, which the customer can review the details of at any time before checking out. This provides instant visual confirmation of what was selected and ensures that items in the order are, in fact, what was intended.

This system also greatly lightens the load on the restaurant’s end, as the entire process of taking orders is automated. Once an order is placed on the webpage, it is entered into the database and then retrieved, in pretty much real-time, by a desktop application on the restaurant’s end. Within this application, all items in the order are displayed, along with their corresponding options and delivery details, in a concise and easy to read manner. This allows restaurant employees to quickly go through the orders as they are placed and produce the necessary items with minimal delay and confusion.

* 1. Problem statement

The challenges encountered by the existing system serve as a major drawback to the realization of efficiency and customer satisfaction. The experience of ordering in most fast food restaurants is not pleasant for the customers. Customers will have to make long queues before placing their orders especially during peak hours and then the ordering staff will record customer orders. Having placed their order, the customer must then wait near the counter until their order is ready for collection. The other problem in the food service industry is that restaurants are not realizing the efficiencies that would result from better application of technology in their daily operations. Fast food business in a very competitive business and one way to stand out from competitors is through improving the business process where business process automation can assist business improvement. The other problem with the current system is that the customers are not able to see the ingredients of the meals before they place their order and also they only have to pay for an order online.

* 1. Objectives
     1. General objectives
* To increase efficiency and improve services provided to the customers through better application of technology in daily operations.
* To be able to stand out from competitors in the food service industry
  + 1. Specific objectives
* To enable customers to order custom meals that aren’t in the menu
* To enable customers to have a visual confirmation that the order was placed correctly
* To enable customers to know food ingredients before ordering
* To reduce restaurant’s food wastage
* To ensure correct placement of orders through visual confirmation
* Improve efficiency of restaurant’s staff
* Eliminate paper work and increase level of accuracy
* Increase speed of service, sales volume and customer satisfaction
  1. Justification
* To increase efficiency by shortening the purchasing time and eliminating paper work like receipts through online transaction
* To be able to stand out from competitors by automating daily operations which will give food service providers the opportunity to increase sales
* To reduce restaurants food wastage and increasing efficiency of the restaurants staff by enabling the restaurants staff to know what food items the customers want in advance.
* To increase customer satisfaction by speeding up food delivery
* To reduce time wasting by eliminating long queues
  1. Project scope

Online ordering system will be a android based application whose main language of programming will be Java. Its main aim is to simplify and improve the efficiency of the ordering process for both customer and restaurant, minimize manual data entry and ensure data accuracy and security during order placement process. Customers will also be able to view product menus and there ingredients and be able to have a visual confirmation that the order was place correctly.

* 1. Limitation of the system
* Requires internet connection and also the user must be computer literate.
* The set back of the system is that the customers targeted are adults with access to computer systems while the minors might have to go physically to the restaurant to purchase the food that they want or order food the food with the help of an adult.
* The other limitation is that the system will only be convenient to people with a small geographical region, basically just around the restaurant i.e. can only help a small area.

CHAPTER 2.0: METHODOLOGY

Top apps are developed in the span of 6–10 weeks, however to develop a basic yet interesting Mobile Application, one will take 5–6 weeks. Let us go through the method of developing a mobile app.

**Week 1: Requirement analysis**

Study the app in detail and list down the features that your app requires. Look into the requirements about what could be the best tools for developing an app. Prepare the list of required items and install the necessary software on your system.

**Week 2: UX/UI designing**

Prepare use case and flowcharts. Break each element into deliverable and lay down a strategy to go ahead with. Work on designing part (UI/UX) and prepare a design that delivers the best user experience. Test it on different devices. Ensure smooth navigation on the Mobile App.

**Week 3 and 4: Coding**

While working on designing, Start implementing back-end Processes. Deploy each functionality one by one and keep testing it. The coding part eats your time here but as you have your plan defined, coding will not be a burden.

**Week 5: Testing and modifications**



Test the functionalities and make necessary changes. Ask dummy users to use the app and make necessary changes according to their inputs.

**Week 6: Launching**



Once you are through with all the above steps, you can now launch your Mobile app. Wrap the app with a strong marketing plan.

CHAPTER 3.0: OVERALL DESCRIPTION

* 1. Weaknesses of the current system
* Inconvenience of customer needing to have a physical copy of the menu
* Time consuming
* Lack of visual confirmation that the order was placed correctly
* Necessity for restaurant to have an employee answering the phone and taking orders
* Difficulty in tracking customers past history
* Manual work and consumes large volumes of data
* Lack of data security
  1. Merits of the proposed system
* Security of data. Data are well protected for personal use.
* Ensures data accuracy during order placement process
* Minimized manual data entry
* Greater efficiency since data processing is very fast
* User friendly and interactive interface with provision for customer to view menus and have a visual confirmation that the order was place correctly.
* Minimized time requirement during the order placement process
* Greatly simplifies the ordering process for both customer and restaurant.
  1. Feasibility study

This is an evaluation and analysis of the potential of the proposed project which is based on extensive investigation and research to support the process of decision making. [1] It assesses the operational, technical and economic merits of the proposed project. The feasibility study is intended to be a preliminary review of the facts to see if it is worthy of proceeding to the analysis phase. From the systems analyst perspective, the feasibility analysis is the primary tool for recommending whether to proceed to the next phase or to discontinue the project.

* + 1. Technical feasibility

This assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the needs of the proposed system. The systems project is considered technically feasible if the internal technical capability is sufficient to support the project requirements. The essential questions that help in testing the technical feasibility of a system include the following:

* Is the project feasible within the limits of current technology?
* Does the technology exist at all?
* Is it available within given resource constraints?
* Is it a practical proposition?
* Is there enough manpower- programmers, testers & debuggers?
* Do the required software and hardware exist?
* Are the current technical resources sufficient for the new system?
* Can they be upgraded to provide the level of technology necessary for the new system?
* Do we possess the necessary technical expertise, and is the schedule reasonable?
* Can the technology be easily applied to current problems?
* Does the technology have the capacity to handle the solution?
* Do we currently possess the necessary technology?
  + 1. Operational feasibility

Operational feasibility is the measure of how well the project will support the customer and the service provider during the operational phase. It is dependent on human resources available for the project and involves projecting whether the system will be used if it is developed and implemented. The essential questions that help in testing the technical feasibility of a system include the following:

* Is the project feasible to operate or not?
* Does current mode of operation provide adequate throughput and response time?
* Could there be a reduction in cost and or an increase in benefits?
* Does current mode of operation offer effective controls to protect against fraud and to guarantee accuracy and security of data and information?
* Does current mode of operation make maximum use of available resources, including people, time, and flow of forms?
* Are the current work practices and procedures adequate to support the new system?
* If the system is developed, will it be used?
* Does it agree with the government regulations
* Will the proposed system really benefit the organization?
* Will the system affect the customers in considerable way?
* How do the end-users feel about their role in the new system?
* How will the working environment of the end-user change?
  + 1. Economic feasibility

This assessment aims to determine the positive economic benefits to the organization that the proposed system will provide. It typically involves a cost/ benefits analysis and it’s the most frequently used method for evaluating the effectiveness of a new proposed system. Possible questions raised in economic analysis are:

* Is the system cost effective?
* Do benefits outweigh costs?
* The cost of doing full system study
* The cost of business employee time
* Estimated cost of hardware
* Estimated cost of software/software development
* Is the project possible, given the resource constraints?
* What are the savings that will result from the system?
* Cost of employees' time for study
* Cost of packaged software/software development
* Selection among alternative financing arrangements (rent/lease/purchase)
  + 1. Schedule feasibility

It is the measure of how reasonable the project time table is or the deadline is reasonable or not. During the lack of time or the time become mandatory, we must finish the project within a given time period. It mainly addresses:

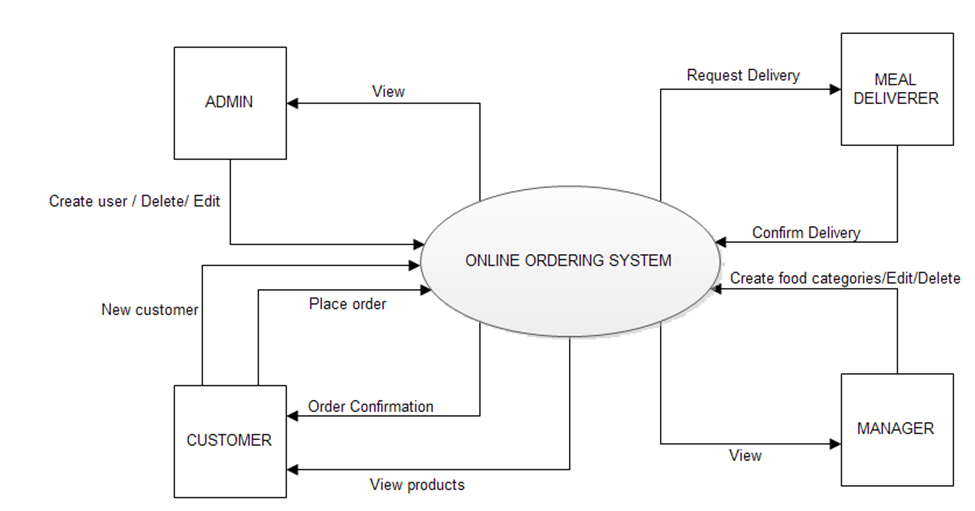
* Can the project really be completed in the given period of time

CHAPTER 4.0: ANALYSIS AND DESIGN

Development of computerized systems requires analysis of the process to be digitized in order to enable a correct system, a system that functions as required and to assist the potential users of the system understand the general functionality of the system. The analysis specifies the system's objectives and constraints to which designers have to comply. The purpose of doing analysis is to transform the system’s major inputs into structured specification.

* 1. Context diagram

This is a brief structure which depicts the environment in which a software system exists and helps in communicating about what lies outside the system boundary.



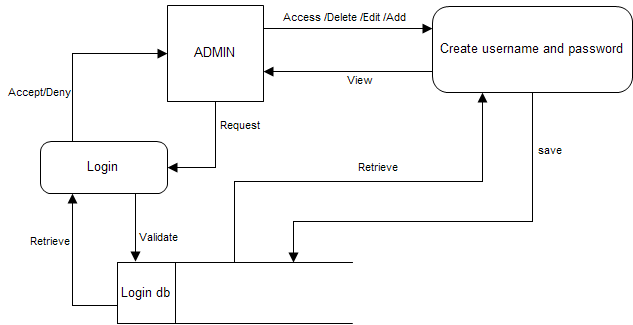
* 1. Data flow diagram

It is a two-dimensional diagram that explains how data is processed and transferred in a system. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output.

Administrator module

Functionalities provided:

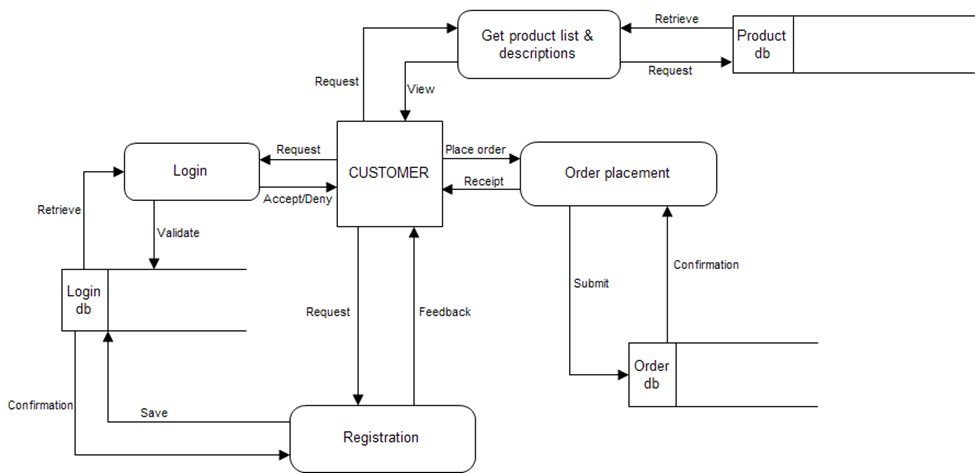
* Create usernames and passwords
* View/ edit / delete user accounts



Customer module

Functionalities provided:

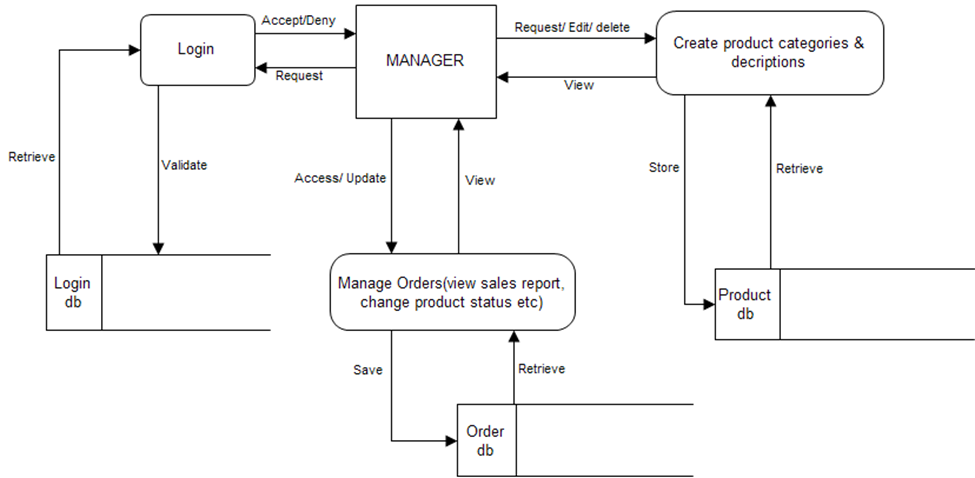
* View product’s list
* Register
* Place orders



Manager module

Functionalities provided:

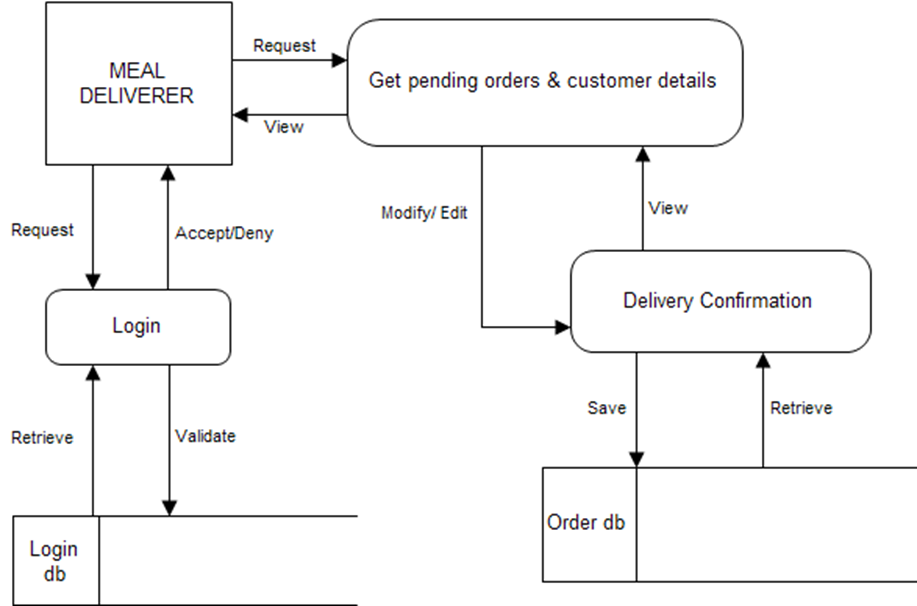
* Create product categories and functionalities
* Edit / delete product categories and descriptions
* View and manage orders and sales report



Meal deliver module

Functionalities provided:

* View pending orders and delivery details
* Confirm order deliveries



* 1. User requirements

The system will be designed to be user friendly. The user friendly and interactive interfaces design helps to achieve this by enabling customers to easily browse through the menus place orders with just a few clicks and also allows restaurant employees to quickly go through the orders as they are placed and produce the necessary items with minimal delay and confusion. The system will be simple to use.

* + 1. Functional requirements

Functional requirements define the capabilities and functions that a system must be able to perform successfully. The functional requirements of this online ordering system include:

* The system shall enable the customer to view the products menu, create an account, login to the system and place an order.
* The customer shall specify whether the order is to be picked up or delivered.
* The system shall display the food items ordered, the individual food item prices and the payment amount calculated.
* The system shall prompt customer to confirm the meal order.
* The system shall provide visual confirmation of the order placement
* The system shall enable the manager to view, create, edit and delete food category and descriptions
* The system shall allow confirmation of pending orders.
* The system shall allow generation of sales report for the orders made.
* The system shall allow the manager to update additional information (description, photo, ingredients etc.) for a given food item.
* The system shall allow the manager to update price for a given food item.
  + 1. Non-functional requirements

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Some of the non-functional requirements include:

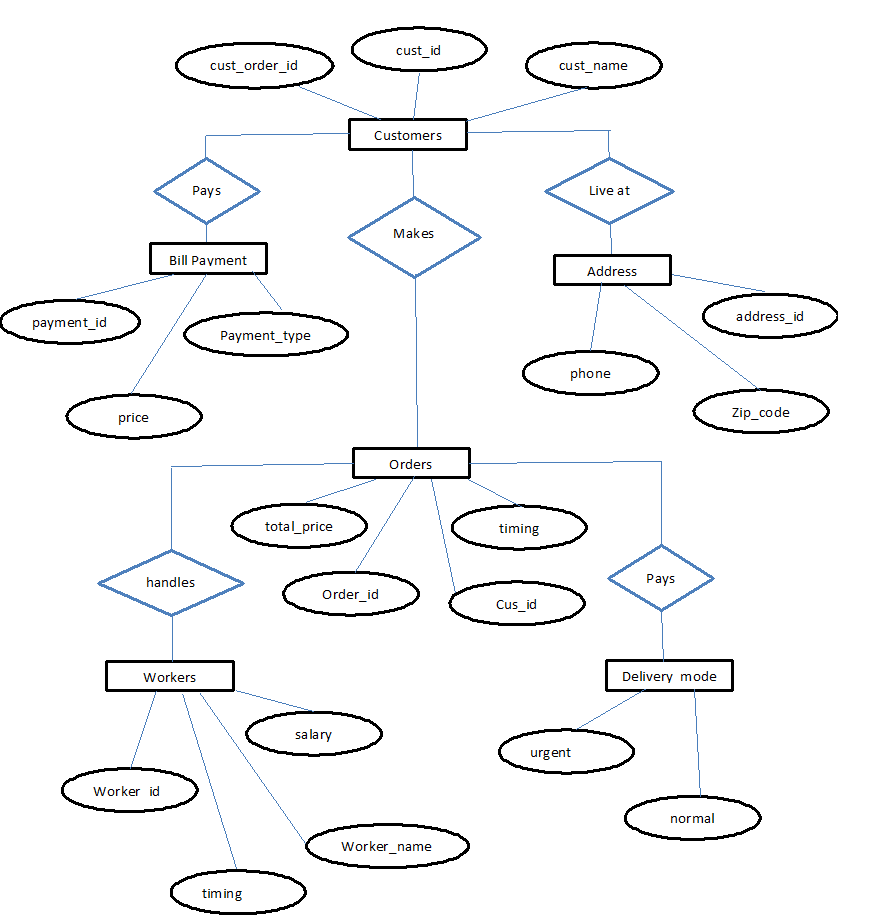
* The should be sufficient network bandwidth
* Backup- provision for data backup
* Maintainability- easy to maintain
* Performance/ response time- fast response
* Usability by target user community- easy to use
* Expandability- needs to be future proof or upgradable
* Safety- should be safe to use

* 1. System requirements

These consist of the hardware and software components of a computer system that are required to install in order to use the software efficiently.

* + 1. Software requirements
* Operating system: Windows 10
* Technology : Java
* Database : Firebase Relational Database
* Tool : Android Studio
* Antivirus software
* Backup & Data Recovery software
  + 1. Hardware requirements
* **Processor:** Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
* **Processor Speed:** 2.5GHz or above
* **RAM:** 8 GB RAM or above
* **Hard Disk:** 1 TB hard disk or above
* Printer for printing reports
* Uninterruptible power supply to ensure a constant access of data.
* USB flash disk( At least 2GB)

## CHAPTER 5.0: ER-DIAGRAM OF ORDER FOOD ONLINE SYSTEM



The above shown is an entity relationship diagram, depicting the working of food ordering system. ER diagram reflects the relationships that various entities involved in the system share among themselves, along with the entities.

Following are the description of entities involved in food ordering system:

### Customer

This represents the set of customers, which are the clients who will be using this application. The customers are for whom the system is being designed. Its attribute set includes:

#### Name:

This is the name of the customer, searching or purchasing the products. When signing up to the website the name of the customer is stored, this is done for the future referencing and maintaining the user’s data record (history).

It is the composite attribute which contains two more attributes that are First\_Name and Last\_Name. That contains user’s first name and last name.

#### Cus\_id:

This is the identification number assigned by the admin to the users so as to identify them uniquely in future. This identity number is helpful in fetching data of the individual user from a big set.

This is mainly to manage the huge database system where the entire data is being stored. It is a permanent identification number given by the admin to the customer to maintain customer history.

#### Cus\_order\_id:

This is the identification number given to determine and manage the sequence of service. Since multiple customers will place orders, so as to schedule whom to give the delivery first is determined by the help of this number, so as to maintain consistency in the system working procedure.

It will be unique for each order a day. But the same id can be repeated on a new day, as it is mainly for the restaurant’s reference and to prevent any type of conflict.

### Address

This field is for the physical address of the customer where the restaurant authority is required to deliver the parcels. It may or may not the same as customer’s permanent address or resident, but can be the office place or any place. Its attribute includes:

#### Address\_id:

An identity through which categorization of places may be done. As address may or may not be unique for each customer registered. But still, this identity helps the delivery person to identify the right place to deliver.

#### Zip\_code:

It is the pin code or the postal code of a region, and which is utmost important to any address, since multiple places, streets, bungalows with the same name exist. This is even important in any national level identification of address.

Also, this will help the owner in surveying that which region has their more demand so as to expand their business in that region.

#### Phone:

The user’s contact number is something that must be correct because if at some point of time delivery person gets confused with the address, it can be used for confirmation.

Also, the restaurant authority can contact to their customers for any type of feed backs or know the delivery service is good or not.

### Orders

The customer’s place order, which is not only still here, there is some work that needs to be done in the database in order to maintain records for keeping track on monthly basis.

#### Order\_id:

This is the identification number given to determine and manage the sequence of service. Since multiple customers will place orders, so as to schedule whom to give the delivery first is determined by the help of this number, so as to maintain consistency in the system working procedure.

It will be unique for each order a day. But the same id can be repeated on a new day, as it is mainly for the restaurant’s reference and to prevent any type of overlapping of thoughts between customers and owners. It is mainly for the chef’s preference.

#### Cus\_id:

This is the identification number assigned by the admin to the users so as to identify them uniquely in future. This identity number is helpful in fetching data of the individual user from a big set.

This is mainly to manage the huge database system where the entire data is being stored. It is a permanent identification number given by the admin to the customer to maintain customer history.

#### Total\_price:

This attribute manages the total price sum of the orders user has made in one attempt.It is one of the most important attributes, since most of the times people change their menu order list contents depending upon their needs, health and economic situation.

#### Timing:

Time is something most important to be valued. And one of the major reason behind the success of this food ordering system. So managing this cause becomes a goal to be completed.

In order to maintain the business work better, the authority must stick to its commitment.

### Payment

It defines the payment to be done by the customer for an order placed from the web store at worth price. Also, various security encryption mechanisms have been used, so the customer details of accounts and other credentials are safe and secure.

#### Payment\_type:

The user is provided with lots of options that he/she can opt for making the payment depending upon their ease. There are many choices available for net banking, use of wallets like pay and i-cash cards, also the credit card and debit card options are available too.

#### Payment\_id:

It is for the benefit of the user as well as the website owners since the payment\_id is helpful in maintaining the payment record in the database, as well as it is also provided to the customer after the successful completion of payment.

As later customer can claim anytime that they have already done the payments and the owners cannot deny. So it is useful to prevent any kind of fraud from both the sides.

