INDIAN INSTITUTE OF
INFORMATION TECHNOLOGY, ALLAHABAD

Social Delivery : An Android Application

Ayush Agnihotri	IIM2015004
Nidheesh Pandey	IIM2015501
Abhishek Pasi	ICM2015002

UNDER SUPERVISION
of
DR. JAGPREET SINGH

DECLARATION

We hereby declare that the work presented in this project report entitled **"Social Delivery"**, submitted towards progress of summer project report of Dual Degree(IT) at **Indian Institute of Information Technology, Allahabad**, is an authenticated record of our original work carried out from **May 2018** to **present** under the guidance of **Dr. Jagpreet Singh**. Due acknowledgements have been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Signature Of Supervisor:

Date:

Signature Of Students:

Contents

List of Figures

List of Tables

Social Delivery

1. Introduction and Motivation

Online Shopping is growing at a fast pace. People prefer shopping for products online rather than offline. However, when you need to buy things from local markets less options are available. Normally, people go to shops or send somebody to buy goods for them. They may also ask someone who is going to market to buy things for them.

We have tried to provide a solution to above problem by developing an android application. With this application we aim to automate the process of communication between potential deliverers and shoppers.

People go for shopping, at the same time their friends and other people may also want to buy things from same or nearby markets. These People going for shopping can help their friends. They will also get some incentive in terms of delivery charge without much effort.

We aim to provide an efficient way to manage the shopping needs of people from local markets. Any user using our app is free to use it as a shopper or deliverer. Sometimes you can be a shopper and at other times you can be a deliverer. Shoppers are driven by their shopping needs and deliverers are motivated to use our app for incentive they get in return of their service.

We named this application as '**Social Delivery**', Since, it creates a social shopping environment .

2. Related work

As you already know, we have made an android application which helps people shop from local shops, it will be helpful to discuss what work has been already done in this space.

Let's have a look at the local shopping market space. While amazon is leader in online shopping and e-commerce, it has also entered into grocery delivering area. Ever heard of Google selling groceries and small daily use items? Yes it is true, Google which is one of the biggest technology company has also entered into this space with its **Google express**. From what we have seen it has partnered with various big store chains like walmart and offers free delivery over a certain cost. Although, it is not available in India as of now.

Taxi booking apps are also taking a step in local delivery space. Uber experimented with **UberFresh** in the past (2014). Now it has successfully launched **UberEats**, which serves worldwide. It aims to deliver fast food from local restaurant chains in about 10 minutes or less. We already know Zomato, Swiggy and Foodpanda are working in this area, but taxi services entering in this area is totally a new thing. Ola has acquired foodpanda and aims to do the same things as UberEats. Gett is another taxi app which brings items from shops and restaurants , it has joined hands with WunWun , a delivery app to facilitate the same. It delivers every product you can buy from a local shop.

Taxi companies are competing with big players like amazon and google . Although google is new in this space , being a tech giant , it has plenty of resources. Apart from that they have are giving major competition to delivery startups like Postmates , DoorDash and Instacart. We will talk about these startups in brief.

An important point to note is, these type of local delivery services are currently not available in India. However, food delivery services like zomato , swiggy and foodpanda are quite popular. Now coming to services which have quite a resemblance with our application. Local delivery startups like Instacart, DoorDash and Postmates are getting quite popular in the US.

Postmates works with both stores and restaurants to provide services to its customers. Currently it serves 14 cities across the US. An order placed by a user , is purchased and delivered by a local courier which they call hipster on wheels . It closely resembles what we have implemented in our android application.

Instacart, it focuses more on grocery shopping. It sends a local shopper to a grocery store you wanted to shop from, goods are delivered within an hour.

DoorDash helps in doing local delivery from restaurants . It takes about 45 minutes and serves areas in Silicon Valley , Los Angeles , Boston and Chicago.

Our application is quite simple and works on more or less on the same concepts described in this section. However , there are some key differences in our design and working style of application.

3. Software Requirement Specification

3.1. Introduction

3.1.1. Purpose

The purpose of this report is to describe and examine all assorted ideas that have come up to define our application and to give a detailed description of software requirements and technologies used to develop our application Social Delivery. It also describes its user interface and specifies all its functional and non-functional requirements as well as design constraints. Various use-cases are discussed and Use-case diagrams are also used to explain the structure of the application and provide all actions performed by various actors.

3.1.2. Scope

Our **Social Delivery** application will help people around the world who are using our app as Shoppers to order market products from their homes and get them delivered at their doorstep for a small delivery charge. Deliveries will be made by other users who are using the app as Deliverers. The users using our app as shoppers will benefit as they are getting their delivery to wherever they want without going anywhere themselves .The users who are using our app as deliverers will also benefit as they will get paid for the delivery in the form of delivery charge. In this way both the parties involved are happy.

3.1.3. Overview

The next section, **Overall Description**, of this document gives an overview of the functionalities of our application. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next section.

The third section, **Requirements**, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product. Both sections of the document describe the same software

product in its entirety, but are intended for different audiences and thus use different language.

3.2. Overall Description

3.2.1. Product perspective

- This app is an Android based application designed on **Android Studio** platform.
- It uses **Google Firebase** Database which is a NoSQL Real-time cloud-hosted Database. We store the profiles of all users and all the data of the orders placed by them in a JSON tree.
- It uses **Google Places API** to set the delivery location of an order and **Google maps** to help the Deliverers to reach the delivery location by showing them the path.
- Our app also uses **GPS Location** to find the current city of the user.
- Notification alerts are also sent to users on every status change of their orders using the **One Signal API**. The One Signal API is an open source service which uses **Google Cloud Messaging(GCM)** in its backend.

3.2.2. Product features

For Users using our app as Shopper :

- Place a new order.
- Edit existing pending orders.
- Filter the orders according to status like All, Active, Pending, Finished, Cancelled, Expired.
- See his/her PayTM wallet balance.
- Check the details of the deliverer who have accepted his/her order.
- Switch to deliverer view.
- Sign-out from the app.

For Users using our app as Deliverer:

◇ Accept a pending order out of a list of all pending orders from his city location.

◇ Reject an accepted order.

◇ See the path to reach the delivery location for an order.

◇ See his/her current location(city).

◇ Filter the orders according to status like All, Active, Pending, Finished, Cancelled, Expired.

◇ See his/her PayTM wallet balance.

◇ Complete the delivery of an accepted order by OTP verification.

◇ Switch to shopper view.

◇ Sign-out from the app.

3.2.3. User characteristics

- The user must have a smartphone with Android OS and must know how to operate basic android apps.

For Users using our app as Shopper:

◇ The user must know how to use pick places from a google maps.

◇ The description of the orders should be as clear as possible.

◇ The estimated price range of order should be given as close as possible to the real price.

◇ Appropriate expiry Date & Time of the order should be given (if any).

◇ Delivery location should be set appropriately.

◇ The user must know how to pay using PayTM wallet.

For Users using our app as Deliverer:

◇ The user should know how to use Google maps app.

◇ The user must know how basic OTP verification process works.

◇ Users using our app as Deliverers should choose wisely what orders they can deliver.

