Project Report on

Multi Vendor Flutter Ecommerce Application

by

GEETA BHAT [2020510007]

Under the guidance of

Internal Supervisor

Prof. Sakina Shaikh

Department of Master of Computer Applications

Sardar Patel Institute of Technology

Autonomous Institute Affiliated to Mumbai University

2021-2022

PROJECT APPROVAL CERTIFICATE

This is to certify that the following student

GEETA BHAT [2020510007]

Have satisfactorily carried out work on the project entitled

"Multi Vendor Flutter Ecommerce Application"

Towards the fulfillment of summer project, as laid down by University of Mumbai during year 2021-22.

Project Guide	H.O.D.	Principal
(Prof. Sakina Shaikh)	(Dr. Pooja Raundale)	(Dr. B.N. Chaudhari)

PROJECT EVALUATION CERTIFICATE

This is to certify that the following student

GEETA BHAT

[2020510007]

Have successfully completed the Project report on **"MULTI VENDOR FLUTTER ECOMMERCE APPLICATION"**,which is found to be satisfactory and is approved

At

SARDAR PATEL INSTITUTE OF TECHNOLOGY,
ANDHERI(W), MUMBAI.

INTERNAL EXAMINER

EXTERNAL EXAMINER

Table of contents

ABSTRACT

The aim of this project is to develop an online system which will satisfy all the requirements for creating a platform for multiple vendors online. The main goal is to develop an application for all shopkeepers to sell their products online in the best possible way and help them to boost their business. The users can order products from their mobile phones through apps in few easy steps.

The users include Shop Owners, Customers, Delivery Boys and the Admin for Vendor Operations. In our developing and technology dependent life we totally rely on our gadgets and other smart devices especially smartphones. Today everyone has a smartphone. With this we get an opportunity to use technology in a better way so that it can be made useful to us. And it plays an important part in our daily life and helps us shop online. Shopping on a single app through multiple vendors gives the customers flexibility to choose from different shops.

There is a need for such an application because it provides the customers with essential products like mineral water, honey and artificial sweeteners which is necessary for people having diabetes as well as for those people who are consious about their health.

Objectives

- To identify the best method to create an ecommerce mobile application for shopkeepers, and delivery boys which will help them buy healthy or nutritious items and sell it directly to the customers.
- To help shop owners earn a better income by giving them a platform to sell their products to a large number of customers through delivery boys.
- Also to provide them a wider reaches so that they can earn a handsome profit and maintain their competition against online shopping brands.

List of figures

List of Tables

1. Introduction

1.1 Problem Definition

Shopkeepers are facing a huge loss and reduction as an impact of the growing ecommerce market. Customers are going more towards online shopping which has resulted in decreasing market especially for small shop owners. Big E-commerce brands have outgrown the market and small shop owners are facing the major effects of this. Big E-commerce brands have a tie-up with multiple brands and shop owners and a huge chain of delivery boys. Small shop owners lack this infrastructure.

The remarkable problem is that customers have moved more towards online shopping. The shopkeepers that sell high quality healthy and local food and consumables are not reaching to all the customers as they lack online presence. Multi Vendor Flutter Ecommerce Application "G-CART" refers to the degree to which a user gets the healthy and low calories food products from their nearby shopkeepers online.

So, we are introducing a Flutter application whose objective is to provide the user with all products from local and leading brands from their nearby retailer shops.

1.2 Objective and Scope

The shopkeepers can register and sell their products directly from this application. So they don't need to sell their products only through the shops. This will help them to sell their products in shops as well as online. This can also help the shopkeepers to take bulk orders according to the customer's demand.

1.3 Existing System

There are multiple apps for small stores which are for single vendors with minimal features. These apps are not efficient for small shop keepers as it cannot help to maintain the system.

1.4 Proposed System

The proposed app is a multi vendor app meant to be used by multiple vendors, customers and a chain of delivery boys. This system will help to create a great business for small shopkeepers and help customers buy good and healthy food products from theor nearby locality.

1.5 System Requirements

1.5.1 Hardware Requirements

- Minimum 4GB RAM
- Windows 7 or higher configuration Device
- Android Phone Version 5.0 or above

1.5.2 Software Requirements

- Android Studio
- Google Firebase

2. Literature Survey

The existing system does not support multi-vendor product management and also doesn't consists of all the latest features which is provided in the proposed application. There are basic applications to view menu of the products but no ordering system. There is no application for ordering products from small retail shops.

2.1 Tools and Techniques

2.1.1 Tools used

- Google Firebase
- Android Studio

2.1.2 Techniques used

- Firebase libraries
- Firebase dependencies

2.2 Project Management Plan

2.2.1 Tasks

The major tasks are:

- Analysis of requirements
- Analysis of major deliveries
- Designing the GUI
- Modeling
- Documentation
- Approval of Documentation
- Testing the project
- Report

2.2.2 Information Gathering

During the information gathering it was observed that we need a separate apps for different users like customers, shop owners and delivery person to maintain the ease of use. It is necessary to make the UI friendly and interactive.

3. SRS and Design

3.1 Introduction

3.1.1 Purpose

The purpose of this application is to provide shopkeepers with a full-fledged online system to sell their products to the users online and deliver the products through delivery agents. Once, the user registers with his/her mobile number he can place order for the products. The vendor approves the orders, assigns delivery boy and the delivery boy collects the parcel from the shop and delivers to the customers.

3.1.2 System Overview

The project is developed using Flutter and Firebase to develop an ecommerce application. The GUI includes three applications: G-CART, G-CART Delivery and G-CART Vendor. The GUI of the G-CART app includes: Home Page, Favorites Page, My Orders Page, and Accounts Page. The G-CART Delivery app is for the delivery person. The GUI includes a Dashboard UI created through flutter and firebase with Filters for All Orders, Accepted Orders, Ordered or Picked Up orders, on the way orders or Delivered Orders. The G-CART Vendor App is a mobile admin app which is connected to the Firebase and other two G-CART apps. The G-CART vendor app has a dashboard, products page, banners page, coupons page, orders page, and product banners page.

3.2 Overall Description

3.2.1 Product Description

G-CART is basically a multi-vendor online ecommerce shopping application which has a complete system for admin, delivery associates and the users or customers. The App is built using Flutter with Dart Programming Language in Android Studio Platform. The Admin panel is

made with Flutter for web panel. The Database and database connection with the application is done using Firebase.

3.2.2 Product functions

The customer application consists of various Ecommerce features like:

- Displaying image slider banners
- Displaying Nearby stores
- Shop by category filter
- Displaying Recently added products, featured products in the app and the best-selling products
- Adding products to favorites and save them for later
- Removing products from favorites category
- Adding products to the cart and updating the cart
- Using voucher code for discount
- Choosing payment option
- Adding card details
- View order details
- Track order
- Manage cards
- Edit account details

The vendor application features are:

- View all orders
- Accept or reject placed orders
- Assign delivery boy for accepted orders
- Select delivery boy
- Add new products to the database
- Publish or Unpublish the products

- Add new banners
- Delete existing banners
- Add new coupon
- Activate or deactivate coupons
- Add new product banners

The delivery application features are:

- View placed orders
- View customer delivery location
- Call the customer

3.2.3 User characteristics

The use case in this system is a customer orders a products, the vendor admin approves the orders, assigns a delivery agent and the delivery agents delivers the order.

3.3 Specific Requirements

3.3.1 External Interface Requirements

The system does not require any specific external interface to run the application.

3.3.2 Logical DB Requirements

The Database used for the application is Google Firebase. Google Firebase is a NoSQL Database used for Flutter Applications to store and maintain the data for applications.

4. Project Analysis and Design

4.1 Methodologies Accepted

4.1.1 Detailed Life Cycle of the Project

We would suggest the use of an Incremental Model for this project. The requirements are broken down into smaller, stand-alone and can be manageable modules and we work on each module right from analysis design till testing and get one module working and then we can start building another module right on top of the existing module. Each iteration passes through the requirements, design, coding and testing phases and each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

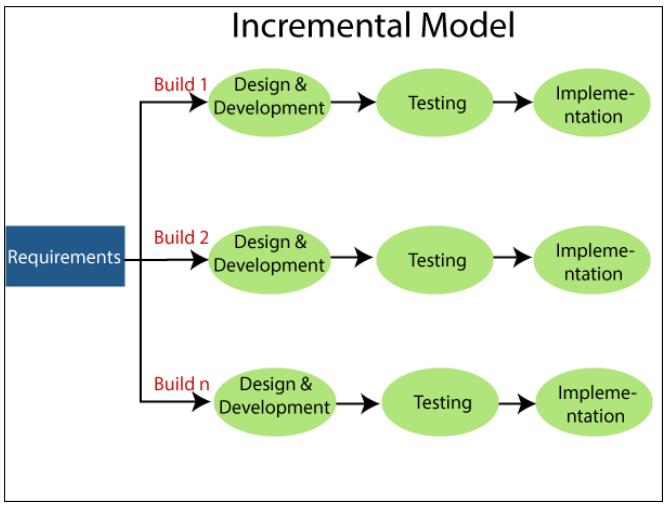


Fig 4.1.1 Incremental Model

The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is plan development for the next increment. Once the requirement is developed, requirements for that increment are frozen. It is majorly selected because of the small team sizes. As there are only 3 people who would be working to make this project a reality, we need a software model which can generate a prototype quickly, is flexible, less expensive and one in which errors are easily identified. And all of these are the foundation stones of the Incremental Model.

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes

through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

The incremental model has four phases:

Requirement Analysis->Design & Development->Testing->Implementation

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product.

