CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

SALESCAPE

by

ABHINAV RATHI (1941003)

Under the Guidance of

Dr Vaidhehi V

Project report submitted in partial fulfilment of the requirements of V Semester BCA, CHRIST (Deemed to be University).

December - 2021

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

CERTIFICATE

*This is to certify that the report titled **Salescape** is a bona fide record of work done by **Abhinav Rathi (1941003)** of CHRIST (Deemed to be University), Bangalore, in partial fulfilment of the requirements of V Semester BCA during the year 2021.*

Head of the Department

Valued-by:

Faculty In charge

Name : Abhinav Rathi

Register Number(s) : 1941003

Examination Centre : CHRIST (Deemed to be University)

Date : 02/12/2021

ACKNOWLEDGEMENTS

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. A special gratitude we give to **Prof. Joy Paulose**, **Head of Department**, Computer Science, CHRIST (Deemed to be University) and **Prof. Deepthi Das**, **UG Coordinator**, Department of Computer Science, CHRIST (Deemed to be University) whose contribution in stimulating suggestions and encouragement, helped us to coordinate my project especially in writing this report.

We are using this opportunity to express our gratitude to the project guide, **Dr. Vaidehi V, Professor,** Department of Computer Science, CHRIST (Deemed to be University). We are thankful for her aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Furthermore, we would also like to acknowledge with much appreciation the crucial role of our classmates in helping us to assemble the parts for the success of "Salescape". We must appreciate the guidance given by other supervisors as well as the panels especially in our project presentation that has improved our presentation skills. Thanks to their valuable comments and advice. Last but not least, many thanks goes to the other faculties who invested their full effort in guiding the team in achieving the goal

ABSTRACT

Salescape is an app-based service that focuses on reducing the gap between the wholesaler and the retailer and digitizing the business trail between them. This service consists of 3 applications for the three different users. Firstly, the admin application is handled by the wholesaler, next the client application is handled by the retailer and the third application, the employee application is used by the logistics worker or the salesmen who work under the wholesaler. The admin app enlists the products of the wholesaler for retailers and also manages the orders placed. The employee app keeps track of the delivery and giving updates related to sales targets. The retailer app is for placing orders and interacting with the wholesaler. The main goal of this project is to minimise the workload in the working of a transaction between a wholesaler and a retailer. It is a digitalized and more efficient way of dealing with the transactions without extra technical complexities.

The Wholesaler app is connected to the entities such as registration entity and order tracking entity which it shares with the retailer, and it also deals with the attendance management entity and location tracking entity along with the employee app. The Salesman app is connected to the registration entity, attendance management entity along with this it is also connected to the target tracking entity and location tracking entity. The basic advantages of this app are that it digitalizes wholesaler and dealer interaction, the margin remains uncompromised, brand establishment is easier and prioritised, and there are no limitations or interference in interaction between the seller and buyer. In conclusion, the idea of Salescape emerges from the fact that a general application with specificity of domain has the capability of working better for wholesalers and retailers than common B2B applications that bring together every product irrespective of domain. This project looks into the various aspects of a business and the importance of digitizing it without just focusing on the profit maximization.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

1. INTRODUCTION

This part of the document specifies all about the project, its objectives, purpose and applicability.

1.1 BACKGROUND OF THE PROJECT

The main idea of this project emerges from the goal of minimizing the workload in the working of a transaction between a wholesaler and a retailer. It aims to form a digitalized and more efficient way of dealing with the transactions, where everything happens online and there is no need of faxing the order to the other party or making a request for order details since all the information will be listed on the cart of the retailer when he/she will select the items. From there everything will be tracked online and that is how this project will solve a few issues of wholesalers and retailers.

1.2 OBJECTIVES

- Digitalize the process between wholesaler and retailer.
- Help the retailer expand his local market.
- Order Management to make it easier for wholesalers and retailers.

1.3 PURPOSE, SCOPE AND APPLICABILITY

Purpose:

The idea of Salescape originates from the realization of the current state of e-commerce. The rapid digitalization is more from the perspective of getting the product to the end consumer and finding out direct and indirect benefits of the said entity, but the ecosystem of the market goes beyond the seller and the buyer, it consists of entities such as wholesalers, dealers, retailers and salesmen who get overshadowed by the major vision of e-commerce. Salescape focuses on bringing about a standard digital product that recognizes these entities while not diminishing their brand identity.

Scope:

Salescape being a common project that can be used by various organisations has certain specific assumptions that have been made for deciding the functionalities and modules. Firstly, it is

assumed that the organisation using the application is a wholesaler who is in lookout for orders from the retailer, the retailer perspective is used by a retailer who looks forward to selling their products either to the end customer or other retailers and is in no way the end customer themselves. The employee perspective assumes a structure of the company where attendance and sales target are running practices. The employees of the wholesaler using the admin perspective are expected to use the employee perspective. The employee app is exclusive for sales purposes and not for managerial or financial operations. The project's limitation can be stated as the fact that the final consumer or the manufacturer are not a part of the application's ecosystem and therefore the application only covers a part of the group. The second limitation is the fact that only selective employee roles are covered by the employee application. The major issue being addressed by our project is that the wholesalers and retailers are being left out of the process of digitization since they either have to function offline or their digitization comes at the cost of losing brand identity under giant companies providing services. The main function of our project is to help multiple retailers place wholesale orders online by judging location and wholesalers benefit by managing orders online while also using a digital solution of managing employees.

Applicability:

Our project can be used by wholesalers, retailers and salesmen. Wholesalers who have not yet become a part of the digital market can use our application for tracking and getting orders and becoming visible for the retailers around them. The retailers get the provision to compare based on location and differences of prices while also identifying who the wholesaler is. The employees can use the application to conveniently manage attendance and their personal sales target. This project contributes to the society by upholding the wholesalers and retailers who have not been in the loop with the rapid digitization by making the process convenient for them.

2. SYSTEM ANALYSIS AND REQUIREMENTS

This part of the document contains all the information related to the existing system, its problems, the proposed solution and its benefits.

2.1 EXISTING SYSTEM

One of the applications related to our project is the Amazon Business application.

Amazon Business-

Amazon Business helps small businesses to grow and start a new business for growing entrepreneurs. Amazon Business provides the following features-

- Purchasing System Integration
- Business-Only Pricing
- Bulk Discounts
- Comparison of Item Pricing
- Office Supplies

2.2 LIMITATIONS OF THE EXISTING SYSTEM

- Giving Up Margin: Amazon charges upto 6-15% of the product price as fees and additionally charges for fulfilled with amazon tag.
- Limited Communication With Buyers: Amazon treats each of its customers as its own and not of the seller, this way the businesses have absolutely no contact with the customers.
- Data and Administration: Listing products on Amazon Business requires a lot of information divided into categories and each category has its own rules which could be problematic while listing products. Also the product data is very important as it has the most impact on whether the customer will buy a product or not.

2.3 PROPOSED SYSTEM

This project is focused on reducing the gap between Wholesalers and the Retailers and also make their work easier.

This project is an application-based service, it consists of 3 applications :

1. Wholesaler

2.Delivery

3. Retailer

2.4 BENEFITS OF THE PROPOSED SYSTEM

- **Digitalizes wholesaler and dealer interaction:**

The vast growth of ecommerce is mostly thriving by digitizing the relation between the end customer and the seller, but our system focuses its scope on the relationship between the wholesaler and the dealer, while also making sure that the users do not face technical complexities.

- **Margin remains uncompromised:**

Since no third-party interference is possible in the interaction between the wholesaler and the dealer, margin for the seller is not compromised and neither is the buyer charged with an exorbitant amount.

- **Brand establishment is easier:**

The lack of third-party interference and the specificity of product domain makes it easier for the brand to form a personal relationship with the buyer and it is identified by the brand not as the most affordable product.

- **No limitation or interference in interaction:**

There are no rules, restrictions or guidelines for the seller to follow when it comes to the interaction with the buyer, this ensures that the seller is in full control of the application and therefore the flow of business, this reduces the dependency of the user on the application and makes sure the product and service are the criterions for judgement.

2.5 FEATURES OF THE PROPOSED SYSTEM

The various features and facilities provided by the proposed system are:

- Tracking the location of the wholesaler.
- Giving multiple options of wholesaler.
- Overview of wholesaler’s stock and products.
- Accepting or rejecting orders from retailers.
- Tracking ongoing orders of retailers.
- Unrestricted channel of communication between wholesaler and retailer.

- Focused on the brand of the wholesaler and not salescape itself.

2.6 SYSTEM REQUIREMENTS SPECIFICATION:

This part of the document describes the system requirements which includes user characteristics, software and hardware requirements, constraints, functional and non-functional requirements.

2.6.1 USER CHARACTERISTICS

- The **Wholesaler** lists products on the app for retailers to look at what kinds of products the wholesalers are selling.
- The **Delivery boy** main work is to keep track of the delivery and giving updates related to the delivery regularly and also, he/she must let the admin and the retailer know about the order placed and the status of the delivery.
- The **Retailer** will place an order through the app provided and will wait for the order approval and delivery status. If the Retailer wants to project any form of review on the order or wants to initiate a conversation with the Dealer they can do so since all the contact details of the dealer will be visible to them.

2.6.2 SOFTWARE AND HARDWARE REQUIREMENTS

- 1. Software Requirements:
  - Operating System: Android
  - Minimum API: 21
- 2.Hardware Requirements:
  - RAM capacity: Minimum 2GB
  - Graphics card: NA
  - Disk capacity: >60mb
  - Processor: Any

2.6.3 CONSTRAINTS

- Hardware Limitations: The main limitations of the project is that every user will have to use a smart-phone that will be able to run this application on their phone.
- Language Requirements for using this application will be English

- Reliability Requirements: The application must be able to function realy well and should have little to no bugs since it has to be made sturdy for the clients as a buggy app will lead to lesser use of the app by the clients.