3.2.4. Constraints

- Active internet connection is required while using this app.
- E-mail ID of the users should be valid.
- GPS should be working properly.
- Google maps application(optional) should be installed in the Android device to be able to use SHOW PATH functionality.
- Database constraints:
  - As we are using the free version of the Firebase, we are limited to a maximum of 100 active connections (users) simultaneously at a time.
  - Currently we can store a total of 1 GB data on our database.

3.2.5. Assumptions and dependencies

- The device which is used for our app should have Android OS version 6 or above.
- We are assuming that the PayTM payment service which we are using for payment of order will be available all the time and will work without any errors.
- We are assuming the GPS location will always give correct results.
- No more than one user who is using our app as Deliverer should Accept a single order at the same time.

3.3. Requirements

3.3.1. Functional Requirements

Module-1 : User Login using E-mail & Password

AIM : User can login to our app using unique E-mail & Password

INPUT : E-mail and password.

OUTPUT : Displays a message if username and password does not match otherwise enter into the Usage mode screen.

PROCESS : Match username and password from database.

Module-2 : User Login using Google Sign-in

AIM : User can login to our app using G-Mail ID.

INPUT : G-Mail ID (E-mail and password).

OUTPUT : enter into the Usage mode screen.

PROCESS : Use GoogleApiClient.

Module-3 : Reset Password

AIM : User can reset password using registered E-mail.

INPUT : Registered E-mail.

OUTPUT : Password reset link sent to registered E-mail.

PROCESS : Use Firebase Authentication functions.

Module-4 : Registration

AIM : A new user can register to our app by filling the form.

INPUT : Name, Mobile No., Alternate Mobile No.(optional), E-mail and password.

OUTPUT : E-mail verification link sent to given E-mail if E-mail valid otherwise asks to enter valid E-mail address.

PROCESS : Use Firebase Authentication functions.

Module-5 : Verify E-mail

AIM : User will follow the instructions given in link for E-mail verification to verify E-mail before using our app.

INPUT : Press Refresh Button.

OUTPUT : Enter into the Usage mode screen if E-mail verification successful else remain in E-mail verification screen.

PROCESS : Save all data in database.

Module-6 : Re-send Verification E-mail

AIM : Re-send the E-mail verification link to entered E-mail while registering.

INPUT : Touch Re-send Verification Mail button.

OUTPUT : Verification E-mail sent to E-mail.

PROCESS : Use Firebase Authentication functions.

Module-7 : Place a new order

AIM : User can place a new order when using our app as Shopper by filling a form.

INPUT : Description or order, Category, Estimated price range minimum and maximum values, Expiry Date and Time of order(optional) and delivery location.

OUTPUT : PayTM payment gateway if all details are valid else displays appropriate message.

PROCESS : Check maximum and minimum value and whether all mandatory fields are filled.

Module-8 : Order payment

AIM : Finish payment of the order by paying from PayTM wallet using PayTM payments SDK.

INPUT : PayTM login details.

OUTPUT : Order details added in Database and details updated in Database if payment successful else no change in database.

PROCESS : Save all data in database and send notification to shopper.

Module-9 : Edit pending order

AIM : User can edit an existing pending order when using our app as Shopper by filling a form.

INPUT : Description of order, Category, Estimated price range minimum value, Expiry Date and Time of order(optional) and delivery location.

OUTPUT : opens order view screen if all details are valid else displays appropriate message.

PROCESS : Save all data in database.

Module-10 : Apply a filter

AIM : Displays the orders according to selected filters.

INPUT : Touch the filter option that you want to apply in the Navigation drawer of Shopper or Deliverer view.

OUTPUT : Recycler view containing cards of orders of chosen status only.

PROCESS : Database query to fetch orders from database of chosen status.

Module-11 : Sign-out

AIM : To Sign-out of the app.

INPUT : Touch Sign Out button in the Navigation drawer of Shopper or Deliverer view.

OUTPUT : Signs out of the app and displays the login screen.

PROCESS : Use Firebase Authentication functions.

Module-12 : View as Deliverer

AIM : Switch to Deliverer mode from the Shopper mode.

INPUT : Touch View as Deliverer button in the Navigation drawer of Shopper view.

OUTPUT : Opens the Deliverer view.

PROCESS : Call the intent for the Deliverer view.

Module-13 : View as Shopper

AIM : Switch to Shopper mode from the Deliverer mode.

INPUT : Touch View as Shopper button in the Navigation drawer of Deliverer view.

OUTPUT : Opens the Shopper view.

PROCESS : Call the intent for the Shopper view.

Module-14 : Show Path

AIM : Shows the path to delivery location of an order to the Deliverer using Google maps.

INPUT : touch Show Path button.

OUTPUT : Opens the path in Google maps.

PROCESS : Calls the Google maps app.

Module-15 : Accept an order

AIM : Accept a pending order.

INPUT : touch Accept button in Deliverer order detailed view.

OUTPUT : Enables Complete Order button.

PROCESS : Update details in database and send notification to shopper.

Module-16 : Reject an order

AIM : Reject an accepted order.

INPUT : touch Reject button in Deliverer order detailed view.

OUTPUT : Disables Complete Order button.

PROCESS : Update details in database and send notification to shopper.

Module-17 : Complete an order

AIM : Complete the delivery of an accepted order.

INPUT : touch Complete Order button in Deliverer order detailed view.

OUTPUT : opens SENT OTP screen.

PROCESS : calls intent for SENT OTP screen.

Module-18 : Send OTP

AIM : Send OTP to the shopper from the deliverer.

INPUT : actual price of item bought.

OUTPUT : opens ENTER OTP screen.

PROCESS : Update details in database and send notification to shopper.

Module-19 : Finish Order

AIM : Finish the delivery of an order. INPUT : OTP as received from shopper.

OUTPUT : Delivery finished if correct OTP entered otherwise displays message.

PROCESS : Update details in database and send notification to shopper.

3.3.2. Non- functional Requirements

3.3.2.1. Performance Requirements

The response time should be less and the actions taken by user should not take much time. The access to database should be fast enough. The amount of graphics/animations should be such that it ensure all the above points.

3.3.2.2. Availability

The application will run 24x7 if internet connection is available.

3.3.2.3. Security

The Database used to store all the application related data should be secure against any malicious activity.

3.3.2.4. Reliability

This application has been tested on Android Oreo, Android Marshmallow, custom UIs like MIUI (Xiaomi phones), EMUI (Honor phones) and will run on any android phone having Android SDK version 6 or above.

3.3.2.5. Fault Tolerance

The data stored in the database should not get corrupted by any means like system crash, power failure or lost internet connectivity. The information stored in database should be resistance to any unwanted changes.

3.3.2.6. Portability

The application should work on any android device like mobile phones, tablets, fablets etc.

3.3.2.7. Maintainability

It should be easy to add any new feature as per requirement anytime.

3.3.3. External Interface Requirements

3.3.3.1. User Interfaces

→ For Front-end : XML(eXtensible Markup Language) and Android(Java)

→ For Back-end : Google Firebase Database

3.3.3.2. Hardware Interfaces

Any device with Android OS like mobile phone, tablet or fablet.

3.3.3.3. Software Interfaces

◇ Softwares :

→ Android Studio : Development environment for Google's Android operating system.