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantages of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

4.1.2 Work Breakdown Structure (WBS)

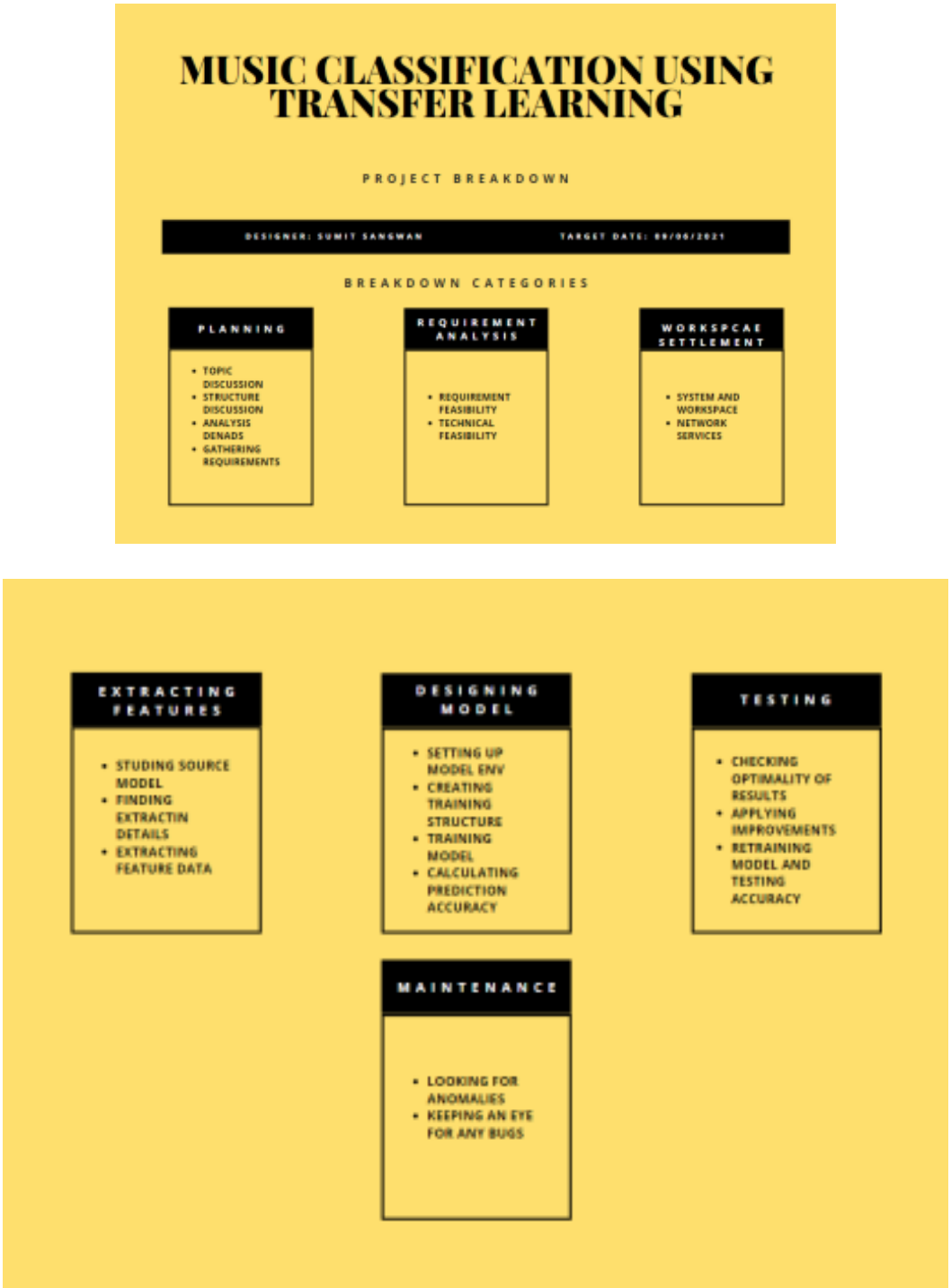


Fig 4.1.2 Work Breakdown Structure (WBS)

4.2 UML Diagram

4.2.1 ER Diagram

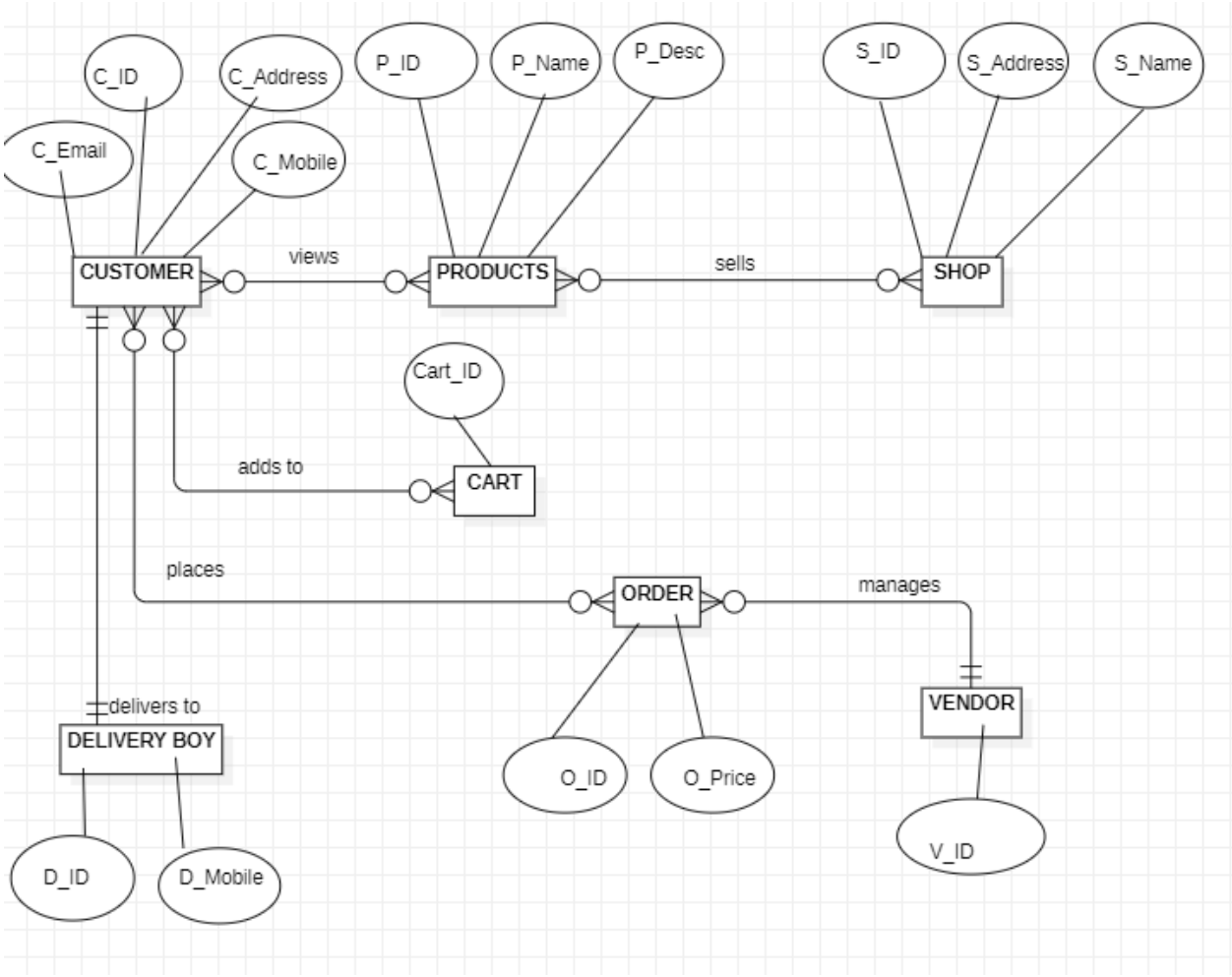


Fig 4.2.1- ER Diagram

4.3 Database Design

4.3.1 Database Schema Design

Table 4.1 User Table (Schema)

Column name	Data Type	Constraints
U_ID	VARCHAR(45)	Unique, NOT NULL
U_Name	VARCHAR(45)	NOT NULL
U_Email	VARCHAR(45)	Unique, NOT NULL
U_Mobile	INT(20)	Primary Key

Table 4.2 Vendor Table (Schema)

Column name	Data Type	Constraints
V_ID	VARCHAR(45)	Unique, NOT NULL
V_Name	VARCHAR(45)	NOT NULL
V_Email	VARCHAR(45)	Unique, NOT NULL
V_Mobile	INT(20)	Primary Key
V_ShopName	VARCHAR(45)	NOT NULL

V_ShopAddress	VARCHAR(45)	NOT NULL

Table 4.3 Product Table (Schema)

Column name	Data Type	Constraints
P_ID	VARCHAR(45)	Primary Key
P_Name	VARCHAR(45)	NOT NULL
P_Price	INT(20)	NOT NULL
P_Description	VARCHAR(45)	NOT NULL
P_VendorName	VARCHAR(45)	NOT NULL
V_ShopAddress	VARCHAR(45)	NOT NULL

5. Project Implementation & Testing

5.1 Gantt Chart

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. Gantt charts help teams to plan work around deadlines and properly allocate resources. Project planners also use Gantt charts to maintain a bird's eye view of projects. They depict, among other things, the relationship between the start and end dates of tasks, milestones, and dependent tasks.

5.2 Pert Chart

A PERT chart is a visual project management tool used to map out and track the tasks and timelines. The name PERT is an acronym for Project (or Program) Evaluation and Review Technique. A PERT chart, sometimes called a PERT diagram, is a project management tool used to schedule, organize and coordinate tasks within a project. It provides a graphical representation of a project's timeline that enables project managers to break down each individual task in the project for analysis.

5.3 Timeline Chart

A timeline chart is a visual rendition of a series of events. It can be created as a chart or a graph. Timeline charts can be created for anything that occurred over a period of time.

A timeline chart makes it easier to conceptualize a process or a sequence of events. It can make it easier for you to understand the intricacies of a project or why something seems to take so long to accomplish.

Here are some common uses of a simple timeline chart:

- Helps to manage and keep complex tasks on time.
- Allows you to visualize a process or events that took place over a period of time.
- Works as an aid to decision-making about current and future projects.
- Allocation of resources at the right time is easier.
- A timeline chart offers a variety of uses.