2.6.4 FUNCTIONAL REQUIREMENTS

Table 2.1: Description of functional requirements of the project

Requirement ID	Requirement	Description
FR1	Stock Management	The wholesalers should be able to manage their stock and their respective prices that is to be displayed to the retailer. The option of updating stock by addition, deletion or modification of stock items or units of stock should be available.
FR2	Order Management	The wholesaler should have the choice of accepting and rejecting individual orders that are placed by the retailers.
FR3	Communication Channel	There should be a convenient communication channel that can be used by the wholesaler and retailer to negotiate or clarify the terms of any order or deal.
FR4	Location tracking (Wholesaler)	The facility of location tracking should be available to present the retailer with a list of wholesalers based on the convenience of location.
FR5	Location tracking (Delivery Agent)	Location tracking is also important to track the live location of delivery agents when they set out for delivery of an order. This functionality is to be enabled for both wholesalers and retailers.
FR6	Order Placement	The retailers should be able to choose from a list of wholesalers and place their order if their requirements are met by a wholesaler.
FR7	Order Tracking	The wholesaler as well as the retailers should be able to

		view the details of all the orders placed in the past, or currently are under processing or transit.
FR8	Search	The search functionality should be enabled so that the retailers can look for specific wholesalers. The search functionality is also required to fetch particular order details for both the wholesaler and retailer and fetch stock details for the wholesaler.

2.6.5 NON-FUNCTIONAL REQUIREMENTS

Table 2.2: Description of non-functional requirements of the project

Requirement ID	**Requirement**	**Description**
NF_R1	Performance	As it's a Android application, the network, hardware and other related infrastructure plays a vital role in determining the application performance. • Data compression approach has been applied to reduce the network burden. • Number of screen navigations and clicks are reduced to the minimum. • Basic and simple color and graphics settings are implemented to help the performance.
NF_R2	Safety/Security	Being a system to manage orders and transaction between unique pairs of wholesaler and retailer its primary character should be security, thus providing secure environment for the patient flow process. This system allows the user to only be accessed over the secure Cerner network and with his/her Cerner access credentials of the system.

NF_R3	Quality Requirements	• **Usability** As the users with different level of functionalities should require better understandability of the system. The interface for each type of user kept very simple and complete for ease of its user. Manuals, demos or the documents made available, simple, clear and definite set of interfaces makes the context easy to understand and use. • **Reliability** This system uses atomicity features where each task flow is performed to complete a process successful, failure of single task halts the transaction and the process fails. This avoids occurrence of incomplete order processing. Data backup ensures the availability of data to the system users all the time. • **Availability** This android application handles multiple service requests and the server is up all the time, which is made sure by the robust algorithms and server design architecture.

2.7 BLOCK DIAGRAM

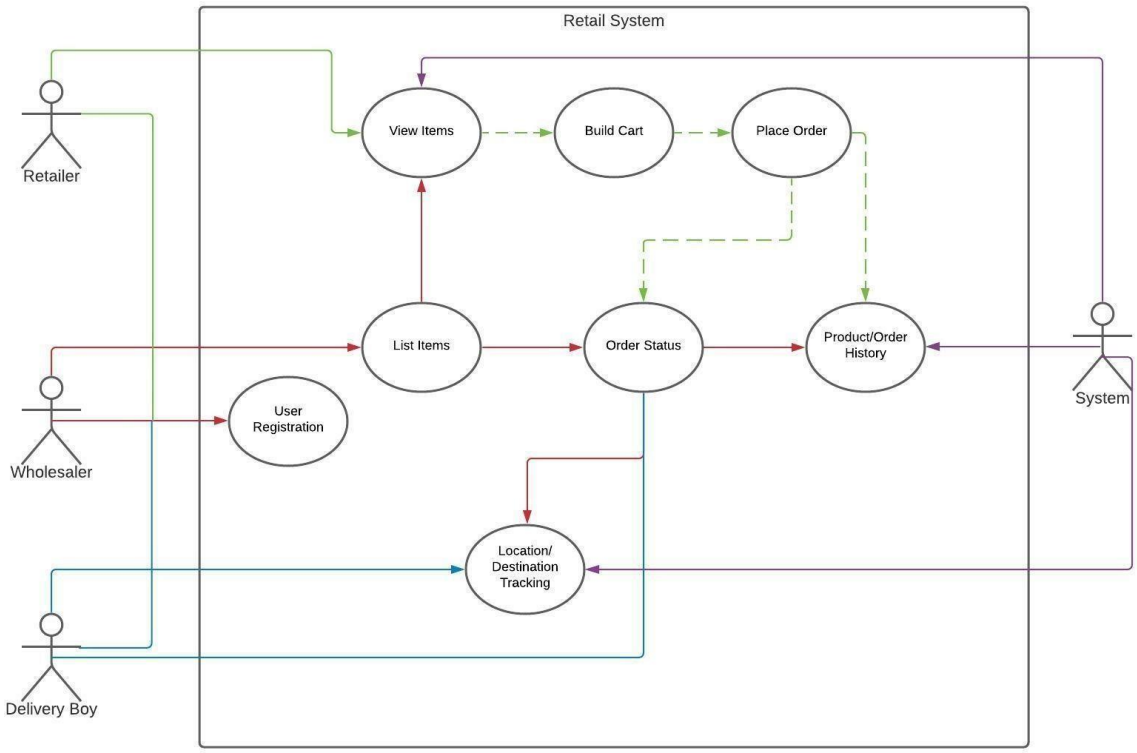

Fig 2.1: Use Case Diagram

- The **Wholesaler** part is connected to the entities such as registration entity which contains users' information, order tracking entity which contains the status of the delivery of orders for specific wholesalers, attendance management entity which contains employee attendance for specific wholesalers and location tracking entity for wholesalers to track salesmen for specific orders.
- The **Retailer** part is connected to certain entities such as the registration entity which contains users' registration information, product cart entity which contains the relationship between the products and the retailer that chose those products and order tracking entity which contains order details of a specific retailer.
- The **Delivery** part is connected to entities such as order entity which updates the order status, and location tracking entity which is used to track the order destination for various orders.

3. SYSTEM DESIGN

This part of the documentation contains all the information related to the modules, system design, flow of data, database design and ER diagram and the procedural design.

3.1 SYSTEM ARCHITECTURE

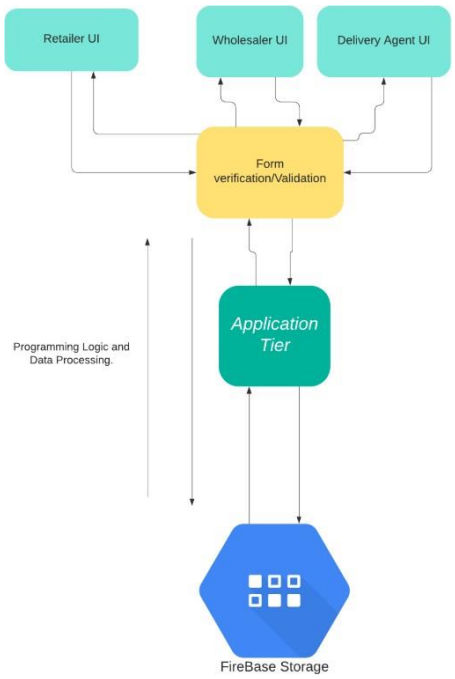

Fig 3.1: System Design

Salescape has a 3-tier architecture:

- **Presentation Tier:** The Presentation Tier consists of the presentation layer which is basically the UI part of the application which we have developed using Dart (programming language) and Material UI. The UI will interact with the backend using API calls. Salescape consists of 3 users that are Wholesaler, Retailer and Delivery Agent and they consist of a separate app for each type of user, this makes it easier on the user's perspective as he/she does not have to bother with the other unnecessary functionalities.

- **Application Tier:** The Application Tier consists of the programming logic that makes the application work properly with the backend and the front-end. Salescape uses Dart for the logic building and the retrieval and processing of the data from the front-end to the backend.
- **Data Tier:** The Data Tier consists of the database and the program for managing read/write/alter/deletion access to the database of the applications. The whole backend of Salescape is being managed by Firebase Database. It is a NoSQL database and helps in a lot of flexibility with the data processing.

3.2 MODULE DESIGN

Salescape aims to be a bridge between wholesalers and retailers and understanding the major steps in the relation between the two makes the recognition of modules convenient for us. The various important phases involved here are the presentation of stock, placement of order based on the stock and the delivery of the same. Keeping these in mind the following modules can be identified:

- **User Registration:** The user registration module is the first step of the process that allows the identification of the three types of end users who are expected to use the application: Wholesaler, Retailer and Delivery partner. The registration module encompasses the registration screen that collects details from the user and stores it in the respective database depending on their role. This module enables all registered individuals to have unique login credentials that are to be used thereafter and will allow the users to access their required functionalities.
- **Profile Management:** Profile Management is an important module that allows the different wholesalers and retailers to provide the necessary information required to go through with the deals and transactions. It allows the wholesaler to manage and present their stock of various products for the retailer to choose from. Allowing customization of profile emphasizes on our objective of establishing the wholesaler's brand. The profile and location are the very factors based on which the retailers decide if an order is to be placed.
- **Order Placement:** The retailers have the option of adding various products to the cart and purchase the needful, based on wholesale rates. The stocks of various products are available for the retailer to select and order from. This module works like a normal online purchase platform where the user, in this case the retailer has the choice of selecting. Adding a step to this,

the wholesaler has the privilege of accepting or rejecting the order placed since this application does not aim to diminish the healthy negotiation that works between traders while settling deals. The additional functionality of a communication through an unrestricted channel will be made available to promote the previously said objective.

- **Order Tracking:** This module ensures that the two parties involved in the deal are always provided with the status of a placed order. The status of an order is displayed as pending or approved depending on the wholesaler's discretion. Once approved the details of the order that has been put to motion for delivery is made available for the retailer. The state of the order is displayed depending on the dispatch, packing, delivery etc so that the retailers can rightly estimate the arrival of their order and the wholesaler can confirm it's delivery once sent out.
- **Catalogue System of selection:** The retailer has the option of selecting the wholesaler based on the convenience of location and this is ensured by the functionality of location tracking system. This location system helps the retailer to filter out the wholesalers that provide the best rates and are in a geographically convenient place. The catalogue system not only helps the retailer to select from a list of wholesalers but also the product catalogue under every wholesaler is also provided to understand the prices and order in the required capacity and to look into the other items for wholesale under the same domain.
- **Delivery System:** The delivery system is a part of the project where we coordinate with delivery agents based on their state. The location tracking makes it possible to assign the nearest delivery agent for the pickup of a particular event. The location of these agents are provided to both the wholesaler and the retailer to oversee the entire delivery process. The application provides the needful information of that particular order to the delivery agent. The module of the delivery system connects the two major entities using the application.