◇ Languages :

→ For Front-end : XML(eXtensible Markup Language) and Android(Java)

→ For Back-end : Java

◇ Tools / Libraries:

→ Google Sign-in

→ E-mail verification (using Firebase Authentication)

→ Google Places API

→ Google Maps

→ Location

→ PayTM payments SDK

→ OneSignal API (for Notifications)

→ Lucidchart.com (for diagrams)

◇ Databases :

→ Google Firebase (NoSQL Real-time cloud-hosted Database) :

1. Data is synced across all clients in realtime, and remains available when your app goes offline.
2. Data is stored as JSON (JavaScript Object Notation) tree.

3.3.3.4. Communications Interfaces

Internet connection in the device through any medium like 3G, 4G or Wifi.

3.3.4. Use-cases

This application is mainly used for shopping for items from local stores.

- In our current design of application we have a community of shoppers and deliverers using our app.
- Any user can choose to become a shopper or deliverer . Some days you can be a shopper and on other days you can be a deliverer.

Use case 1 : as a shopper.
You want to buy some item from a shop but you are too busy to go there . You can simply use this application and someone will deliver the product at your doorstep.

Use case 2 : as a shopper with some urgency.
You want some item before a certain time . Just put your deadline as expiry date . It is useful when you dont want anyone to accept your order after your deadline . It is also obvious that deliverers will process the order which has more incentive. We can have provision for tips also . Although , it is not included in current version of app.

Use case 3: as a deliverer who is in market.
You are currently in main market of your city and you are done with your shopping . You open our app and find many users want goods which are easily available in this market . Its an earning opportunity for you.

Use case 4: as a deliverer with a will to earn.
Today you have some free time . You open the application and see many orders with a familiar destination and marketplace. You decide it is a good opportunity to earn something . You accept some orders and process them to earn

incentives (delivery charge).
The above four use cases clearly describe the utility of our application . However , with the same design and some changes we can easily convert this application to have more use cases.

Example of few such use cases :-

a. Partnering with local shops and having centralised system of delivery between the shop and its customers.

b. Having listing of shops and using the application to buy goods from shops.

c. Partnering with restaurants for booking tables or ordering food items from their menu.

4. Application Design

4.1. UI Design

We have made an android application which helps people shop from local stores. Using our application you can order any goods or food items and it will be delivered to you by one of our deliverer.

As our app is a kind of a transportation app, we initially thought of making the user interface similar to that of the **OLA** app.
The operations would have been like follows:

For placing a new order:

- A map would have been there which will show all the deliverers, out of which a deliverer would be alloted to us for our delivery.
- the selected deliverer would be notified of our order and that deliverer would buy our order and deliver it to us at the location chosen at the time of placing the order.

But that would type of interface would have brought a lot of problems like:

- As the deliverer is selected randomly and not of his free will, there are high chances of order rejection, as that deliverer may not want to go to the place from where the order is to be purchased or does not intend to go to the delivery location.
- The deliverer may not want to deliver your order because of low delivery charge (delivery incentive).
- So, we may have to get our order rejected a lot of times before we could actually get a deliverer who is willing to deliver our order to us.

So, we needed to ensure that the deliverer **chooses** to deliverer our order.

We solved these problems by choosing a different User interface:
The new operations are like follows:
For placing a new order:

- Instead of using a map and choosing a deliverer at the time of placing the order, we created a list of all pending orders, just like unread mails

in **google mail**. This method also reduces our dependency on google apis which are costly.

- We also have filters for pending , active , cancelled , finished and expired orders which can be compared with different kind of mail filters we have in gmail app like inbox , outbox , drafts , social etc.
- So now every newly placed order will be placed in the list of pending orders which is a global list visible to all the users using our app as a deliverer.
- All deliverers can check the details of your order, and the deliverers who are interested in delivering your order will **Accept** your order and will deliver it to you.
- You will be notified of as soon as someone **Accepts** your order.

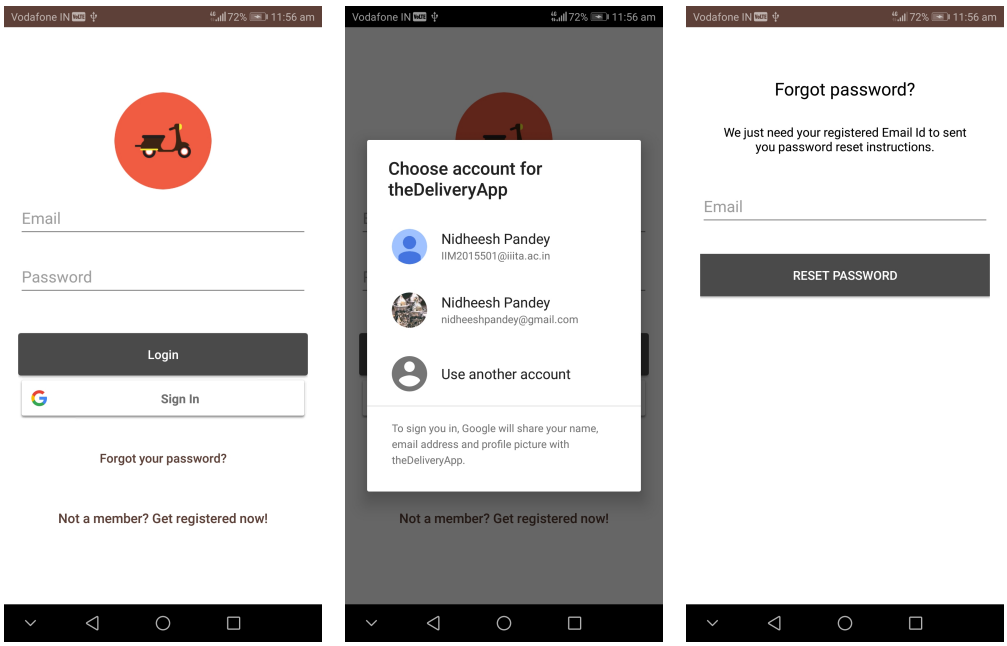
This list of pending orders is implemented using **Recycler View of cards** which is a very efficient method for displaying a big list of objects in any android application.

4.1.1. Login Screen

- This is the first screen which is displayed when the app is opened.
- There are two methods for login:

  → E-mail & Password

  → Google Sign-in



(a) Login screen using E-Mail & Password (b) Reset Password screen (c) Forgot password screen

Fig - 4.1: Login Screen and Password Reset

4.1.2. Forgot Password Screen

- If you ever forget your password, just enter your registered E-mail and the password reset instructions will be sent to that E-mail.