5.4 Code Pages

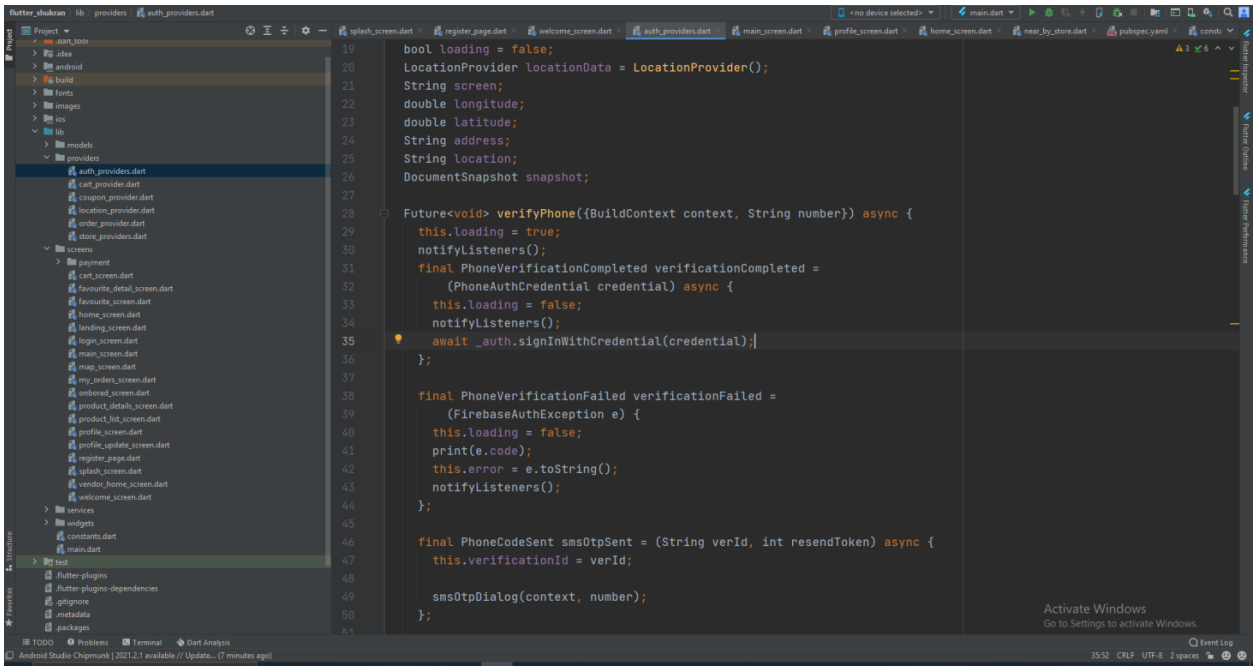
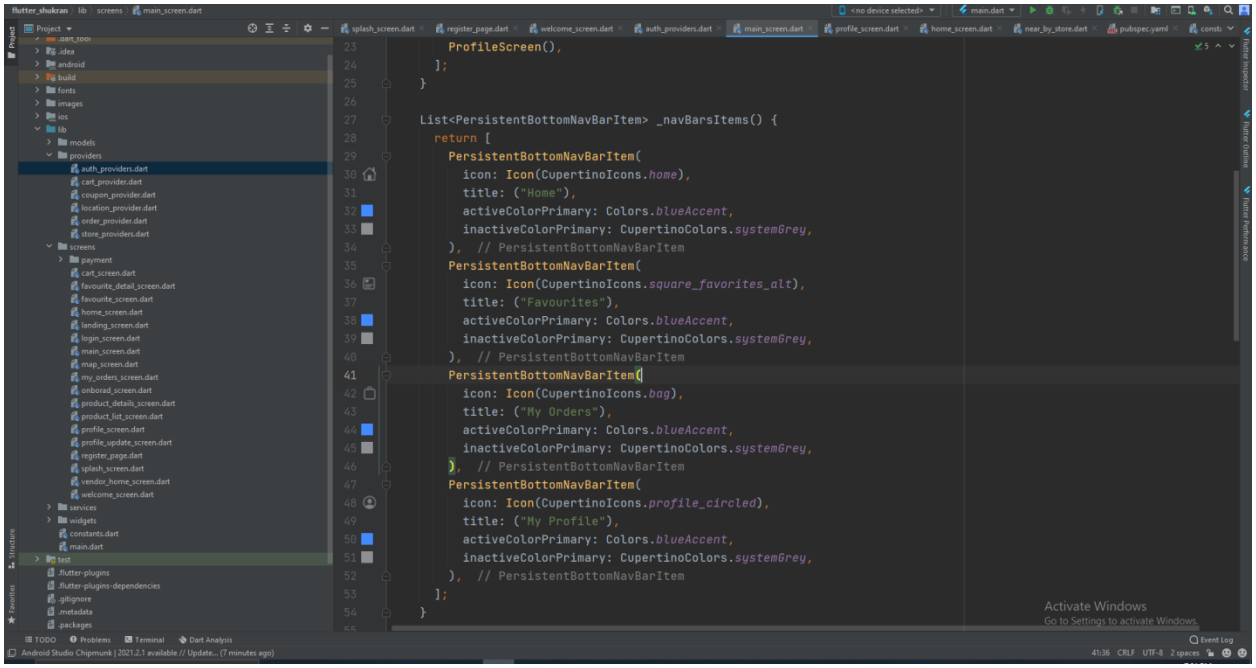
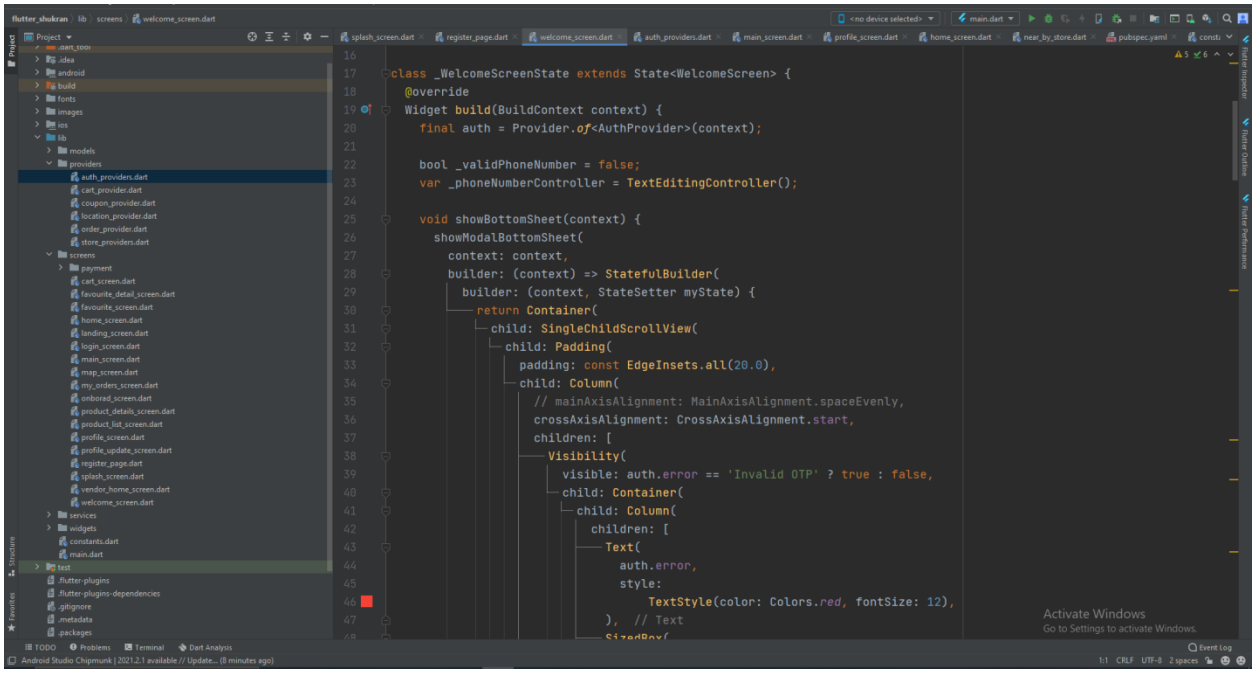


Figure 5.4.1 User Code-1



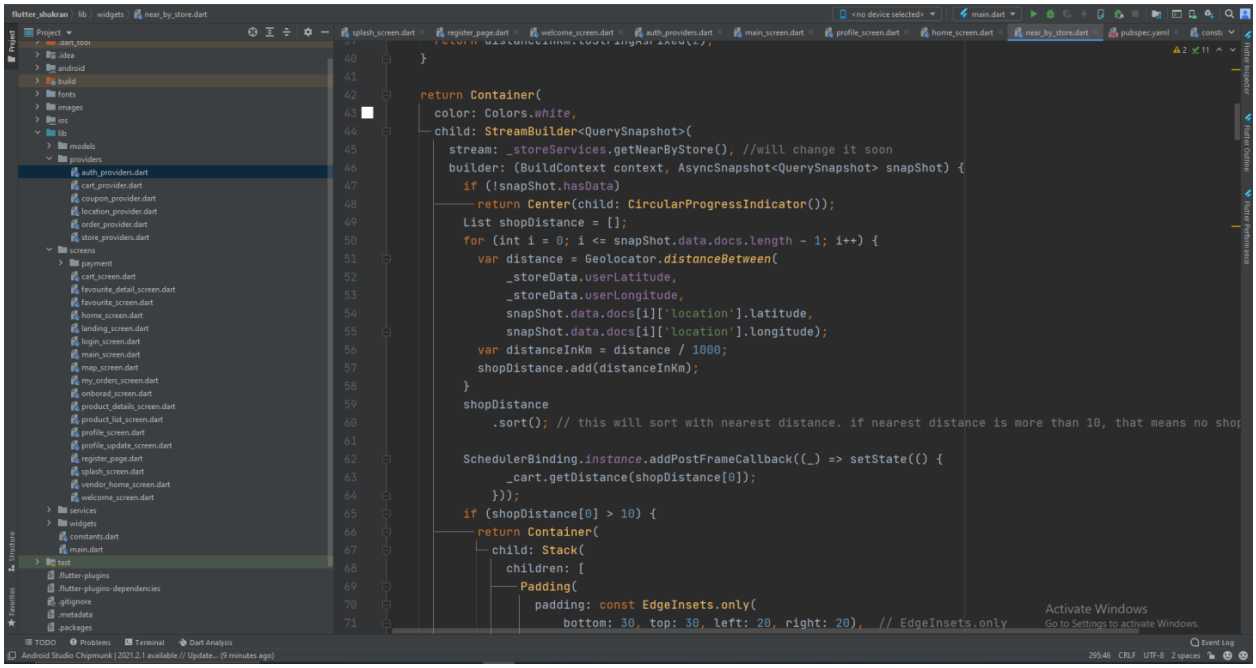
```
      ProfileScreen(),
    ];
  }

  List<PersistentBottomNavBarItem> _navBarsItems() {
    return [
      PersistentBottomNavBarItem(
        icon: Icon(CupertinoIcons.home),
        title: ("Home"),
        activeColorPrimary: Colors.blueAccent,
        inactiveColorPrimary: CupertinoColors.systemGrey,
      ),  // PersistentBottomNavBarItem
      PersistentBottomNavBarItem(
        icon: Icon(CupertinoIcons.square_favorites_alt),
        title: ("Favourites"),
        activeColorPrimary: Colors.blueAccent,
        inactiveColorPrimary: CupertinoColors.systemGrey,
      ),  // PersistentBottomNavBarItem
      PersistentBottomNavBarItem(
        icon: Icon(CupertinoIcons.bag),
        title: ("My Orders"),
        activeColorPrimary: Colors.blueAccent,
        inactiveColorPrimary: CupertinoColors.systemGrey,
      ),  // PersistentBottomNavBarItem
      PersistentBottomNavBarItem(
        icon: Icon(CupertinoIcons.profile_circled),
        title: ("My Profile"),
        activeColorPrimary: Colors.blueAccent,
        inactiveColorPrimary: CupertinoColors.systemGrey,
      ),  // PersistentBottomNavBarItem
    ];
  }
```