3.3 DATA FLOW DIAGRAM

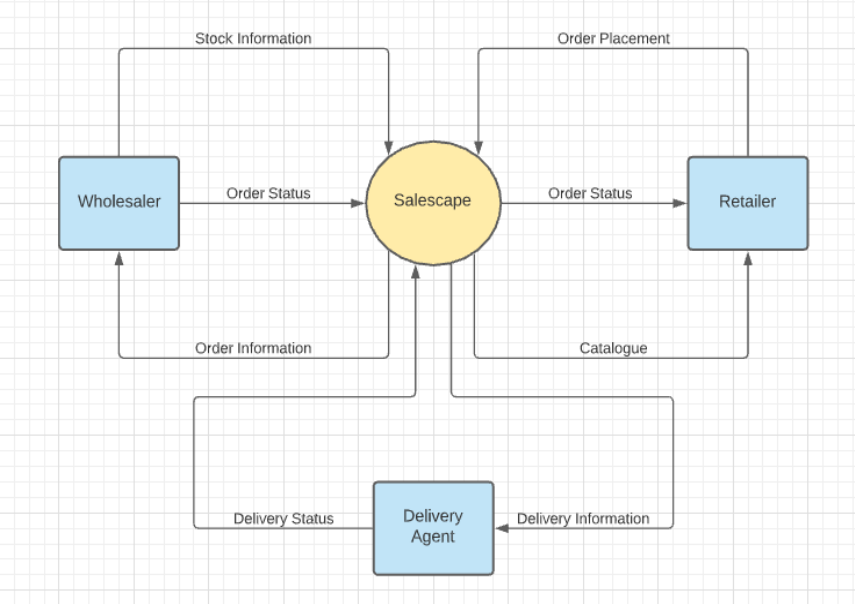

Fig 3.2:Data Flow Diagram (Level 0)

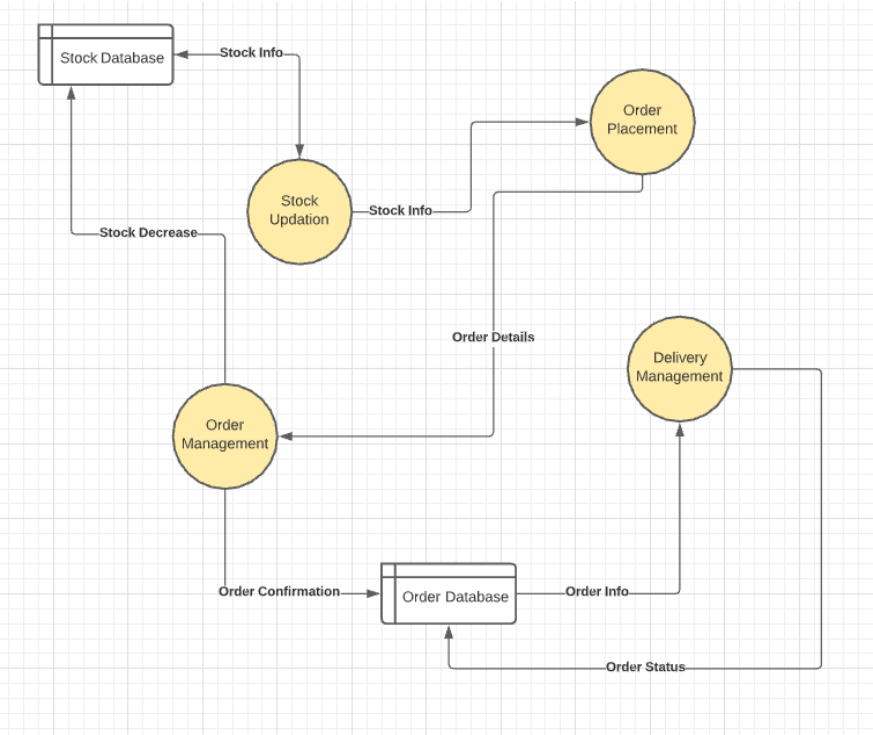

Fig 3.3:Data Flow Diagram (Level 1)

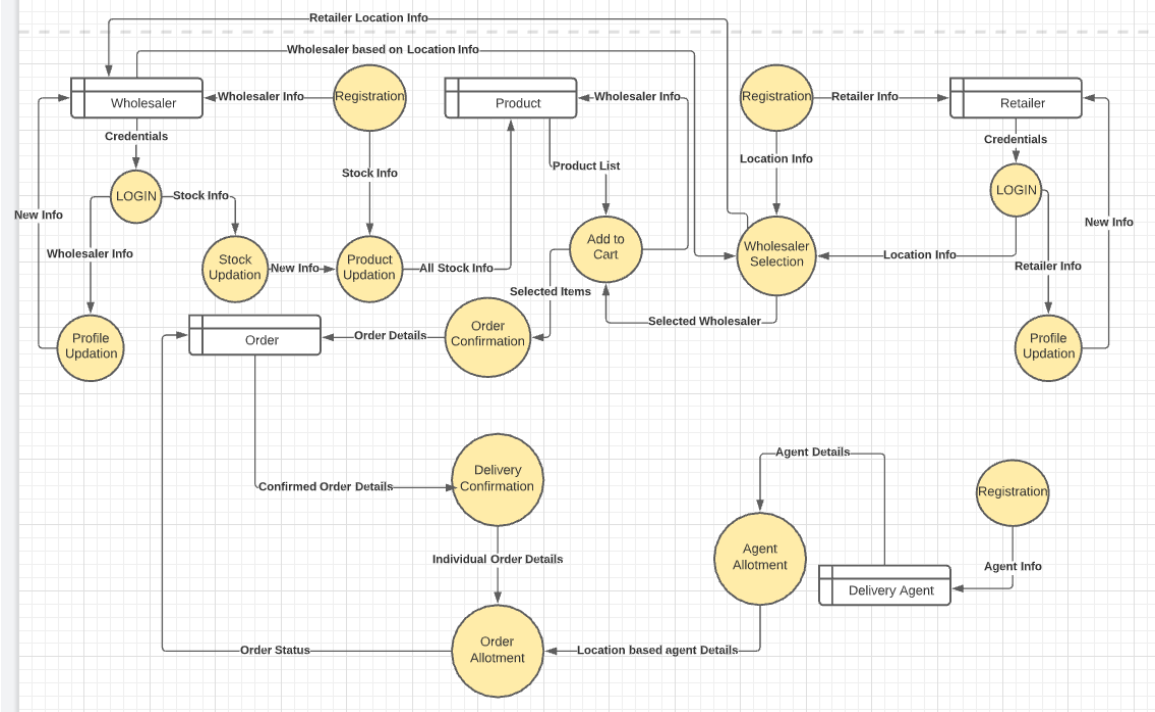

Fig 3.4:Data Flow Diagram (Level 2)

3.4 ER DIAGRAM

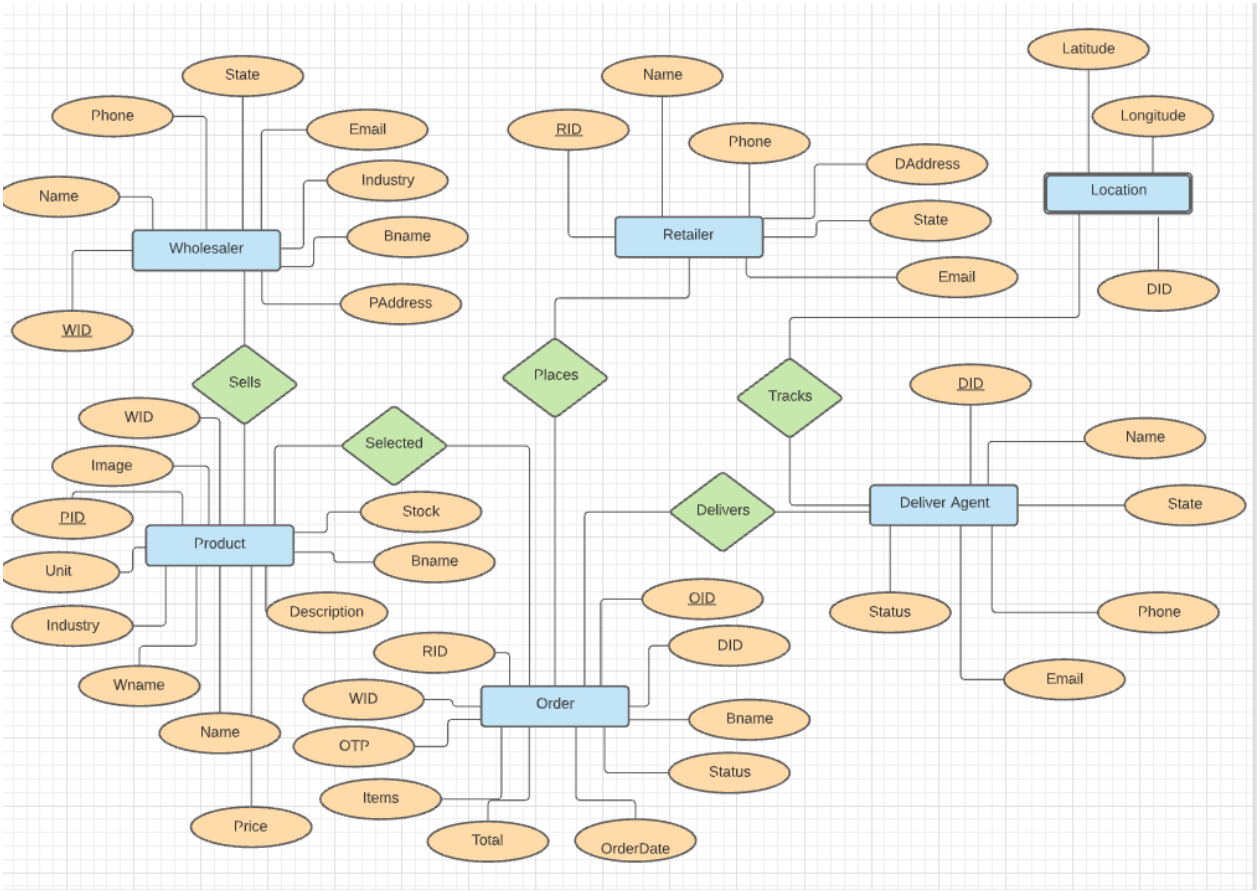

Fig 3.5:Entity Relationship Diagram

3.5 DATABASE DESIGN

The database is designed according to the requirements of all the 3 apps- for wholesaler, retailer and delivery boy. Since there is a separate app for different users, the database contains separate entities for each of the users. These user entities have all the information related to the specific user.