#### Price:

It is the record of the total sum amount the user needs to pay, and after the payment, it is used to update in the server-side database to keep the record of the net profit or loss on daily basis.

### Worker

The base of any company, restaurant or hotel is its employee. It is said that an organization is known by its employee and work. Employees will work honestly and with complete dedication if they are paid sufficient enough money.

On the whole, it’s just like a food cycle, everyone depends on somebody.

#### Worker\_name:

The name of the worker is important to maintain their database of work and payment record. Also if any complaints are filed then it is required.

#### Timing:

Time is something most important to be valued. And one of the major reason behind the success of this food ordering system. So managing this cause becomes a goal to be completed.

In order to maintain the business work better, the authority must stick to its commitment. Workers are paid for their good work and more than that for completion of work before time.

#### Worker\_id:

To uniquely identify each worker and prevent any type of redundancy in records.

#### Salary:

The amount of money to be paid to the workers for their effective and on time work was done.

### Delivery\_mode

The delivery sequence and choice is not same for everyone but varies person to person. It may happen that even sometimes a person says no to home delivery as he/she is passing by and can pick the parcel themselves. But it is almost an ideal case.

#### Urgent:

In some cases like un invited guest arrival, late night, people prefer to pay more and get the order delivered urgently. So restaurants manage such situations by not following the sequence of order placement, as they are getting more than usual.

And with another customer whom they have delayed, they manage it with some small gifts or offers.

#### Normal:

The usual mode of delivery, that is followed by the sequence of orders placed. It is the normal and majority case. The hotels manage do not need to put extra efforts to manage these.

CHAPTER 6.0: PLANNING

* 1. Time scheduling

|  |
| --- |
| Online Food and Cab Services |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | October 2018 | November 2018 | December  2018 | January 2019 | February 2019 | March 2019 | April 2019 |
| Research & Data Collection |  |  |  |  |  |  |  |
| Proposal Waiting |  |  |  |  |  |  |  |
| Proposal approval & project starting time |  |  |  |  |  |  |  |
| Coding, design and Testing |  |  |  |  |  |  |  |
| Deployment and documentation |  |  |  |  |  |  |  |
| Project presentation |  |  |  |  |  |  |  |

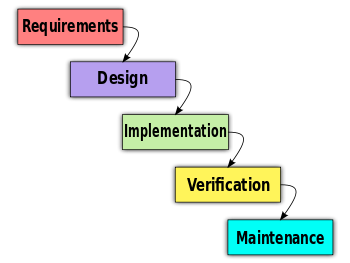
* 1. Budget

|  |  |
| --- | --- |
| **ITEM DESCRIPTION** | **AMOUNT** |
| FIREBASE CONSOLE | FREE |
| GOOGLE CONSOLE | FREE |
| ANDROID STUDIO | FREE |
| LAPTOP | 45,000 |
| STATIONERY AND PRINTING | 1000 |
| **TOTAL:** | **46,000** |

Table: Budget

* 1. Process model

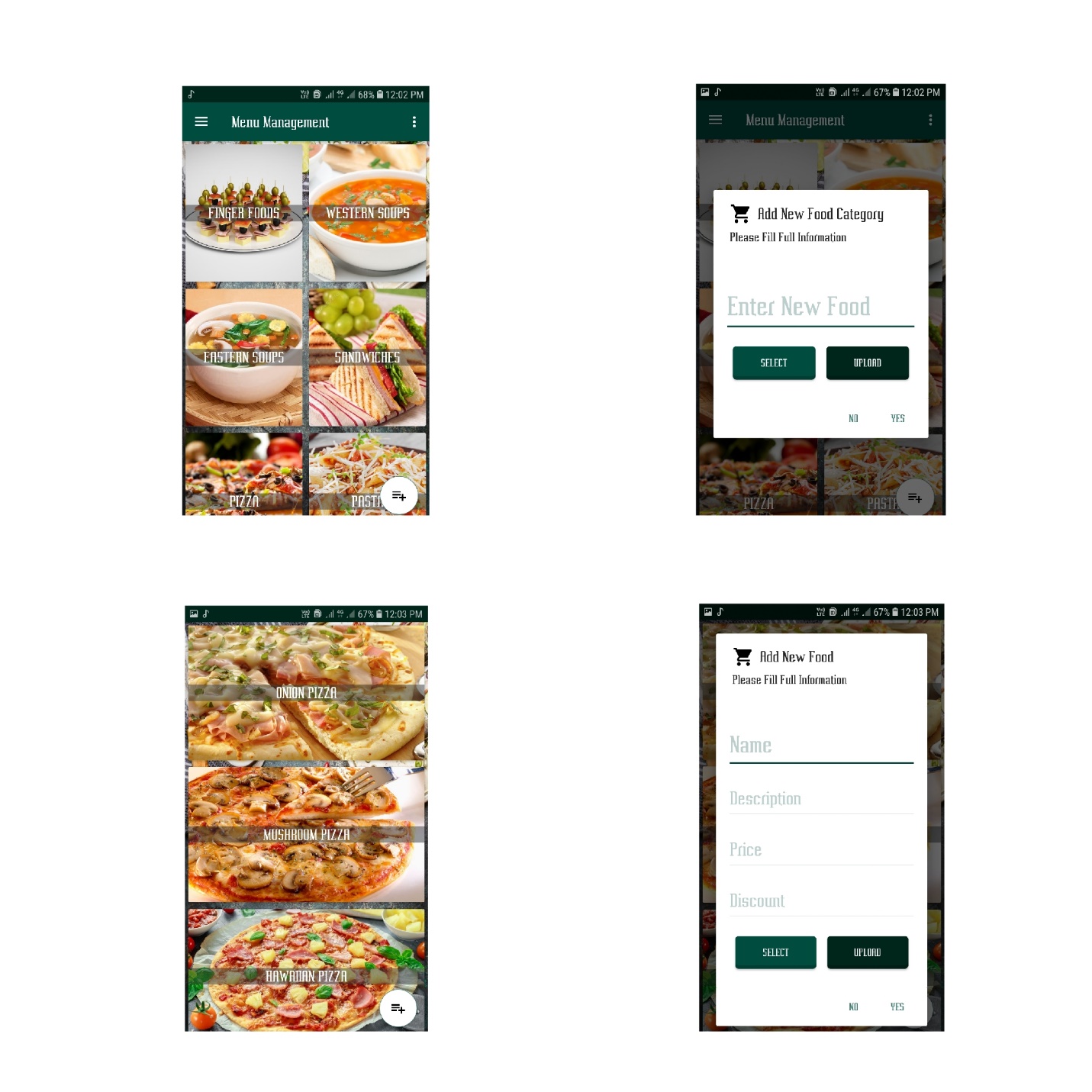
For my project I plan to use waterfall as a process model. The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing and Maintenance.



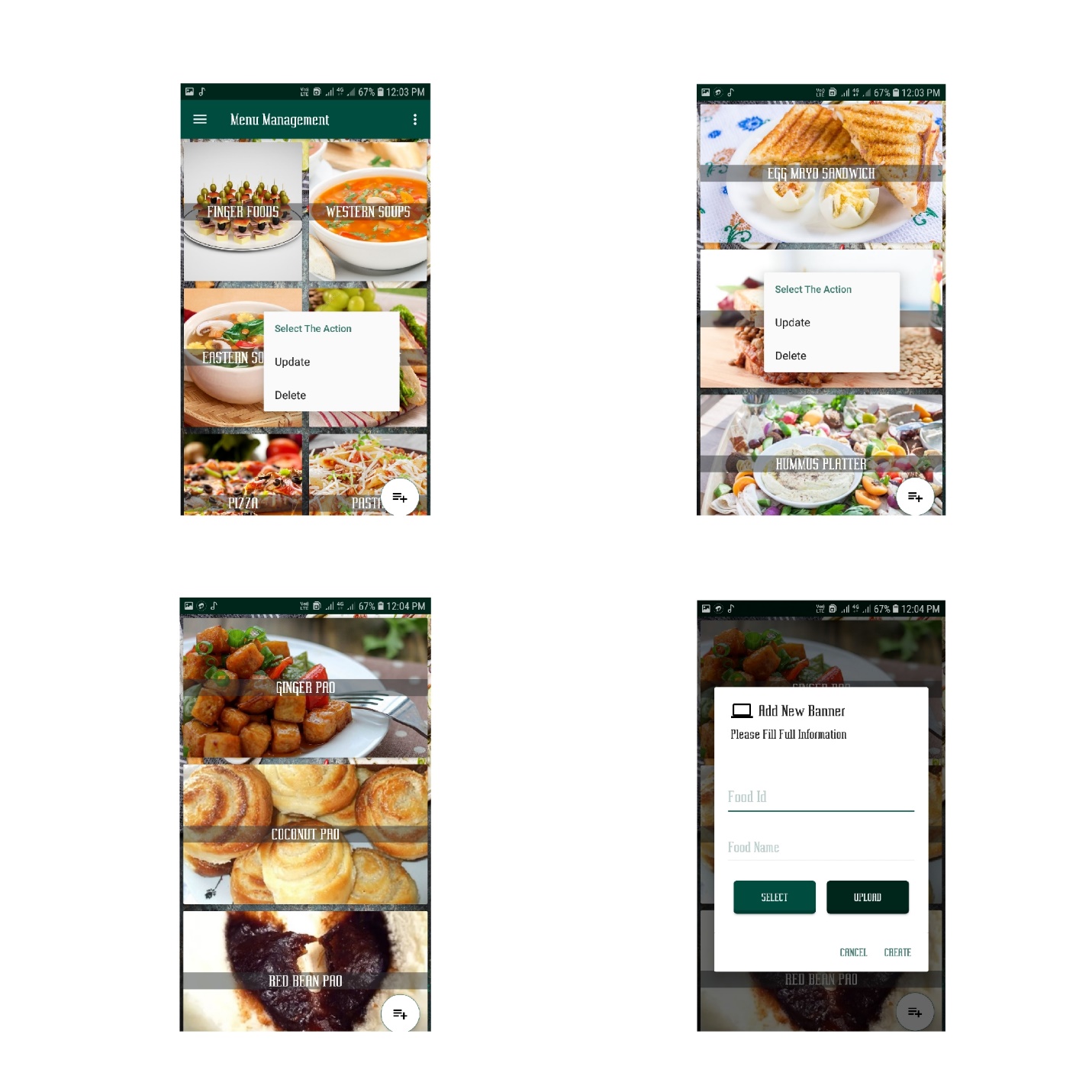
CHAPTER 7.0: IMPLEMENTATION AND TESTING

7.1: Admin App

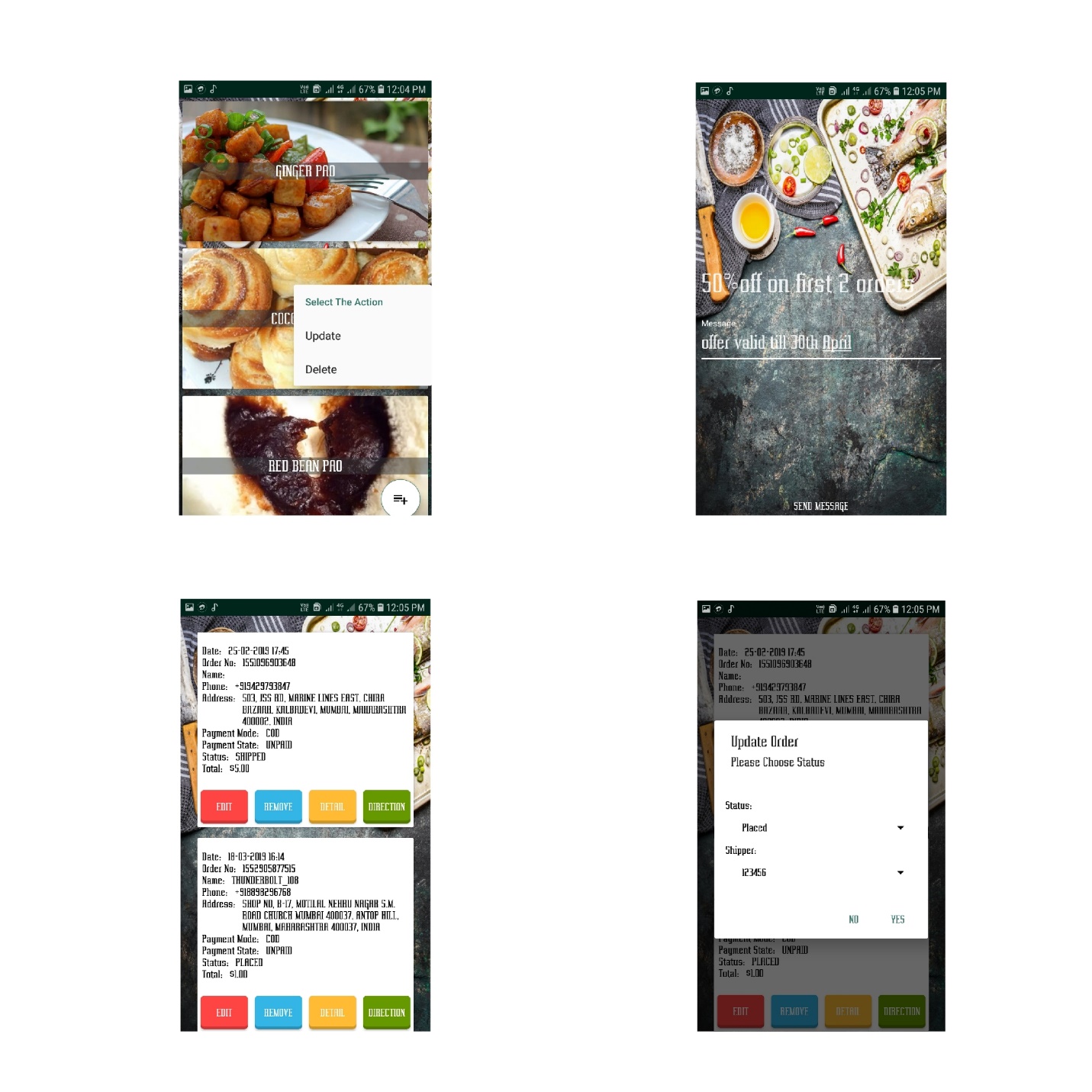
This is the admin view where admin can create food category and within food category admin can create respective food items which will reflect in customer app.



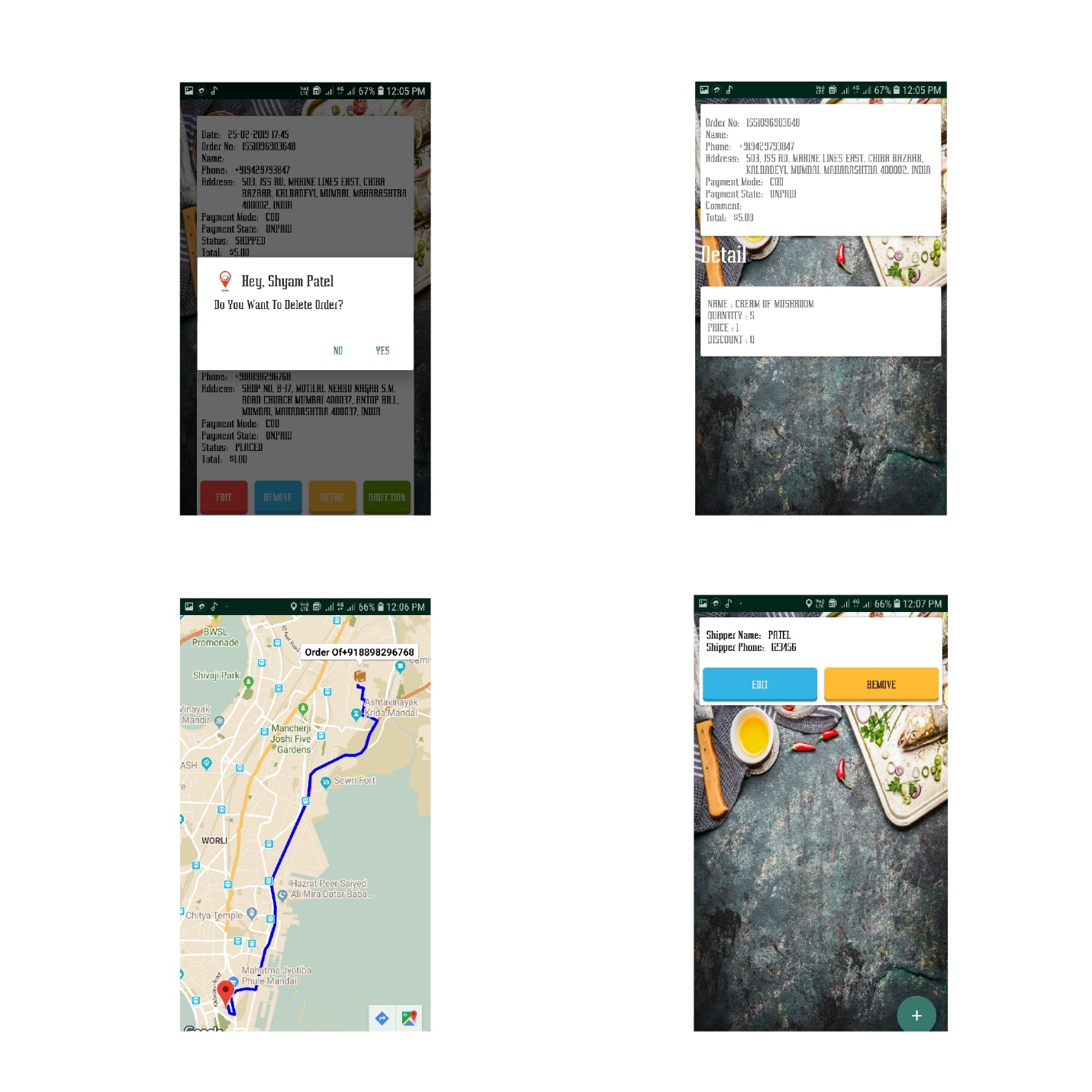
Admin can perform all create, delete, update actions in food category and food items. Admin can also create, update and delete banner of food items which is going to reflect in customer app as shown below



Admin can send broadcast message to all user about offers and other related information, admin can receive orders from customer app and he/she can update order in 3 ways i.e. Placed, On Process and Shipping which will notify the customers regarding order updates.

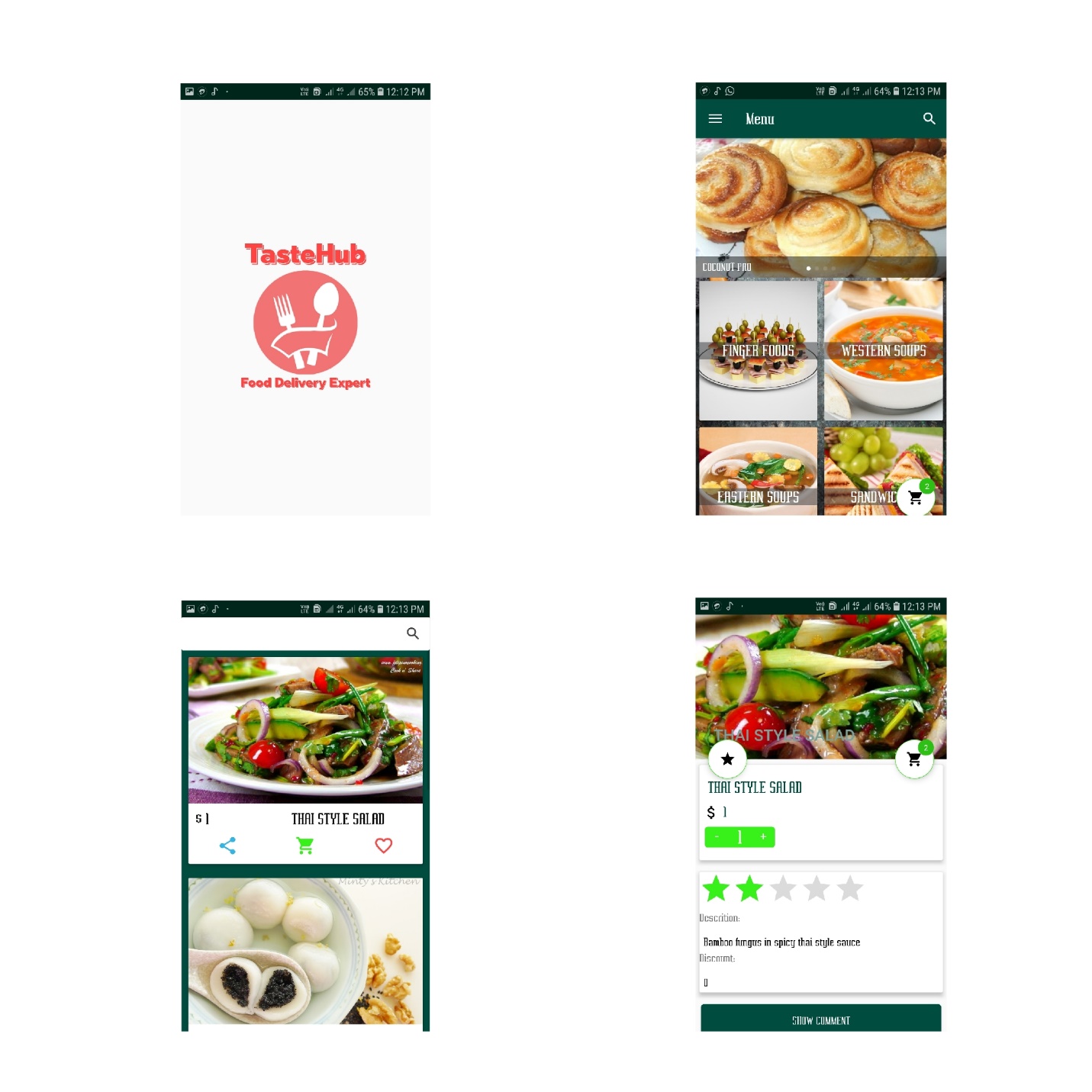


At the same time admin can view the order details and customer details on order receipt, admin can also delete the order and admin can have access to see the directions of order which is placed by particular customer as shown below