4.1.3. New Registration Screen

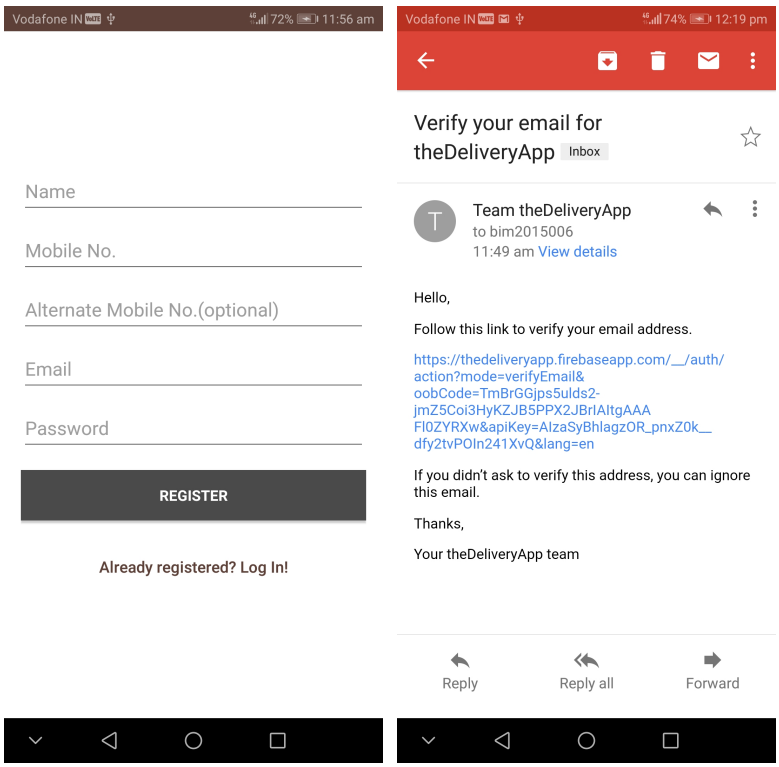
- This is the screen for registration for the E-mail & Password method in our database.

4.1.4. Choose Mode of Usage

- This is the screen where you choose whether you want to act as a Shopper or as a Deliverer.
- The interface of both the modes is very similar.

4.1.5. Shopper View

- All of your orders will be displayed in this screen in the form of a Recycler View of cards where each card is used to show an order.



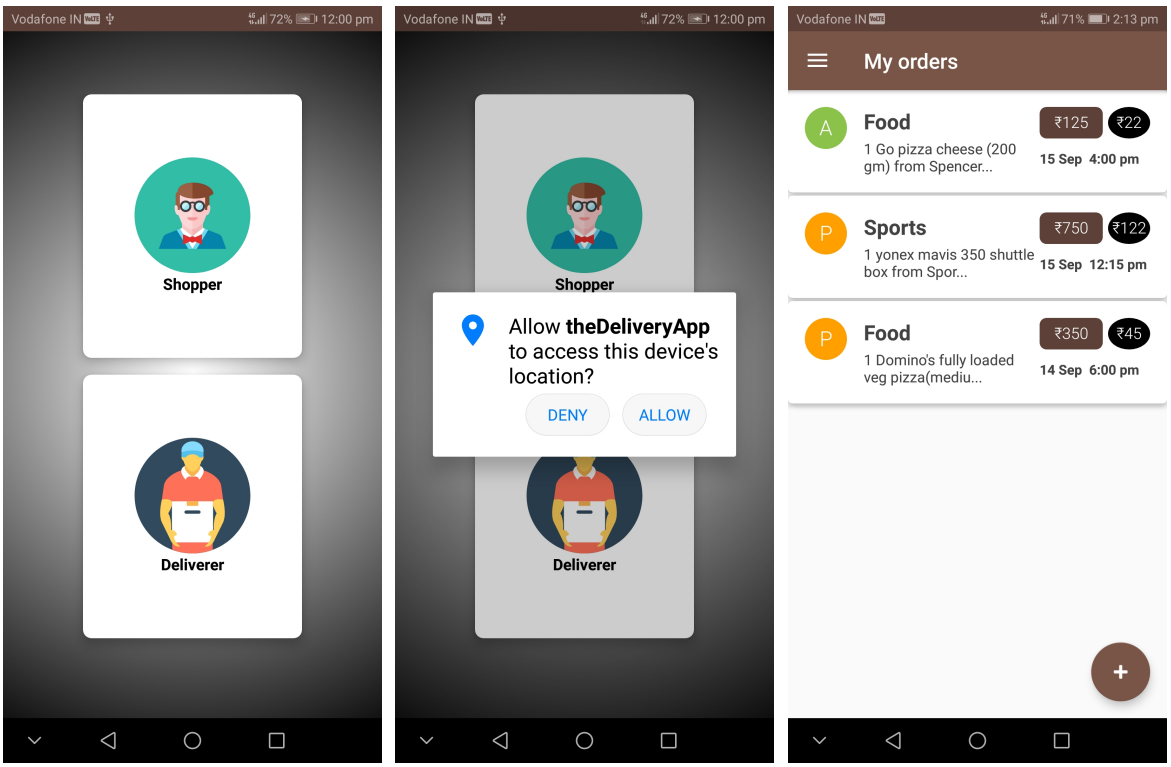
(a) User Registration form (b) E-Mail verification Mail

Fig - 4.2: Registration screen and E-Mail verification mail

- The default filter shows all your Pending and Active orders on this screen.
- Each order is summarised in the form of a card in the Recycler view showing its Status, Category, Description, Maximum range price, Delivery Charge and the Expiry Date & time.
- On the top left there is the toolbar menu button for displaying the Navigation Drawer.
- On the bottom right there is a PLUS Floating Action Button which is used for placing a new order.

4.1.6. Shopper Order Detail View

- Clicking an order in the Shopper View will open the detailed view of that order in a new activity.
- All the details of that order can be found here in this screen which also contains a nice Collapsing Toolbar Layout.
- There is also an EDIT floating Action Button on the top right of the screen using which we can edit the details of that order.
- When an order is accepted, the OTP will also be displayed in this screen and the details of the deliverer will also be shown.