These entities are not connected to each other directly, instead these are connected using the order entity. The order entity contains all the details about the orders such as products, status, etc. The product entity contains the list of products with respect to its wholesalers.

The location entity contains information about the accurate location for the delivery boy.

3.5.1 Table Design:

Table 3.1: Wholesaler Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	WID	String	Wholesaler Id	Primary Key
2	Name	String	Name of Wholesaler	Not Null
3	Phone	Integer	Phone No of Wholesaler	Not Null, Unique
4	State	String	State of Wholesaler	Not Null
5	Email	String	Email of Wholesaler	Not Null, Unique, Check(validity)
6	Industry	String	Industry Domain	Not Null
7	Bname	String	Business Name	Not Null
8	PAddress	String	Pickup Address	Not Null

Table 3.2: Retailer Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	RID	String	Retailer Id	Primary Key
2	Name	String	Name of Retailer	Not Null
3	Phone	Integer	Phone No of Retailer	Not Null, Unique
4	State	String	State of Retailer	Not Null
5	Email	String	Email of Retailer	Not Null, Unique, Check(validity)
6	DAddress	String	Delivery Address	Not Null

Table 3.3: Delivery Agent Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	DID	String	Delivery Boy Id	Primary Key
2	Name	String	Name of Delivery Boy	Not Null
3	Phone	Integer	Phone No of Delivery Boy	Not Null, Unique
4	State	String	State of Delivery Boy	Not Null
5	Email	String	Email of Delivery Boy	Not Null, Unique, Check(validity)
6	Status	String	Delivery Boy Status	Not Null, Check (value in (available, not available))

Table 3.4: Order Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	OID	String	Order Id	Primary Key
2	RID	String	Retailer Id	Foreign Key
3	WID	String	Wholesaler Id	Foreign Key
4	DID	String	Delivery Boy Id	Foreign Key
5	Status	String	Status of Order	Not Null, Check(value in (pending, approved, rejected))
6	Order Date	DateTime	Date of Placing Order	Not Null
7	Total	Integer	Total Price of Items	Not Null
8	Items	List	List of Items/Product associated with the order	Not Null

9	Bname	String	Business Name	Not Null
10	OTP	Integer	One Time Password	Not Null

Table 3.5: Product Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	PID	String	Product Id	Primary Key
2	WID	String	Wholesaler Id	Foreign Key
3	Name	String	Name of Product	Not Null
4	Price	Integer	Price of Product	Not Null
5	Stock	Integer	Stock/Inventory of Product	Not Null
6	Bname	String	Business Name	Not Null
7	Description	String	Description of product	Not Null
8	ImageURL	String	URL for image of product	Not Null
9	Unit	String	Unit of product	Not Null
10	Industry	String	Industry	Not Null
11	Wname	String	Wholesaler name	Not Null

Table 3.6: Location Table Design

S.No	Attribute Name	Data Type	Description	Constraint
1	Latitude	String	Latitude of Delivery Boy	Not Null, Check(value >= -90 and <=90)
2	Longitude	String	Longitude of Delivery Boy	Not Null, Check(value >= -180 and <=180)
3	DID	String	Deliver Boy Id	Foreign Key

3.5.2 Data Integrity and Constraints:

Primary Key:

The primary key is the unique identifier attribute in a table. The primary key for the tables is defined as-

Wholesaler- WID (Wholesaler Id)

Retailer- RID (Retailer Id)

Delivery_Boy= DID (Delivery Boy Id)

Order- OID (Order Id)

Product- PID (Product Id)

Foreign Key:

The foreign key is the attribute that acts as a link between 2 tables. The foreign keys the tables are defined as-

Order- WID (Wholesaler Id), RID (Retailer Id), DID (Delivery Boy Id)

Product- WID (Wholesaler Id)

Unique:

The unique constraint is used where duplications are not allowed. The following attributes have been given the unique constraint- phone and email.

Not Null:

The not null constraint is used for all the attributes which are not a primary or foreign key, this ensures that there are no null values anywhere in the database.

Validity Check for Email:

A validity check is given for email which will determine whether the email given is correct or not using regular expressions.

Check condition for password:

A validity check is given for password which involves the following conditions- the length of the password should be greater than 6 characters, it should contain at least 1 upper case letter, at least 1 lower case letter and at least 1 symbol.

Check condition for delivery boy status:

The condition for delivery boy status makes sure that the value is either available or not available.

Check condition for order status:

The condition for order status makes sure that the value is either pending, approved or rejected.

Check condition for value of latitude:

The condition for latitude status makes sure that the value lies between -90 and 90.

Check condition for value of longitude:

The condition for longitude status makes sure that the value lies between -180 and 180.

3.5.3 Data Dictionary

Table 3.7: Data Dictionary

S.No	Attribute Name	Min Value	Max Value	Default Value
1	Phone	-	-	-
2	Total	**0**	-	**0**
3	Price	**0**	-	**0**
4	Stock	**0**	-	0