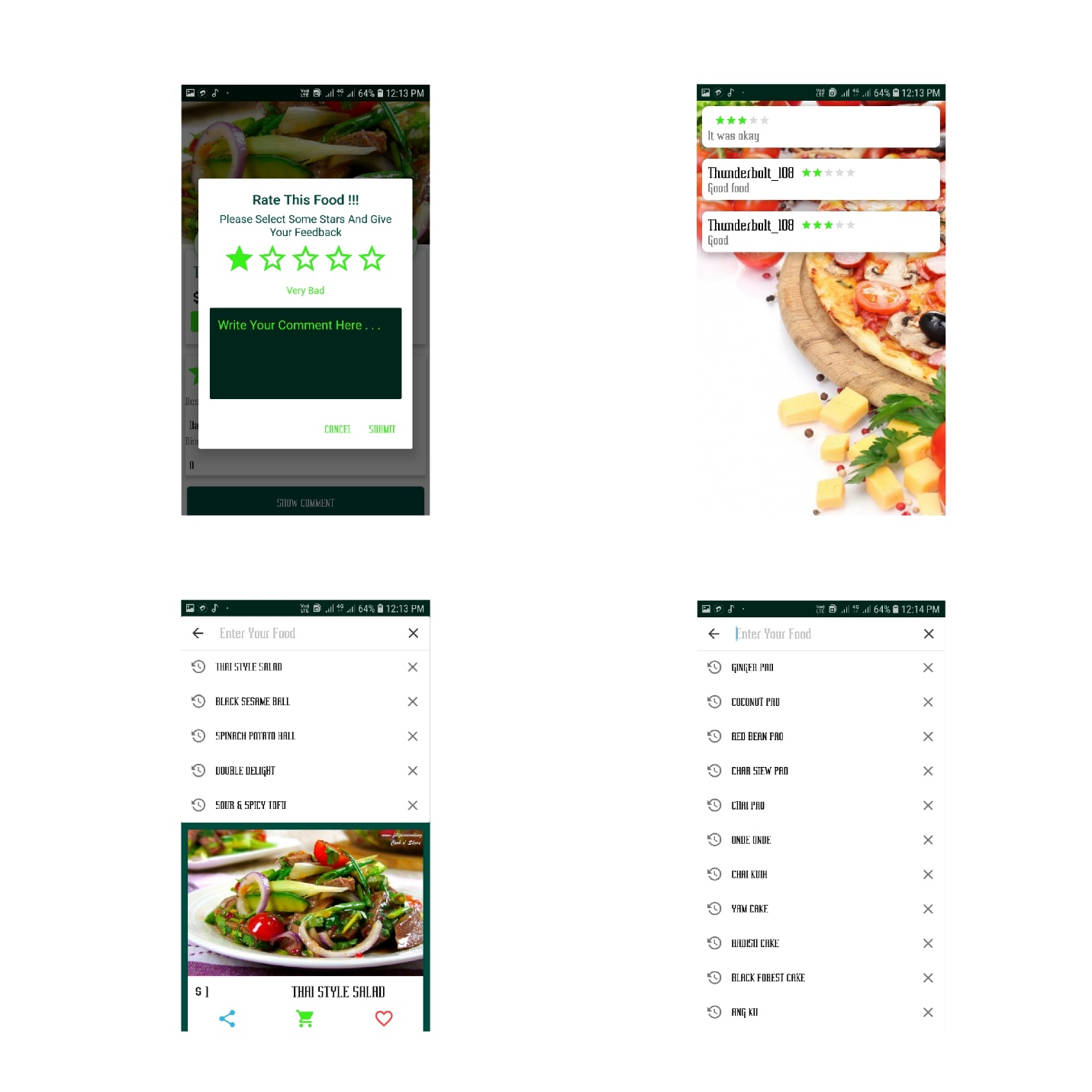


7.2: Customer App

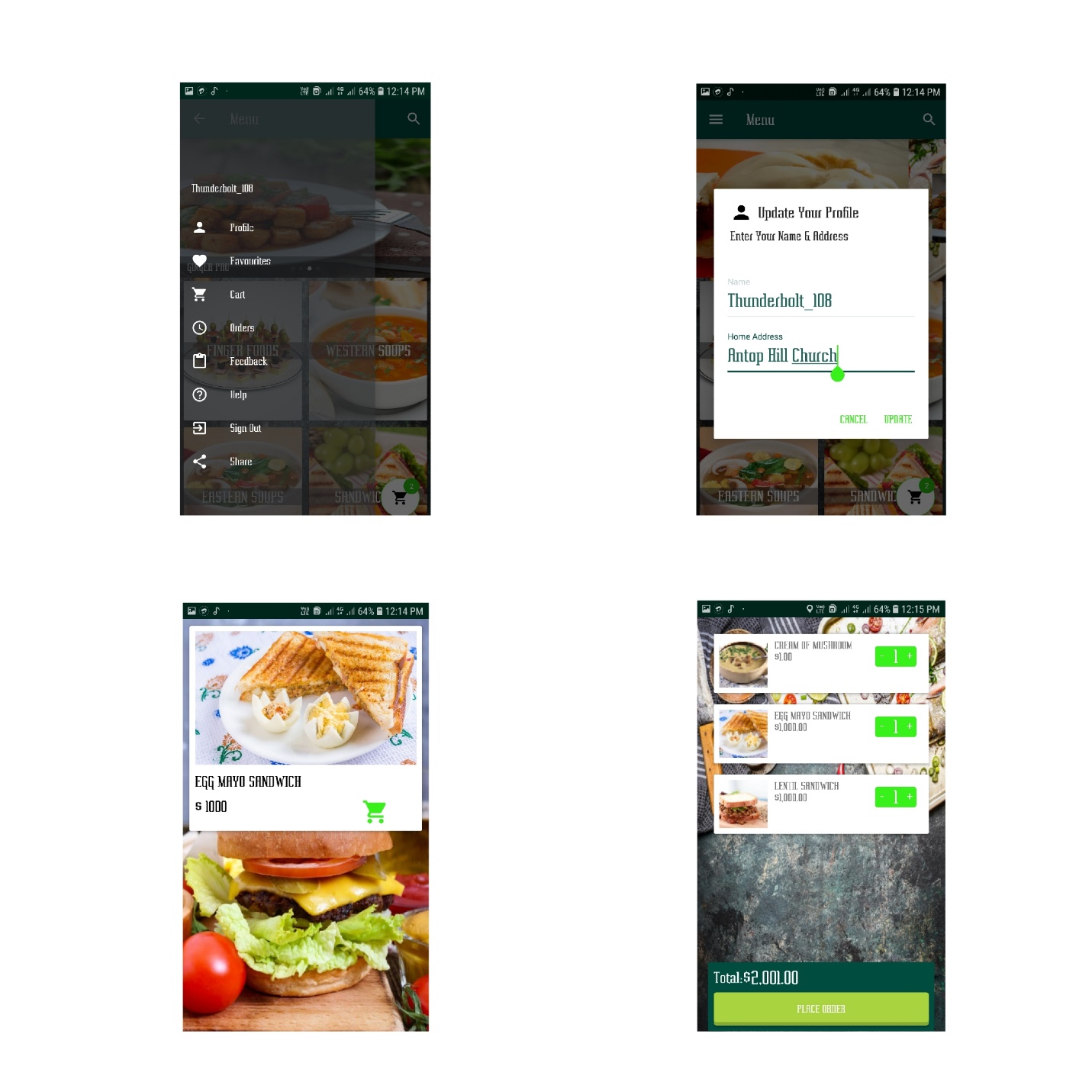
Below is the customer app where it has splash screen with logo, customer can see home page having food category, banners of popular food items. Customers can see the details of particular food items such as description of food, price and discount details as shown below



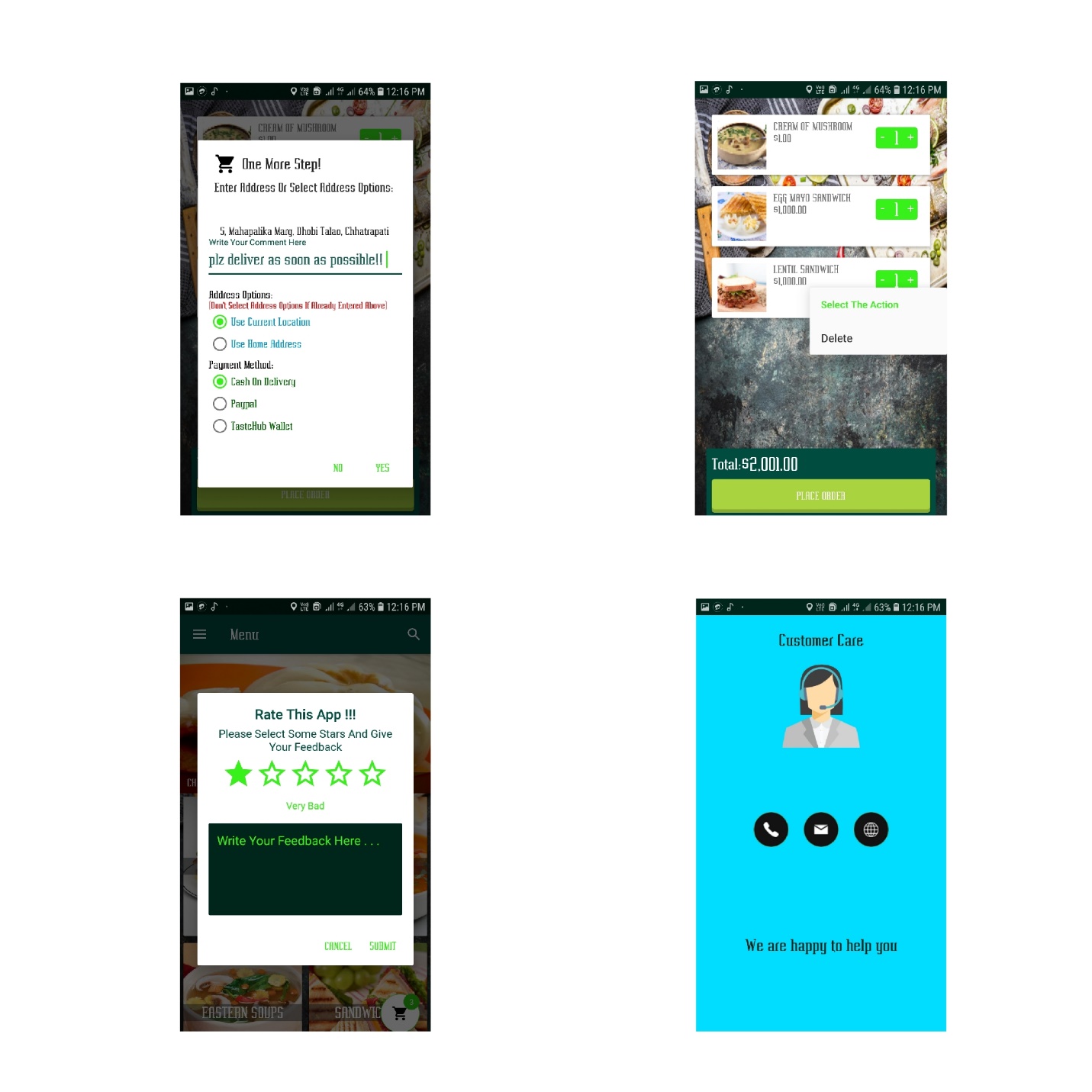
Customer can rate the specific food by giving desired stars to food items and can also comment on food items so that others can see the reviews of particular food. Customer can search food from home page as well as from respective food category as shown below



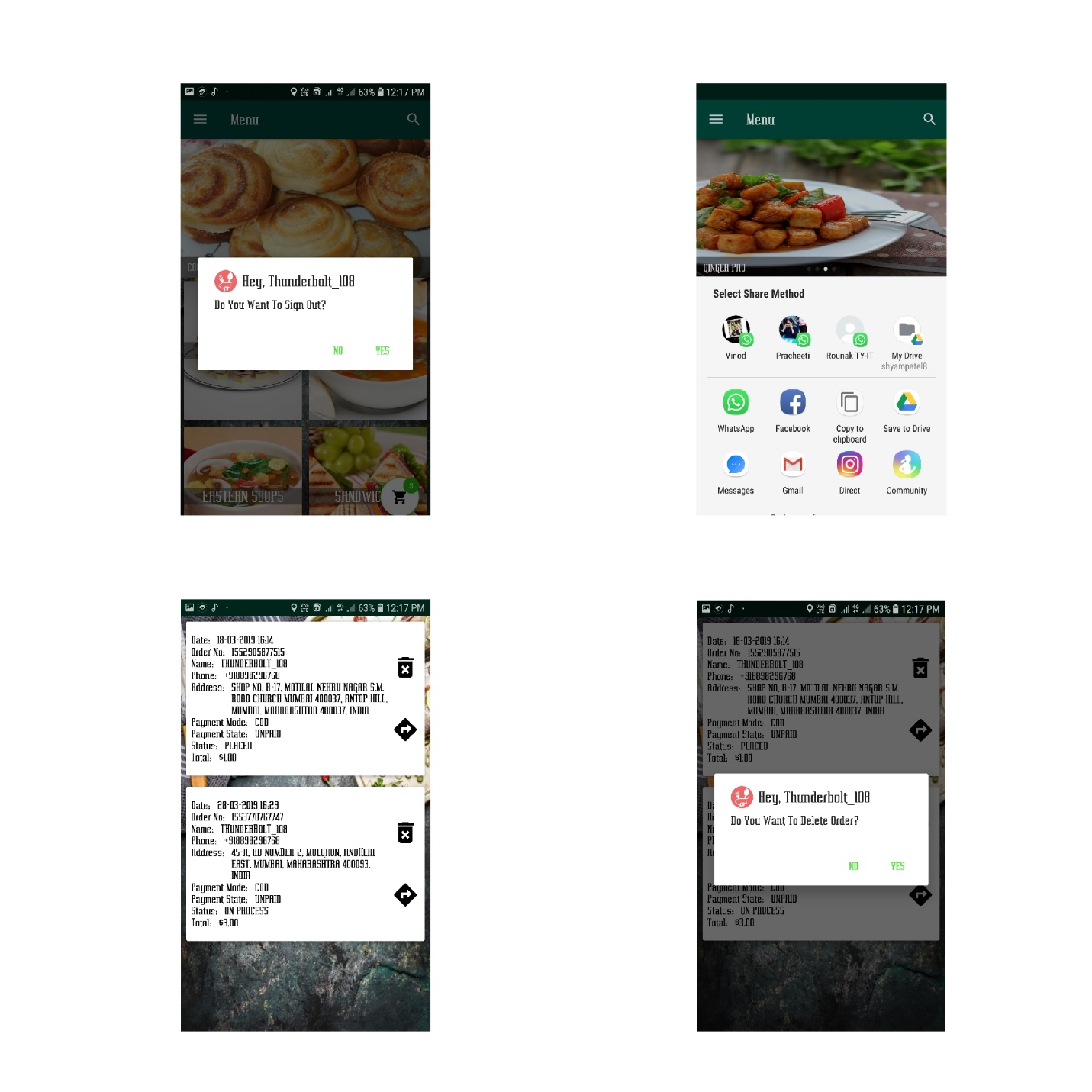
Customer app is very user friendly where he/she can tap on menu button to see all the features of app where he/she can update his/her personal details like name, address, etc. If customer likes any food item then they can keep them into favorites activity where it will be easy for them to place order for the second time, after adding food to cart customer can also update their order such as increase the number of food items, delete the food items.



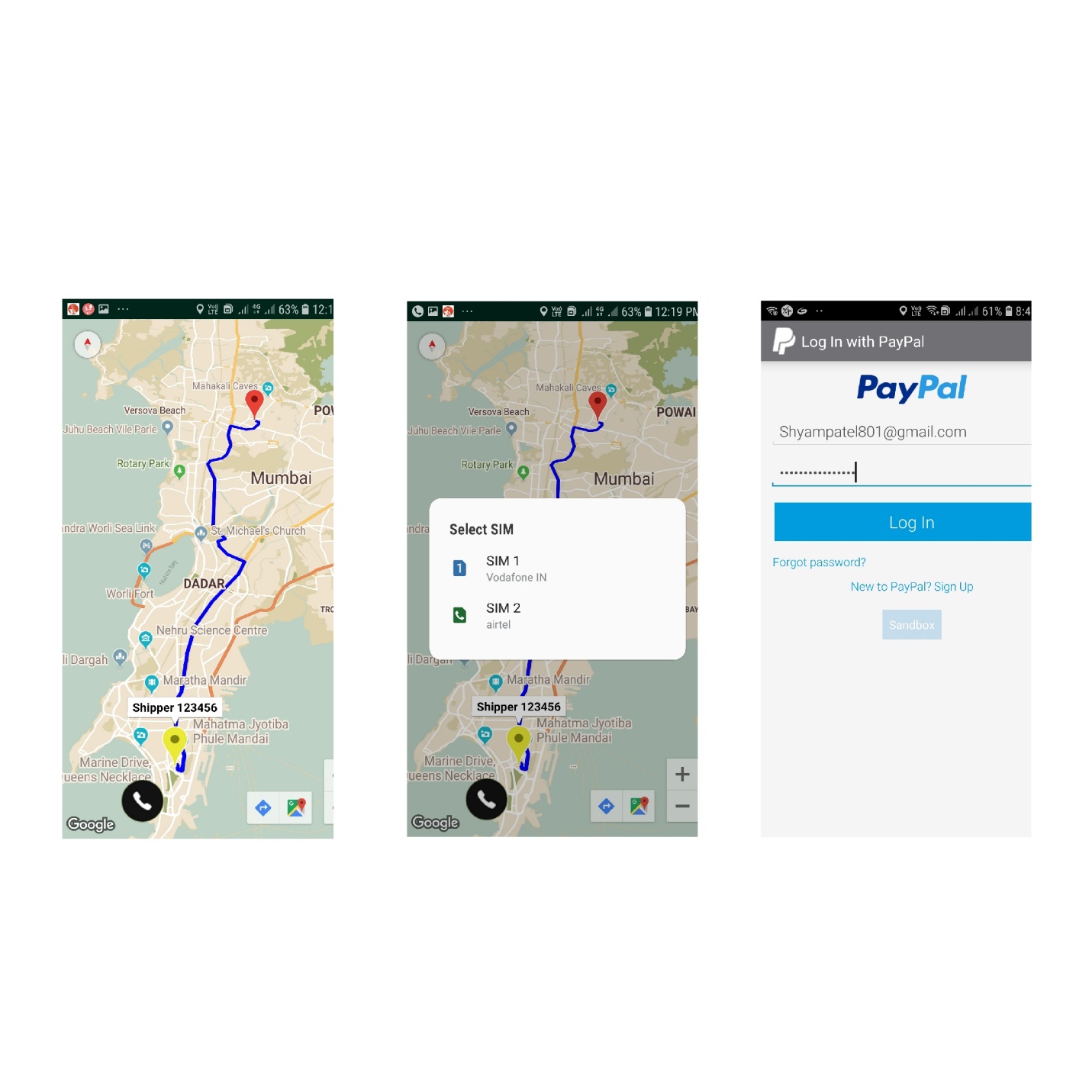
Customer can place order with address options such as they can type desired address, they can select their home address or they can simply use current location which is faster as compared to others. If customers want to give some feedbacks then they can give the feedback by selecting feedback options from menu, also if customers having any problems regarding orders then they can use customer support where admin or its team will reach them shortly.



Customer can delete the food after placing if they do not want the order due to some reasons, customers app provides phone number authentication where they can use their phone number to login to the app and later they can sign out after placing order meanwhile if any customer would like to share app with their friends, family then he/she can share the app with just one tap.

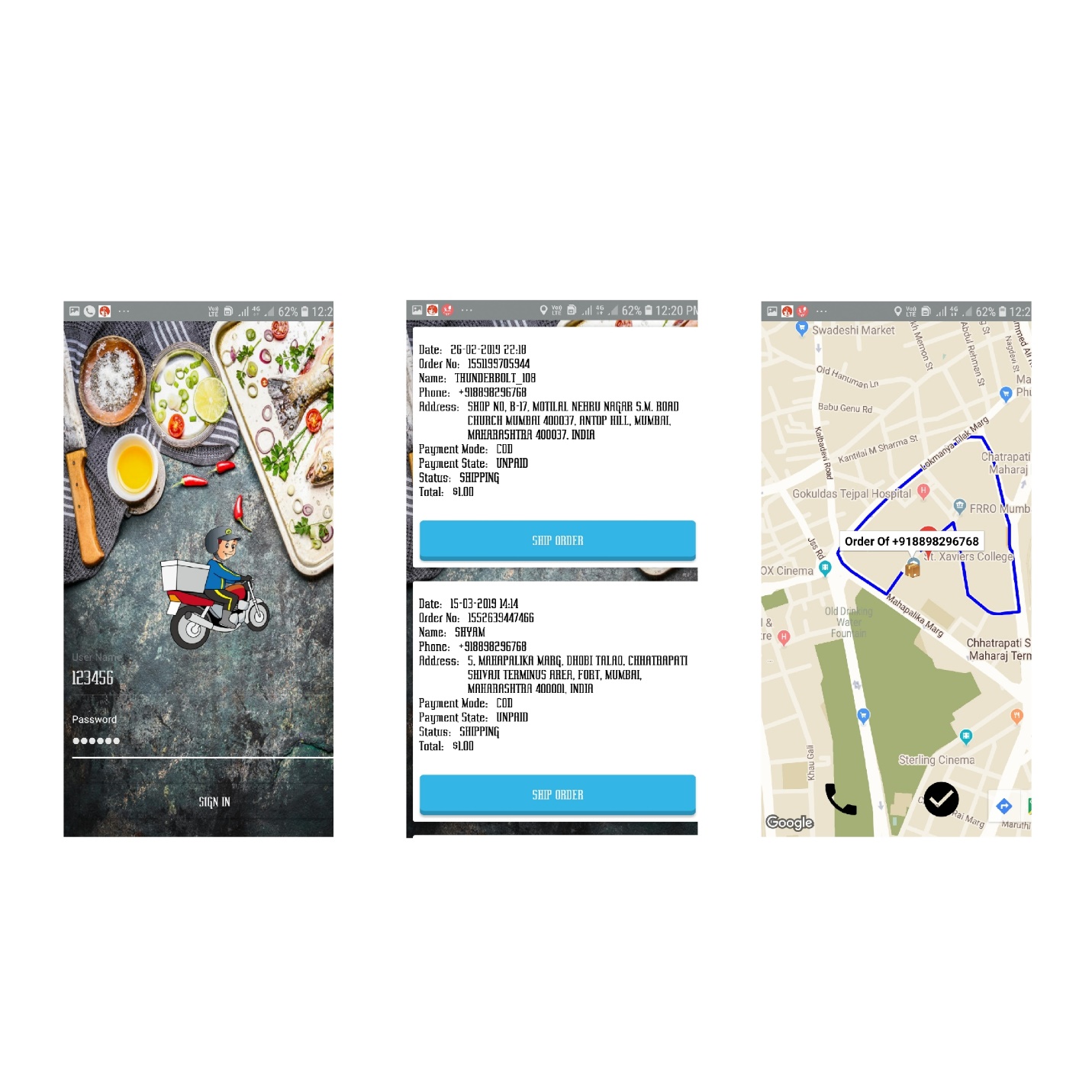


Customer can track the order where he/she will be able to see the real time tracking of shipper meanwhile customer can also can shipper if any problem happened. Customer can also make online payments through Paypal and also cash on delivery is available if user do not want to pay before order get placed.



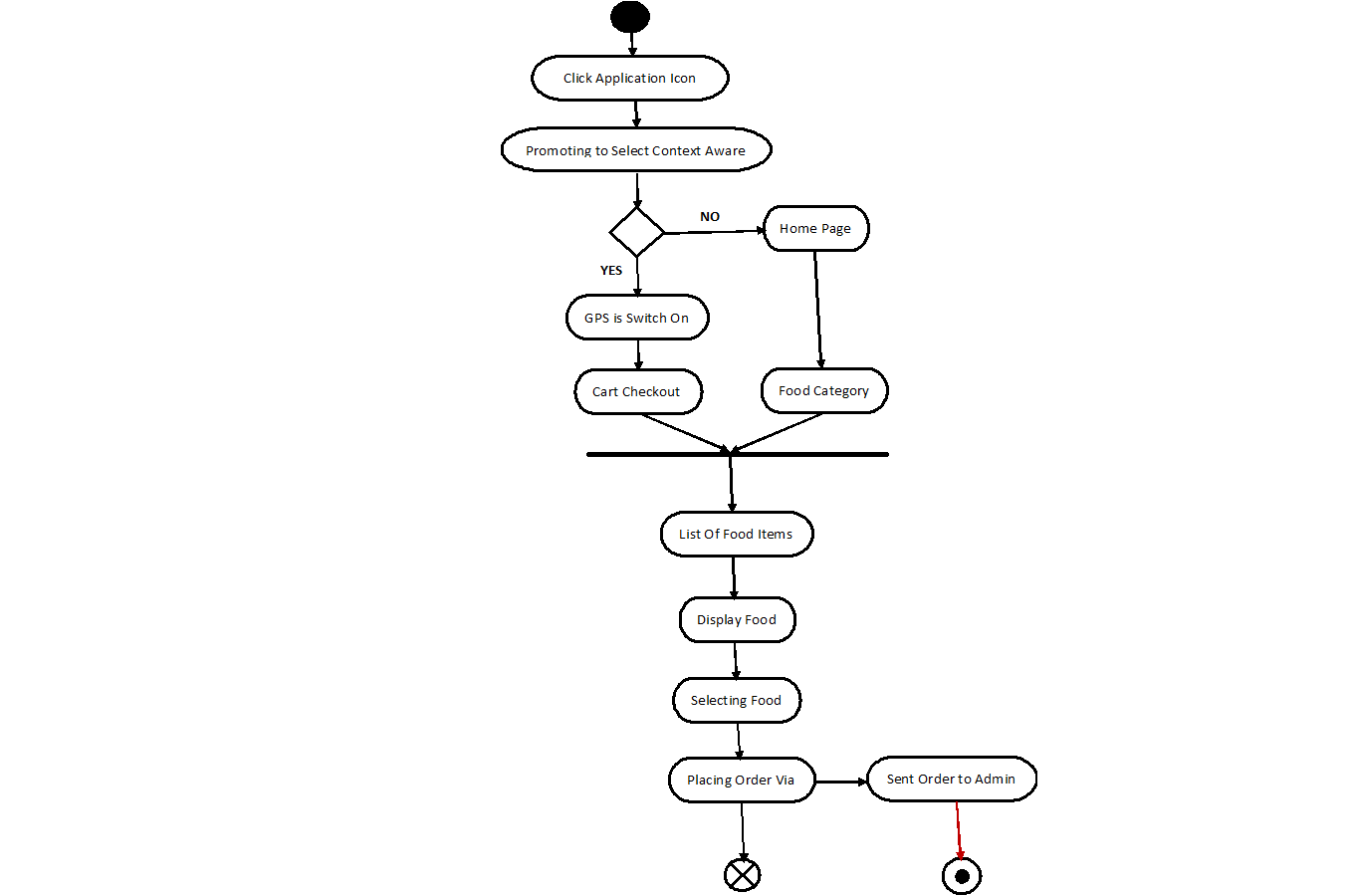
7.3: Shipper App

Shippers are created by admin where admin can forward the orders to specified shipper and that shipper will get the order with customer’s order details. Shipper can also track the customer address which will be easier for shipper to deliver the order as soon as possible, shipper can call customer if he need any help for directions.