Figure 5.4.2 User Code-2



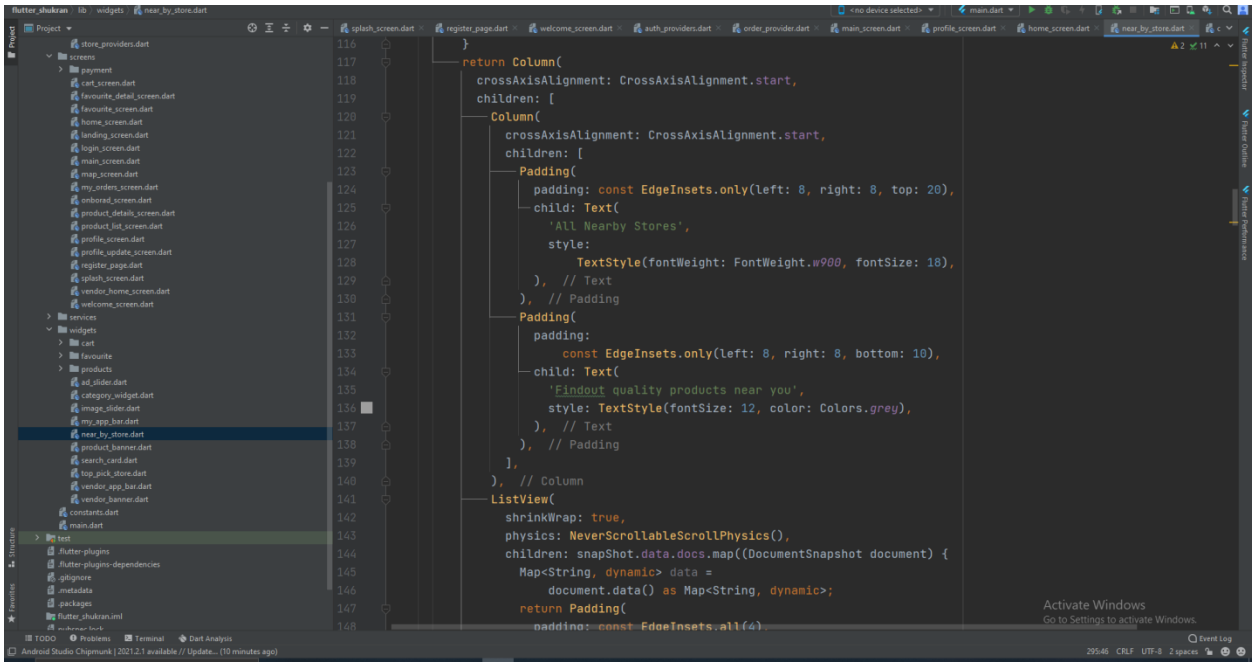
```
class _WelcomeScreenState extends State<WelcomeScreen> {
  @override
  Widget build(BuildContext context) {
    final auth = Provider.of<AuthProvider>(context);

    bool _validPhoneNumber = false;
    var _phoneNumberController = TextEditingController();

    void showBottomSheet(context) {
      showModalBottomSheet(
        context: context,
        builder: (context) => StatefulBuilder(
          builder: (context, StateSetter myState) {
            return Container(
              child: SingleChildScrollView(
                child: Padding(
                  padding: const EdgeInsets.all(20.0),
                  child: Column(
                    // mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Visibility(
                        visible: auth.error == 'Invalid OTP' ? true : false,
                        child: Container(
                          child: Column(
                            children: [
                              Text(
                                auth.error,
                                style:
                                    TextStyle(color: Colors.red, fontSize: 12),
                              ),  // Text
                              SizedBox(
```

Figure 5.4.3 User Code-3



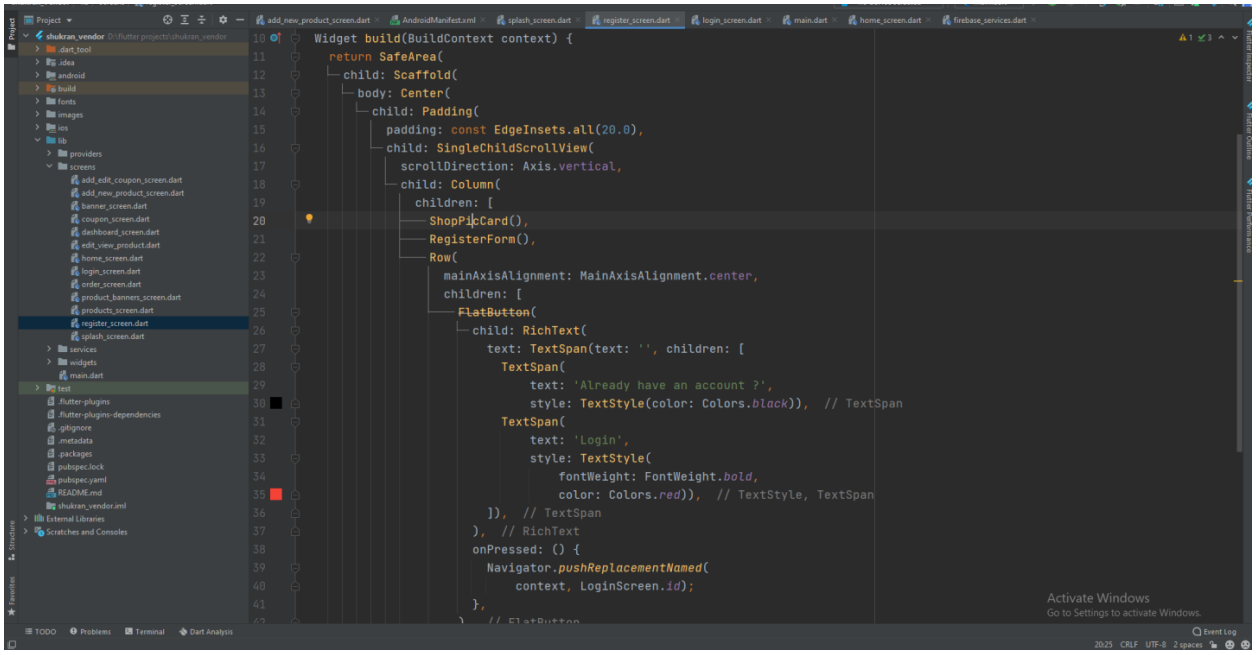
```
    }

    return Container(
      color: Colors.white,
      child: StreamBuilder<QuerySnapshot>(
        stream: _storeServices.getNearByStore(), //will change it soon
        builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapShot) {
          if (!snapShot.hasData)
            return Center(child: CircularProgressIndicator());
          List shopDistance = [];
          for (int i = 0; i <= snapShot.data.docs.length - 1; i++) {
            var distance = Geolocator.distanceBetween(
                _storeData.userLatitude,
                _storeData.userLongitude,
                snapShot.data.docs[i]['location'].latitude,
                snapShot.data.docs[i]['location'].longitude);
            var distanceInKm = distance / 1000;
            shopDistance.add(distanceInKm);
          }
          shopDistance
              .sort(); // this will sort with nearest distance. if nearest distance is more than 10, that means no shop

          SchedulerBinding.instance.addPostFrameCallback((_) => setState(() {
                _cart.getDistance(shopDistance[0]);
              }));
          if (shopDistance[0] > 10) {
            return Container(
              child: Stack(
                children: [
                  Padding(
                    padding: const EdgeInsets.only(
                        bottom: 30, top: 30, left: 20, right: 20),  // EdgeInsets.only
```

Figure 5.4.4 User Code-4



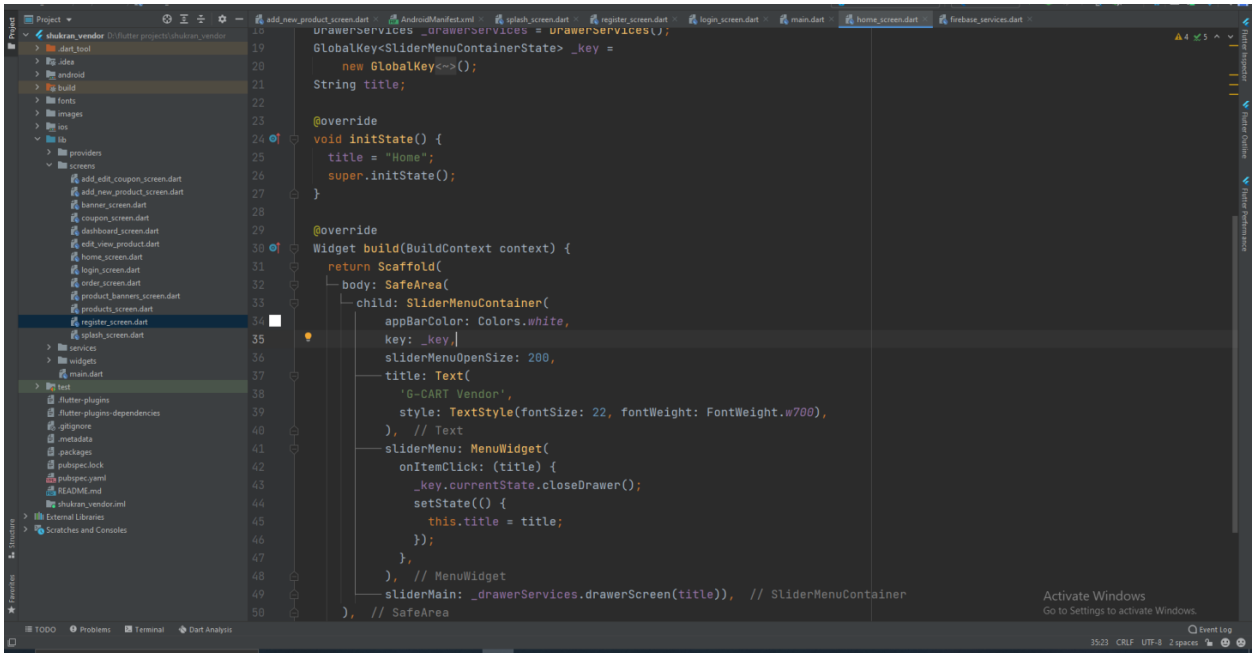
```
      }
      return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Padding(
                padding: const EdgeInsets.only(left: 8, right: 8, top: 20),
                child: Text(
                  'All Nearby Stores',
                  style:
                      TextStyle(fontWeight: FontWeight.w900, fontSize: 18),
                ),  // Text
              ),  // Padding
              Padding(
                padding:
                    const EdgeInsets.only(left: 8, right: 8, bottom: 10),
                child: Text(
                  'Findout quality products near you',
                  style: TextStyle(fontSize: 12, color: Colors.grey),
                ),  // Text
              ),  // Padding
            ],
          ),  // Column
          ListView(
            shrinkWrap: true,
            physics: NeverScrollableScrollPhysics(),
            children: snapShot.data.docs.map((DocumentSnapshot document) {
              Map<String, dynamic> data =
                  document.data() as Map<String, dynamic>;
              return Padding(
                padding: const EdgeInsets.all(4),
```