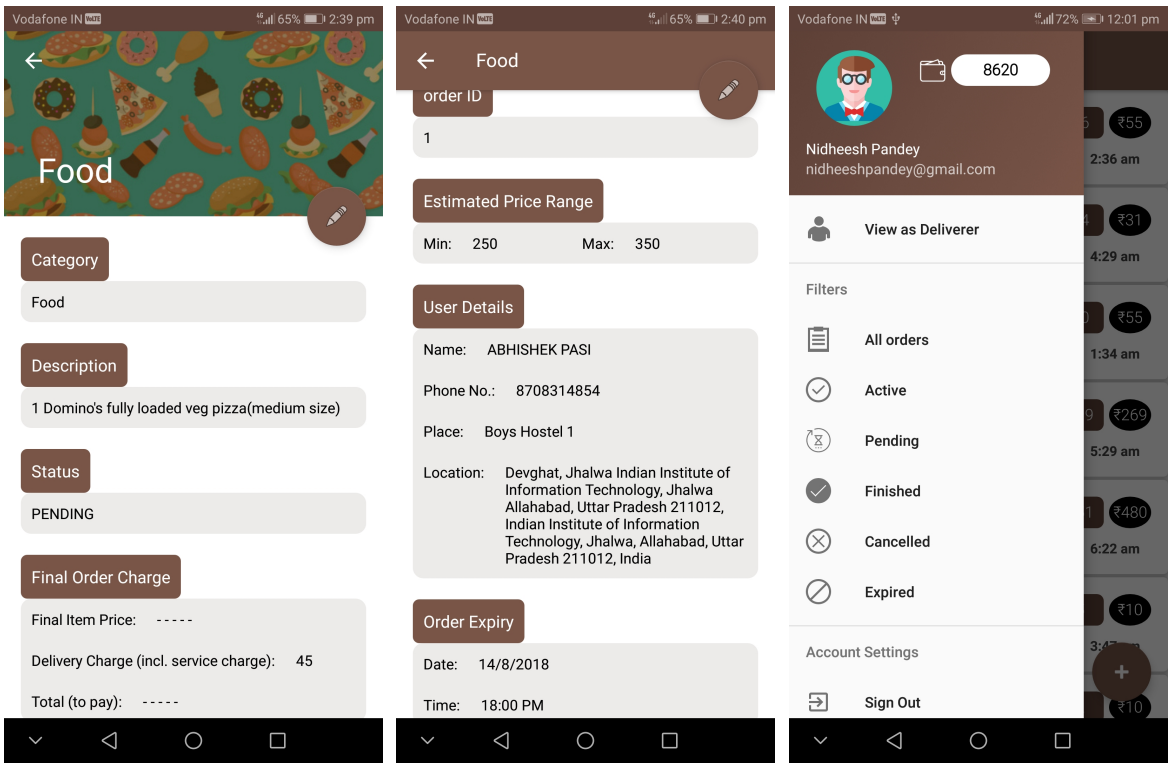


(a) Usage mode screen (b) GPS Location permission (c) Shopper view showing Active & Pending orders

Fig - 4.3: Usage mode screen and Shopper view screen

4.1.7. Shopper Navigation Drawer

- Touching the top left toolbar Menu button opens the Navigation Drawer for the Shopper which can also be opened by sliding the screen from the left boundary of the mobile to a slight right.
- This Navigation Drawer shows your basic profile details along with your current PayTM wallet balance.
- There is the option to switch to the Deliverer View i.e. use the app as a Deliverer to deliver orders.
- We also have various filters for order display like All orders, Active, Pending, Finished, Cancelled and Expired orders. Touching any one of them results in the display of only those type of orders(if any) in the Recycler view.
- On the bottom of the Navigation Drawer is the option to Sign-Out of the app.

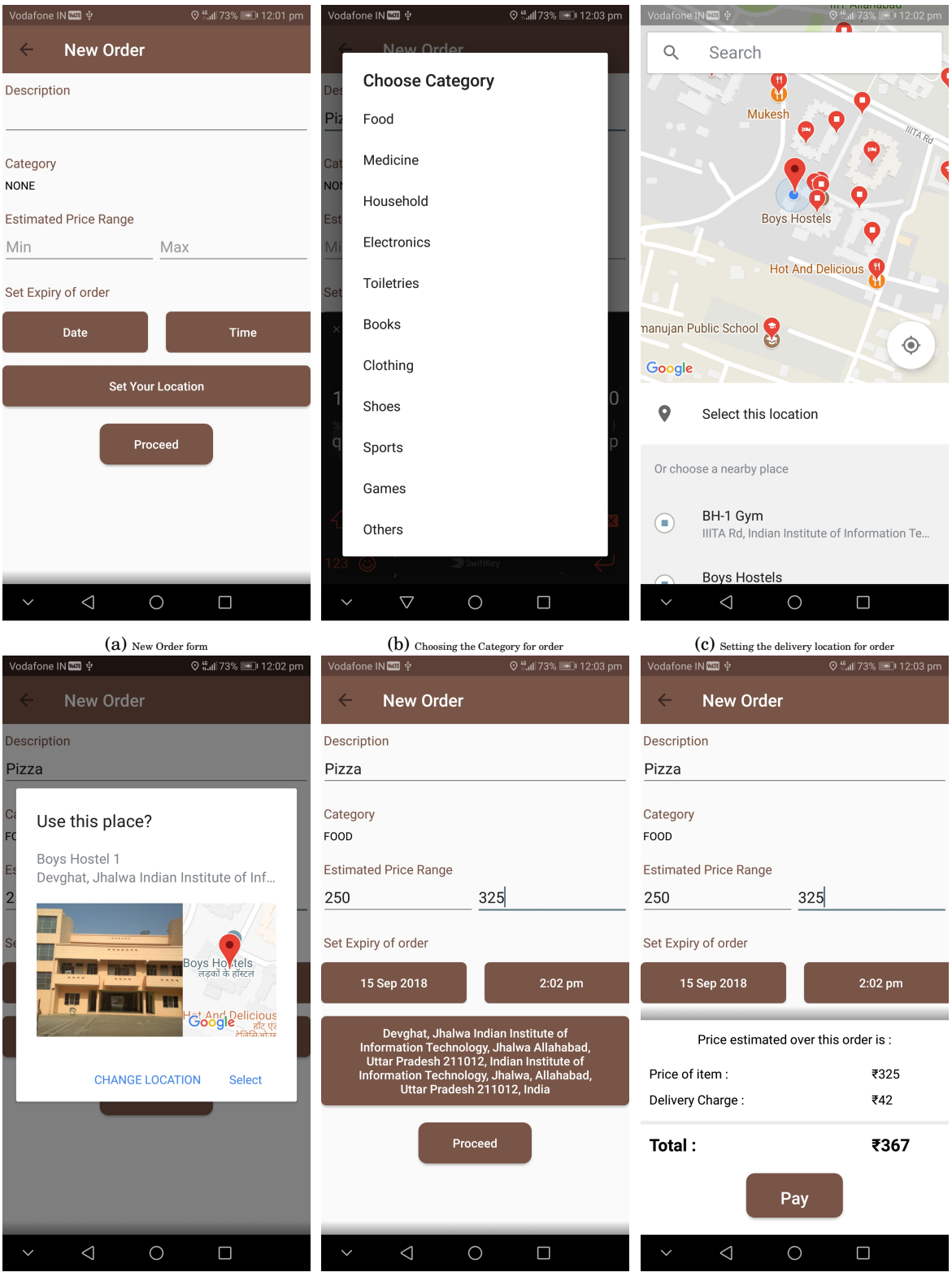


(a) Order detail view (b) Order detail view (c) Shopper Navigation

Fig - 4.4: Shopper order Detail view and Shopper Navigation drawer

4.1.8. Placing New Order

- Touching the PLUS Floating Action Button on the bottom right on the Shopper View screen opens up the form for placing a new order.
- We need to give the order Description, its Category, an estimated price range.
- We can choose to give the expiry Date and Time for that order, expiration of after which that order will automatically be removed from the global pending orders list of Deliverers.
- We also need to set the delivery location for our order i.e the location where we want our order to be delivered. We are using Google Places API for doing this.
- We are storing this Delivery location in our database, so that it can be used by the deliverer to track the delivery location.
- Payment of the order is being done from the shoppers PayTM wallet using PayTM payments SDK.
- After successful payment and placing of order, that order is added as a card in the Recycler view in the Shopper View screen with status as PENDING.