CHAPTER 8.0: DIAGRAM AND VISUALISATION

8.1: Activity Diagram



8.2: Class Diagram

|  |
| --- |
| **Welcome PopUp** |
|  |
|  |

|  |
| --- |
| **FoodMenu** |
|  |
| +displayListFood() |

|  |
| --- |
| **Welcome PopUp** |
|  |
|  |

|  |
| --- |
| **ListOfCategory** |
| +foodId  +category  +menuId |
| +onCreate() |

|  |
| --- |
| **Home** |
| -Food Category  -Cart  -Banner  -Other Settings |
|  |

|  |
| --- |
| **StorageHelper** |
| +Cotext |
| +Create Tables() |

|  |
| --- |
| **PlaceOrder** |
|  |
| +placeOrder() |

|  |
| --- |
| **FoodItems** |
| +foodName  +price  +description  +discount |
|  |

|  |
| --- |
| **SendSMS** |
| +ReceviedFoodSelected |
|  |

8.3: Collaboration Diagram

|  |
| --- |
| 2) If Cart Items Found Then |

|  |
| --- |
| 1)User Selects Context Aware Computing() |

|  |
| --- |
| 3) Selecting Food And Placing Order |

|  |
| --- |
| 4) Sending SMS to Admin For Ordering |

|  |
| --- |
| 5) If User Clicks On |

Food Menu

Hotel

User

Cart Iems

Category

GPS

8.4: Sequence Diagram

Admin

foodMenu

Category

foodList

GPS

user

|  |
| --- |
| 1. User Selects Contents |

|  |
| --- |
| 1. If Category Found Then |

|  |
| --- |
| 3) SelectionFound and PlaceOrder |

|  |
| --- |
| 1. If User Clicks On Category |

|  |
| --- |
| 4) Sending SMS to Admin |

8.5: Case Diagram

|  |
| --- |
| User |

|  |
| --- |
| Admin |

CHAPTER 9.0: Car Booking App

9.1: Introduction

As the technology increasing day by day most of the work is made easier by software applications run on mobile. We are living in the 21st century which is known as the “Mobile World”, where almost every kind of work is carried out with the help of mobiles. In this project, we are introducing a concept of having remote access to book a vehicle of our desired wish for a long tour or short tour with payment through our mobile application which is supported by Android operating system. Our Booking Vehicle application is designed with the registration facility in which the customers are registered with personal details. This application is designed with functionalities such as Login, Drive mode, (4 wheeler), Booking Report, Online Payment, List of the vehicles and their costs, Access.

The purpose of this Booking Vehicle project is to describe the Driver4U Android Application.

The Driver4U application is a android application where all information about the vehicles (4 wheelers) which customers want for long tours and short tours. The problem mostly is most people want to drive their vehicles by their own and some want compulsorily want drivers. Driver4U application provide both facilities. And the main feature of the application as it provides both 4 wheelers according to the users choice.

It provides a friendly environment to maintain the details of vehicles, Availability details, Images of vehicles conducted, List of the drivers and their contact details, User Access,  Updating the vehicles details and showing the vehicles to user boarding to the selected boarding point. The main purpose of this project is to maintain easy Booking of Vehicles for user desired Transportation.

9.2: Motivation

In this project, we are introducing a concept of having remote access to book a vehicle of our desired wish for a  long tour or short tour with payment through our mobile application which is supported by android operating system. Our application is designed with the registration facility in which the customers are registered with personal details. This application is designed with functionalities such as Login, Drive mode, (4 wheeler), Booking Report, Online Payment, Driver details, List of the vehicles and their costs, Access. Whereas an administrator can log in, verify, modify and update the vehicle details, upload driver details through a Driver4U.

9.3: ProblemDefinition

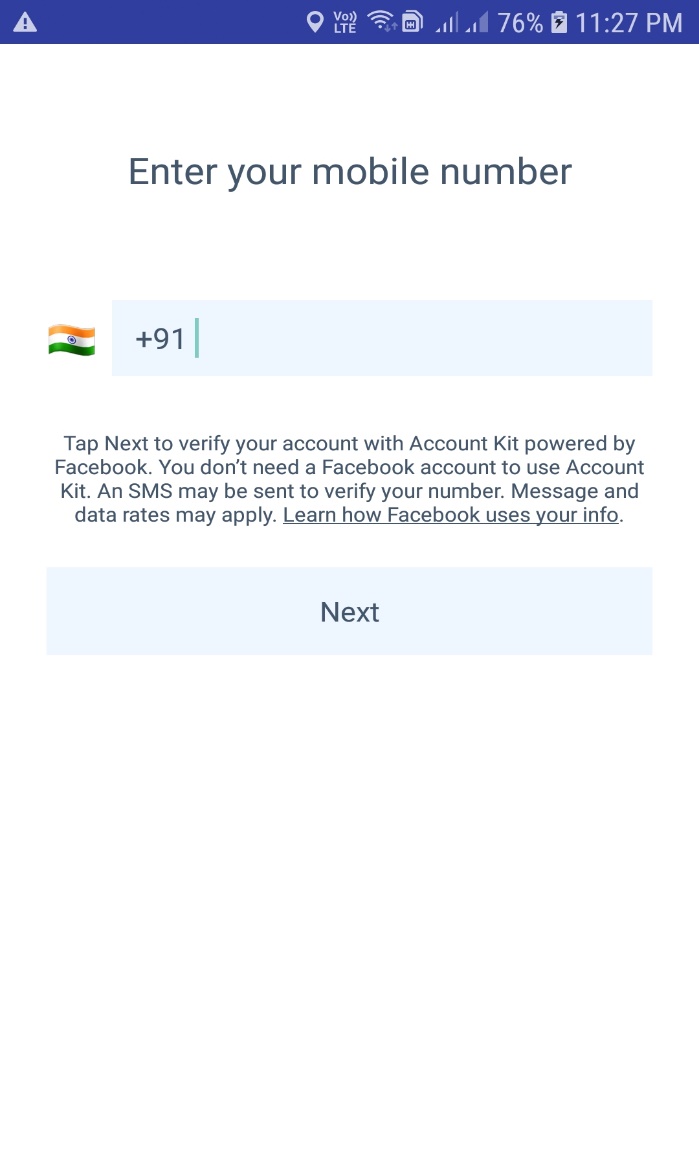
In the present scenario, whenever we want to book a vehicle it takes more payment and no user satisfaction. Existing system existing not having any application with the selection of users desired vehicle. The selection of the driver also not available in the existing systems. The travelers face a lot of distress with the driver and there is no mutual understanding between the driver and passenger. Self-driving option may not available in the existing system, it enables users to handle the tour according to their wish.

9.4:Objective

In this Booking Vehicle project, we are providing an individual account for users with a username and a unique password by giving them registration options. They can log in into their own account through the internet from any remote access area. Driver4U is available in a mobile which runs on Android OS as an Application on your fingertips. This Application has all facilities for the user to select a vehicle, select a driver, book a vehicle, start a trip, end the trip, payment through online. Login facility both for user and admin for viewing, modifying vehicles.

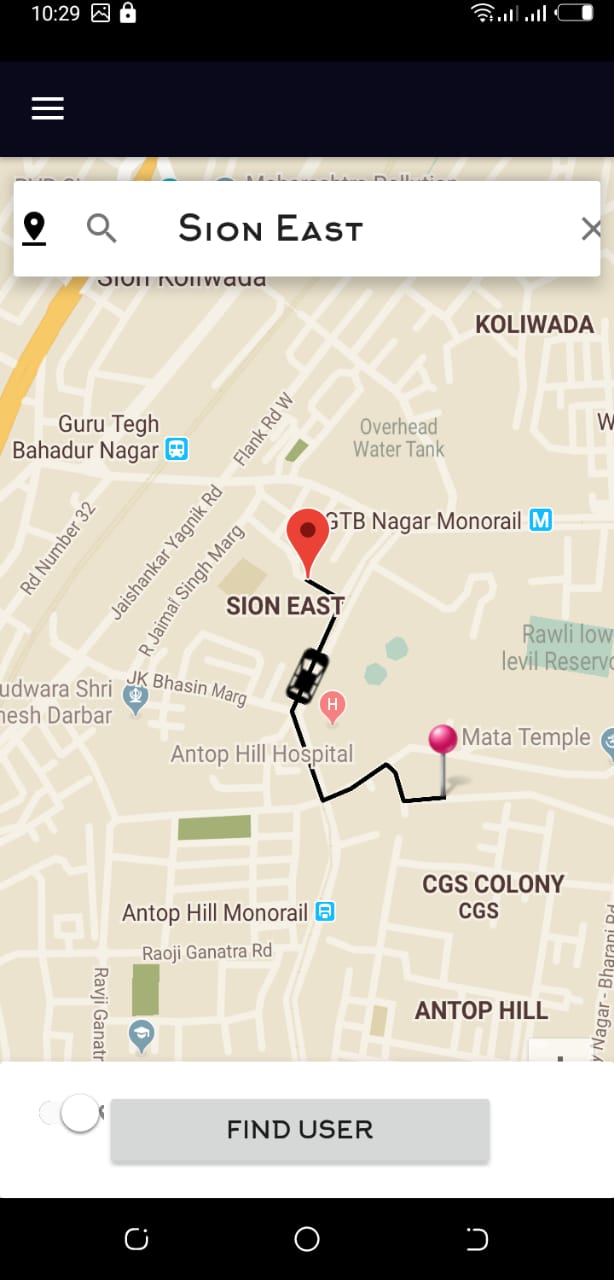
9.5: Login Module

The purpose of this module is to provide entry into the Driver4U application. Based on the type of login, the user is provided with various facilities and functionalities. The main function of this module is to allow the user to use the event information system.



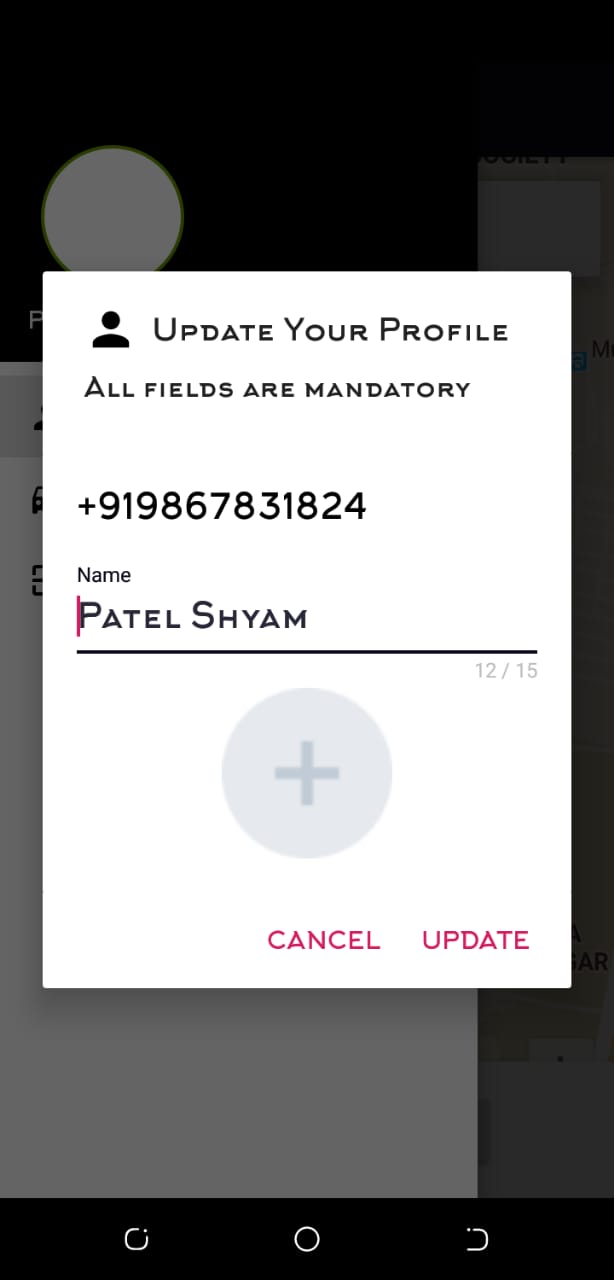
9.6: Administrator Module

In this module, if the administrator enters a particular phone number, then that admin is logged in.



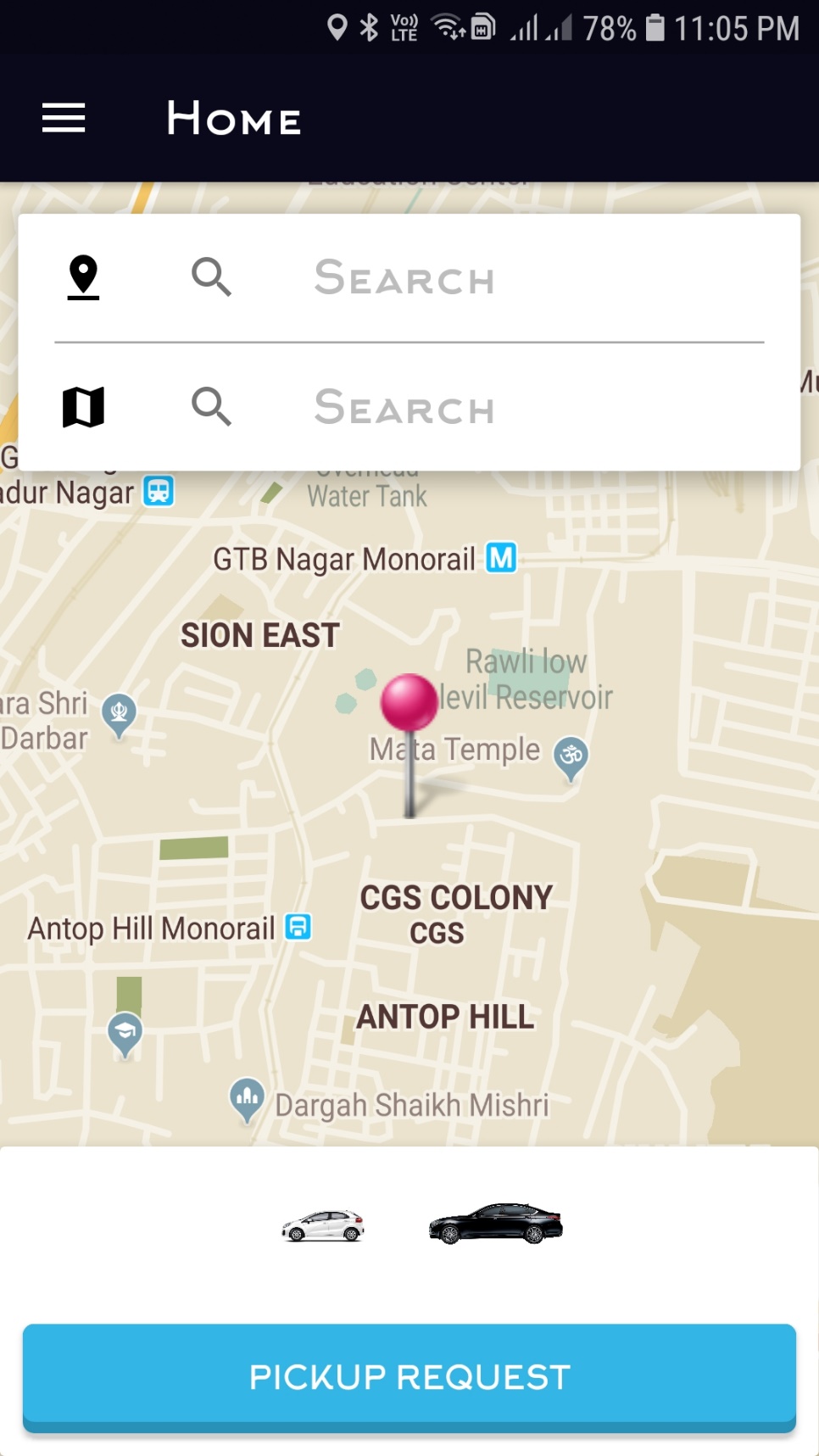
9.7: Updates/Adds/Displays vehicles and drivers list

Admin can modify vehicles details, can update vehicles, and add driver’s details and display cars list, drivers list.



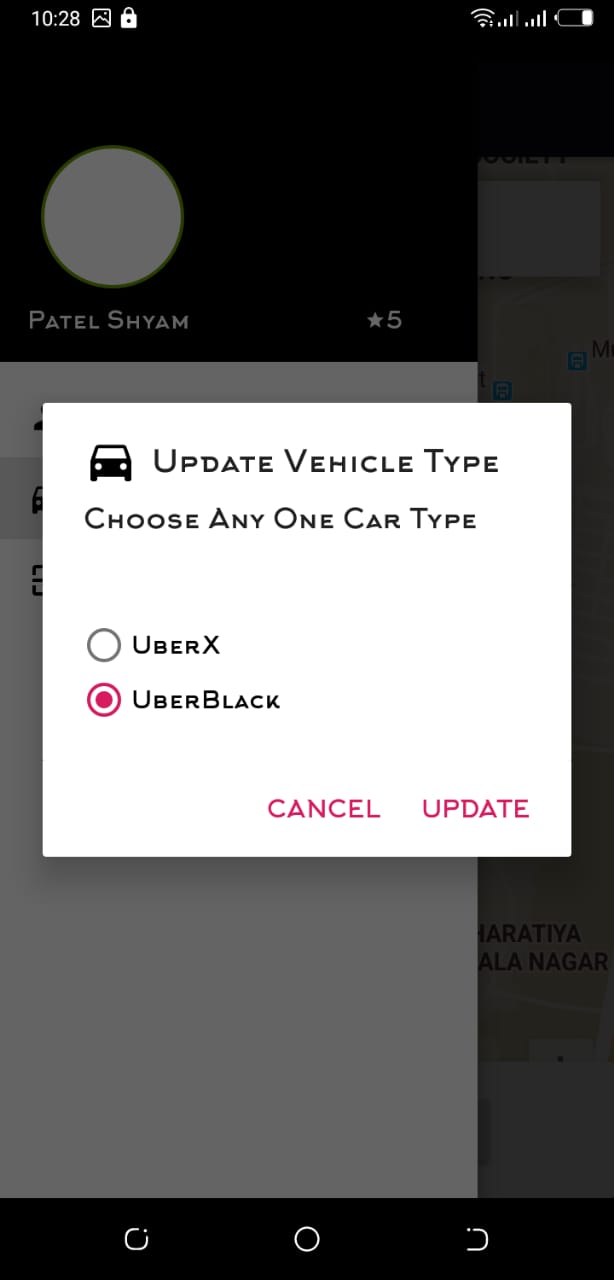
9.8: User Module

In this module when a user enters his phone number, then he can visit the entire following page:



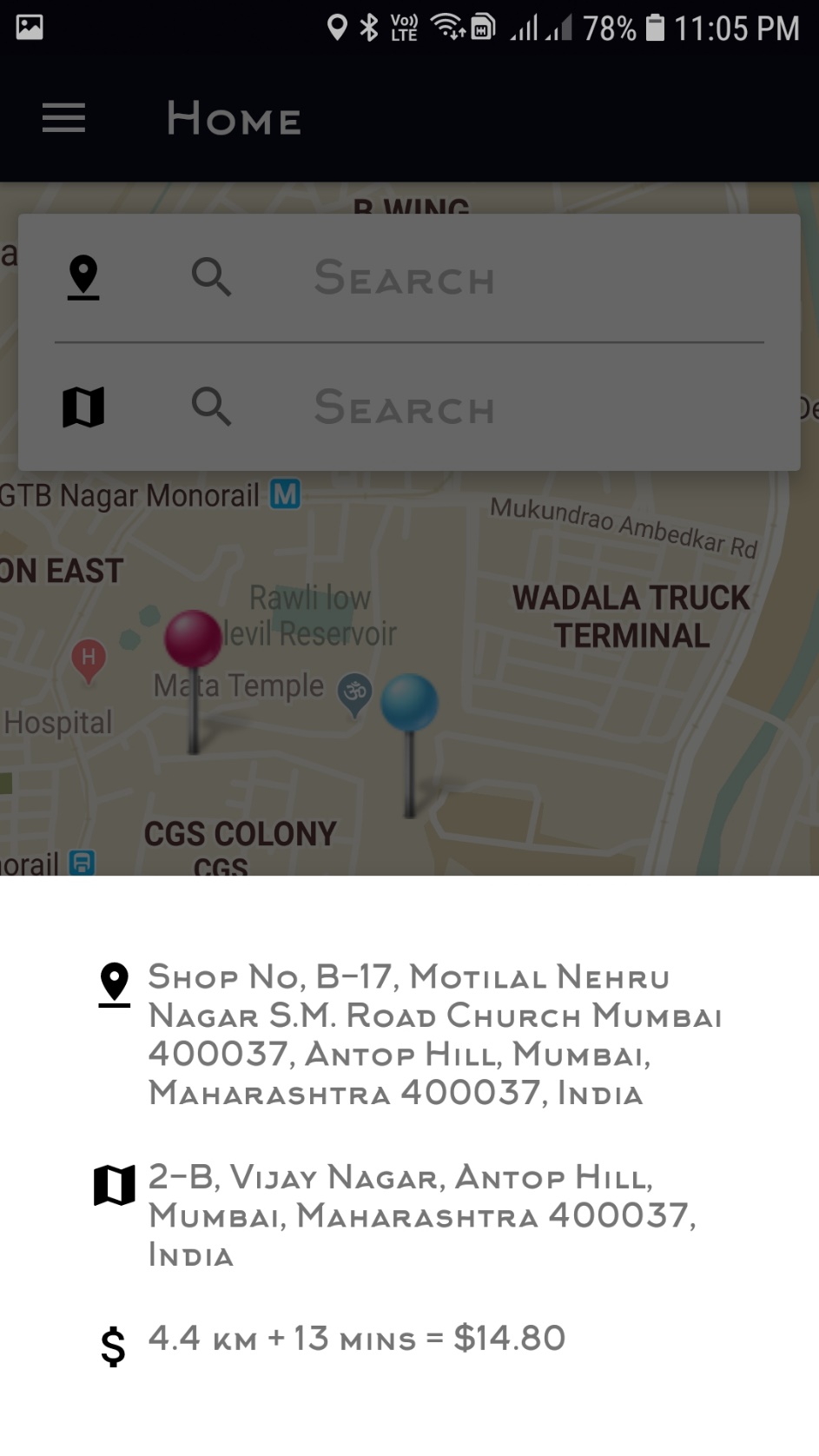
9.9: Selection of Drive Mode

When the driver clicks on vehicle type then he/she will get two options of car types. They may select any one.



9.10: Selection of point area

When the user tap on this map, the source and destination is displayed here along with ride cost. They can select any destination by tapping on map.



9.11: Cars View

When the user tap on the view continue button, cars in the selected area are displayed.