3.6 INTERFACE AND PROCEDURAL DESIGN

3.6.1 USER INTERFACE DESIGN:

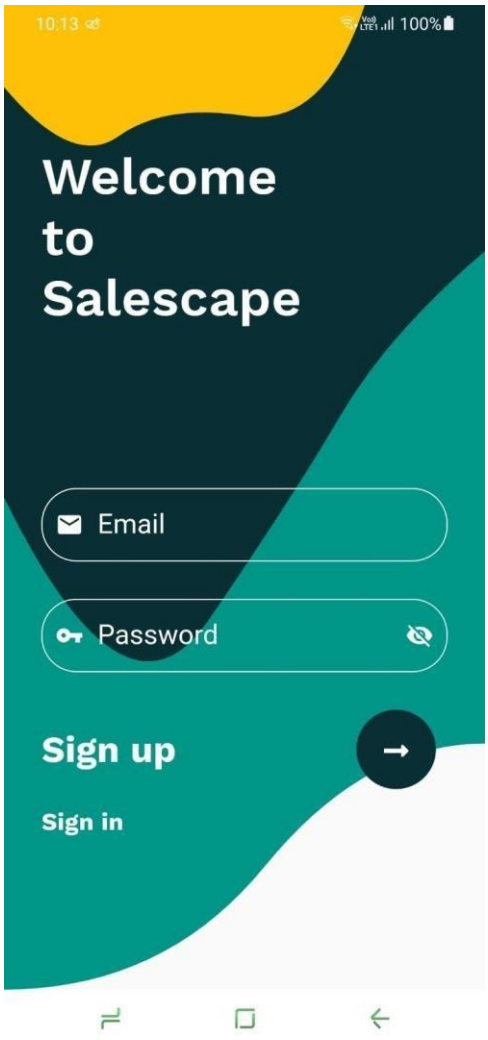

Fig 3.6: Sign up Interface

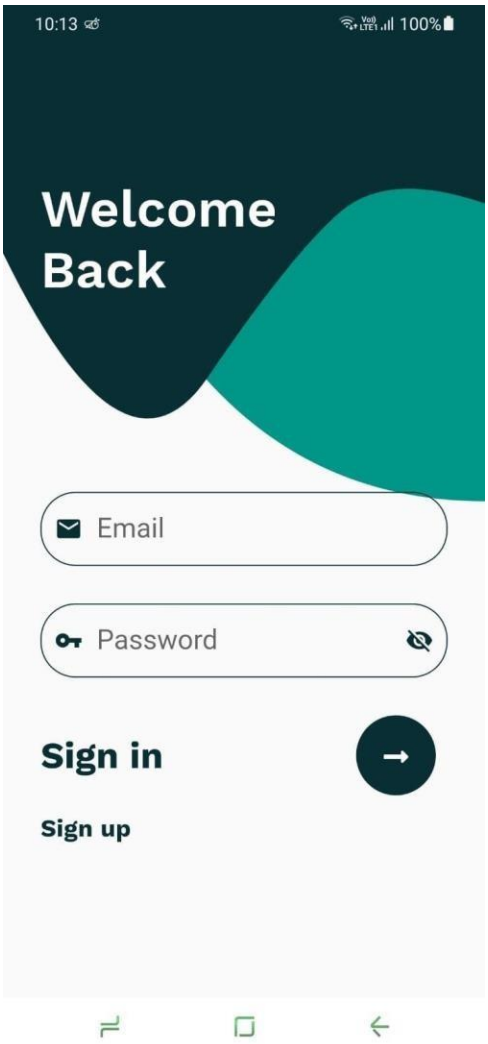

Fig 3.7: Sign in Interface

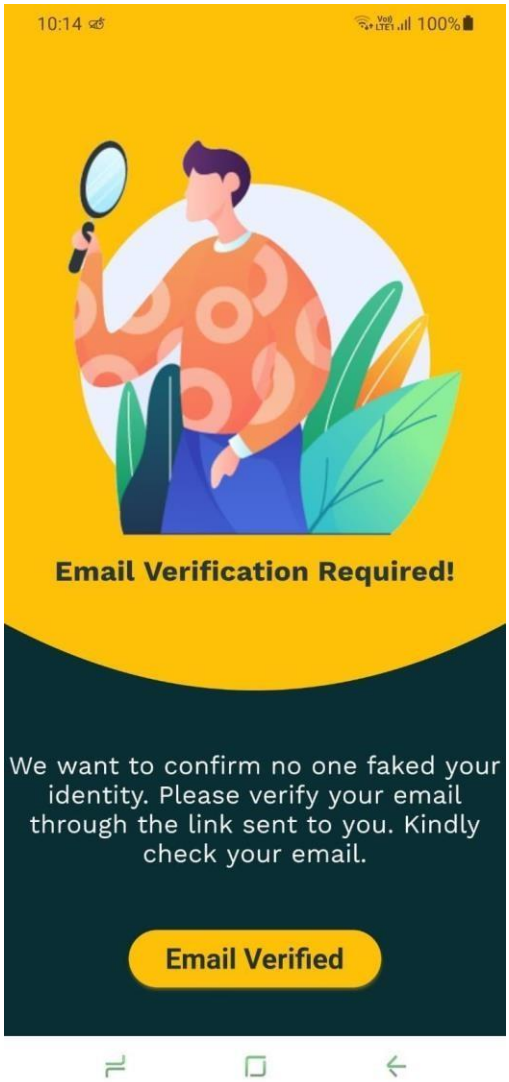

Fig 3.8:Email Verification Interface

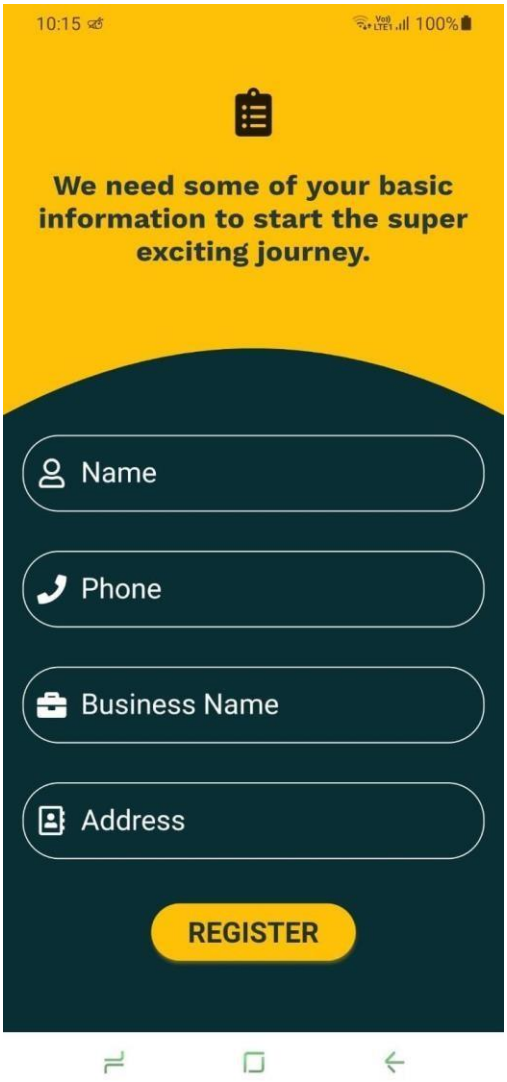

Fig 3.9: Registration Interface

3.6.2 PROCEDURAL DESIGN:

- **Sign Up:**
  - User registers with email and password needed for Firebase Authentication and will generate a user id.
  - Then the user will receive a verification mail in the registered email.
  - After the user verifies, he/she has to provide registration details such as name, phone, state etc.
  - Then the provided user data will be stored in Cloud Firestore under User document with document id same as user id.
- **Placing Order:**
  - Retailers can add products of a wholesaler with the required quantity in their cart.
  - They can confirm to place an order in the cart screen.
  - After confirmation the order will be converted to json format and will be sent to Cloud Firestore under the Order document.

3.7 REPORTS DESIGN:

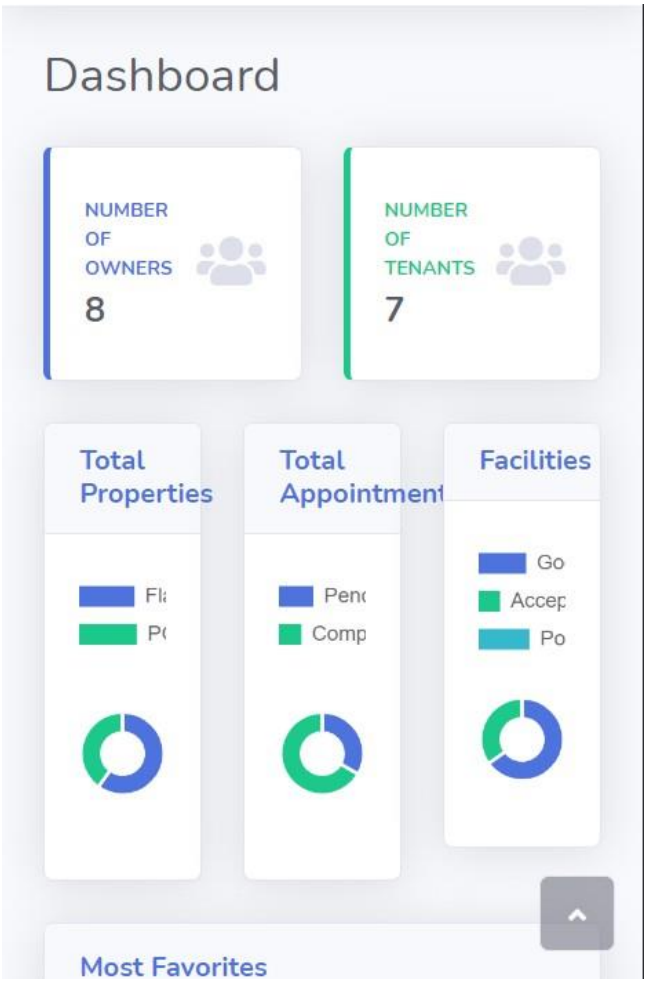

Fig 3.10: Report Design

1. **Number of Users:**

- **Input-**
  - This information will be taken from the Wholesaler and retailer table.
- **Output-**
  - Total users in retailers and wholesalers.

2. **Sales Report (Wholesaler):**

- **Product Sales Report:**
  - **Input-**
    - The report will be processed by taking the data from the order table and the reference of the product from the products table and its unique ID to determine the number of sales made.
  - **Output-**
    - This report will consist of the number of sales of each product in between a given period of time. This report will also give the data of the most sold product.
- **Top Retailer:**
  - **Input-**
    - The report will be processed by taking the data from the item, price, RID fields of the order table.
  - **Output-**
    - This report will consist of the data of the top retailers according to the number of purchases they have made.

3. **Industry Report:**

- **Industry Demand Report:**
  - **Input-**
    - This report will be derived from the order table as it contains the data of all the sales transactions that happen and by the help of the PID we can derive the industry trend.
  - **Output-**

- This report will summarize the trends of each industry and will show which industry is most in demand.

4. Total Orders:

- **Input-**
  - This report will be derived from the records of the Order table.
- **Output-**
  - This report will consist of the total number of orders.

4. IMPLEMENTATION

The various modules specified and elucidated in the design document are coded in the implementation phase. This chapter explains the coding standards followed and the important code that was put together to create the project.

4.1 CODING STANDARDS

Good software development practices include the maintenance of certain coding standards. The coding standards are important so as to maintain a uniformity in the code written by different people working on the same project. It increases the readability of the code and makes it more maintainable. Following coding standards also makes error detection convenient. There were numerous coding standards that were followed in this project. The following are the significant ones:-

- **Limited use of globals:** Limited data was declared as global.
- **Naming conventions for local variables, global variables, constants and functions:**

Naming conventions are an important determinant of code readability and therefore there were certain naming conventions that were followed. Some of the naming conventions are given below:

  - Local have been named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names start with a capital letter (e.g. **GlobalData**). Constant names have been formed using capital letters only (e.g. **CONSDATA**).
  - Digits have been avoided in variable names.
  - The names of the function have been written in camel case starting with small letters.
  - The name of the function describes the reason for using the function clearly and briefly.

- **Indentation:**

Proper indentation has been maintained to increase the readability of the code. Some of the spacing conventions are given below:

  - A space has been given after giving a comma between two function arguments.

- Each nested block has been properly indented and spaced.
- Proper Indentation has been maintained at the beginning and at the end of each block in the program.
- All braces start from a new line and the code following the end of braces also starts from a new line.

- **Error return values and exception handling conventions:**

All functions that encounter an error condition either return a 0 or 1 for simplifying the debugging.

- **Complex code has been avoided:**

Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive, which is an important detail kept in mind while coding.

- **Using an identifier for multiple purposes has been avoided:**

Each variable has been given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.

- **Function length has been kept moderate:**

Lengthy functions are very difficult to understand. That's why functions have been kept small enough to carry out small work and lengthy functions have been broken into small ones for completing small tasks.

- **GOTO statement has been avoided:**

The GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

4.2 CODING DETAILS

The following are the important snippets of code that the applications constitute of. The important models in the database:

```
//The Product Model
factory Product.fromJson(Map<String, Object?> json) {
    return Product(
      pid: json['pid'] as String,
      wid: json['wid'] as String,
      wname: json['wname'] as String,
      bname: json['bname'] as String,
      industry: json['industry'] as String,
      name: json['name'] as String,
      description: json['description'] as String,
      price: double.parse(json['price'].toString()),
      stock: json['stock'] as int,
      unit: json['unit'] as String,
      imageUrl: json['imageUrl'] as String,
   );
  }

//The Wholesaler Model
  factory Wholesaler.fromJson(Map<String, Object?> json) {
    return Wholesaler(
      wid: json['wid'] as String,
      name: json['name'] as String,
      bname: json['bname'] as String,
```

```
      phone: json['phone'] as String,
      state: json['state'] as String,
      email: json['email'] as String,
      industry: json['industry'] as String,
      pickup_address: json['PAddress'] as String,
    );
  }
//The Order Model
  factory Order.fromJson(Map<String, Object?> json) {
    return Order(
      rid: json['rid'] as String,
      oid: json['oid'] as String,
      wid: json['wid'] as String,
      did: json['did'] == null ? null : json['did'] as String,
      items: (jsonDecode(json['items'] as String) as List)
          .map((i) => CartItem.fromJson(i))
          .toList(),
      total: json['total'] as double,
      status: OrderStatus.values.elementAt(json['status'] as int),
      dateTime: (json['date'] as Timestamp).toDate(),
otp: (json['otp'] as String),
    );
  }
```