(a) New Order form

(b) Choosing the Category for order

(c) Setting the delivery location for order

(d) COnfirming the delivery location for order

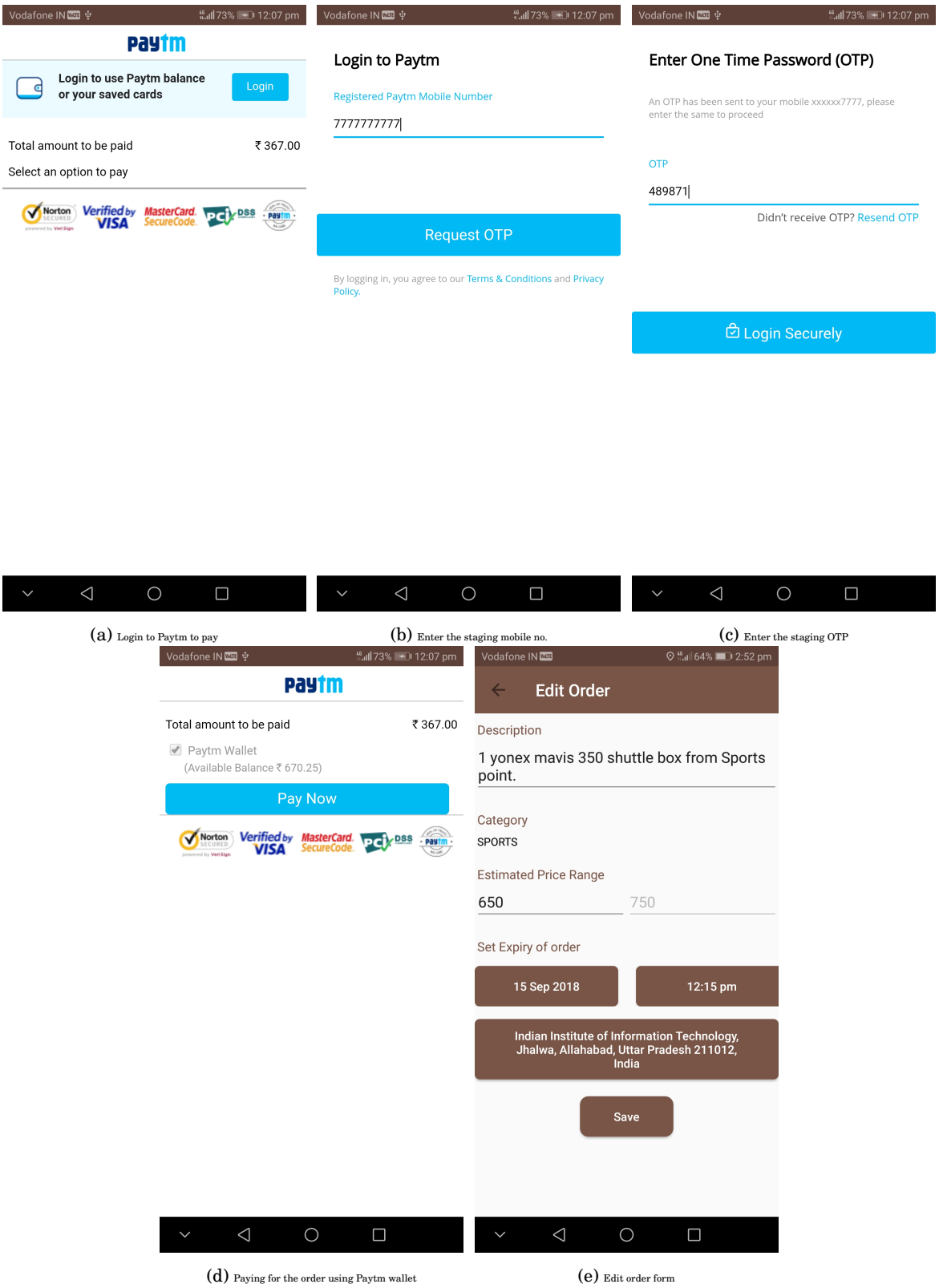
(e) Filled form

(f) Order Price display using Bottomsheet

Fig - 4.5: Placing new order

4.1.9. Editing an Order

- On touching the EDIT floating Action Button on the top right of the Shopper Order Detail View screen, an edit form will be opened for that order



(a) Login to Paytm to pay (b) Enter the staging mobile no. (c) Enter the staging OTP

(d) Paying for the order using Paytm wallet (e) Edit order form

Fig - 4.6: Payment of order and Editing of order

using which we can edit the details of that order.

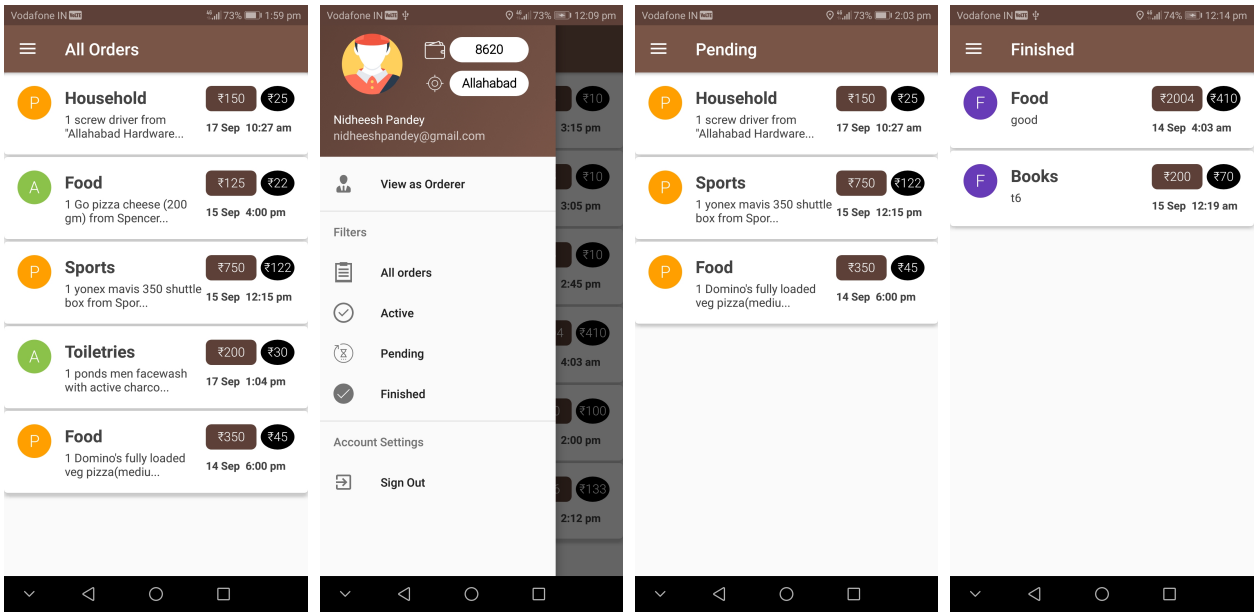
4.1.10. Deliverer View

- This screen is just like the Shopper View but according to a Deliverer.

- You can only see the PENDING orders of the shoppers who are in the same city as yours.
- Along with pending orders of the shoppers of your city, the default filter shows all of your Active orders, i.e. the order which you have accepted to deliver.
- For Accepting an order from the pending orders, you have to open the order's detailed view by touching the card for that order.

4.1.11. Deliverer Navigation Drawer

- Along with your basic profile information, wallet info. , filters and Sign-out option, here, the Navigation Drawer also displays your current location (city).