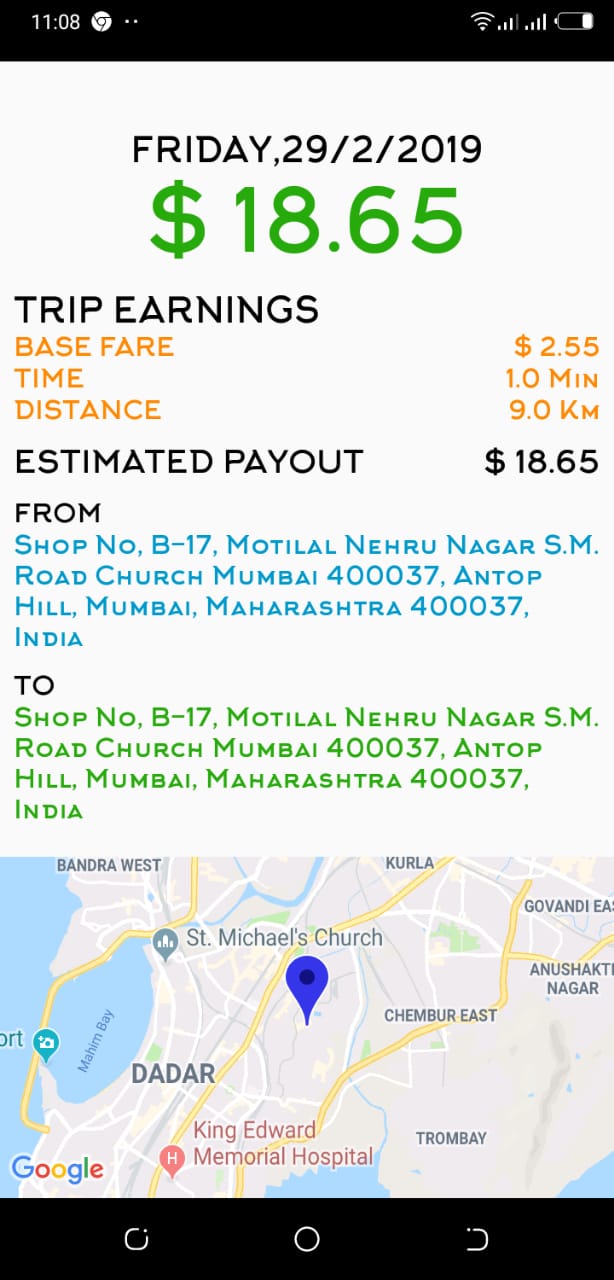
9.12: Drivers View

This displays drivers call where distance and address of the rider appeared with two buttons ACCEPT and DECLINE along with 30 sec timer



9.13: Payment Success View

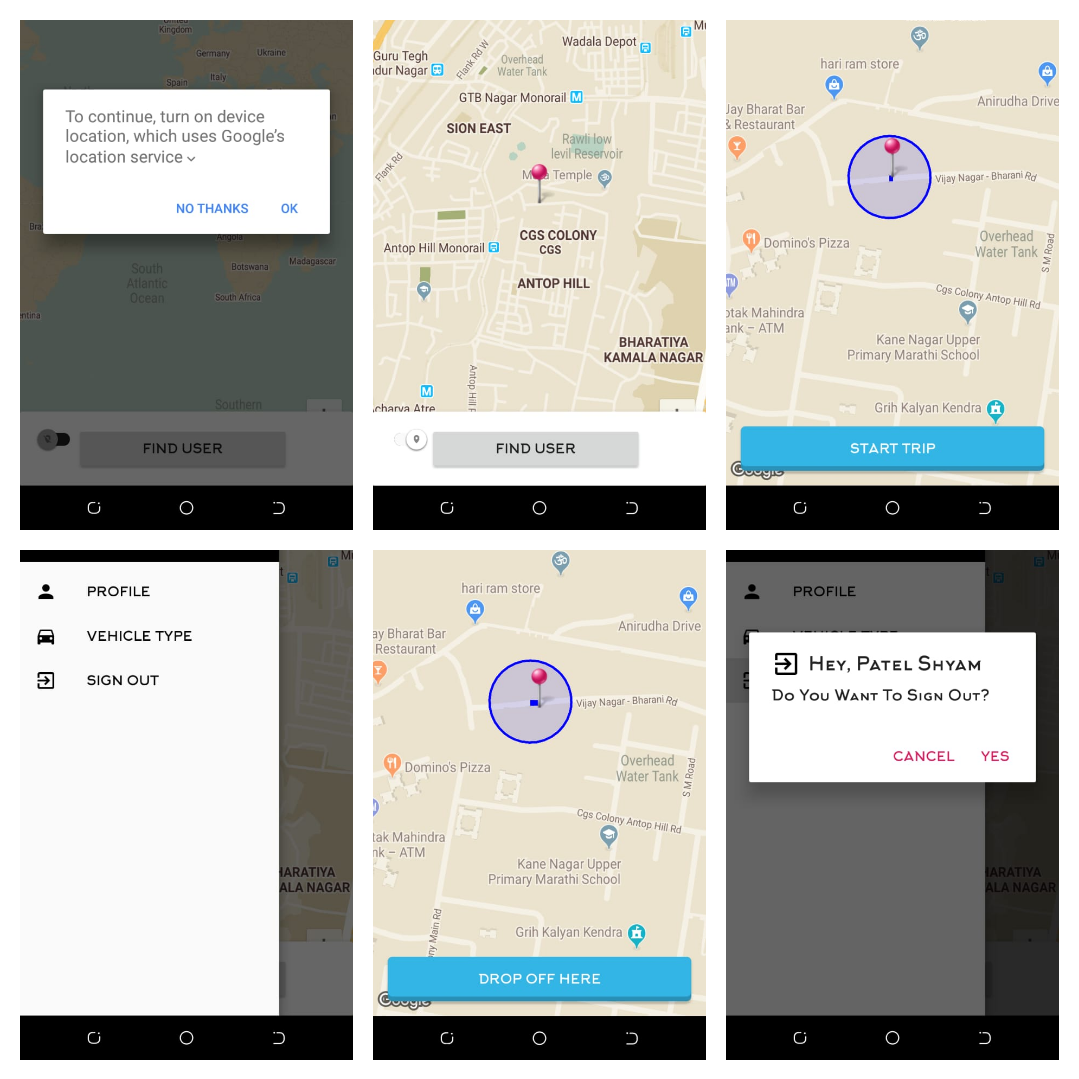
This notifies the user with payment successful text and a button to drive to home again.



9.14: Functionalities

User Accessing application

The user is of center focus; because in every application, the user plays a very important role.Users can have access to view only if they log in. Users can access the information of the vehicles, select a vehicle, and Select a driver, Start and stop a trip, make payment.

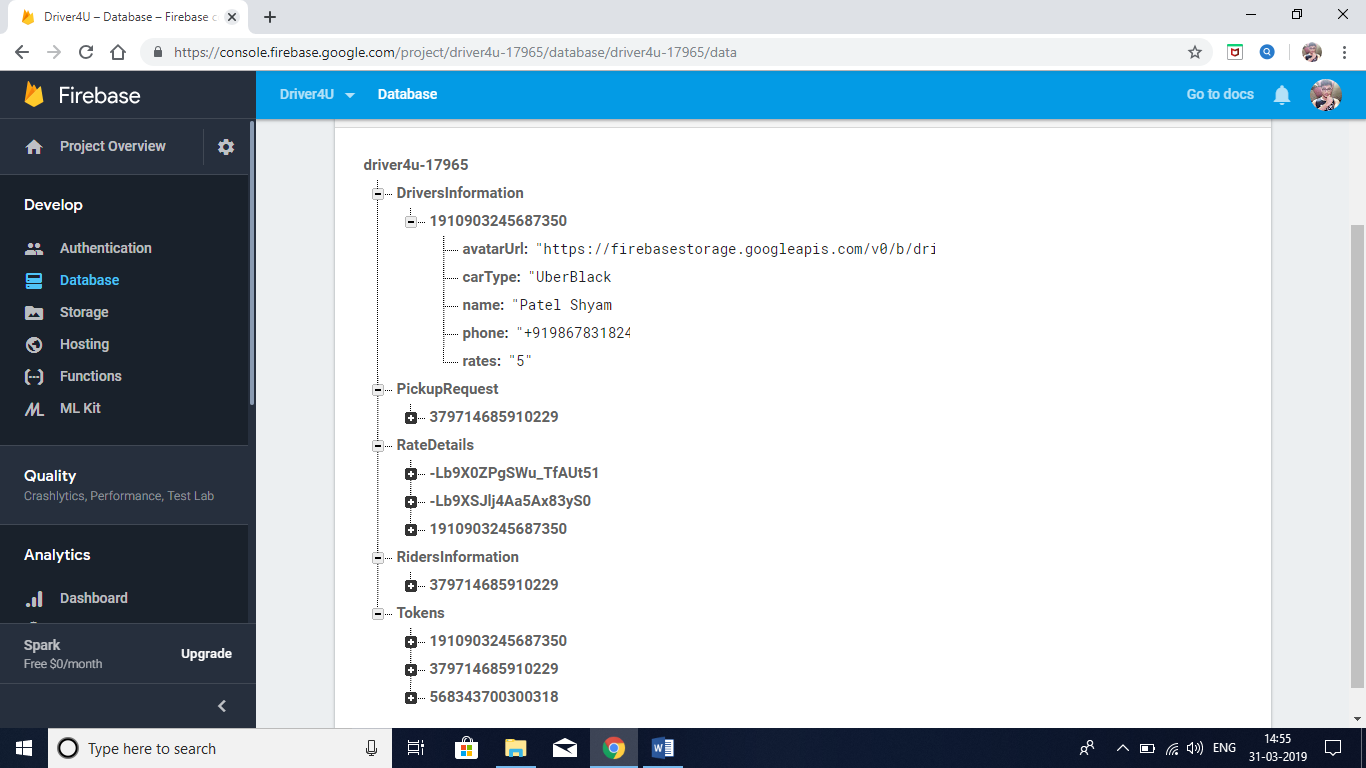


CHAPTER 10.0: DATABASES

10.1: Food Delivery App



10.2: Car Booking App



CHAPTER 11.0: SYSTEM CODING

11.1: Food Delivery App

11.1.1: MainActivity.java

**package** com.example.admin.cafeapp;  
  
**import** android.app.AlertDialog;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.Intent;  
**import** android.content.pm.PackageInfo;  
**import** android.content.pm.PackageManager;  
**import** android.content.pm.Signature;  
**import** android.graphics.Color;  
**import** android.graphics.Typeface;  
**import** android.support.annotation.NonNull;  
**import** android.support.annotation.Nullable;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.util.Base64;  
**import** android.util.Log;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.example.admin.cafeapp.Common.Common;  
**import** com.example.admin.cafeapp.Model.User;  
**import** com.facebook.FacebookSdk;  
**import** com.facebook.accountkit.Account;  
**import** com.facebook.accountkit.AccountKit;  
**import** com.facebook.accountkit.AccountKitCallback;  
**import** com.facebook.accountkit.AccountKitError;  
**import** com.facebook.accountkit.AccountKitLoginResult;  
**import** com.facebook.accountkit.ui.AccountKitActivity;  
**import** com.facebook.accountkit.ui.AccountKitConfiguration;  
**import** com.facebook.accountkit.ui.LoginType;  
**import** com.google.android.gms.tasks.OnCompleteListener;  
**import** com.google.android.gms.tasks.Task;  
**import** com.google.firebase.database.DataSnapshot;  
**import** com.google.firebase.database.DatabaseError;  
**import** com.google.firebase.database.DatabaseReference;  
**import** com.google.firebase.database.FirebaseDatabase;  
**import** com.google.firebase.database.ValueEventListener;  
**import** com.rengwuxian.materialedittext.MaterialEditText;  
  
**import** java.security.MessageDigest;  
**import** java.security.NoSuchAlgorithmException;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**import** cc.cloudist.acplibrary.ACProgressConstant;  
**import** cc.cloudist.acplibrary.ACProgressFlower;  
**import** dmax.dialog.SpotsDialog;  
**import** info.hoang8f.widget.FButton;  
**import** io.paperdb.Paper;  
**import** retrofit2.Call;  
**import** retrofit2.Callback;  
**import** retrofit2.Response;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyConfig;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyContextWrapper;  
  
  
**public class** MainActivity **extends** AppCompatActivity {  
 FButton **btnContinue**;  
 TextView **txtslogan**;  
 **private static final int *REQUEST\_CODE***=7171;  
  
 FirebaseDatabase **database**;  
 DatabaseReference **users**;  
  
 @Override  
 **protected void** attachBaseContext(Context newBase) {  
 **super**.attachBaseContext(CalligraphyContextWrapper.*wrap*(newBase));  
 }  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 CalligraphyConfig.*initDefault*(**new** CalligraphyConfig.Builder()  
 .setDefaultFontPath(**"fonts/restaurant\_font.otf"**)  
 .setFontAttrId(R.attr.***fontPath***)  
 .build());  
 setContentView(R.layout.***activity\_main***);  
  
 FacebookSdk.*sdkInitialize*(getApplicationContext());  
  
 AccountKit.*initialize*(**this**);  
  
 *//Init* **database**=FirebaseDatabase.*getInstance*();  
 **users**=**database**.getReference(**"User"**);  
  
 **btnContinue**=(FButton)findViewById(R.id.***btn\_continue***);  
  
 **txtslogan**=(TextView)findViewById(R.id.***txtSlogan***);  
 Typeface typeface=Typeface.*createFromAsset*(getAssets(),**"fonts/Vahika.ttf"**);  
 **txtslogan**.setTypeface(typeface);  
  
 **btnContinue**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 startLoginSystem();  
  
 }  
 });  
  
 *//Check session facebook account kit* **if** (AccountKit.*getCurrentAccessToken*() !=**null**)  
 {  
 **final** AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(MainActivity.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Please Wait . . ."**);  
  
 AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 *//Login* **users**.child(account.getPhoneNumber().toString())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 User localUser=dataSnapshot.getValue(User.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, Home.**class**);  
 Common.*currentUser* = localUser;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
  
 }  
 }  
  
 **private void** startLoginSystem() {  
 Intent intent=**new** Intent(MainActivity.**this**, AccountKitActivity.**class**);  
 AccountKitConfiguration.AccountKitConfigurationBuilder configurationBuilder=  
 **new** AccountKitConfiguration.AccountKitConfigurationBuilder(LoginType.***PHONE***,  
 AccountKitActivity.ResponseType.***TOKEN***);  
 intent.putExtra(AccountKitActivity.***ACCOUNT\_KIT\_ACTIVITY\_CONFIGURATION***,configurationBuilder.build());  
 startActivityForResult(intent,***REQUEST\_CODE***);  
 }  
  
 @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, @Nullable Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
  
 **if** (requestCode==***REQUEST\_CODE***)  
 {  
 AccountKitLoginResult result=data.getParcelableExtra(AccountKitLoginResult.***RESULT\_KEY***);  
 **if** (result.getError()!=**null**)  
 {  
 Toast.*makeText*(**this**, **""**+result.getError().getErrorType().getMessage(), Toast.***LENGTH\_SHORT***).show();  
 **return**;  
 }  
 **else if**(result.wasCancelled())  
 {  
 Toast.*makeText*(**this**, **"Cancel"**, Toast.***LENGTH\_SHORT***).show();  
 **return**;  
 }  
 **else** {  
 **if** (result.getAccessToken()!=**null**){  
  
 *//show Dialog* **final** AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(MainActivity.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Please Wait . . ."**);  
  
  
 *//Get User Phone and check exists on server* AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(**final** Account account) {  
  
  
 **final** String userPhone=account.getPhoneNumber().toString();  
  
 *//Check user if exists on fire base* **users**.orderByKey().equalTo(userPhone)  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 **if** (!dataSnapshot.child(userPhone).exists())  
 { *//if not exits then create new user* User newUser=**new** User();  
 newUser.setPhone(userPhone);  
 newUser.setName(**""**);  
 newUser.setBalance(String.*valueOf*(0.0));  
  
 *//Add to firebase* **users**.child(userPhone)  
 .setValue(newUser)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful())  
 {  
 }  
  
 *//Login* **users**.child(userPhone)  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 User localUser=dataSnapshot.getValue(User.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, Profile.**class**);  
 Common.*currentUser* = localUser;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
 }  
 });  
  
 }  
 **else** {  
 *//If exists  
  
 //Login* **users**.child(userPhone)  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 User localUser=dataSnapshot.getValue(User.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, Home.**class**);  
 Common.*currentUser* = localUser;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
  
  
 }  
  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
 Toast.*makeText*(MainActivity.**this**, **""**+accountKitError.getErrorType().getMessage(), Toast.***LENGTH\_SHORT***).show();  
  
 }  
 });  
 }  
 }  
 }  
  
 }  
  
 **private void** printKeyHash() {  
 **try**{  
 PackageInfo info=getPackageManager().getPackageInfo(**"com.example.admin.cafeapp"**,  
 PackageManager.***GET\_SIGNATURES***);  
 **for** (Signature signature:info.**signatures**)  
 {  
 MessageDigest md=MessageDigest.*getInstance*(**"SHA"**);  
 md.update(signature.toByteArray());  
 Log.*d*(**"KeyHash"**,Base64.*encodeToString*(md.digest(),Base64.***DEFAULT***));  
  
 }  
 } **catch** (PackageManager.NameNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
}

11.1.2: Home.java

**package** com.example.admin.cafeapp;  
  
**import** android.app.ProgressDialog;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.Intent;  
**import** android.graphics.Color;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.support.annotation.NonNull;  
**import** android.support.annotation.Nullable;  
**import** android.support.v4.widget.SwipeRefreshLayout;  
**import** android.support.v7.app.AlertDialog;  
**import** android.support.v7.widget.GridLayoutManager;  
**import** android.support.v7.widget.RecyclerView;  
**import** android.text.TextUtils;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.support.design.widget.NavigationView;  
**import** android.support.v4.view.GravityCompat;  
**import** android.support.v4.widget.DrawerLayout;  
**import** android.support.v7.app.ActionBarDrawerToggle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.support.v7.widget.Toolbar;  
**import** android.view.Menu;  
**import** android.view.MenuItem;  
**import** android.view.ViewGroup;  
**import** android.view.animation.AnimationUtils;  
**import** android.view.animation.LayoutAnimationController;  
**import** android.widget.CheckBox;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.andremion.counterfab.CounterFab;  
**import** com.daimajia.slider.library.Animations.DescriptionAnimation;  
**import** com.daimajia.slider.library.SliderLayout;  
**import** com.daimajia.slider.library.SliderTypes.BaseSliderView;  
**import** com.daimajia.slider.library.SliderTypes.TextSliderView;  
**import** com.example.admin.cafeapp.Common.Common;  
**import** com.example.admin.cafeapp.Database.MyData;  
**import** com.example.admin.cafeapp.Interface.ItemClickListener;  
**import** com.example.admin.cafeapp.Model.Banner;  
**import** com.example.admin.cafeapp.Model.Category;  
**import** com.example.admin.cafeapp.Model.Feedback;  
**import** com.example.admin.cafeapp.Model.Rating;  
**import** com.example.admin.cafeapp.Model.Token;  
**import** com.example.admin.cafeapp.Model.User;  
**import** com.example.admin.cafeapp.ViewHolder.MenuViewHolder;  
**import** com.facebook.accountkit.Account;  
**import** com.facebook.accountkit.AccountKit;  
**import** com.facebook.accountkit.AccountKitCallback;  
**import** com.facebook.accountkit.AccountKitError;  
**import** com.firebase.ui.database.FirebaseRecyclerAdapter;  
**import** com.firebase.ui.database.FirebaseRecyclerOptions;  
**import** com.google.android.gms.tasks.OnCompleteListener;  
**import** com.google.android.gms.tasks.OnSuccessListener;  
**import** com.google.android.gms.tasks.Task;  
**import** com.google.firebase.database.DataSnapshot;  
**import** com.google.firebase.database.DatabaseError;  
**import** com.google.firebase.database.DatabaseReference;  
**import** com.google.firebase.database.FirebaseDatabase;  
**import** com.google.firebase.database.ValueEventListener;  
**import** com.google.firebase.iid.FirebaseInstanceId;  
**import** com.google.firebase.messaging.FirebaseMessaging;  
**import** com.rengwuxian.materialedittext.MaterialEditText;  
**import** com.squareup.picasso.Picasso;  
**import** com.stepstone.apprating.AppRatingDialog;  
**import** com.stepstone.apprating.listener.RatingDialogListener;  
  
**import** org.jetbrains.annotations.NotNull;  
  
**import** java.util.Arrays;  
**import** java.util.HashMap;  
**import** java.util.Map;  
**import** java.util.UUID;  
  
**import** cc.cloudist.acplibrary.ACProgressConstant;  
**import** cc.cloudist.acplibrary.ACProgressFlower;  
**import** io.paperdb.Paper;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyConfig;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyContextWrapper;  
  
**public class** Home **extends** AppCompatActivity  
 **implements** NavigationView.OnNavigationItemSelectedListener, RatingDialogListener {  
  
 FirebaseDatabase **database**;  
 FirebaseRecyclerAdapter **adapter**;  
 DatabaseReference **category**;  
 TextView **txtFullName**;  
 RecyclerView **recycler\_menu**;  
 RecyclerView.LayoutManager **layoutManager**;  
  
 SwipeRefreshLayout **swipeRefreshLayout**;  
  
 CounterFab **fab**;  
 **private long backPressedTime**;  
  
 *//Slider* HashMap<String,String> **image\_list**;  
 SliderLayout **mSlider**;  
  
 @Override  
 **protected void** attachBaseContext(Context newBase) {  
 **super**.attachBaseContext(CalligraphyContextWrapper.*wrap*(newBase));  
 }  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 CalligraphyConfig.*initDefault*(**new** CalligraphyConfig.Builder()  
 .setDefaultFontPath(**"fonts/Vahika.ttf"**)  
 .setFontAttrId(R.attr.***fontPath***)  
 .build());  
 setContentView(R.layout.***activity\_home***);  
  
 Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 toolbar.setTitle(**"Menu"**);  
 setSupportActionBar(toolbar);  
  
 *//View* **swipeRefreshLayout**=(SwipeRefreshLayout)findViewById(R.id.***swipe\_layout***);  
 **swipeRefreshLayout**.setColorSchemeResources(R.color.colorPrimary,  
 android.R.color.holo\_green\_dark,  
 android.R.color.holo\_orange\_dark,  
 android.R.color.holo\_blue\_dark);  
 swipeRefreshLayout.setOnRefreshListener(**new** SwipeRefreshLayout.OnRefreshListener() {  
 @Override  
 **public void** onRefresh() {  
  
 **if** (Common.isConnectedToInternet(getBaseContext()))  
 loadMenu();  
 **else** {  
 Toast.makeText(getBaseContext(), **"Check Your Connection !!!"**, Toast.LENGTH\_SHORT).show();  
 **return**;  
 }  
 }  
 });  
  
 *//Default load for 1st time* swipeRefreshLayout.post(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **if** (Common.isConnectedToInternet(getBaseContext()))  
 loadMenu();  
 **else** {  
 Toast.makeText(getBaseContext(), **"Check Your Connection !!!"**, Toast.LENGTH\_SHORT).show();  
 **return**;  
 }  
 }  
 });  
  
 *//Init firebase* database=FirebaseDatabase.getInstance();  
 category=database.getReference(**"Category"**);  
  
 FirebaseRecyclerOptions<Category>options=**new** FirebaseRecyclerOptions.Builder<Category>().setQuery(category,Category.**class**).build();  
 adapter= **new** FirebaseRecyclerAdapter<Category, MenuViewHolder>(options) {  
 @NonNull  
 @Override  
 **public** MenuViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, **int** i) {  
 View itemView=LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.menu\_item,viewGroup,**false**);  
 **return new** MenuViewHolder(itemView);  
 }  
  
 @Override  
 **protected void** onBindViewHolder(@NonNull MenuViewHolder holder, **int** position, @NonNull Category model) {  
  
 holder.txtMenuName.setText(model.getName());  
 Picasso.with(getBaseContext()).load(model.getImage()).into(holder.imageView);  
 **final** Category clickItem=model;  
 holder.setItemClickListener(**new** ItemClickListener() {  
 @Override  
 **public void** onClick(View view, **int** position, **boolean** isLongClick) {  
 *//Get Category Id and send to new activity* Intent foodList=**new** Intent(Home.**this**,FoodList.**class**);  
 *// Category ID is key so we just get key of this item* foodList.putExtra(**"CategoryId"**,adapter.getRef(position).getKey());  
 startActivity(foodList);  
 }  
 });  
  