Figure 5.4.5 User Code-5



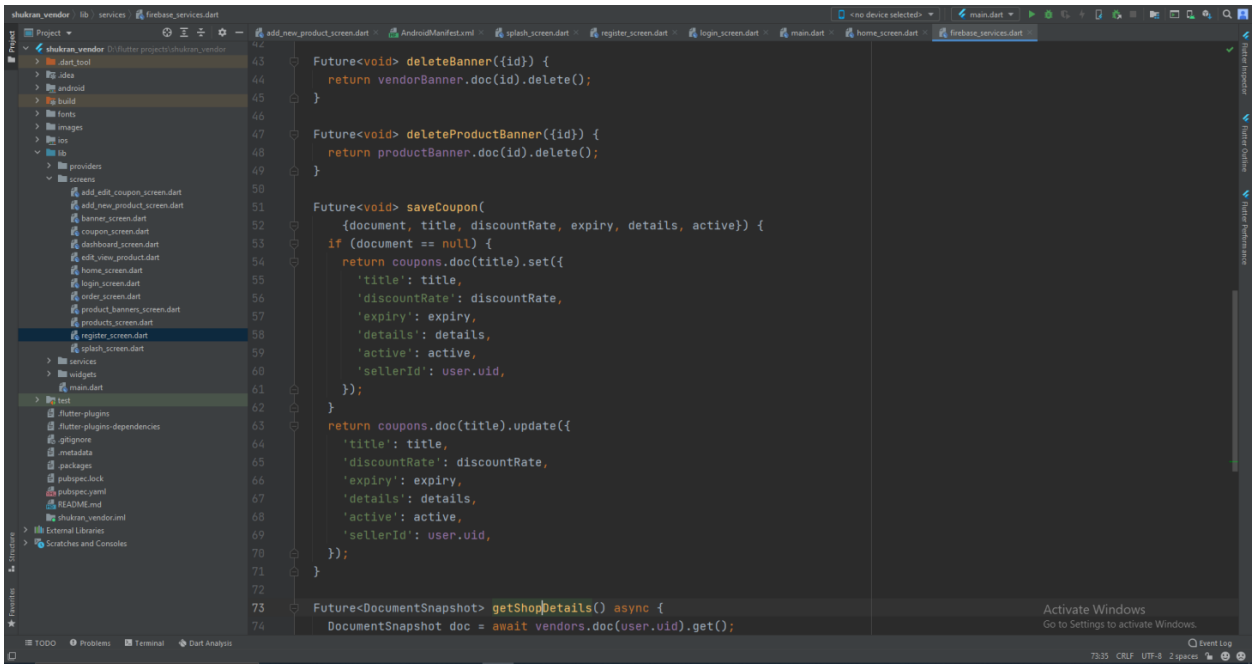
```
Widget build(BuildContext context) {
  return SafeArea(
    child: Scaffold(
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: SingleChildScrollView(
            scrollDirection: Axis.vertical,
            child: Column(
              children: [
                ShopPicCard(),
                RegisterForm(),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    FlatButton(
                      child: RichText(
                          text: TextSpan(text: '', children: [
                            TextSpan(
                                text: 'Already have an account ?',
                                style: TextStyle(color: Colors.black)),  // TextSpan
                            TextSpan(
                                text: 'Login',
                                style: TextStyle(
                                    fontWeight: FontWeight.bold,
                                    color: Colors.red)),  // TextStyle, TextSpan
                          ]),  // TextSpan
                      ),  // RichText
                      onPressed: () {
                        Navigator.pushReplacementNamed(
                            context, LoginScreen.id);
                      },
```

Figure 5.4.6 Vendors Code-1



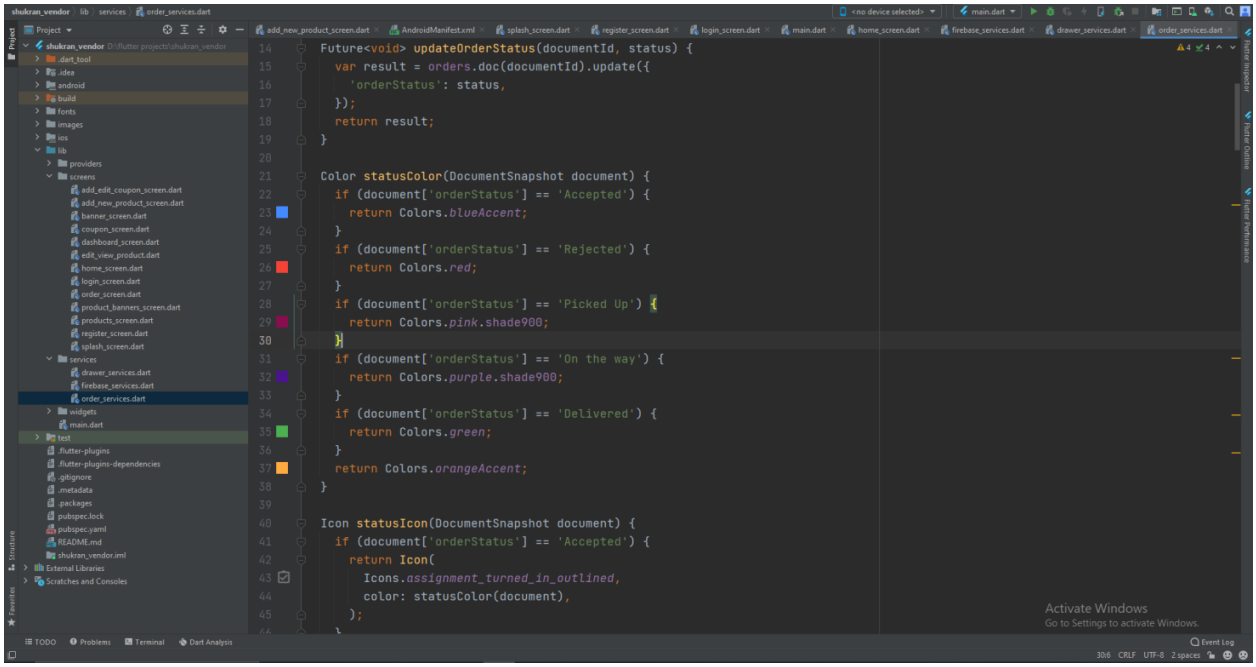
```
GlobalKey<SliderMenuContainerState> _key =
    new GlobalKey<>();
String title;

@override
void initState() {
  title = "Home";
  super.initState();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SafeArea(
      child: SliderMenuContainer(
          appBarColor: Colors.white,
          key: _key,
          sliderMenuOpenSize: 200,
          title: Text(
            'G-CART Vendor',
            style: TextStyle(fontSize: 22, fontWeight: FontWeight.w700),
          ),  // Text
          sliderMenu: MenuWidget(
            onItemClick: (title) {
              _key.currentState.closeDrawer();
              setState(() {
                this.title = title;
              });
            },
          ),  // MenuWidget
          sliderMain: _drawerServices.drawerScreen(title)),  // SliderMenuContainer
    ),  // SafeArea
```

Figure 5.4.7 Vendors Code-2



```
Future<void> deleteBanner({id}) {
  return vendorBanner.doc(id).delete();
}

Future<void> deleteProductBanner({id}) {
  return productBanner.doc(id).delete();
}

Future<void> saveCoupon(
    {document, title, discountRate, expiry, details, active}) {
  if (document == null) {
    return coupons.doc(title).set({
      'title': title,
      'discountRate': discountRate,
      'expiry': expiry,
      'details': details,
      'active': active,
      'sellerId': user.uid,
    });
  }
  return coupons.doc(title).update({
    'title': title,
    'discountRate': discountRate,
    'expiry': expiry,
    'details': details,
    'active': active,
    'sellerId': user.uid,
  });
}

Future<DocumentSnapshot> getShopDetails() async {
  DocumentSnapshot doc = await vendors.doc(user.uid).get();
```

Figure 5.4.8 Vendors Code-3



```
Future<void> updateOrderStatus(documentId, status) {
  var result = orders.doc(documentId).update({
    'orderStatus': status,
  });
  return result;
}

Color statusColor(DocumentSnapshot document) {
  if (document['orderStatus'] == 'Accepted') {
    return Colors.blueAccent;
  }
  if (document['orderStatus'] == 'Rejected') {
    return Colors.red;
  }
  if (document['orderStatus'] == 'Picked Up') {
    return Colors.pink.shade900;
  }
  if (document['orderStatus'] == 'On the way') {
    return Colors.purple.shade900;
  }
  if (document['orderStatus'] == 'Delivered') {
    return Colors.green;
  }
  return Colors.orangeAccent;
}

Icon statusIcon(DocumentSnapshot document) {
  if (document['orderStatus'] == 'Accepted') {
    return Icon(
      Icons.assignment_turned_in_outlined,
      color: statusColor(document),
    );
```