(a) Deliverer order view (b) Deliverer Navigation drawer (c) Pending orders (d) Finished orders

Fig - 4.7: Deliverer view

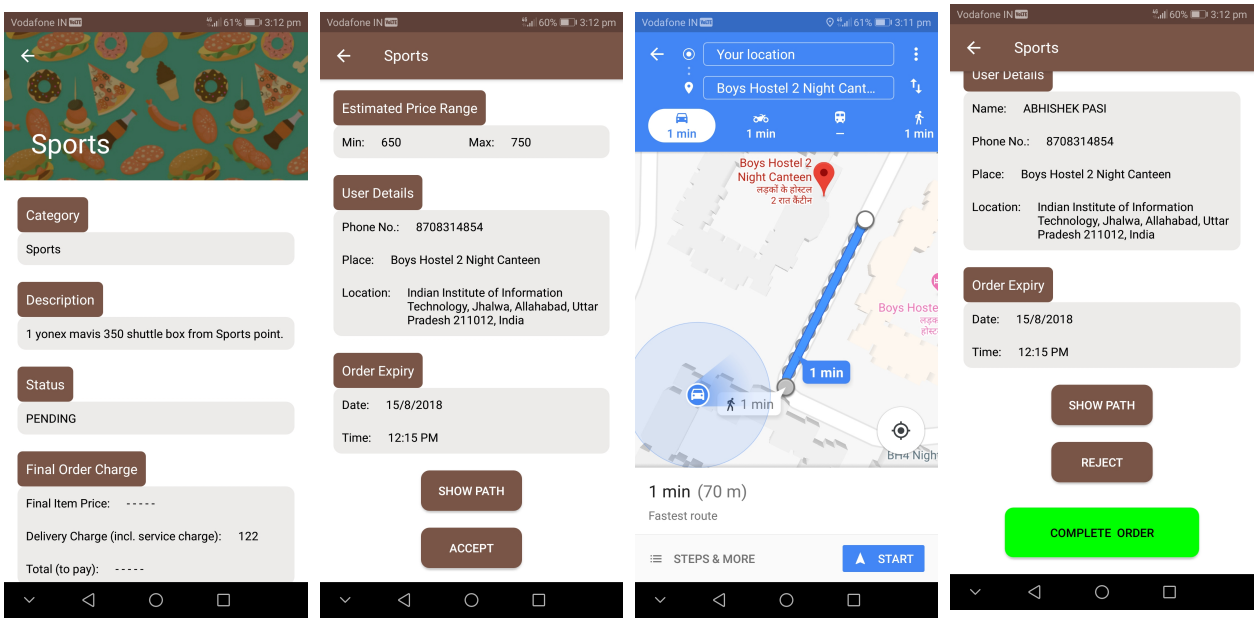
4.1.12. Deliverer Order Detail View

- It is similar to the Detailed order view of Shopper but according to a deliverer.
- There is a SHOW PATH button which shows the path from your current location to the delivery location of that order.
- When the order is pending, there is an ACCEPT button.

- When the order is ACCEPTED, there is a button to reject as well as a button to COMPLETE ORDER.

4.1.13. Accept an Order

- A deliverer can Accept a pending order by touching the ACCEPT button in the detailed view of that order.



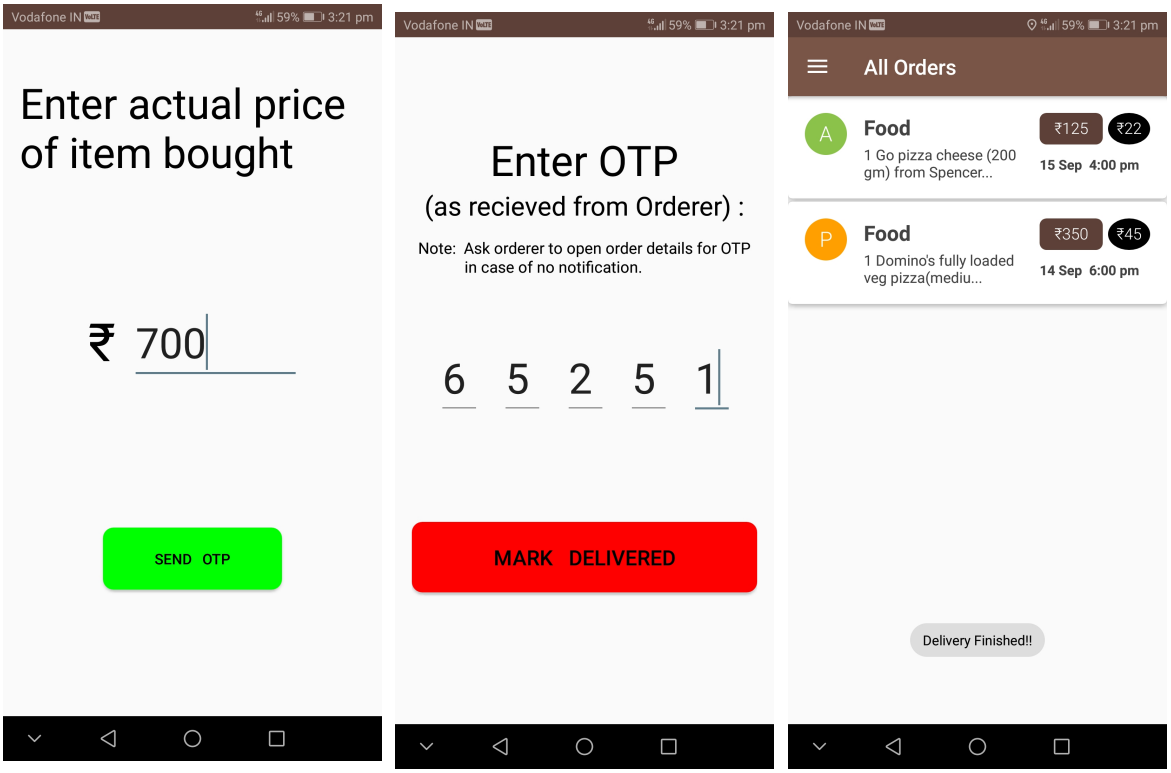
(a) Deliverer Order detail view (b) Deliverer Order detail view (c) Show Path functionality using Google Maps (d) Accepting an order by touching the Accept button

Fig - 4.8: Accepting an order

4.1.14. Complete Order

- By touching the COMPLETE ORDER button, we start the process of completing the order.
- First a screen is displayed which asks the deliverer to enter the Actual price paid to buy the order, after filling the price, the SEND OTP button is activated, using which an OTP along with the price as filled by the deliverer is sent to the respective shopper of that order in the form of a notification.
- Then, a new screen is displayed to the deliverer, asking to enter the OTP as received from the shopper.
- The OTP is working as a confirmation from the shopper. So, a shopper will give the OTP to the deliverer only when he/she is satisfied with the price and quality of the delivery.

- After entering the correct OTP by the deliverer, the delivery of that order is considered to be finished.



(a) Sending OTP to Shopper after entering the Actual price of order

(b) Entering the OTP as received from Shopper to finish Delivery

(c) Message confirming the Finishing of Delivery of order

Fig - 4.9: Completing the Delivery of an order

5. Technology used

◇ Softwares :

→ Android Studio : Development environment for Google's Android operating system.