 }  
 };  
  
 *//Init Paper* Paper.init(**this**);  
  
 fab = (CounterFab) findViewById(R.id.fab);  
 fab.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 Intent cartIntent=**new** Intent(Home.**this**,Cart.**class**);  
 startActivity(cartIntent);  
 }  
 });  
  
 fab.setCount(**new** MyData(**this**).getCountCart(Common.currentUser.getPhone()));  
  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  
 ActionBarDrawerToggle toggle = **new** ActionBarDrawerToggle(  
 **this**, drawer, toolbar, R.string.navigation\_drawer\_open, R.string.navigation\_drawer\_close);  
 drawer.addDrawerListener(toggle);  
 toggle.syncState();  
  
 NavigationView navigationView = (NavigationView) findViewById(R.id.nav\_view);  
 navigationView.setNavigationItemSelectedListener(**this**);  
  
 *//Set Name For User* View headerView=navigationView.getHeaderView(0);  
 txtFullName=(TextView)headerView.findViewById(R.id.txtFullName);  
 txtFullName.setText(Common.currentUser.getName());  
  
 *//Load Menu* recycler\_menu=(RecyclerView)findViewById(R.id.recycler\_menu);  
 recycler\_menu.setLayoutManager(**new** GridLayoutManager(**this**,2));  
 LayoutAnimationController controller=AnimationUtils.loadLayoutAnimation(recycler\_menu.getContext(),  
 R.anim.layout\_slide\_from\_bottom);  
 recycler\_menu.setLayoutAnimation(controller);  
  
 *//News System* **final** CheckBox ckb\_subrcribe\_new=(CheckBox)findViewById(R.id.ckb\_sub\_new);  
 FirebaseMessaging.getInstance().subscribeToTopic(Common.topicName);  
  
 updateToken(FirebaseInstanceId.getInstance().getToken());  
  
 *//SetUpSlider* setupSlider();  
  
 }  
  
 **private void** setupSlider() {  
 mSlider=(SliderLayout)findViewById(R.id.slider);  
 image\_list=**new** HashMap<>();  
  
 **final** DatabaseReference banners=database.getReference(**"Banner"**);  
  
 banners.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
  
 **for** (DataSnapshot postSnapShot:dataSnapshot.getChildren())  
 {  
 Banner banner=postSnapShot.getValue(Banner.**class**);  
 *//We Will concatstring name and id like  
 //pizza \_01 => and we will use PIZZA for showing description,01 for food id to click* image\_list.put(banner.getName()+**"@@@"**+banner.getId(),banner.getImage());  
 }  
 **for** (String key:image\_list.keySet())  
 {  
 String[] keySplit=key.split(**"@@@"**);  
 String nameOfFood=keySplit[0];  
 String idOfFood=keySplit[1];  
  
 *//Create Slider* **final** TextSliderView textSliderView=**new** TextSliderView(getBaseContext());  
 textSliderView  
 .description(nameOfFood)  
 .image(image\_list.get(key))  
 .setScaleType(BaseSliderView.ScaleType.Fit)  
 .setOnSliderClickListener(**new** BaseSliderView.OnSliderClickListener() {  
 @Override  
 **public void** onSliderClick(BaseSliderView slider) {  
 Intent intent=**new** Intent(Home.**this**,FoodDetail.**class**);  
 *//we will send food id to food detail* intent.putExtras(textSliderView.getBundle());  
 startActivity(intent);  
  
 }  
 });  
 *//Add Extra bundle* textSliderView.bundle(**new** Bundle());  
 textSliderView.getBundle().putString(**"FoodId"**,idOfFood);  
  
 mSlider.addSlider(textSliderView);  
  
 *//Remove after event finishes* banners.removeEventListener(**this**);  
 }  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 mSlider.setPresetTransformer(SliderLayout.Transformer.Background2Foreground);  
 mSlider.setPresetIndicator(SliderLayout.PresetIndicators.Center\_Bottom);  
 mSlider.setCustomAnimation(**new** DescriptionAnimation());  
 mSlider.setDuration(4000);  
 }  
  
 @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 fab.setCount(**new** MyData(**this**).getCountCart(Common.currentUser.getPhone()));  
 **if** (adapter!=**null**)  
 adapter.startListening();  
  
 }  
  
 **private void** updateToken(String token) {  
 FirebaseDatabase db=FirebaseDatabase.getInstance();  
 DatabaseReference tokens=db.getReference(**"Tokens"**);  
 Token data=**new** Token(token,**false**); *//false becausenthis token is sent from client app* tokens.child(Common.currentUser.getPhone()).setValue(data);  
 }  
  
 **private void** loadMenu() {  
  
  
  
 adapter.startListening();  
 recycler\_menu.setAdapter(adapter);  
 swipeRefreshLayout.setRefreshing(**false**);  
  
 *//Animation* recycler\_menu.getAdapter().notifyDataSetChanged();  
 recycler\_menu.scheduleLayoutAnimation();  
  
 }  
  
 @Override  
 **protected void** onStart() {  
 **super**.onStart();  
 mSlider.startAutoCycle();  
 }  
  
 @Override  
 **protected void** onPostResume() {  
 **super**.onPostResume();  
 loadMenu();  
 }  
  
 @Override  
 **protected void** onStop() {  
 **super**.onStop();  
 adapter.stopListening();  
 mSlider.stopAutoCycle();  
 }  
  
 @Override  
 **public void** onBackPressed() {  
  
 **if** (backPressedTime + 2000 > System.currentTimeMillis()){  
 **super**.onBackPressed();  
 **return**;  
  
  
 }  
 **else** {  
 Toast.makeText(**this**, **"Press Back Again To Exit !!!"**, Toast.LENGTH\_SHORT).show();  
  
 }  
 backPressedTime=System.currentTimeMillis();  
  
 }  
  
 @Override  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 *// Inflate the menu; this adds items to the action bar if it is present.* getMenuInflater().inflate(R.menu.home, menu);  
 **return true**;  
 }  
  
 @Override  
 **public boolean** onOptionsItemSelected(MenuItem item) {  
  
 **if** (item.getItemId()==R.id.menu\_search)  
 startActivity(**new** Intent(Home.**this**,SearchActivity.**class**));  
 **return super**.onOptionsItemSelected(item);  
 }  
  
 @SuppressWarnings(**"StatementWithEmptyBody"**)  
 @Override  
 **public boolean** onNavigationItemSelected(MenuItem item) {  
 *// Handle navigation view item clicks here.* **int** id = item.getItemId();  
  
 **if** (id == R.id.nav\_cart) {  
 Intent cartIntent=**new** Intent(Home.**this**,Cart.**class**);  
 startActivity(cartIntent);  
  
 } **else if** (id == R.id.nav\_orders) {  
 Intent orderIntent=**new** Intent(Home.**this**,OrderStatus.**class**);  
 startActivity(orderIntent);  
  
 } **else if** (id == R.id.nav\_log\_out) {  
  
 AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(Home.**this**);  
 alertDialog.setTitle(**"Hey, "**+Common.currentUser.getName());  
 alertDialog.setMessage(**"Do You Want To Sign Out?"**);  
 alertDialog.setCancelable(**false**);  
  
 alertDialog.setPositiveButton(**"YES"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 *//Delete Remember user & Password* AccountKit.logOut();  
 *//Log Out* Intent signIn=**new** Intent(Home.**this**,MainActivity.**class**);  
 signIn.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK|Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);  
 startActivity(signIn);  
 }  
 });  
 alertDialog.setNegativeButton(**"NO"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.makeText(Home.**this**, **"Cancel"**, Toast.LENGTH\_SHORT).show();  
 }  
 });  
 alertDialog.setIcon(R.mipmap.unnamed);  
 alertDialog.show();  
  
  
 }  
 **else if** (id==R.id.nav\_feedback){  
 showRatingDialog();  
 }  
 **else if** (id==R.id.nav\_Favourites){  
 startActivity(**new** Intent(Home.**this**,FavouritesActivity.**class**));  
 }  
 **else if** (id==R.id.nav\_profile){  
 *// startActivity(new Intent(Home.this,Profile.class));* showProfileDialog();  
 }  
 **else if** (id==R.id.nav\_share){  
 shareAPK();  
  
 }  
 **else if** (id==R.id.nav\_help){  
 startActivity(**new** Intent(Home.**this**,HelpActivity.**class**));  
 }  
  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  
 drawer.closeDrawer(GravityCompat.START);  
 **return true**;  
 }  
  
 **private void** shareAPK() {  
 **try** {  
 Intent i = **new** Intent(Intent.ACTION\_SEND);  
 i.setType(**"text/plain"**);  
 i.putExtra(Intent.EXTRA\_SUBJECT, **"Taste Hub:Food Delivery Expert"**);  
 String sAux = **"\nLet Me Recommend You Taste Hub Application\n\n"**;  
 sAux = sAux + **"https://drive.google.com/open?id=1-3Y7w05blyAx1vRHFcwx\_TmZxDcTvPIr\n\n"**;  
 i.putExtra(Intent.EXTRA\_TEXT, sAux);  
 startActivity(Intent.createChooser(i, **"Select Share Method"**));  
 } **catch**(Exception e) {  
 }  
  
 }  
  
 **private void** showProfileDialog() {  
 **final** AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(Home.**this**);  
 alertDialog.setTitle(**"Update Your Profile"**);  
 alertDialog.setMessage(**"Enter Your Name & Address"**);  
 alertDialog.setCancelable(**false**);  
  
 LayoutInflater inflater=LayoutInflater.from(**this**);  
 **final** View layout\_profile=inflater.inflate(R.layout.profile\_layout,**null**);  
  
 **final** MaterialEditText edtName=(MaterialEditText)layout\_profile.findViewById(R.id.Profile\_name);  
 **final** MaterialEditText edtHomeAddress=(MaterialEditText)layout\_profile.findViewById(R.id.Profile\_address);  
 alertDialog.setView(layout\_profile);  
  
 edtName.setText(Common.currentUser.getName());  
 edtHomeAddress.setText(Common.currentUser.getHomeAddress());  
  
 *//Button* alertDialog.setPositiveButton(**"UPDATE"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
  
 **if**(edtName.getText().toString().trim().length()==0 || edtHomeAddress.getText().toString().trim().length()==0){  
 Toast.makeText(Home.**this**, **"Please Enter All Details !!!"**, Toast.LENGTH\_SHORT).show();  
  
 }  
  
 **else** {  
 *//show Dialog* **final** ACProgressFlower waitingDialog = **new** ACProgressFlower.Builder(Home.**this**)  
 .direction(ACProgressConstant.DIRECT\_CLOCKWISE)  
 .themeColor(Color.WHITE)  
 .text(**"Loading. . ."**)  
 .fadeColor(Color.DKGRAY).build();  
 waitingDialog.show();  
  
 *//Update Name* Map<String, Object> update\_profile = **new** HashMap<>();  
 update\_profile.put(**"name"**, edtName.getText().toString());  
 update\_profile.put(**"homeAddress"**, edtHomeAddress.getText().toString());  
  
  
 FirebaseDatabase.*getInstance*()  
 .getReference(**"User"**)  
 .child(Common.*currentUser*.getPhone())  
 .updateChildren(update\_profile)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 *//Dismiss Dialog* waitingDialog.dismiss();  
 **if** (task.isSuccessful())  
 Toast.*makeText*(Home.**this**, **"Profile Successfully Updated !!!"**, Toast.***LENGTH\_SHORT***).show();  
  
 }  
 });  
  
 *//Refresh Driver Data* DatabaseReference userInformation=FirebaseDatabase.*getInstance*().getReference(**"User"**);  
 userInformation.child(Common.*currentUser*.getPhone())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
  
 Common.*currentUser*=dataSnapshot.getValue(User.**class**);  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 }  
 });  
  
 alertDialog.setNegativeButton(**"CANCEL"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.makeText(Home.**this**, **"Cancel"**, Toast.LENGTH\_SHORT).show();  
 }  
 });  
 alertDialog.setIcon(R.drawable.ic\_person\_black\_24dp);  
 alertDialog.show();  
 }  
  
 **private void** showSettingDialog() {  
  
 AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(Home.**this**);  
 alertDialog.setTitle(**"Notification Setting"**);  
 alertDialog.setCancelable(**false**);  
  
 LayoutInflater inflater=LayoutInflater.from(**this**);  
 View layout\_setting=inflater.inflate(R.layout.setting\_layout,**null**);  
  
  
  
 *//Add Code remember state of checkbox* Paper.init(**this**);  
 String isSubscribe=Paper.book().read(**"sub\_new"**);  
  
 **final** CheckBox ckb\_subrcribe\_new=(CheckBox)layout\_setting.findViewById(R.id.ckb\_sub\_new);  
  
 Paper.book().write(**"sub\_new"**,**"true"**);  
 **if** (isSubscribe==**null** || TextUtils.isEmpty(isSubscribe) || isSubscribe.equals(**"false"**))  
 ckb\_subrcribe\_new.setChecked(**false**);  
 **else** ckb\_subrcribe\_new.setChecked(**true**);  
  
  
  
 alertDialog.setView(layout\_setting);  
 alertDialog.setPositiveButton(**"OK"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
  
 **if** (ckb\_subrcribe\_new.isChecked())  
 {  
 ckb\_subrcribe\_new.setChecked(**true**);  
 FirebaseMessaging.getInstance().subscribeToTopic(Common.topicName);  
 *//Write Value* Paper.book().write(**"sub\_new"**,**"true"**);  
 Toast.makeText(Home.**this**, **"News Is Enabled !!!"**, Toast.LENGTH\_SHORT).show();  
 }  
 **else** {  
 ckb\_subrcribe\_new.setChecked(**false**);  
 FirebaseMessaging.getInstance().unsubscribeFromTopic(Common.topicName);  
 *//write value* Paper.book().write(**"sub\_new"**,**"false"**);  
 Toast.makeText(Home.**this**, **"News Is Disabled !!!"**, Toast.LENGTH\_SHORT).show();  
 }  
 }  
 });  
  
 alertDialog.setNegativeButton(**"CANCEL"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.makeText(Home.**this**, **"Cancel"**, Toast.LENGTH\_SHORT).show();  
  
 }  
 });  
  
 alertDialog.setIcon(R.drawable.ic\_settings\_black\_24dp);  
 alertDialog.show();  
  
 }  
  
  
 @Override  
 **public void** onNegativeButtonClicked() {  
  
 }  
  
 @Override  
 **public void** onNeutralButtonClicked() {  
  
 }  
  
 @Override  
 **public void** onPositiveButtonClicked(**int** i, @NotNull String s) {  
  
  
 *//Set Rating and Upload To Firebase* **final** Feedback feedback=**new** Feedback(Common.currentUser.getName(),  
 Common.currentUser.getHomeAddress(),  
 Common.currentUser.getDOB(),  
 Common.currentUser.getEmailAddress(),  
 Common.currentUser.getGender(),  
 String.valueOf(i),  
 s);  
  
 FirebaseDatabase.getInstance().getReference(**"Feedback"**).child(Common.currentUser.getPhone())  
 .push()  
 .setValue(feedback)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
  
 Toast.makeText(Home.**this**, **"Thank You For Your Feedback !!!"**, Toast.LENGTH\_SHORT).show();  
 }  
 });  
 }  
  
 **private void** showRatingDialog() {  
  
 **new** AppRatingDialog.Builder()  
 .setCancelable(**false**)  
 .setCanceledOnTouchOutside(**false**)  
 .setWindowAnimation(R.anim.item\_animation\_slide\_from\_right)  
 .setPositiveButtonText(**"Submit"**)  
 .setNegativeButtonText(**"Cancel"**)  
 .setNoteDescriptions(Arrays.asList(**"Very Bad"**,**"Not Good"**,**"Quite Ok"**,**"Very Good"**,**"Excellent"**))  
 .setDefaultRating(1)  
 .setTitle(**"Rate This App !!!"**)  
 .setDescription(**"Please Select Some Stars And Give Your Feedback"**)  
 .setTitleTextColor(R.color.colorPrimary)  
 .setDescriptionTextColor(R.color.colorPrimary)  
 .setHint(**"Write Your Feedback Here . . ."**)  
 .setHintTextColor(R.color.colorAccent)  
 .setCommentTextColor(android.R.color.white)  
 .setCommentBackgroundColor(R.color.colorPrimaryDark)  
 .setWindowAnimation(R.style.RatingDialogFadeAnim)  
 .create(Home.**this**)  
 .show();  
  
  
 }  
}

11.2: Car Booking App

11.2.1: MainActivity.java

**package** com.example.thunderbolt\_108.driver4u;  
  
**import** android.app.AlertDialog;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.content.pm.PackageInfo;  
**import** android.content.pm.PackageManager;  
**import** android.content.pm.Signature;  
**import** android.support.annotation.NonNull;  
**import** android.support.annotation.Nullable;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.util.Base64;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.RelativeLayout;  
**import** android.widget.Toast;  
  
**import** com.example.thunderbolt\_108.driver4u.Common.Common;  
**import** com.example.thunderbolt\_108.driver4u.Model.UberDriver;  
**import** com.facebook.accountkit.Account;  
**import** com.facebook.accountkit.AccountKit;  
**import** com.facebook.accountkit.AccountKitCallback;  
**import** com.facebook.accountkit.AccountKitError;  
**import** com.facebook.accountkit.AccountKitLoginResult;  
**import** com.facebook.accountkit.ui.AccountKitActivity;  
**import** com.facebook.accountkit.ui.AccountKitConfiguration;  
**import** com.facebook.accountkit.ui.LoginType;  
**import** com.google.android.gms.tasks.OnCompleteListener;  
**import** com.google.android.gms.tasks.OnFailureListener;  
**import** com.google.android.gms.tasks.Task;  
**import** com.google.firebase.auth.FirebaseAuth;  
**import** com.google.firebase.database.DataSnapshot;  
**import** com.google.firebase.database.DatabaseError;  
**import** com.google.firebase.database.DatabaseReference;  
**import** com.google.firebase.database.FirebaseDatabase;  
**import** com.google.firebase.database.ValueEventListener;  
  
**import** java.security.MessageDigest;  
**import** java.security.NoSuchAlgorithmException;  
  
**import** dmax.dialog.SpotsDialog;  
**import** info.hoang8f.widget.FButton;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyConfig;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyContextWrapper;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 RelativeLayout **rootLayout**;  
 FButton **btnContinue**;  
 **private static final int *REQUEST\_CODE***=7181;  
  
 FirebaseAuth **auth**;  
 FirebaseDatabase **db**;  
 DatabaseReference **users**;  
  
 @Override  
 **protected void** attachBaseContext(Context newBase) {  
 **super**.attachBaseContext(CalligraphyContextWrapper.*wrap*(newBase));  
 }  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 CalligraphyConfig.initDefault(**new** CalligraphyConfig.Builder()  
 .setDefaultFontPath(**"fonts/Arkhip\_font.ttf"**)  
 .setFontAttrId(R.attr.fontPath)  
 .build());  
 setContentView(R.layout.activity\_main);  
  
 auth=FirebaseAuth.getInstance();  
 db=FirebaseDatabase.getInstance();  
 users=db.getReference(Common.user\_driver\_tbl);  
  
 btnContinue=(FButton)findViewById(R.id.btnContinue);  
  
 rootLayout=(RelativeLayout)findViewById(R.id.rootLayout);  
  
 btnContinue.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 signInWithPhone();  
 }  
 });  
  
 *//Auto login for second time* **if** (AccountKit.getCurrentAccessToken() !=**null**)  
 {  
 **final** AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(MainActivity.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Please Wait . . ."**);  
 waitingDialog.setCancelable(**false**);  
  
 AccountKit.getCurrentAccount(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 *//Login* users.child(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 UberDriver localUberDriver =dataSnapshot.getValue(UberDriver.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, DriverHome.**class**);  
 Common.currentUberDriver = localUberDriver;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
  
 }  
  
  
 }  
  
 **private void** signInWithPhone() {  
 Intent intent=**new** Intent(MainActivity.**this**, AccountKitActivity.**class**);  
 AccountKitConfiguration.AccountKitConfigurationBuilder configurationBuilder=  
 **new** AccountKitConfiguration.AccountKitConfigurationBuilder(LoginType.PHONE,  
 AccountKitActivity.ResponseType.TOKEN);  
 intent.putExtra(AccountKitActivity.ACCOUNT\_KIT\_ACTIVITY\_CONFIGURATION,configurationBuilder.build());  
 startActivityForResult(intent,REQUEST\_CODE);  
 }  
  
 @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, @Nullable Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
  
 **if** (requestCode==REQUEST\_CODE)  
 {  
 AccountKitLoginResult result=data.getParcelableExtra(AccountKitLoginResult.RESULT\_KEY);  
 **if** (result.getError()!=**null**)  
 {  
 Toast.makeText(**this**, **""**+result.getError().getErrorType().getMessage(), Toast.LENGTH\_SHORT).show();  
 **return**;  
 }  
 **else if**(result.wasCancelled())  
 {  
 Toast.makeText(**this**, **"Cancel"**, Toast.LENGTH\_SHORT).show();  
 **return**;  
 }  
 **else** {  
 **if** (result.getAccessToken()!=**null**){  
  
 *//show Dialog* **final** AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(MainActivity.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Please Wait . . ."**);  
 waitingDialog.setCancelable(**false**);  
  
  
 *//Get Current Phone* AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(**final** Account account) {  
  
  
 **final** String userId=account.getId();  
  
 *//Check user if exists on firebase* **users**.orderByKey().equalTo(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 **if** (!dataSnapshot.child(account.getId()).exists())  
 { *//if not exits then create new user* UberDriver newUberDriver =**new** UberDriver();  
 newUberDriver.setPhone(account.getPhoneNumber().toString());  
 newUberDriver.setName(**""**);  
 newUberDriver.setRates(**"0.0"**);  
 newUberDriver.setAvatarUrl(**""**);  
 newUberDriver.setCarType(**"UberX"**);  
  
 *//Add to firebase* users.child(account.getId())  
 .setValue(newUberDriver)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful())  
 {  
 Toast.makeText(MainActivity.**this**, **"UberDriver Registered Successfully !!!"**, Toast.LENGTH\_SHORT).show();  
  
  
 }  
  
 *//Login* users.child(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 UberDriver localUberDriver =dataSnapshot.getValue(UberDriver.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, DriverHome.**class**);  
 Common.currentUberDriver = localUberDriver;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
 }  
 }).addOnFailureListener(**new** OnFailureListener() {  
 @Override  
 **public void** onFailure(@NonNull Exception e) {  
 Toast.makeText(MainActivity.**this**, **""**+e.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 });  
  
 }  
 **else** {  
 *//If exists  
  
 //Login* users.child(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 UberDriver localUberDriver =dataSnapshot.getValue(UberDriver.**class**);  
 Intent homeIntent = **new** Intent(MainActivity.**this**, DriverHome.**class**);  
 Common.currentUberDriver = localUberDriver;  
 startActivity(homeIntent);  
 waitingDialog.dismiss();  
 finish();  
  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
  
  
 }  
  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
 Toast.makeText(MainActivity.**this**, **""**+accountKitError.getErrorType().getMessage(), Toast.LENGTH\_SHORT).show();  
  
 }  
 });  
 }  
 }  
 }  
  
 }  
  
 **private void** printKeyHash() {  
 **try**{  
 PackageInfo info=getPackageManager().getPackageInfo(**"com.example.thunderbolt\_108.driver4u"**,  
 PackageManager.GET\_SIGNATURES);  
 **for** (Signature signature:info.signatures)  
 {  
 MessageDigest md=MessageDigest.getInstance(**"SHA"**);  
 md.update(signature.toByteArray());  
 Log.d(**"KeyHash"**, Base64.encodeToString(md.digest(),Base64.DEFAULT));  
  
 }  
 } **catch** (PackageManager.NameNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
}