Figure 5.4.9 Vendors Code-4

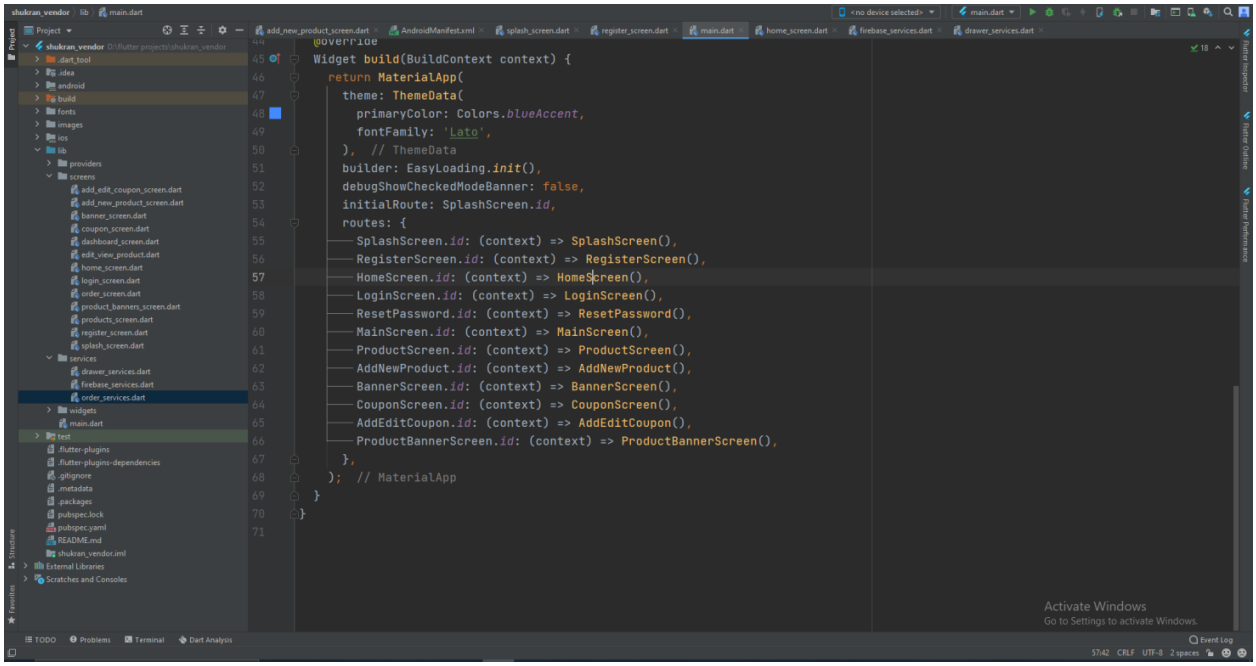


Figure 5.4.10 Vendors Code-5

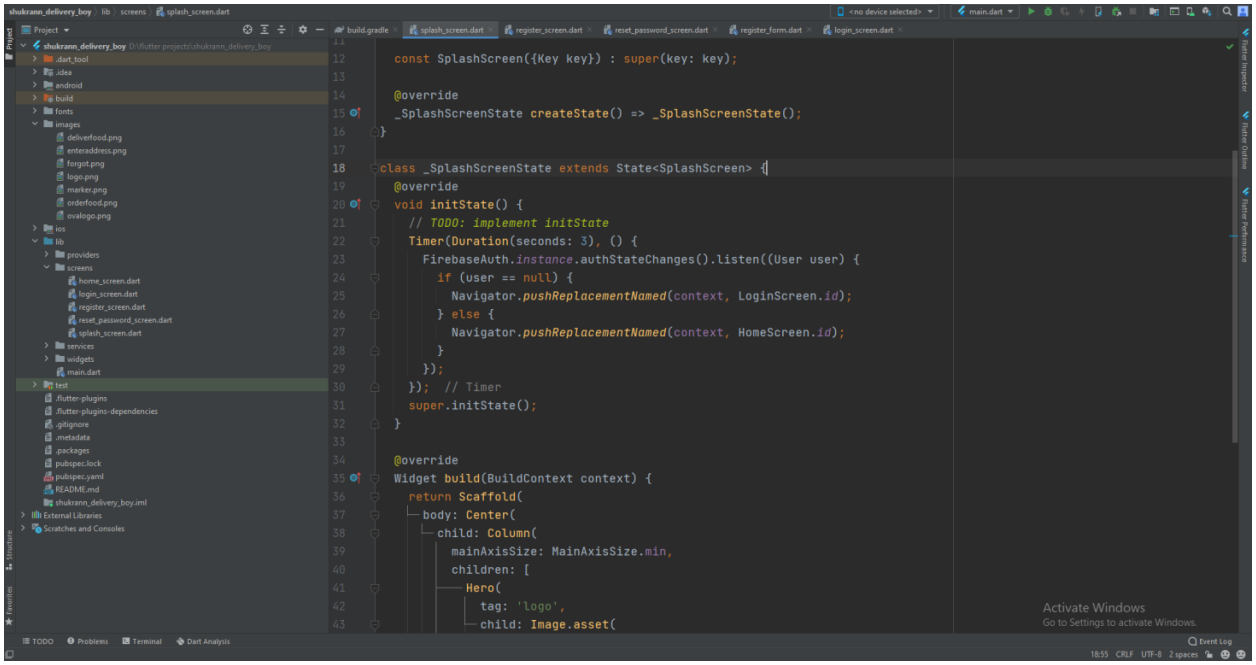
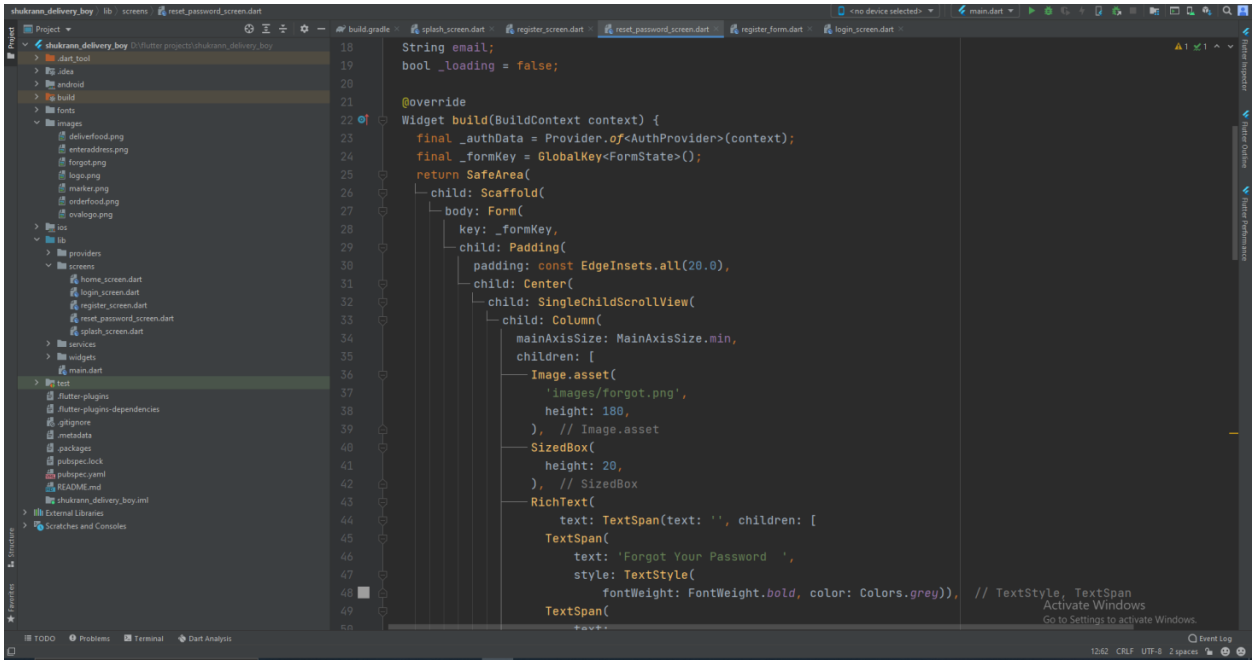
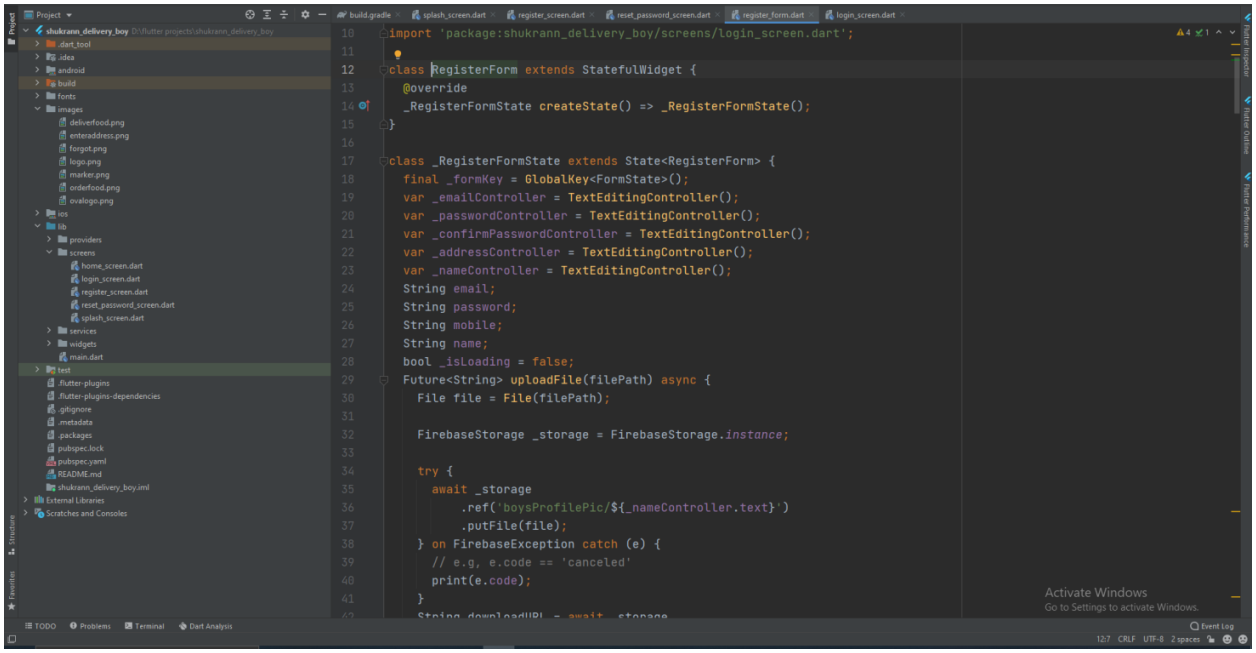


Figure 5.4.11 Delivery Code-1



```
String email;
bool _loading = false;

@override
Widget build(BuildContext context) {
  final _authData = Provider.of<AuthProvider>(context);
  final _formKey = GlobalKey<FormState>();
  return SafeArea(
    child: Scaffold(
      body: Form(
        key: _formKey,
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Center(
            child: SingleChildScrollView(
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Image.asset(
                    'images/forgot.png',
                    height: 180,
                  ),  // Image.asset
                  SizedBox(
                    height: 20,
                  ),  // SizedBox
                  RichText(
                      text: TextSpan(text: '', children: [
                    TextSpan(
                        text: 'Forgot Your Password  ',
                        style: TextStyle(
                            fontWeight: FontWeight.bold, color: Colors.grey)),  // TextStyle, TextSpan
                    TextSpan(
```

Figure 5.4.12 Delivery Code-2



```
import 'package:shukrann_delivery_boy/screens/login_screen.dart';

class RegisterForm extends StatefulWidget {
  @override
  _RegisterFormState createState() => _RegisterFormState();
}

class _RegisterFormState extends State<RegisterForm> {
  final _formKey = GlobalKey<FormState>();
  var _emailController = TextEditingController();
  var _passwordController = TextEditingController();
  var _confirmPasswordController = TextEditingController();
  var _addressController = TextEditingController();
  var _nameController = TextEditingController();
  String email;
  String password;
  String mobile;
  String name;
  bool _isLoading = false;
  Future<String> uploadFile(filePath) async {
    File file = File(filePath);

    FirebaseStorage _storage = FirebaseStorage.instance;

    try {
      await _storage
          .ref('boysProfilePic/${_nameController.text}')
          .putFile(file);
    } on FirebaseException catch (e) {
      // e.g, e.code == 'canceled'
      print(e.code);
    }
    String downloadURL = await storage
```