◇ Languages :

→ For Front-end : XML(eXtensible Markup Language) and Android(Java)

→ For Back-end : Java

◇ Tools / Libraries:

→ Google Sign-in

→ E-mail verification (using Firebase Authentication)

→ Google Places API

→ Google Maps

→ Location

→ PayTM payments SDK

→ OneSignal API (for Notifications)

→ Lucidchart.com (for diagrams)

◇ Databases :

→ Google Firebase (NoSQL Real-time cloud-hosted Database) :

1. Data is synced across all clients in realtime, and remains available when your app goes offline.
2. Data is stored as JSON (JavaScript Object Notation) tree.

6. Business Model

Our application can be used as a Business application or it can be used among friends for their logistic and shopping needs.

As a business plan we are charging shoppers some delivery charge for the service we provide. The deliverer gets some incentive for delivering the order.

The delivery charge can be shared between us and the delivery person. When we find less deliverers are willing to use our app, we may increase their incentive to make up for the number of deliverers.

Also, we initially planned to launch this app in our college. We didn't do so due to time constraints and some other reasons.

For more smooth functioning of our application we may collaborate with third party delivery services.

We will discuss more business opportunities in future work section of our report.

7. Current status of Project

Currently all the basic features mentioned in this report are working properly.

We have tested our application for various scenarios like

i) if network is not working properly we are not allowing any action such as accept or reject as it may lead to confusion between users.

ii) we are asking for location so that one can only accept same city orders.

iii) Updating the user screen as soon as status of any of the order changes to avoid ambiguity that may arise if a user cancels an active order.

iv) Since our app is quite realtime (order status might change anytime) , we have taken care of any of the corner cases that may cause conflicts in status of orders. However , it may happen that some cases did not cross our mind, you may suggest if you find any of those. We will fix issues as soon as we find them.

We don't have access to all the features that PayTm gateway provides . Since we have testing credentials for the gateway , we can only simulate the process of making of payment currently . However, the gateway is fully functional . To get a genuine merchant id , we need to register a company name.

8. Future scope

While discussing Use case section of Software requirement Specification (SRS) of our project , we told about some use cases which can be thought of . We may modify our application to suit specific needs.

Example of few such use cases :-

I Partnering with local shops and having centralised system of delivery between the shop and its customers.

II Having listing of shops and using the application to buy goods from shops.

III Partnering with restaurants for booking tables or ordering food items from their menu.

9. How to use our App

Here we present a simple and summarised way on how to use our app. Our app has two modes of usage namely Shopper and Deliverer and corresponding two main functionalities namely Place an order and Deliver an order. Below we give summarised steps on how to execute these two main functionalities.

9.1. Place an order

→ Choose Shopper mode.

→ Click on the floating action button ('+').

→ Fill all the details.

→ Add location from the map. (uses google place picker api).

→ Add expiry date and expiry time[1].

→ Click proceed button after you have completed.

→ A bottom sheet appears which calculates and shows delivery charge.

→ After that you need to pay for the order using Paytm wallet[2].

→ You will be prompted to login into Paytm. (Don't login into you real Paytm account instead use test credentials provided by us).

Mobile No.	–	7777777777
OTP	–	489871

Few things to note :-

- ◇ You can left swipe to cancel this order.
- ◇ You can undo the cancel operations within 3 seconds.

→ When payment is complete you will be redirected back to User Screen.

→ You can see your order on the screen.

[1] You order expires as soon as clock hits the expiry time and date. No deliverer will be able to see an expired order.
[2] Right now we are in sandbox mode, we have a demo merchant id.

9.1.1. Circle Notation :

- P - pending orders
- A - active orders
- F - Finished orders
- C - Cancelled orders
- E - Expired Orders

These all type of orders can be accessed via filters in navigation bar.

9.2. Deliver an order

→ Choose Deliverer mode.

→ You will see various pending orders on your screen.

→ Open the order you are interested to deliver.

→ Accept the order. (A notification goes to customer associated with that order)

→ You may also reject, if you change your mind. (Again, a notification goes to user)

→ A bright green button named COMPLETE ORDER appears on the order detail page of the order you accepted.

→ After you purchase the item press the bright green COMPLETE ORDER button.

→ You will be asked to enter actual price of item you bought (excluding delivery charges).

→ You press a send OTP button.

→ Enter the OTP provided by user to finish the order.

10. About Database

- We are using Google Firebase Database which is a NoSQL Real-time cloud-hosted Database.
- As our app is quite real-time (order status might change anytime) we needed the properties of a real-time database.
- Real-time syncing makes it easy for our users to access their data from any device. Real-time Database also helps your users collaborate with one another as data is synced across all clients in real-time, and remains available when our app goes offline.
- All the database queries like reading and writing data are fully asynchronous. This means that the call always returns immediately, without blocking the code to wait for a result. The results come some time later, whenever theyre ready, this results in overall faster response time for our app. Also asynchronous tasks run on parallel threads which make use of multiprocessing power of our device.
- If queries were synchronous , it will hold on the app till the result is obtained . This may lead to crashes and freezes in user interface .

10.1. NoSQL v.s SQL database

- NoSQL databases were created to overcome the limitations of relational database technology.
- NoSQL databases are more flexible, scalable and also superior in the performance when compared to relational databases.
- NoSQL helps to store information without doing upfront schema design.
- NoSQL data model has fewer restrictions and easily allow new changes without creating much fuss.
- NoSQL is cost effective compared to RDBMS and allows to process and store more data at a much lower cost because RDBMS rely on overpriced storage and proprietary server systems while NoSQL databases commonly rely on clusters of cheap commodity servers.

10.2. Cloud-hosted vs Local server

- The application will run 24x7 if internet connection is available.
- Data is synced across all clients in real time, and remains available when our app goes offline.
- As we are using third-party (Google Firebase) as our cloud hosting provider, we do not need to worry about security and where the data is stored.
- Whereas in local server we would have to maintain it fully and ensure that the server is available 24x7.

10.3. Database Structure

All the data is stored as JSON (JavaScript Object Notation) tree.

- JSON is preferred over XML as the primary data format for transmitting back data, because it is much more lightweight.
- It is easy for humans to read and write.
- It is easy for machines to parse and generate.
- It is a text format that is completely language independent.

JSON is built on two universal data structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

10. References

[1] Cong-Kai Lin, Yang-Yin Lee, Chi-Hsin Yu, and Hsin-Hsi Chen. Exploring ensemble of models in taxonomy-based cross-domain sentiment classification. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1279–1288. ACM, 2014.

[2] Arun Reddy Nelakurthi, Hanghang Tong, Ross Maciejewski, Nadya Bliss, and Jingrui He. User-guided cross-domain sentiment classification. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 471–479. SIAM, 2017.

[3] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web*, pages 751–760. ACM, 2010.

[4] Fangzhao Wu, Yongfeng Huang, and Zhigang Yuan. Domain-specific sentiment classification via fusing sentiment knowledge from multiple sources. volume 35, pages 26–37. Elsevier, 2017.

[5] Fangzhao Wu, Sixing Wu, Yongfeng Huang, Songfang Huang, and Yong Qin. Sentiment domain adaptation with multi-level contextual sentiment knowledge. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 949–958. ACM, 2016.