11.2.2: Home.java

**package** com.example.thunderbolt\_108.driver4u;  
  
**import** android.Manifest;  
**import** android.animation.ValueAnimator;  
**import** android.app.ProgressDialog;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.Intent;  
**import** android.content.IntentSender;  
**import** android.content.pm.PackageManager;  
**import** android.content.res.Resources;  
**import** android.graphics.Color;  
**import** android.location.Location;  
**import** android.location.LocationManager;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.Looper;  
**import** android.support.annotation.NonNull;  
**import** android.support.annotation.Nullable;  
**import** android.support.design.widget.Snackbar;  
**import** android.support.v4.app.ActivityCompat;  
**import** android.support.v7.app.AlertDialog;  
**import** android.text.TextUtils;  
**import** android.util.Log;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.support.design.widget.NavigationView;  
**import** android.support.v4.view.GravityCompat;  
**import** android.support.v4.widget.DrawerLayout;  
**import** android.support.v7.app.ActionBarDrawerToggle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.support.v7.widget.Toolbar;  
**import** android.view.Menu;  
**import** android.view.MenuItem;  
**import** android.view.animation.LinearInterpolator;  
**import** android.widget.ImageView;  
**import** android.widget.RadioButton;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.example.thunderbolt\_108.driver4u.Common.Common;  
**import** com.example.thunderbolt\_108.driver4u.Model.Token;  
**import** com.example.thunderbolt\_108.driver4u.Model.UberDriver;  
**import** com.example.thunderbolt\_108.driver4u.Remote.IGoogleAPI;  
**import** com.facebook.accountkit.Account;  
**import** com.facebook.accountkit.AccountKit;  
**import** com.facebook.accountkit.AccountKitCallback;  
**import** com.facebook.accountkit.AccountKitError;  
**import** com.firebase.geofire.GeoFire;  
**import** com.firebase.geofire.GeoLocation;  
**import** com.github.glomadrian.materialanimatedswitch.MaterialAnimatedSwitch;  
**import** com.google.android.gms.common.ConnectionResult;  
**import** com.google.android.gms.common.api.GoogleApiClient;  
**import** com.google.android.gms.common.api.PendingResult;  
**import** com.google.android.gms.common.api.ResultCallback;  
**import** com.google.android.gms.common.api.Status;  
**import** com.google.android.gms.location.FusedLocationProviderClient;  
**import** com.google.android.gms.location.LocationCallback;  
**import** com.google.android.gms.location.LocationRequest;  
**import** com.google.android.gms.location.LocationResult;  
**import** com.google.android.gms.location.LocationServices;  
**import** com.google.android.gms.location.LocationSettingsRequest;  
**import** com.google.android.gms.location.LocationSettingsResult;  
**import** com.google.android.gms.location.LocationSettingsStatusCodes;  
**import** com.google.android.gms.location.places.AutocompleteFilter;  
**import** com.google.android.gms.location.places.Place;  
**import** com.google.android.gms.location.places.ui.PlaceAutocompleteFragment;  
**import** com.google.android.gms.location.places.ui.PlaceSelectionListener;  
**import** com.google.android.gms.maps.CameraUpdate;  
**import** com.google.android.gms.maps.CameraUpdateFactory;  
**import** com.google.android.gms.maps.GoogleMap;  
**import** com.google.android.gms.maps.OnMapReadyCallback;  
**import** com.google.android.gms.maps.SupportMapFragment;  
**import** com.google.android.gms.maps.model.BitmapDescriptorFactory;  
**import** com.google.android.gms.maps.model.CameraPosition;  
**import** com.google.android.gms.maps.model.JointType;  
**import** com.google.android.gms.maps.model.LatLng;  
**import** com.google.android.gms.maps.model.LatLngBounds;  
**import** com.google.android.gms.maps.model.MapStyleOptions;  
**import** com.google.android.gms.maps.model.Marker;  
**import** com.google.android.gms.maps.model.MarkerOptions;  
**import** com.google.android.gms.maps.model.Polyline;  
**import** com.google.android.gms.maps.model.PolylineOptions;  
**import** com.google.android.gms.maps.model.SquareCap;  
**import** com.google.android.gms.tasks.OnCompleteListener;  
**import** com.google.android.gms.tasks.OnSuccessListener;  
**import** com.google.android.gms.tasks.Task;  
**import** com.google.firebase.database.DataSnapshot;  
**import** com.google.firebase.database.DatabaseError;  
**import** com.google.firebase.database.DatabaseReference;  
**import** com.google.firebase.database.FirebaseDatabase;  
**import** com.google.firebase.database.ValueEventListener;  
**import** com.google.firebase.iid.FirebaseInstanceId;  
**import** com.google.firebase.storage.FirebaseStorage;  
**import** com.google.firebase.storage.OnProgressListener;  
**import** com.google.firebase.storage.StorageReference;  
**import** com.google.firebase.storage.UploadTask;  
**import** com.google.maps.android.SphericalUtil;  
**import** com.rengwuxian.materialedittext.MaterialEditText;  
**import** com.squareup.picasso.Picasso;  
  
**import** org.json.JSONArray;  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**import** java.util.ArrayList;  
**import** java.util.HashMap;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.UUID;  
  
**import** de.hdodenhof.circleimageview.CircleImageView;  
**import** dmax.dialog.SpotsDialog;  
**import** retrofit2.Call;  
**import** retrofit2.Callback;  
**import** retrofit2.Response;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyConfig;  
**import** uk.co.chrisjenx.calligraphy.CalligraphyContextWrapper;  
  
**import static** com.example.thunderbolt\_108.driver4u.Common.Common.*mLastlocation*;  
  
**public class** DriverHome **extends** AppCompatActivity  
 **implements** NavigationView.OnNavigationItemSelectedListener, OnMapReadyCallback  
 {  
  
 **private** GoogleMap **mMap**;  
  
 FusedLocationProviderClient **fusedLocationProviderClient**;  
 LocationCallback **locationCallback**;  
  
 *//Play Services* **private static final int *MY\_PERMISSION\_REQUEST\_CODE***=7000;  
 **private static final int *PLAY\_SERVICE\_RES\_REQUEST***=7001;  
  
 **private** LocationRequest **mLocationRequest**;  
 **private** GoogleApiClient **mGoogleApiClient**;  
  
  
 **private static int** *UPDATE\_INTERVAL*=5000;  
 **private static int** *FASTEST\_INTERVAL*=3000;  
 **private static int** *DISPLACEMENT*=10;  
  
 DatabaseReference **drivers**;  
 GeoFire **geoFire**;  
 Marker **mCurrent**;  
  
 SupportMapFragment **mapFragment**;  
 MaterialAnimatedSwitch **location\_switch**;  
  
 *//Car Animation* **private** List<LatLng> **polyLineList**;  
 **private** Marker **carMarker**;  
 **private float v**;  
 **private double lat**,**lng**;  
 **private** Handler **handler**;  
 **private** LatLng **startPosition**,**endPosition**,**currentPosition**;  
 **private int index**,**next**;  
 **private** PlaceAutocompleteFragment **places**;  
 AutocompleteFilter **typeFilter**;  
 **private** String **destination**;  
 **private** PolylineOptions **polylineOptions**,**blackPolylineOptions**;  
 **private** Polyline **blackPolyline**,**grayPolyline**;  
  
 **private** IGoogleAPI **mService**;  
 **private** GoogleApiClient **googleApiClient**;  
 **final static int *REQUEST\_LOCATION*** = 199;  
  
 FirebaseStorage **firebaseStorage**;  
 StorageReference **storageReference**;  
  
 *//Presence System* DatabaseReference **onlineRef**,**currentUserRef**;  
  
  
 Runnable **drawPathRunnable**=**new** Runnable() {  
 @Override  
 **public void** run() {  
 **if** (**index**<**polyLineList**.size()-1)  
 {  
 **index**++;  
 **next**=**index**+1;  
 }  
 **if** (**index**<**polyLineList**.size()-1)  
 {  
 **startPosition**=**polyLineList**.get(**index**);  
 **endPosition**=**polyLineList**.get(**next**);  
 }  
  
 **final** ValueAnimator valueAnimator=ValueAnimator.*ofFloat*(0,1);  
 valueAnimator.setDuration(3000);  
 valueAnimator.setInterpolator(**new** LinearInterpolator());  
 valueAnimator.addUpdateListener(**new** ValueAnimator.AnimatorUpdateListener() {  
 @Override  
 **public void** onAnimationUpdate(ValueAnimator animation) {  
 **v**=animation.getAnimatedFraction();  
 **lng**=**v**\***endPosition**.**longitude**+(1-**v**)\***startPosition**.**longitude**;  
 **lat**=**v**\***endPosition**.**latitude**+(1-**v**)\***startPosition**.**latitude**;  
 LatLng newPos=**new** LatLng(**lat**,**lng**);  
 **carMarker**.setPosition(newPos);  
 **carMarker**.setAnchor(0.5f,0.5f);  
 **carMarker**.setRotation(getBearing(**startPosition**,newPos));  
 **mMap**.moveCamera(CameraUpdateFactory.*newCameraPosition*(  
 **new** CameraPosition.Builder()  
 .target(newPos)  
 .zoom(15.5f)  
 .build()  
 ));  
 }  
 });  
 valueAnimator.start();  
 **handler**.postDelayed(**this**,3000);  
 }  
 };  
  
 **private float** getBearing(LatLng startPosition, LatLng endPosition) {  
 **double** lat=Math.*abs*(startPosition.**latitude**-endPosition.**latitude**);  
 **double** lng=Math.*abs*(startPosition.**longitude**-endPosition.**longitude**);  
  
 **if** (startPosition.**latitude** < endPosition.**latitude** && startPosition.**longitude** < endPosition.**longitude**)  
 **return** (**float**)(Math.*toDegrees*(Math.*atan*(lng/lat)));  
 **else if** (startPosition.**latitude** >= endPosition.**latitude** && startPosition.**longitude** < endPosition.**longitude**)  
 **return** (**float**)((90-Math.*toDegrees*(Math.*atan*(lng/lat)))+90);  
 **else if** (startPosition.**latitude** >= endPosition.**latitude** && startPosition.**longitude** >= endPosition.**longitude**)  
 **return** (**float**)(Math.*toDegrees*(Math.*atan*(lng/lat))+180);  
 **else if** (startPosition.**latitude** < endPosition.**latitude** && startPosition.**longitude** >= endPosition.**longitude**)  
 **return** (**float**)((90-Math.*toDegrees*(Math.*atan*(lng/lat)))+270);  
 **return** -1;  
  
 }  
  
 @Override  
 **protected void** attachBaseContext(Context newBase) {  
 **super**.attachBaseContext(CalligraphyContextWrapper.*wrap*(newBase));  
 }  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 CalligraphyConfig.*initDefault*(**new** CalligraphyConfig.Builder()  
 .setDefaultFontPath(**"fonts/Arkhip\_font.ttf"**)  
 .setFontAttrId(R.attr.***fontPath***)  
 .build());  
 setContentView(R.layout.***activity\_driver\_home***);  
 Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(toolbar);  
  
 *//Init Firebase* **firebaseStorage**=FirebaseStorage.*getInstance*();  
 **storageReference**=**firebaseStorage**.getReference();  
  
 **fusedLocationProviderClient**=LocationServices.*getFusedLocationProviderClient*(**this**);  
  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 ActionBarDrawerToggle toggle = **new** ActionBarDrawerToggle(  
 **this**, drawer, toolbar, R.string.***navigation\_drawer\_open***, R.string.***navigation\_drawer\_close***);  
 drawer.addDrawerListener(toggle);  
 toggle.syncState();  
  
 NavigationView navigationView = (NavigationView) findViewById(R.id.***nav\_view***);  
 navigationView.setNavigationItemSelectedListener(**this**);  
  
 View navigationHeaderView=navigationView.getHeaderView(0);  
 TextView txtName=(TextView)navigationHeaderView.findViewById(R.id.***txtDriverName***);  
 TextView txtStars=(TextView)navigationHeaderView.findViewById(R.id.***txtStars***);  
 CircleImageView imageAvatar=(CircleImageView)navigationHeaderView.findViewById(R.id.***image\_avatar***);  
 imageAvatar.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 chooseImage();  
 }  
 });  
  
 txtName.setText(Common.*currentUberDriver*.getName());  
 txtStars.setText(Common.*currentUberDriver*.getRates());  
  
 **if** (Common.*currentUberDriver*.getAvatarUrl()!=**null** && !TextUtils.*isEmpty*(Common.*currentUberDriver*.getAvatarUrl())) {  
 Picasso.*get*()  
 .load(Common.*currentUberDriver*.getAvatarUrl())  
 .into(imageAvatar);  
 }  
  
  
 *// Obtain the SupportMapFragment and get notified when the map is ready to be used.* **mapFragment** = (SupportMapFragment) getSupportFragmentManager()  
 .findFragmentById(R.id.***map***);  
 **mapFragment**.getMapAsync(**this**);  
  
 *//Init View* **location\_switch**=(MaterialAnimatedSwitch)findViewById(R.id.***location\_switch***);  
 **location\_switch**.setOnCheckedChangeListener(**new** MaterialAnimatedSwitch.OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(**boolean** isOnline) {  
  
 **if** (isOnline)  
 {  
 FirebaseDatabase.*getInstance*().goOnline();  
 **if** (ActivityCompat.*checkSelfPermission*(DriverHome.**this**,Manifest.permission.***ACCESS\_COARSE\_LOCATION***)!=PackageManager.***PERMISSION\_GRANTED*** &&  
 ActivityCompat.*checkSelfPermission*(DriverHome.**this**,Manifest.permission.***ACCESS\_FINE\_LOCATION***)!=PackageManager.***PERMISSION\_GRANTED***)  
 {  
 **return**;  
 }  
  
 buildLocationCallBack();  
 buildLocationRequest();  
 *//Update Location* **fusedLocationProviderClient**.requestLocationUpdates(**mLocationRequest**,**locationCallback**, Looper.*myLooper*());  
  
 *//Geofire* **drivers**=FirebaseDatabase.*getInstance*().getReference(Common.***driver\_tbl***)  
 .child(Common.*currentUberDriver*.getCarType());  
 **geoFire**=**new** GeoFire(**drivers**);  
  
 displayLocation();  
 Snackbar.*make*(**mapFragment**.getView(),**"You're Online"**,Snackbar.***LENGTH\_SHORT***).show();  
 }  
 **else** {  
 FirebaseDatabase.*getInstance*().goOffline();  
 **fusedLocationProviderClient**.removeLocationUpdates(**locationCallback**);  
 **if** (**mCurrent**!=**null**)  
 **mCurrent**.remove();  
  
 **mMap**.clear();  
 **if** (**handler**!=**null**)  
 **handler**.removeCallbacks(**drawPathRunnable**);  
 Snackbar.*make*(**mapFragment**.getView(),**"You're Offline"**,Snackbar.***LENGTH\_SHORT***).show();  
 }  
 }  
 });  
  
 **polyLineList**=**new** ArrayList<>();  
  
 **typeFilter**=**new** AutocompleteFilter.Builder()  
 .setTypeFilter(AutocompleteFilter.***TYPE\_FILTER\_ADDRESS***)  
 .setTypeFilter(3)  
 .build();  
  
 **places**=(PlaceAutocompleteFragment)getFragmentManager().findFragmentById(R.id.***place\_autocomplete\_fragment***);  
 **places**.setOnPlaceSelectedListener(**new** PlaceSelectionListener() {  
 @Override  
 **public void** onPlaceSelected(Place place) {  
 **if** (**location\_switch**.isChecked())  
 {  
 **destination**=place.getAddress().toString();  
 **destination**=**destination**.replace(**" "**,**"+"**); *//Replace space with + to fetch data* Log.*d*(**"SHYAM PATEL"**,**destination**);  
  
 getDirection();  
 }  
 **else** {  
 Toast.*makeText*(DriverHome.**this**, **"Please Change Your Status To ONLINE"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 @Override  
 **public void** onError(Status status) {  
  
 Log.*e*(**"ERROR"**,status.getStatusMessage());  
 }  
  
  
 });  
  
  
  
 *// Todo Location Already on ... start* **final** LocationManager manager = (LocationManager) DriverHome.**this**.getSystemService(Context.***LOCATION\_SERVICE***);  
 **if** (manager.isProviderEnabled(LocationManager.***GPS\_PROVIDER***) && hasGPSDevice(DriverHome.**this**)) {  
 Toast.*makeText*(DriverHome.**this**,**"GPS Is Enabled"**,Toast.***LENGTH\_SHORT***).show();  
 }  
 *// Todo Location Already on ... end* **if**(!hasGPSDevice(DriverHome.**this**)){  
 Toast.*makeText*(DriverHome.**this**,**"Gps not Supported"**,Toast.***LENGTH\_SHORT***).show();  
 }  
  
 **if** (!manager.isProviderEnabled(LocationManager.***GPS\_PROVIDER***) && hasGPSDevice(DriverHome.**this**)) {  
 Log.*e*(**"Shyam"**,**"Gps already enable"**);  
 Toast.*makeText*(DriverHome.**this**,**"Please Enable Your GPS"**,Toast.***LENGTH\_SHORT***).show();  
 enableLoc();  
 }**else**{  
 Log.*e*(**"Shyam"**,**"Gps already enabled"**);  
 Toast.*makeText*(DriverHome.**this**,**"GPS Is Enabled"**,Toast.***LENGTH\_SHORT***).show();  
 }  
  
 setUpLocation();  
  
 **mService**=Common.*getGoogleAPI*();  
  
 updateFirebaseToken();  
 }  
  
 @Override  
 **protected void** onPostResume() {  
 **super**.onPostResume();  
 *//Presence System* AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 **onlineRef**= FirebaseDatabase.*getInstance*().getReference().child(**".info/connected"**);  
 **currentUserRef**=FirebaseDatabase.*getInstance*().getReference(Common.***driver\_tbl***)  
 .child(Common.*currentUberDriver*.getCarType())  
 .child(account.getId());  
 **onlineRef**.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 **currentUserRef**.onDisconnect().removeValue();  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
 }  
  
 @Override  
 **protected void** onDestroy() {  
 FirebaseDatabase.*getInstance*().goOffline();  
 **fusedLocationProviderClient**.removeLocationUpdates(**locationCallback**);  
 **if** (**mCurrent**!=**null**)  
 **mCurrent**.remove();  
  
 **mMap**.clear();  
 **if** (**handler**!=**null**)  
 **handler**.removeCallbacks(**drawPathRunnable**);  
 **super**.onDestroy();  
 }  
  
 @Override  
 **public void** onBackPressed() {  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 **if** (drawer.isDrawerOpen(GravityCompat.***START***)) {  
 drawer.closeDrawer(GravityCompat.***START***);  
 } **else** {  
 **super**.onBackPressed();  
 }  
 }  
  
 @Override  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 *// Inflate the menu; this adds items to the action bar if it is present.* getMenuInflater().inflate(R.menu.***driver\_home***, menu);  
 **return true**;  
 }  
  
 @Override  
 **public boolean** onOptionsItemSelected(MenuItem item) {  
 *// Handle action bar item clicks here. The action bar will  
 // automatically handle clicks on the Home/Up button, so long  
 // as you specify a parent activity in AndroidManifest.xml.* **int** id = item.getItemId();  
  
 **return super**.onOptionsItemSelected(item);  
 }  
  
 @SuppressWarnings(**"StatementWithEmptyBody"**)  
 @Override  
 **public boolean** onNavigationItemSelected(MenuItem item) {  
 *// Handle navigation view item clicks here.* **int** id = item.getItemId();  
  
 **if** (id == R.id.***nav\_sign\_out***) {  
 signOut();  
 *// Handle the camera action* }  
 **else if** (id == R.id.***nav\_update\_info***) {  
 showDialogUpdateInfo();  
 }  
 **else if** (id == R.id.***nav\_car\_type***) {  
 showDialogUpdateCarType();  
 }  
 **return true**;  
 }  
  
 **private void** showDialogUpdateCarType() {  
 AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(DriverHome.**this**);  
 alertDialog.setTitle(**"Update Vehicle Type"**);  
 alertDialog.setMessage(**"Choose Any One Car Type"**);  
 alertDialog.setCancelable(**false**);  
  
 LayoutInflater inflater=LayoutInflater.*from*(**this**);  
 **final** View car\_type=inflater.inflate(R.layout.***layout\_update\_car\_type***,**null**);  
  
 **final** RadioButton rdi\_uberX=(RadioButton)car\_type.findViewById(R.id.***rdi\_uberX***);  
 **final** RadioButton rdi\_uberBlack=(RadioButton)car\_type.findViewById(R.id.***rdi\_uber\_black***);  
  
 *//Load Default Data from user information* **if** (Common.*currentUberDriver*.getCarType().equals(**"UberX"**))  
 rdi\_uberX.setChecked(**true**);  
 **if** (Common.*currentUberDriver*.getCarType().equals(**"UberBlack"**))  
 rdi\_uberBlack.setChecked(**true**);  
  
 alertDialog.setView(car\_type);  
  
 alertDialog.setPositiveButton(**"UPDATE"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
  
 dialog.dismiss();  
 **final** android.app.AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(DriverHome.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Updating. . ."**);  
 waitingDialog.setCancelable(**false**);  
  
 AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(**final** Account account) {  
  
 Map<String,Object>updateInfo=**new** HashMap<>();  
 **if** (rdi\_uberX.isChecked())  
 updateInfo.put(**"carType"**,rdi\_uberX.getText().toString());  
 **if** (rdi\_uberBlack.isChecked())  
 updateInfo.put(**"carType"**,rdi\_uberBlack.getText().toString());  
  
 DatabaseReference driverInformation=FirebaseDatabase.*getInstance*().getReference(Common.***user\_driver\_tbl***);  
 driverInformation.child(account.getId())  
 .updateChildren(updateInfo)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful()) {  
 **currentUserRef**=FirebaseDatabase.*getInstance*().getReference(Common.***driver\_tbl***)  
 .child(Common.*currentUberDriver*.getCarType())  
 .child(account.getId());  
 Toast.*makeText*(DriverHome.**this**, **"Vehicle Type Updated Successfully"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 **else** Toast.*makeText*(DriverHome.**this**, **"Vehicle Type Update Failed"**, Toast.***LENGTH\_SHORT***).show();  
 waitingDialog.dismiss();  
 }  
 });  
  
 *//Refresh Driver Data* driverInformation.child(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
  
 Common.*currentUberDriver*=dataSnapshot.getValue(UberDriver.**class**);  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
  
 }  
 });  
 alertDialog.setNegativeButton(**"CANCEL"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.*makeText*(DriverHome.**this**, **"Cancel"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 alertDialog.setIcon(R.drawable.***ic\_directions\_car\_black\_24dp***);  
 alertDialog.show();  
  
 }  
 **private void** signOut() {  
 AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(DriverHome.**this**);  
 alertDialog.setTitle(**"Hey, "**+Common.*currentUberDriver*.getName());  
 alertDialog.setMessage(**"Do You Want To Sign Out?"**);  
 alertDialog.setCancelable(**false**);  
  
 alertDialog.setPositiveButton(**"YES"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 *//Delete Remember user & Password* AccountKit.*logOut*();  
 *//Log Out* Intent signIn=**new** Intent(DriverHome.**this**,MainActivity.**class**);  
 signIn.addFlags(Intent.***FLAG\_ACTIVITY\_NEW\_TASK***|Intent.***FLAG\_ACTIVITY\_CLEAR\_TASK***);  
 startActivity(signIn);  
 }  
 });  
 alertDialog.setNegativeButton(**"CANCEL"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.*makeText*(DriverHome.**this**, **"Cancel"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 alertDialog.setIcon(R.drawable.***ic\_exit\_to\_app\_black\_24dp***);  
 alertDialog.show();  
 }  
  
 **private void** showDialogUpdateInfo() {  
 **final** AlertDialog.Builder alertDialog=**new** AlertDialog.Builder(DriverHome.**this**);  
 alertDialog.setTitle(**"Update Your Profile"**);  
 alertDialog.setMessage(**"All fields are mandatory"**);  
 alertDialog.setCancelable(**false**);  
  
 LayoutInflater inflater=LayoutInflater.*from*(**this**);  
 **final** View layout\_profile=inflater.inflate(R.layout.***layout\_update\_information***,**null**);  
  