Figure 5.4.13 Delivery Code-3

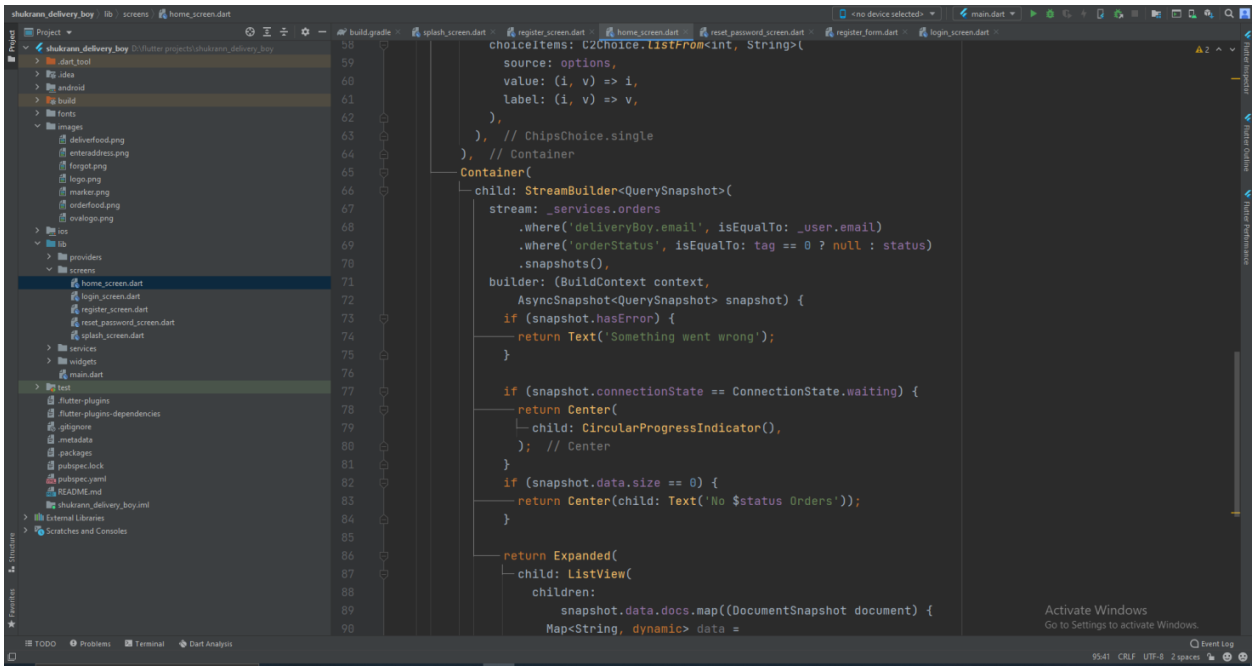


Figure 5.4.14 Delivery Code-4

5.5 Snapshot of Application

5.5.1 Application

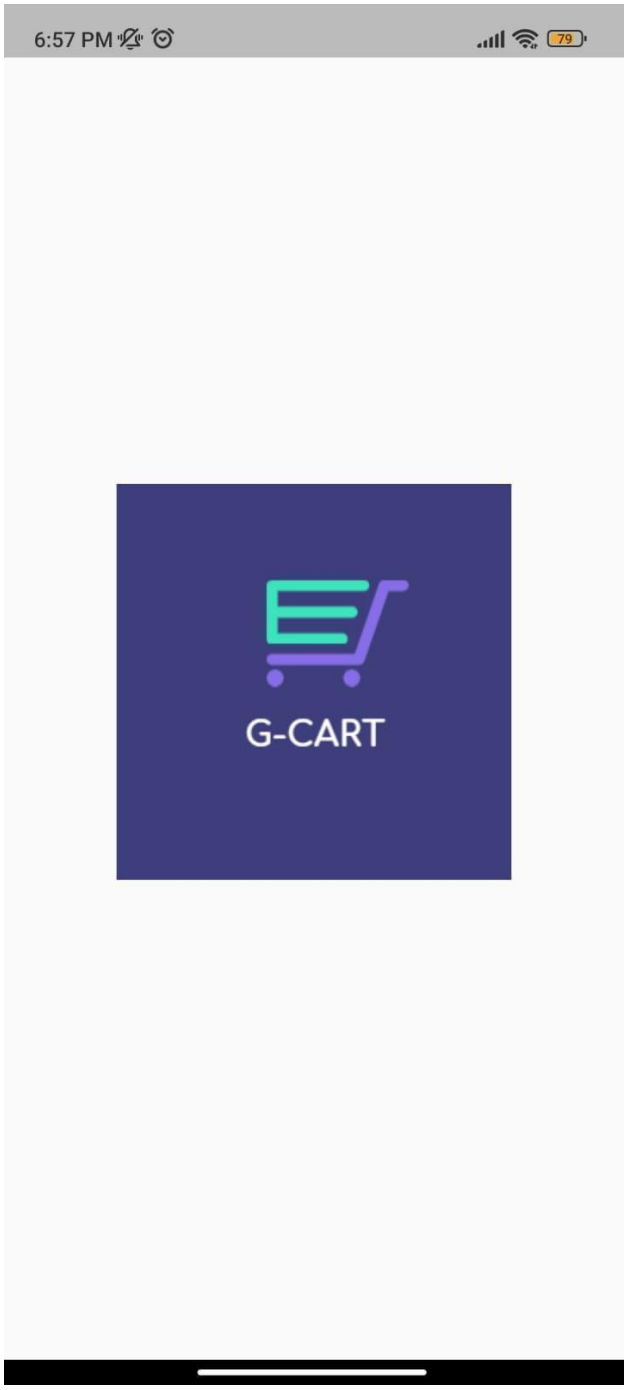


Figure 5.5.1 Logo Page

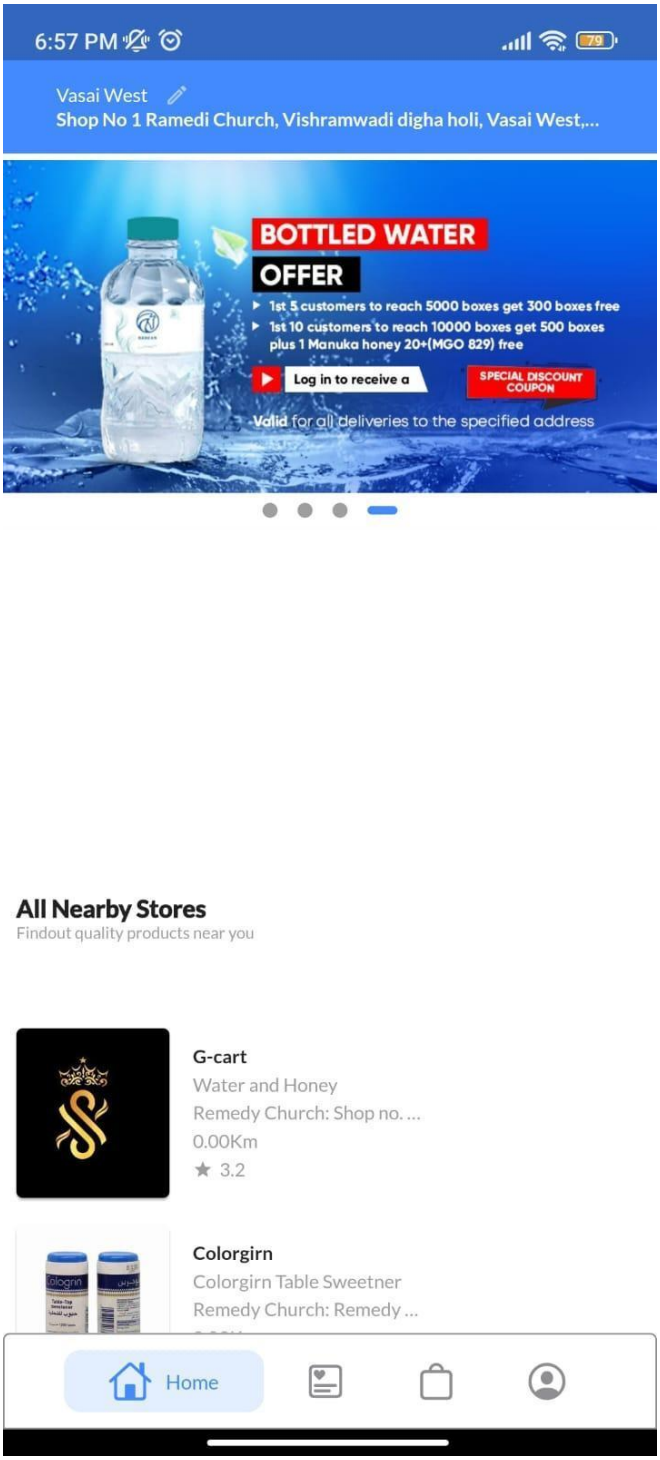


Figure 5.5.2 Home Page 1

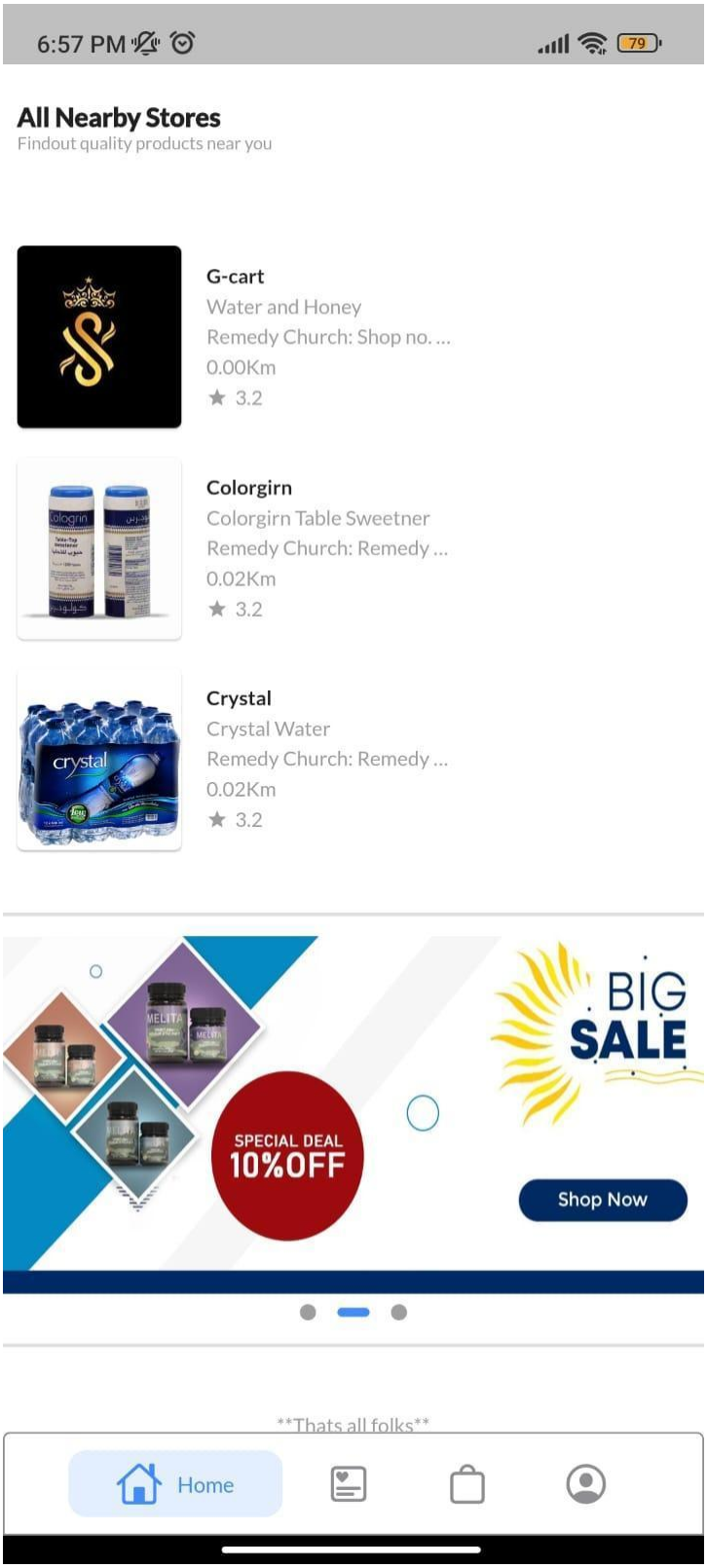


Figure 5.5.3 Home Page 2

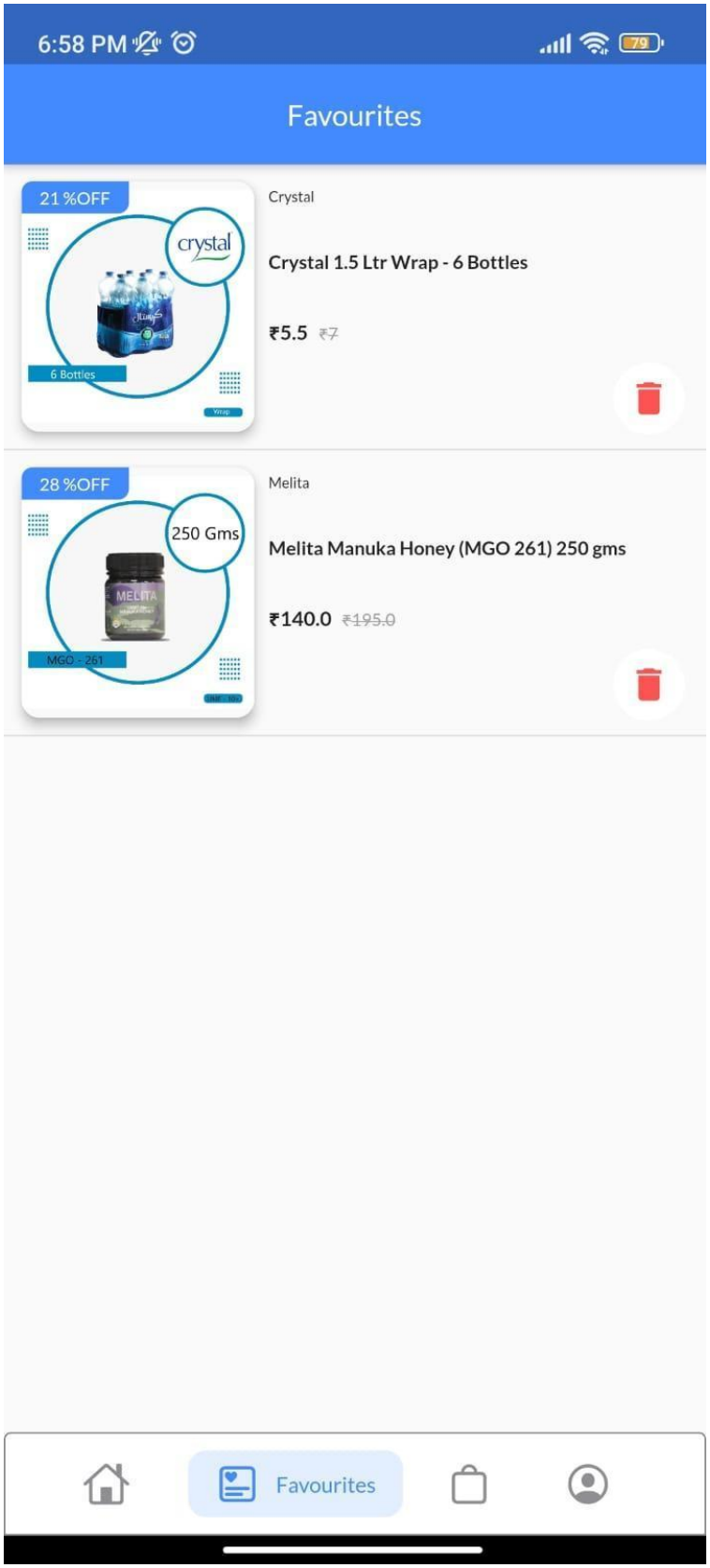

6:58 PM
Favourites
21 %OFF
crystal
6 Bottles