User Services performed by making changes in the database

```
class FirestoreService {
  final _firestore = FirebaseFirestore.instance;
```

```
// User Services

Future<void> addUser(Wholesaler userData) async {
  var _userRef =
      _firestore.collection('wholesalers').withConverter<Wholesaler>(
            fromFirestore: (snapshots, _) =>
                Wholesaler.fromJson(snapshots.data()!),
            toFirestore: (user, _) => user.toJson(),
          );
  await _userRef.doc(userData.wid).set(userData);
}

Future<Wholesaler?> getUser(String wid) async {
  var _userRef =
      _firestore.collection('wholesalers').withConverter<Wholesaler>(
            fromFirestore: (snapshots, _) =>
                Wholesaler.fromJson(snapshots.data()!),
            toFirestore: (user, _) => user.toJson(),
          );
  return (await _userRef.doc(wid).get()).data();
}

Future<void> updateUser(Wholesaler userData) async {
  var _userRef =
      _firestore.collection('wholesalers').withConverter<Wholesaler>(
            fromFirestore: (snapshots, _) =>
                Wholesaler.fromJson(snapshots.data()!),
            toFirestore: (user, _) => user.toJson(),
          );
  await _userRef.doc(userData.wid).set(userData);
```

```
}

Future<void> updateDeliveryStatus(Delivery userData) async {
  var _userRef = _firestore
      .collection('deliveryboys')
      .withConverter<Delivery>(
        fromFirestore: (snapshots, _) => Delivery.fromJson(snapshots.data()!),
        toFirestore: (user, _) => user.toJson(),
      );
  await _userRef.doc(userData.did).set(userData);
}

Future<Retailer> getRetailer(String rid) async {
  var _userRef = _firestore.collection('retailers').withConverter<Retailer>(
        fromFirestore: (snapshots, _) => Retailer.fromJson(snapshots.data()!),
        toFirestore: (user, _) => user.toJson(),
      );
  return (await _userRef.doc(rid).get()).data()!;
}

Future<Delivery?> getDelivery(String did) async {
  var _userRef = _firestore
      .collection('deliveryboys')
      .withConverter<Delivery>(
        fromFirestore: (snapshots, _) => Delivery.fromJson(snapshots.data()!),
        toFirestore: (user, _) => user.toJson(),
      );
  return (await _userRef.doc(did).get()).data();
}
```

```
Future<List<Delivery>> getAllFreeDelivery() async {
  var _userRef = _firestore
      .collection('deliveryboys')
      .withConverter<Delivery>(
        fromFirestore: (snapshots, _) => Delivery.fromJson(snapshots.data()!),
        toFirestore: (user, _) => user.toJson(),
      );
  List<Delivery> deliveryList = [];
  List<QueryDocumentSnapshot<Delivery>> delivery =
      await _userRef.where('status', isEqualTo: 0).get().then((d) => d.docs);
  for (var del in delivery) {
    deliveryList.add(del.data());
  }
  return deliveryList;
}

Stream<DeliveryLocation> getDeliveryLocation(String did) {
  return _firestore
      .collection('locations')
      .doc(did)
      .snapshots()
      .map((snapshot) => DeliveryLocation.fromMap(snapshot.data()!));
}

//Product Services

Future<void> addProduct(Product product) async {
  var _productRef = _firestore.collection('products').withConverter<Product>(
        fromFirestore: (snapshots, _) => Product.fromJson(snapshots.data()!),
        toFirestore: (product, _) => product.toJson(),
```

```
      );
  await _productRef.doc(product.pid).set(product);
}

Future<Product?> getProduct(String pid) async {
  var _productRef = _firestore.collection('products').withConverter<Product>(
        fromFirestore: (snapshots, _) => Product.fromJson(snapshots.data()!),
        toFirestore: (product, _) => product.toJson(),
      );
  return (await _productRef.doc(pid).get()).data();
}

Future<List<Product>> getAllProducts(String wid) async {
  var _productRef = _firestore.collection('products').withConverter<Product>(
        fromFirestore: (snapshots, _) => Product.fromJson(snapshots.data()!),
        toFirestore: (product, _) => product.toJson(),
      );
  List<Product> productList = [];
  List<QueryDocumentSnapshot<Product>> products = await _productRef
      .where('wid', isEqualTo: wid)
      .get()
      .then((products) => products.docs);
  for (var product in products) {
  productList.add(product.data());
  }
  return productList;
}

Future<void> deleteProduct(String pid) async {
  var _productRef = _firestore.collection('products').withConverter<Product>(
```

```
        fromFirestore: (snapshots, _) => Product.fromJson(snapshots.data()!),
        toFirestore: (product, _) => product.toJson(),
      );
  return _productRef
      .doc(pid)
      .delete()
      .then((value) => print("Product Deleted"))
      .catchError((error) => print("Failed to product user: $error"));
}

// Order Services

Future<List<Order>> getAllUserOrders(String wid) async {
  var _orderRef = _firestore.collection('orders').withConverter<Order>(
        fromFirestore: (snapshots, _) => Order.fromJson(snapshots.data()!),
        toFirestore: (order, _) => order.toJson(),
      );
  List<Order> orderList = [];
  List<QueryDocumentSnapshot<Order>> orders = await _orderRef
      .where('wid', isEqualTo: wid)
      .get()
      .then((orders) => orders.docs);
  for (var order in orders) {
  orderList.add(order.data());
  }
  return orderList;
}

Future<List<Order>> getPendingUserOrders(String wid) async {
  List<Order> orderList = await getAllUserOrders(wid);
```

```
  List<Order> pendingList = [];
  for (var order in orderList) {
    if (order.status == OrderStatus.pending) {
      pendingList.add(order);
    }
  }
  return pendingList;
}

Future<List<Order>> getAcceptedUserOrders(String wid) async {
  List<Order> orderList = await getAllUserOrders(wid);
  List<Order> acceptedList = [];
  for (var order in orderList) {
    if (order.status == OrderStatus.accepted ||
      order.status == OrderStatus.start) {
      acceptedList.add(order);
    }
  }
  return acceptedList;
}

Future<List<Order>> getCompletedUserOrders(String wid) async {
  List<Order> orderList = await getAllUserOrders(wid);
  List<Order> completedList = [];
  for (var order in orderList) {
    if (order.status == OrderStatus.completed) {
      completedList.add(order);
    }
  }
  return completedList;
```

```
}

Future<void> updateOrder(Order order) async {
  var _orderRef = _firestore.collection('orders').withConverter<Order>(
        fromFirestore: (snapshots, _) => Order.fromJson(snapshots.data()!),
        toFirestore: (order, _) => order.toJson(),
      );
  await _orderRef.doc(order.oid).set(order);
}

Future<void> deleteOrder(String oid) async {
  var _orderRef = _firestore.collection('orders').withConverter<Order>(
        fromFirestore: (snapshots, _) => Order.fromJson(snapshots.data()!),
        toFirestore: (order, _) => order.toJson(),
      );
  return _orderRef
      .doc(oid)
      .delete()
      .then((value) => print("Order Deleted"))
      .catchError((error) => print("Failed to delete order: $error"));
}

//Report Service
Future<Report> getReport(String wid) async {
  int total;
  List<String> pidList = [];
  List<Order> pendingList = [];
  List<Order> acceptedList = [];
  List<Order> completedList = [];
  List<ProductReport> prodrep = [];
```

```
  var value = await getAllUserOrders(wid);
  total = value.length;
  for (var order in value) {
    if (order.status == OrderStatus.pending) {
      pendingList.add(order);
    } else if (order.status == OrderStatus.accepted ||
      order.status == OrderStatus.start) {
      acceptedList.add(order);
    } else {
      completedList.add(order);
    }
    for (var item in order.items) {
      if (!pidList.contains(item.pid)) {
        pidList.add(item.pid);
        var prod = await getProduct(item.pid);
        prodrep.add(ProductReport(pid: item.pid, name: prod!.name, qty: 1));
      } else {
        var i = prodrep.lastIndexWhere((e) => e.pid == item.pid);
        prodrep[i].qty = prodrep[i].qty + 1;
      }
    }
  }
  return Report(
      torder: total.toString(),
      pending: pendingList.length,
      accepted: acceptedList.length,
      completed: completedList.length,
      prodrep: prodrep);
}
```

```
}
```

Email Authentication with Firebase

```
class AuthProvider with ChangeNotifier {
bool _isLoading = false;
  bool get isLoading => _isLoading;

  User? _user;
  User? get getUser => _user;

  FirebaseAuth firebaseAuth = FirebaseAuth.instance;

  Future<String?> register(String email, String password) async {
    setLoading(true);
    try {
      UserCredential authResult = await firebaseAuth
          .createUserWithEmailAndPassword(email: email, password: password);
      _user = authResult.user;
      setLoading(false);
      notifyListeners();
      return _user != null ? null : 'An error Occurred';
    } on SocketException {
      setLoading(false);
      notifyListeners();
      return "No internet, please connect to internet";
    } on FirebaseAuthException catch (e) {
      setLoading(false);
      notifyListeners();
      return e.message;
    }
```

```
}

Future<String?> login(String email, String password) async {
  setLoading(true);
  try {
    UserCredential authResult = await firebaseAuth.signInWithEmailAndPassword(
        email: email, password: password);
    _user = authResult.user;
    setLoading(false);
    notifyListeners();
    return _user != null ? null : 'An error Occurred';
  } on SocketException {
    setLoading(false);
    notifyListeners();
    return "No internet, please connect to internet";
  } on FirebaseAuthException catch (e) {
    setLoading(false);
    notifyListeners();
    return e.message;
  }
}

Future logout() async {
  _user = null;
  await firebaseAuth.signOut();
}

Future<bool> isVerified() async {
  await firebaseAuth.currentUser!.reload();
  return firebaseAuth.currentUser!.emailVerified;
```

```
  }

  void setLoading(val) {
    _isLoading = val;
    notifyListeners();
  }

  Stream<User?> get user =>
      firebaseAuth.authStateChanges().map((event) => event);
}
```

4.3 SCREENSHOTS

Following are the screenshots of the various screens that receive input from and display results of various functionalities.