 **final** MaterialEditText edtName=(MaterialEditText)layout\_profile.findViewById(R.id.***edtName***);  
 **final** TextView edtPhone=(TextView)layout\_profile.findViewById(R.id.***edtPhone***);  
 **final** ImageView image\_upload=(ImageView)layout\_profile.findViewById(R.id.***image\_upload***);  
 image\_upload.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 chooseImage();  
 }  
 });  
  
 alertDialog.setView(layout\_profile);  
  
 edtName.setText(Common.*currentUberDriver*.getName());  
 edtPhone.setText(Common.*currentUberDriver*.getPhone());  
  
 *//Set Button* alertDialog.setPositiveButton(**"UPDATE"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 **final** android.app.AlertDialog waitingDialog=**new** SpotsDialog.Builder().setContext(DriverHome.**this**).build();  
 waitingDialog.show();  
 waitingDialog.setMessage(**"Updating. . ."**);  
  
 AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 String name=edtName.getText().toString();  
 String phone=edtPhone.getText().toString();  
  
 Map<String,Object>updateInfo=**new** HashMap<>();  
 **if** (!TextUtils.*isEmpty*(name))  
 updateInfo.put(**"name"**,name);  
 **if** (!TextUtils.*isEmpty*(phone))  
 updateInfo.put(**"phone"**,phone);  
  
 DatabaseReference driverInformation=FirebaseDatabase.*getInstance*().getReference(Common.***user\_driver\_tbl***);  
 driverInformation.child(account.getId())  
 .updateChildren(updateInfo)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful())  
 Toast.*makeText*(DriverHome.**this**, **"Profile Updated Successfully"**, Toast.***LENGTH\_SHORT***).show();  
 **else** Toast.*makeText*(DriverHome.**this**, **"Profile Update Failed"**, Toast.***LENGTH\_SHORT***).show();  
 waitingDialog.dismiss();  
 }  
 });  
  
 *//Refresh Driver Data* driverInformation.child(account.getId())  
 .addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
  
 Common.*currentUberDriver*=dataSnapshot.getValue(UberDriver.**class**);  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
 }  
 });  
  
 alertDialog.setNegativeButton(**"CANCEL"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 dialog.dismiss();  
 Toast.*makeText*(DriverHome.**this**, **"Cancel"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
  
 alertDialog.setIcon(R.drawable.***ic\_person\_black\_24dp***);  
 alertDialog.show();  
 }  
  
 **private void** chooseImage() {  
 Intent intent=**new** Intent();  
 intent.setType(**"image/\*"**);  
 intent.setAction(Intent.***ACTION\_GET\_CONTENT***);  
 startActivityForResult(Intent.*createChooser*(intent,**"select picture"**),Common.***PICK\_IMAGE\_REQUEST***);  
 }  
  
 @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, @Nullable Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
 **if** (requestCode==Common.***PICK\_IMAGE\_REQUEST*** && resultCode==***RESULT\_OK*** && data!=**null** && data.getData()!=**null**)  
 {  
 Uri saveUri=data.getData();  
 **if** (saveUri!=**null**)  
 {  
 **final** ProgressDialog mDialog=**new** ProgressDialog(**this**);  
 mDialog.setMessage(**"Uploading . . ."**);  
 mDialog.setCancelable(**false**);  
 mDialog.setCanceledOnTouchOutside(**false**);  
 mDialog.show();  
  
 String imageName= UUID.*randomUUID*().toString();*//Random Name to image* **final** StorageReference imageFolder=**storageReference**.child(**"images/"**+imageName);  
 imageFolder.putFile(saveUri)  
 .addOnSuccessListener(**new** OnSuccessListener<UploadTask.TaskSnapshot>() {  
 @Override  
 **public void** onSuccess(UploadTask.TaskSnapshot taskSnapshot) {  
 AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(**final** Account account) {  
 mDialog.dismiss();  
  
 imageFolder.getDownloadUrl().addOnSuccessListener(**new** OnSuccessListener<Uri>() {  
 @Override  
 **public void** onSuccess(Uri uri) {  
 Map<String,Object> avatarUpdate=**new** HashMap<>();  
 avatarUpdate.put(**"avatarUrl"**,uri.toString());  
  
 DatabaseReference driverInformation=FirebaseDatabase.*getInstance*().getReference(Common.***user\_driver\_tbl***);  
 driverInformation.child(account.getId())  
 .updateChildren(avatarUpdate)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful())  
 Toast.*makeText*(DriverHome.**this**, **"Image Uploaded Successfully"**, Toast.***LENGTH\_SHORT***).show();  
 **else** Toast.*makeText*(DriverHome.**this**, **"Image Upload Failed"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
 }  
 })  
 .addOnProgressListener(**new** OnProgressListener<UploadTask.TaskSnapshot>() {  
 @Override  
 **public void** onProgress(UploadTask.TaskSnapshot taskSnapshot) {  
 **double** progress=(100.0\*taskSnapshot.getBytesTransferred()/taskSnapshot.getTotalByteCount());  
 mDialog.setMessage(**"Uploaded "**+progress+**" %"**);  
 }  
 });  
 }  
 }  
 }  
  
 **private void** updateFirebaseToken() {  
  
 AccountKit.*getCurrentAccount*(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 Token token=**new** Token(FirebaseInstanceId.*getInstance*().getToken());  
 FirebaseDatabase db=FirebaseDatabase.*getInstance*();  
 DatabaseReference tokens=db.getReference(Common.***token\_tbl***);  
 tokens.child(account.getId()).setValue(token);  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
  
 }  
  
 **private void** getDirection() {  
 **currentPosition**=**new** LatLng(*mLastlocation*.getLatitude(),*mLastlocation*.getLongitude());  
 String requestApi=**null**;  
 **try**{  
 requestApi=**"https://maps.googleapis.com/maps/api/directions/json?"**+  
 **"mode=driving&"**+  
 **"transit\_routing\_preference=less\_driving&"**+  
 **"origin="**+currentPosition.latitude+**","**+currentPosition.longitude+**"&"**+  
 **"destination="**+destination+**"&"**+  
 **"key="**+getResources().getString(R.string.google\_direction\_api);  
 Log.d(**"SHYAM PATEL"**,requestApi);  
  
 mService.getPath(requestApi)  
 .enqueue(**new** Callback<String>() {  
 @Override  
 **public void** onResponse(Call<String> call, Response<String> response) {  
  
 **try** {  
 JSONObject jsonObject=**new** JSONObject(response.body().toString());  
 JSONArray jsonArray=jsonObject.getJSONArray(**"routes"**);  
 **for** (**int** i=0;i<jsonArray.length();i++)  
 {  
 JSONObject route=jsonArray.getJSONObject(i);  
 JSONObject poly=route.getJSONObject(**"overview\_polyline"**);  
 String polyline=poly.getString(**"points"**);  
 polyLineList=decodePoly(polyline);  
  
 *//Adjusting bounds* LatLngBounds.Builder builder= **new** LatLngBounds.Builder();  
 **for**(LatLng latLng:polyLineList)  
 builder.include(latLng);  
 LatLngBounds bounds=builder.build();  
 CameraUpdate mCameraUpdate=CameraUpdateFactory.newLatLngBounds(bounds,2);  
 mMap.animateCamera(mCameraUpdate);  
  
 polylineOptions=**new** PolylineOptions();  
 polylineOptions.color(Color.GRAY);  
 polylineOptions.width(5);  
 polylineOptions.startCap(**new** SquareCap());  
 polylineOptions.endCap(**new** SquareCap());  
 polylineOptions.jointType(JointType.ROUND);  
 polylineOptions.addAll(polyLineList);  
 grayPolyline=mMap.addPolyline(polylineOptions);  
  
 blackPolylineOptions=**new** PolylineOptions();  
 blackPolylineOptions.color(Color.BLACK);  
 blackPolylineOptions.width(5);  
 blackPolylineOptions.startCap(**new** SquareCap());  
 blackPolylineOptions.endCap(**new** SquareCap());  
 blackPolylineOptions.jointType(JointType.ROUND);  
 blackPolyline=mMap.addPolyline(blackPolylineOptions);  
  
 mMap.addMarker(**new** MarkerOptions()  
 .position(polyLineList.get(polyLineList.size()-1))  
 .title(**"Pickup Location"**));  
  
 *//Animation* ValueAnimator polyLineAnimator=ValueAnimator.ofInt(0,100);  
 polyLineAnimator.setDuration(2000);  
 polyLineAnimator.setInterpolator(**new** LinearInterpolator());  
 polyLineAnimator.addUpdateListener(**new** ValueAnimator.AnimatorUpdateListener() {  
 @Override  
 **public void** onAnimationUpdate(ValueAnimator animation) {  
 List<LatLng>points=grayPolyline.getPoints();  
 **int** percentValue=(**int**)animation.getAnimatedValue();  
 **int** size=points.size();  
 **int** newPoints=(**int**)(size\*(percentValue/100.0f));  
 List<LatLng> p=points.subList(0,newPoints);  
 blackPolyline.setPoints(p);  
 }  
 });  
 polyLineAnimator.start();  
  
 carMarker=mMap.addMarker(**new** MarkerOptions().position(currentPosition)  
 .flat(**true**)  
 .icon(BitmapDescriptorFactory.fromResource(R.drawable.car)));  
  
 handler=**new** Handler();  
 index=-1;  
 next=1;  
 handler.postDelayed(drawPathRunnable,3000);  
  
 }  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 **public void** onFailure(Call<String> call, Throwable t) {  
 Toast.makeText(DriverHome.**this**, **" "**+t.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 });  
  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }  
 }  
  
 **private** List decodePoly(String encoded) {  
  
 List poly = **new** ArrayList();  
 **int** index = 0, len = encoded.length();  
 **int** lat = 0, lng = 0;  
  
 **while** (index < len) {  
 **int** b, shift = 0, result = 0;  
 **do** {  
 b = encoded.charAt(index++) - 63;  
 result |= (b & 0x1f) << shift;  
 shift += 5;  
 } **while** (b >= 0x20);  
 **int** dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));  
 lat += dlat;  
  
 shift = 0;  
 result = 0;  
 **do** {  
 b = encoded.charAt(index++) - 63;  
 result |= (b & 0x1f) << shift;  
 shift += 5;  
 } **while** (b >= 0x20);  
 **int** dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));  
 lng += dlng;  
  
 LatLng p = **new** LatLng((((**double**) lat / 1E5)),  
 (((**double**) lng / 1E5)));  
 poly.add(p);  
 }  
  
 **return** poly;  
 }  
  
 @Override  
 **public void** onRequestPermissionsResult(**int** requestCode, @NonNull String[] permissions, @NonNull **int**[] grantResults) {  
 **super**.onRequestPermissionsResult(requestCode, permissions, grantResults);  
 **switch** (requestCode)  
 {  
 **case** MY\_PERMISSION\_REQUEST\_CODE:  
 **if** (grantResults.length>0 && grantResults[0]== PackageManager.PERMISSION\_GRANTED)  
 {  
 buildLocationCallBack();  
 buildLocationRequest();  
 **if** (location\_switch.isChecked())  
 displayLocation();  
 }  
 }  
 }  
  
 **private void** setUpLocation() {  
 **if** (ActivityCompat.checkSelfPermission(**this**, Manifest.permission.ACCESS\_COARSE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED &&  
 ActivityCompat.checkSelfPermission(**this**,Manifest.permission.ACCESS\_FINE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED)  
 {  
 *//Rquest Run time permission* ActivityCompat.requestPermissions(**this**,**new** String[]{  
 Manifest.permission.ACCESS\_COARSE\_LOCATION,  
 Manifest.permission.ACCESS\_FINE\_LOCATION  
 },MY\_PERMISSION\_REQUEST\_CODE);  
 }  
 **else** {  
 buildLocationRequest();  
 buildLocationCallBack();  
 **if** (location\_switch.isChecked()) {  
 *//Geofire* drivers = FirebaseDatabase.getInstance().getReference(Common.driver\_tbl)  
 .child(Common.currentUberDriver.getCarType());  
 geoFire = **new** GeoFire(drivers);  
 displayLocation();  
 }  
 }  
 }  
  
 **private void** buildLocationCallBack() {  
 locationCallback = **new** LocationCallback() {  
 @Override  
 **public void** onLocationResult(LocationResult locationResult) {  
 **for** (Location location:locationResult.getLocations())  
 {  
 Common.mLastlocation=location;  
 }  
 displayLocation();  
 }  
 };  
 }  
  
 **private void** buildLocationRequest() {  
 mLocationRequest = **new** LocationRequest();  
 mLocationRequest.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);  
 mLocationRequest.setSmallestDisplacement(DISPLACEMENT);  
 mLocationRequest.setInterval(UPDATE\_INTERVAL);  
 mLocationRequest.setFastestInterval(FASTEST\_INTERVAL);  
 }  
  
 **private void** displayLocation() {  
 **if** (ActivityCompat.checkSelfPermission(**this**,Manifest.permission.ACCESS\_COARSE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED &&  
 ActivityCompat.checkSelfPermission(**this**,Manifest.permission.ACCESS\_FINE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED)  
 {  
 **return**;  
 }  
  
 fusedLocationProviderClient.getLastLocation()  
 .addOnSuccessListener(**new** OnSuccessListener<Location>() {  
 @Override  
 **public void** onSuccess(Location location) {  
 Common.mLastlocation=location;  
 **if** (mLastlocation!=**null**)  
 {  
 **if** (location\_switch.isChecked())  
 {  
  
 **final double** latitude=mLastlocation.getLatitude();  
 **final double** longitude=mLastlocation.getLongitude();  
  
 LatLng center=**new** LatLng(latitude,longitude);  
 LatLng northSide= SphericalUtil.computeOffset(center,100000,0);  
 LatLng southSide=SphericalUtil.computeOffset(center,100000,180);  
  
 LatLngBounds bounds=LatLngBounds.builder()  
 .include(northSide)  
 .include(southSide)  
 .build();  
  
 places.setBoundsBias(bounds);  
 places.setFilter(typeFilter);  
  
 *//Update to firebase* AccountKit.getCurrentAccount(**new** AccountKitCallback<Account>() {  
 @Override  
 **public void** onSuccess(Account account) {  
 geoFire.setLocation(account.getId(), **new** GeoLocation(latitude, longitude), **new** GeoFire.CompletionListener() {  
 @Override  
 **public void** onComplete(String key, DatabaseError error) {  
 *//Add marker* **if** (mCurrent!=**null**)  
 mCurrent.remove();  
 mCurrent=mMap.addMarker(**new** MarkerOptions()  
 .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker))  
 .position(**new** LatLng(latitude,longitude))  
 .title(**"Your Location"**));  
  
 *//Move Camera to this position* mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(**new** LatLng(latitude,longitude),15.0f));  
  
 }  
 });  
 }  
  
 @Override  
 **public void** onError(AccountKitError accountKitError) {  
  
 }  
 });  
 }  
 }  
 **else** {  
 Log.d(**"ERROR"**,**"Cannot Get Your Location"**);  
 }  
 }  
 });  
  
  
 }  
  
 @Override  
 **public void** onMapReady(GoogleMap googleMap) {  
 **try**{  
 **boolean** isSuccess=googleMap.setMapStyle(  
 MapStyleOptions.loadRawResourceStyle(**this**,R.raw.my\_map\_style)  
 );  
 **if** (!isSuccess)  
 Log.e(**"ERROR"**,**"Map Style Load Failed"**);  
 }**catch** (Resources.NotFoundException e){  
 e.printStackTrace();  
 }  
  
 mMap = googleMap;  
 mMap.setMapType(GoogleMap.MAP\_TYPE\_NORMAL);  
 mMap.setTrafficEnabled(**false**);  
 mMap.setIndoorEnabled(**false**);  
 mMap.setBuildingsEnabled(**false**);  
 mMap.getUiSettings().setZoomControlsEnabled(**true**);  
  
 **if** (ActivityCompat.checkSelfPermission(**this**,Manifest.permission.ACCESS\_COARSE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED &&  
 ActivityCompat.checkSelfPermission(**this**,Manifest.permission.ACCESS\_FINE\_LOCATION)!=PackageManager.PERMISSION\_GRANTED)  
 {  
 **return**;  
 }  
  
 buildLocationCallBack();  
 buildLocationRequest();  
 *//Update Location* fusedLocationProviderClient.requestLocationUpdates(mLocationRequest,locationCallback, Looper.myLooper());  
  
 }  
  
 **private boolean** hasGPSDevice(Context context) {  
 **final** LocationManager mgr = (LocationManager) context  
 .getSystemService(Context.LOCATION\_SERVICE);  
 **if** (mgr == **null**)  
 **return false**;  
 **final** List<String> providers = mgr.getAllProviders();  
 **if** (providers == **null**)  
 **return false**;  
 **return** providers.contains(LocationManager.GPS\_PROVIDER);  
 }  
 **private void** enableLoc() {  
  
 **if** (googleApiClient == **null**) {  
 googleApiClient = **new** GoogleApiClient.Builder(DriverHome.**this**)  
 .addApi(LocationServices.API)  
 .addConnectionCallbacks(**new** GoogleApiClient.ConnectionCallbacks() {  
 @Override  
 **public void** onConnected(Bundle bundle) {  
  
 }  
  
 @Override  
 **public void** onConnectionSuspended(**int** i) {  
 googleApiClient.connect();  
 }  
 })  
 .addOnConnectionFailedListener(**new** GoogleApiClient.OnConnectionFailedListener() {  
 @Override  
 **public void** onConnectionFailed(ConnectionResult connectionResult) {  
  
 Log.d(**"Location error"**,**"Location error "** + connectionResult.getErrorCode());  
 }  
 }).build();  
 googleApiClient.connect();  
  
 LocationRequest locationRequest = LocationRequest.create();  
 locationRequest.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);  
 locationRequest.setInterval(30 \* 1000);  
 locationRequest.setFastestInterval(5 \* 1000);  
 LocationSettingsRequest.Builder builder = **new** LocationSettingsRequest.Builder()  
 .addLocationRequest(locationRequest);  
  
 builder.setAlwaysShow(**true**);  
  
 PendingResult<LocationSettingsResult> result =  
 LocationServices.SettingsApi.checkLocationSettings(googleApiClient, builder.build());  
 result.setResultCallback(**new** ResultCallback<LocationSettingsResult>() {  
 @Override  
 **public void** onResult(LocationSettingsResult result) {  
 **final** Status status = result.getStatus();  
 **switch** (status.getStatusCode()) {  
 **case** LocationSettingsStatusCodes.RESOLUTION\_REQUIRED:  
 **try** {  
 *// Show the dialog by calling startResolutionForResult(),  
 // and check the result in onActivityResult().* status.startResolutionForResult(DriverHome.**this**, REQUEST\_LOCATION);  
  
 *//\* finish();* } **catch** (IntentSender.SendIntentException e) {  
 *// Ignore the error.* }  
 **break**;  
 }  
 }  
 });  
 }  
 }  
}

CONCLUSION

Food Delivery App:

* It seems like food ordering apps are going to become even more popular in the next few years.
* Luckily for you, there are still some free niches left which you can take.
* We hope our article will encourage you to create a food delivery app and win the market!

Car Booking App:

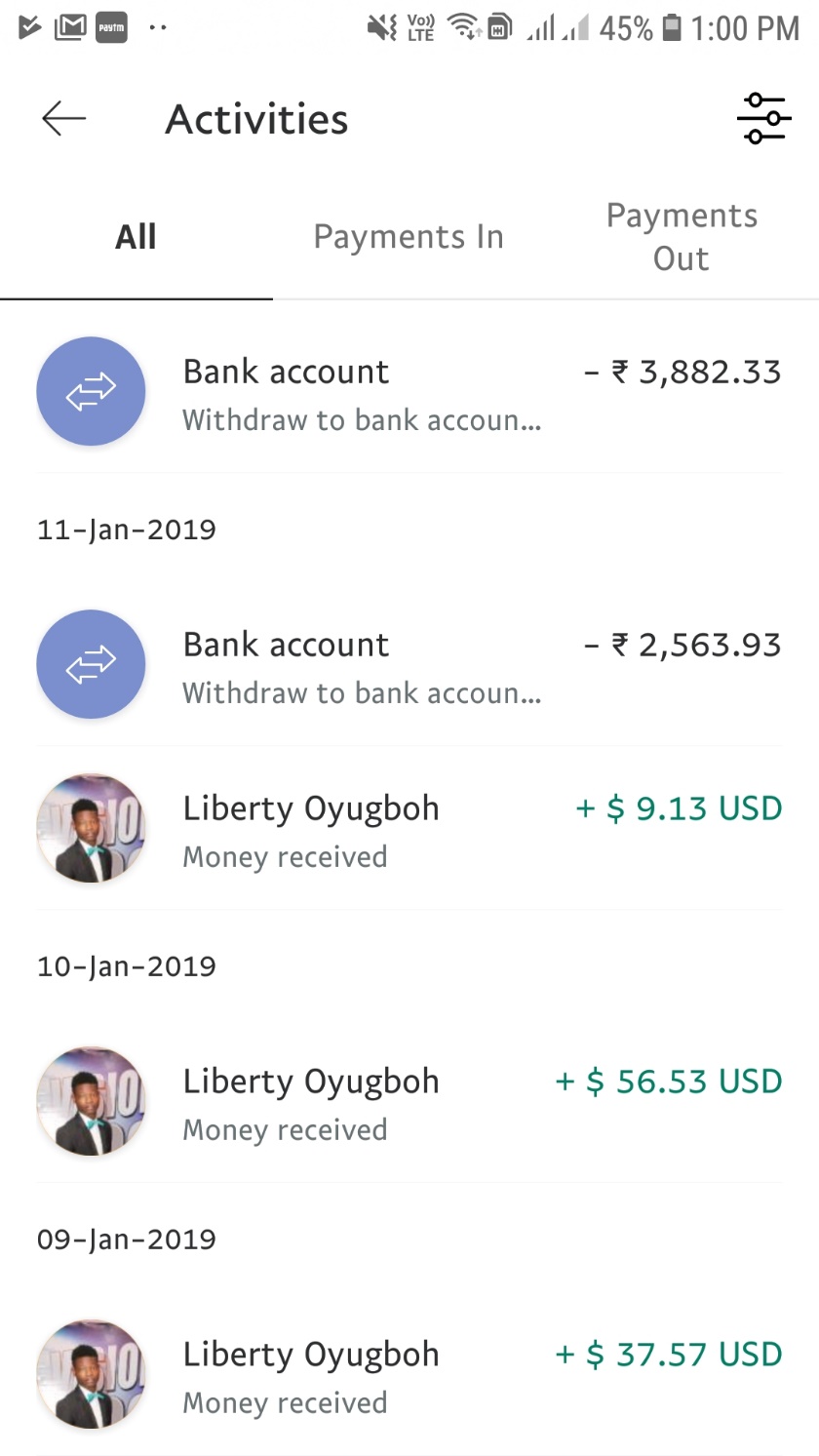
* Online car booking apps offer an easy way to book a car.
* To become the best car service provider, one needs to develop the car booking app patiently while keeping the customer’s basic requirements in mind. Eventually, in a customer-centric business, customer satisfaction is the essential factor to achieve success.
* So, offer the best services to the customers with the help of car booking app and let them spread the word of mouth about their experience.

Future Enhancement

* I will be developing more attractive UI/UX design which will be more user friendly.
* I will be working on app performances which will consume less CPU/RAM on all the devices.
* Better functionalities like 24/7 customer support.
* I will be integrating more payment gateways for quick payment and google API’s is going to be improved for live tracking updates with all the required information.
* I will also plan to add more services to app like dashboard, monthly reports and I will also migrate to web application and cloud storage.
* I will also integrate the advertising functionality which will be helpful for customers to find the services easily.
* With respect to requirements I will be implementing the required functionalities which will save time and work load.

Goals and Achievement

This project is made according to the requirements for the client (Liberty Oyugboh) which is successfully sold to Nigerian Citizen worth 110 USD i.e. Rs 7,480. The payment is received through Paypal which is a online payment gateway where people can send/receive money almost anywhere in the world for business and personal purposes. Below is the paypal payment statement for reference.



REFERENCES

1. [www.stackoverflow.com](http://www.stackoverflow.com)
2. [www.developer.android.com](http://www.developer.android.com)
3. [www.youtube.com/edmtdev](http://www.youtube.com/edmtdev)
4. [www.firebase.google.com](http://www.firebase.google.com)
5. [www.console.developers.google.com](http://www.console.developers.google.com)
6. [www.github.com](http://www.github.com)