Fig 4.1 Retailer app homepage

Fig 4.2 Retailer app search page

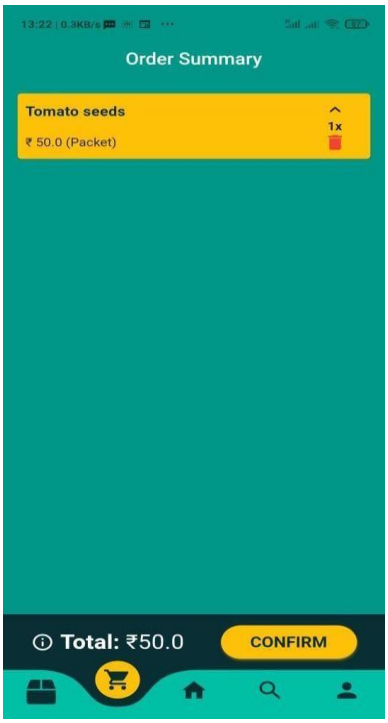

Fig 4.3 Retailer Cart Screen

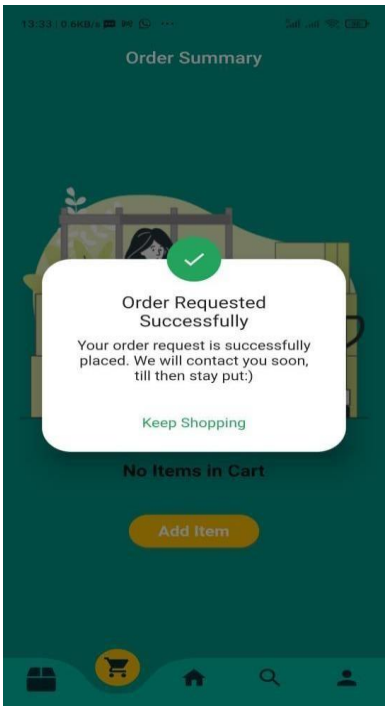

Fig 4.4 Retailer Order confirmation message

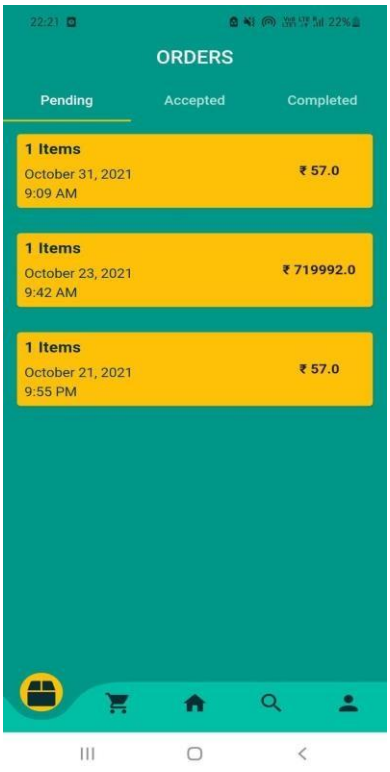

Fig 4.5 Order history screen

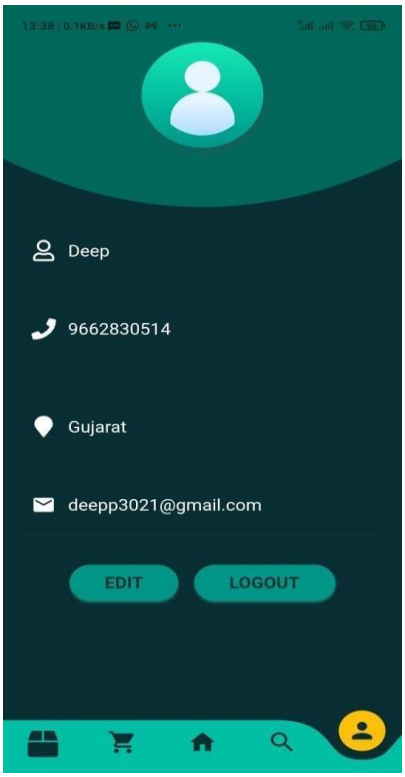

Fig 4.6 Retailer Profile Screen

Fig 4.7 Wholesaler Home Page

Fig 4.8 Wholesaler Add product screen

Fig 4.9 Wholesaler Product Screen

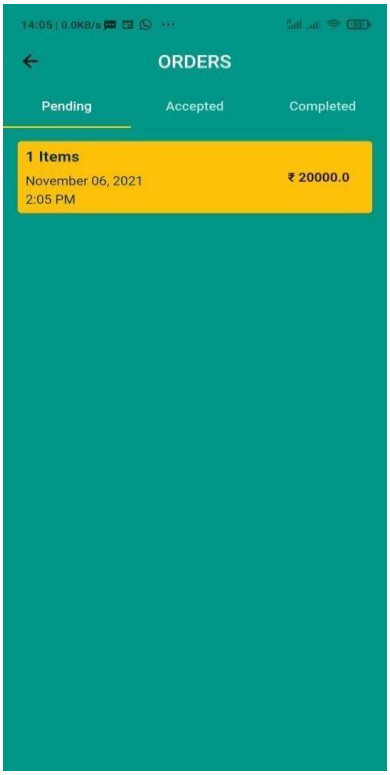

Fig 4.10 Wholesaler Orders Screen

5. TESTING

After the implementation of the software has been completed the Testing phase begins, in this phase the software is tested to see if it functions the ways it is designed and expected to function, test cases are set at various points to check if it is functioning correctly.

5.1 TESTING APPROACHES

The testing of Salescape has been performed with great focus on the modules. When testing the software the test cases were designed in a way to understand what are the various problems that a user might face while using the applications and in what way the user would expect the app to react. Both unit testing and integration testing were performed on the application:

- **Unit Testing:** The main focus of unit testing is to see if the smallest functions of the application are responding the way it is supposed to. For the unit testing of our project it has been made sure that the test cases cover all the expected validations and the various functions that build the module.
- **Integration Testing:** The integration testing has been performed after unit testing, once it is ensured that all the functions are functioning correctly, it is then tested that whether the modules work the way they are designed to and do the modules come together to work as a project.

5.2 TEST CASES

Table 5.1: Test Case Description

Sl. No.	Module Name	Test Case No	Test Case Description	Expected Result
1	User Registration	TC_1	The details entered by the user such as Name, Phone Number, Email ID etc should be validated to accept valid credentials only.	Error message is to be displayed on the input of invalid data along with the reason for data invalidity.

2	User Registration	TC_2	Sign in or Sign up with incomplete credentials should not be allowed.	Error message asking the user to fill an empty field should be displayed.
3	User Registration	TC_3	Signing up of users should be recorded in the database.	On successful sign up the database with user details is supposed to reflect an additional record.
4	User Registration	TC_4	User's email id should be authenticated.	Authentication mail is supposed to be sent to the user's email id and opening the application without authentication should not be allowed.
5	User Registration	TC_5	One email can't be used for signing up more than once.	Error message displaying the requirement of an unique email id is to be displayed.
6	Profile Management	TC_6	User details should be editable.	The option of editing user details should be available and the changes should be reflected in the database as well the app itself.
7	Catalogue System	TC_7	The details entered by the user about the product such as Name, Price etc should be validated to accept valid credentials only.	Error message is to be displayed on the input of invalid data along with the reason for data invalidity.
8	Catalogue System	TC_8	The price of a product entered by a user can't be less than or equal to 0.	Error message is to be displayed stating that the product price has to be higher than 0.
9	Catalogue System	TC_9	Addition of a product should be displayed in the list of added products for the wholesaler.	Product details are to be added with the preexisting products for the wholesaler.

10	Catalogue System	TC_10	Products added by a wholesaler should be visible to retailers.	Product details of newly added products should be displayed in the preexisting list of products according to industry.
11	Catalogue System	TC_11	Deleting and editing of products should be an available option for the wholesaler.	The wholesaler should be able to edit and delete products and the changes should be reflected in the database as well as retailer and wholesaler app.
12	Order Placement	TC_12	Products from only one wholesaler can be added at a time.	Message saying that the product is not from the same wholesaler is to be displayed.
13	Order Placement	TC_13	One product can't be added to the cart twice.	The order should not be displayed as added twice instead the quantity of the first product is supposed to be increased.
14	Order Placement	TC_14	Order can't be placed for a quantity greater than that available in the stock.	Error message saying product out of stock should be displayed and the quantity of order freezes at maximum available stock.

5.3 TEST REPORTS

Explain the test results and reports based on your test cases, which would prove that your project will produce the expected results under different conditions. Produce the test reports as a screen shot for different test cases by giving different sample inputs and showing the corresponding outputs.

Table 5.2: Test Report Index

Sl. No.	Test Case No	Test Status	Test Report
1	TC_1	Successful	Fig 5.1
2	TC_2	Successful	Fig 5.2
3	TC_3	Successful	Fig 5.3
4	TC_4	Successful	Fig 5.4
5	TC_5	Successful	Fig 5.5
6	TC_6	Successful	Fig 5.6
7	TC_7	Successful	Fig 5.7
8	TC_8	Successful	Fig 5.8
9	TC_9	Successful	Fig 5.9
10	TC_10	Successful	Fig 5.10
11	TC_11	Successful	Fig 5.11
12	TC_12	Successful	Fig 5.12
13	TC_13	Successful	Fig 5.13
14	TC_14	Successful	Fig 5.14

Test Report:

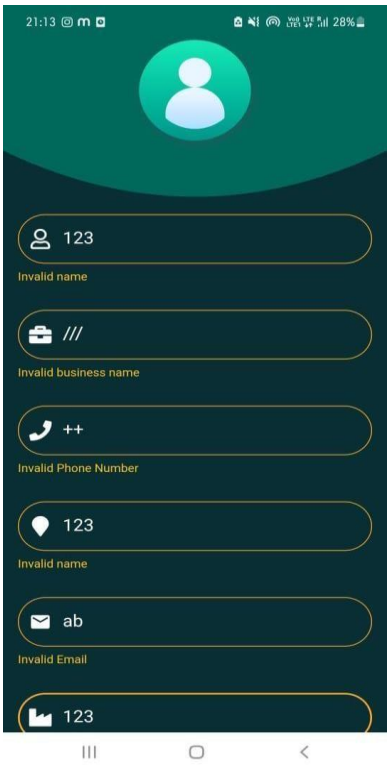

Fig 5.1 Error validations

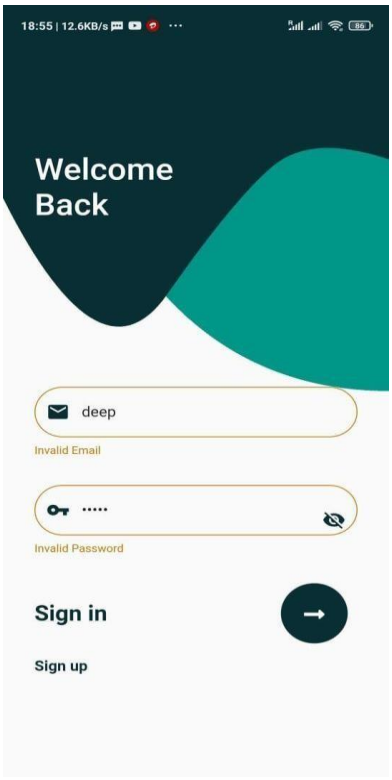

Fig 5.2 User credential validation

Fig 5.3 Data gets added in database

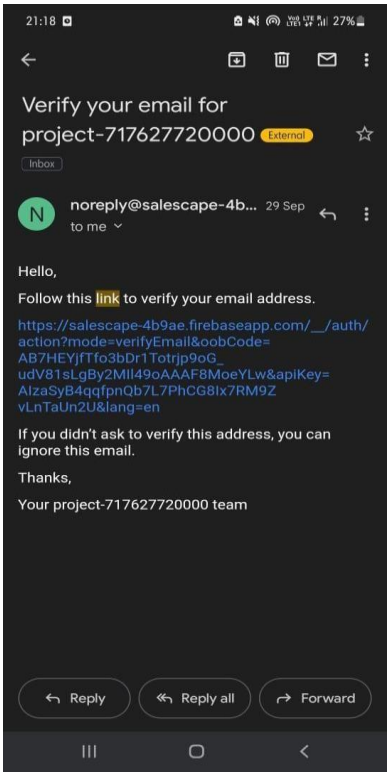

Fig 5.4 Email Verification

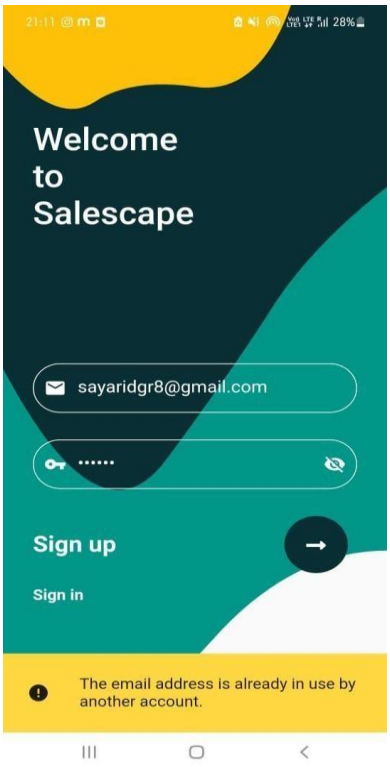

Fig 5.5 One mail id cant be used twice

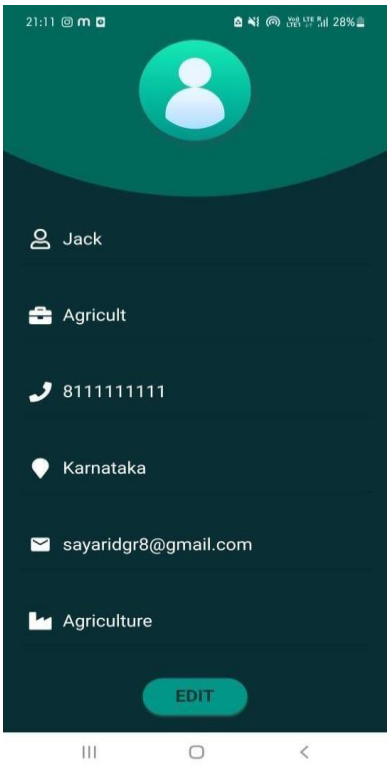

Fig 5.6 User Details can be editable

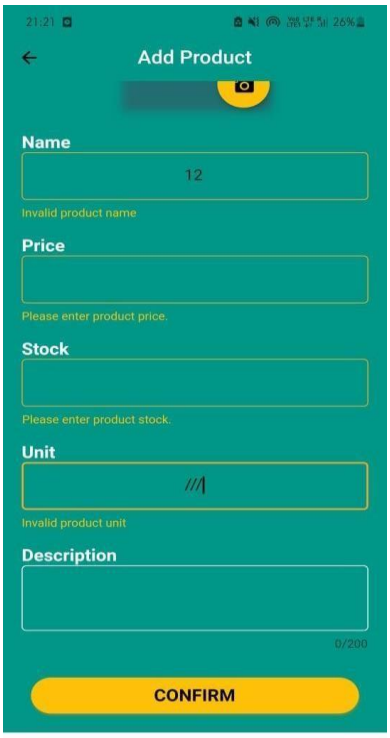

Fig 5.7 Product detail validations

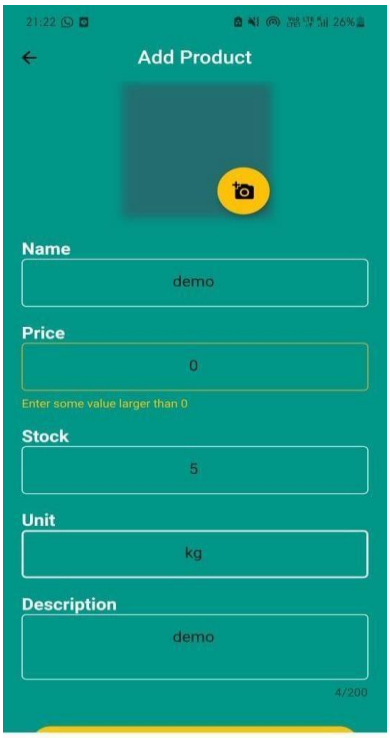

Fig 5.8 Price validation

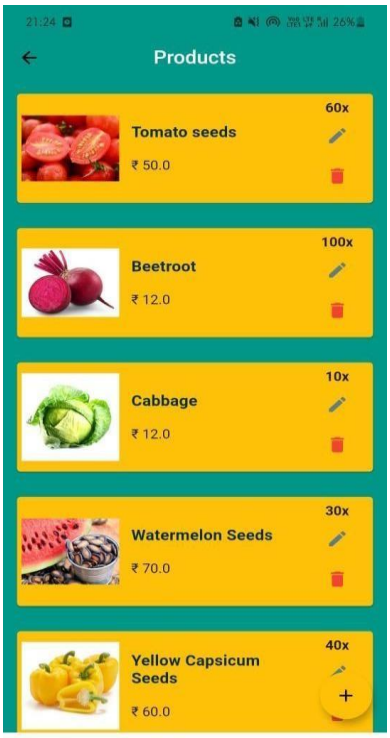

Fig 5.9 Products Screen (Wholesaler)

Fig 5.10 Product Screen (Retailer)

Fig 5.11 Single Wholesaler per order

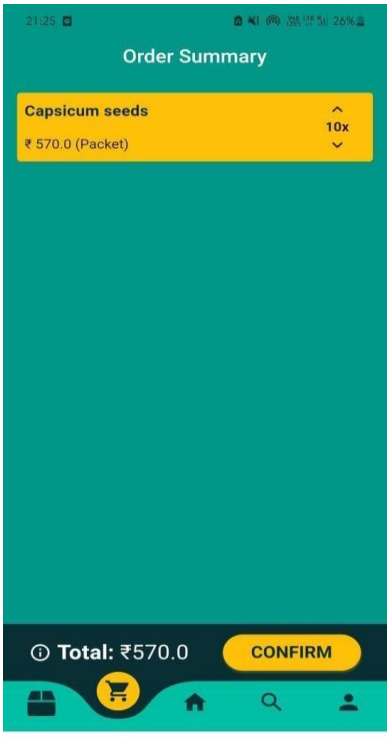

Fig 5.12 Same item cannot be duplicated

6. CONCLUSION

The following chapter elaborates on the issues faced in the lifetime of the project and the advantages and disadvantages of the system along with the future scope of the project. This chapter gives an overall view of the entire project.

6.1 DESIGN AND IMPLEMENTATION ISSUES

Design and implementation are the most important parts of the project and determine everything, right from what the project constitutes of, till how the project actually functions. Design being the plan and implementation being the actual execution of the plan have certain issues while matching up to both the user and the developer's expectations, these issues have been explained ahead.

6.1.1 DESIGN ISSUES

The issues faced in the design phase are usually of the nature where the difficulty mainly arises in balancing the requirements and feasibility of the project, the following were the problems faced in the design phase:

- Designing a system that is sustainable yet maintains the USP of focusing on the wholesaler.
- Determination of a system where 3 entities are correlated without complexity.
- Maintaining simplicity yet making the process efficient.
- Keeping the features of real life trading without inhibiting technological and convenience features.
- Determining tech stack based on requirement.
- Delegating order with limited delivery agents.
- Delegating delivery order based on state and inter state deliveries.

6.1.2 IMPLEMENTATION ISSUES

The issues faced during implementation were related to the code and were technical in nature:

- Requirement of extra email id due to authentication process.
- Real time check on stock and order by simultaneous entities.
- Tracking and order location wise.
- Updating edited information in the database continuously and in multiple collections.
- Dealing with custom industries.

6.2 ADVANTAGES AND LIMITATIONS

Advantages: There are various advantages that Salescape looked forward to covering, most of them being flaws in the current system that created the objective of the project, they are as follows:

- Though the three entities are connected it does not make the user experience complex.
- The simplicity of user experience has been maintained in the entire functioning of the application thus making it convenient for users without a high technical know-how.
- Wholesalers and their brands are focused on by the system.
- Retailers are provided with more options and the options are provided in an organised form.
- Salescape does not affect the margin of the wholesaler in any sort.
- The process is simplified like a normal end user purchase except that it is performed with the motive and features of wholesale-retailer trading.

Limitations: There were certain limitations that persisted in the project and can be worked on in future.

They are as follows:

- Real time communication could be implemented between the wholesaler and retailer.
- Making the categorization based on industry more efficient by adding custom industries and industry descriptions to avoid confusion.
- A social-media like platform could be incorporated for advertising the wholesaler.
- Incorporation of discount coupons and deals required.
- Subjectivity and bargaining involved in trading to be technically simulated.

6.3 FUTURE SCOPE OF PROJECT

The project can be further on scaled in various ways:

- Web and iOS applications could be designed with the help of Flutter.
- Real time communication and chat bots could be implemented to further on make the process simple.
- Increasing the efficiency of the delivery system could be focused on.
- Comparison of cost in different purchases by considering the delivery charges and product prices could be presented to the retailer as a report.