Crystal
Crystal 1.5 Ltr Wrap - 6 Bottles
₹5.5 ₹7
28 %OFF
250 Gms
MELITA
MGO - 261
Melita
Melita Manuka Honey (MGO 261) 250 gms
₹140.0 ₹195.0
Favourites

Figure 5.5.4 Favourites Page

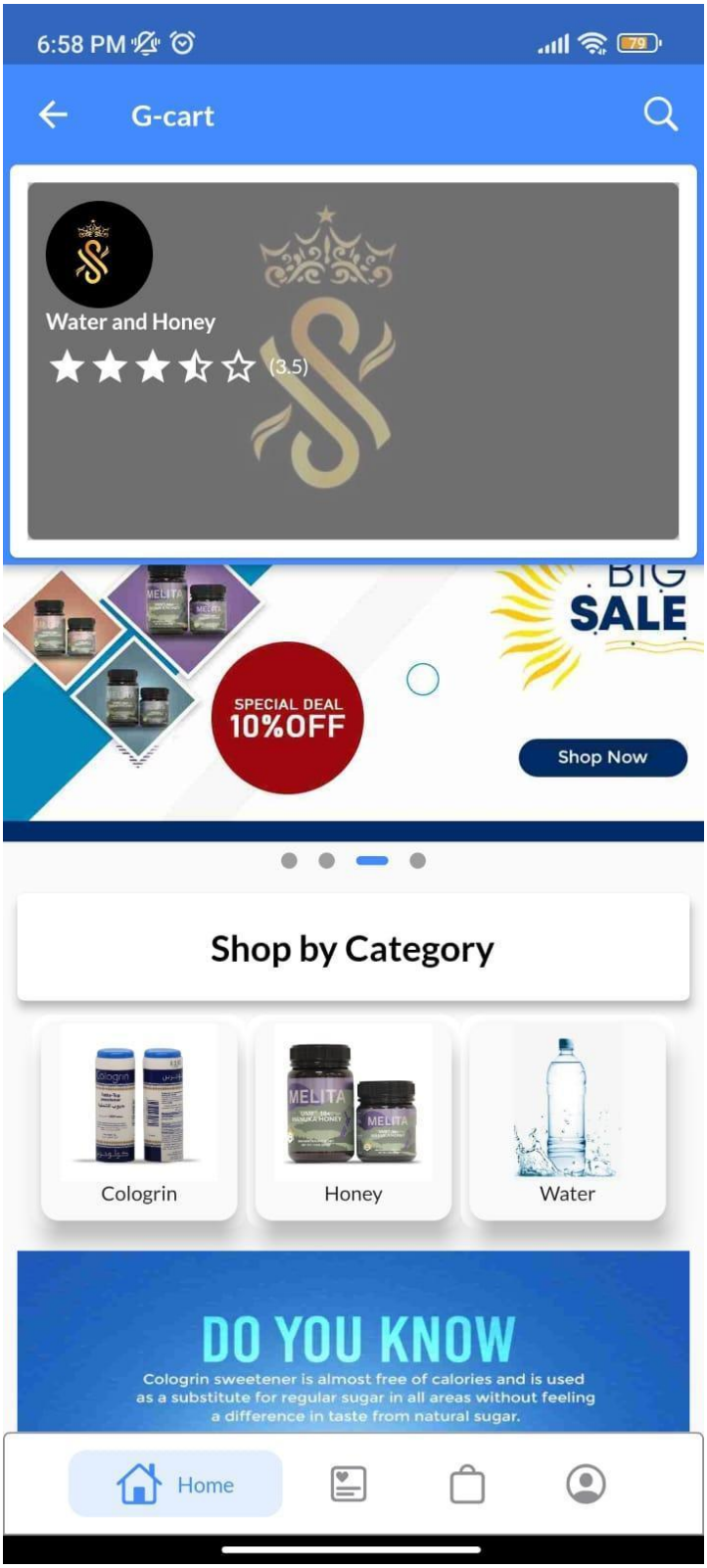


Figure 5.5.4 Home Page 3

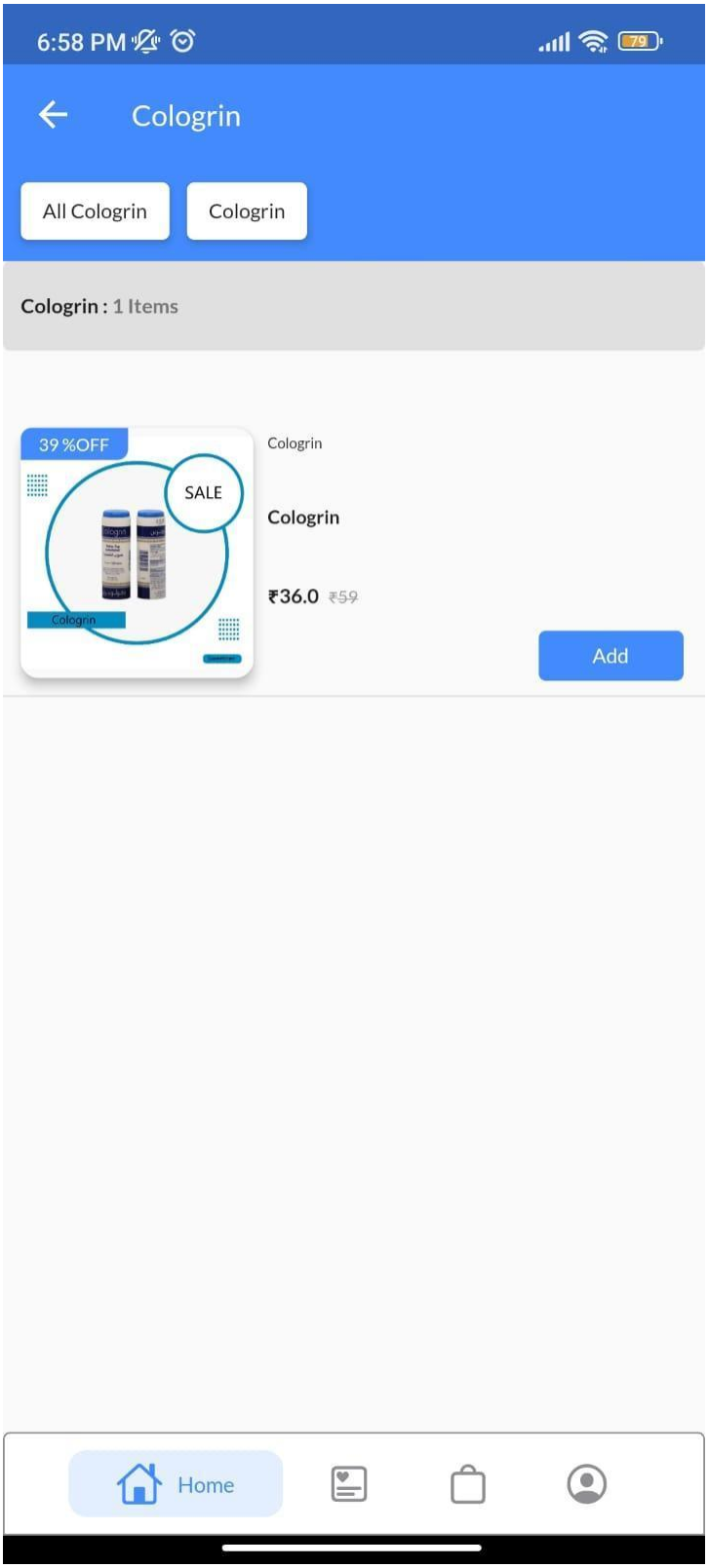

6:58 PM
Cologrin
All Cologrin
Cologrin
Cologrin : 1 Items
39 %OFF
SALE
Cologrin
Cologrin
Cologrin
₹36.0 ₹59
Add
Home

Figure 5.5.6 Category Page

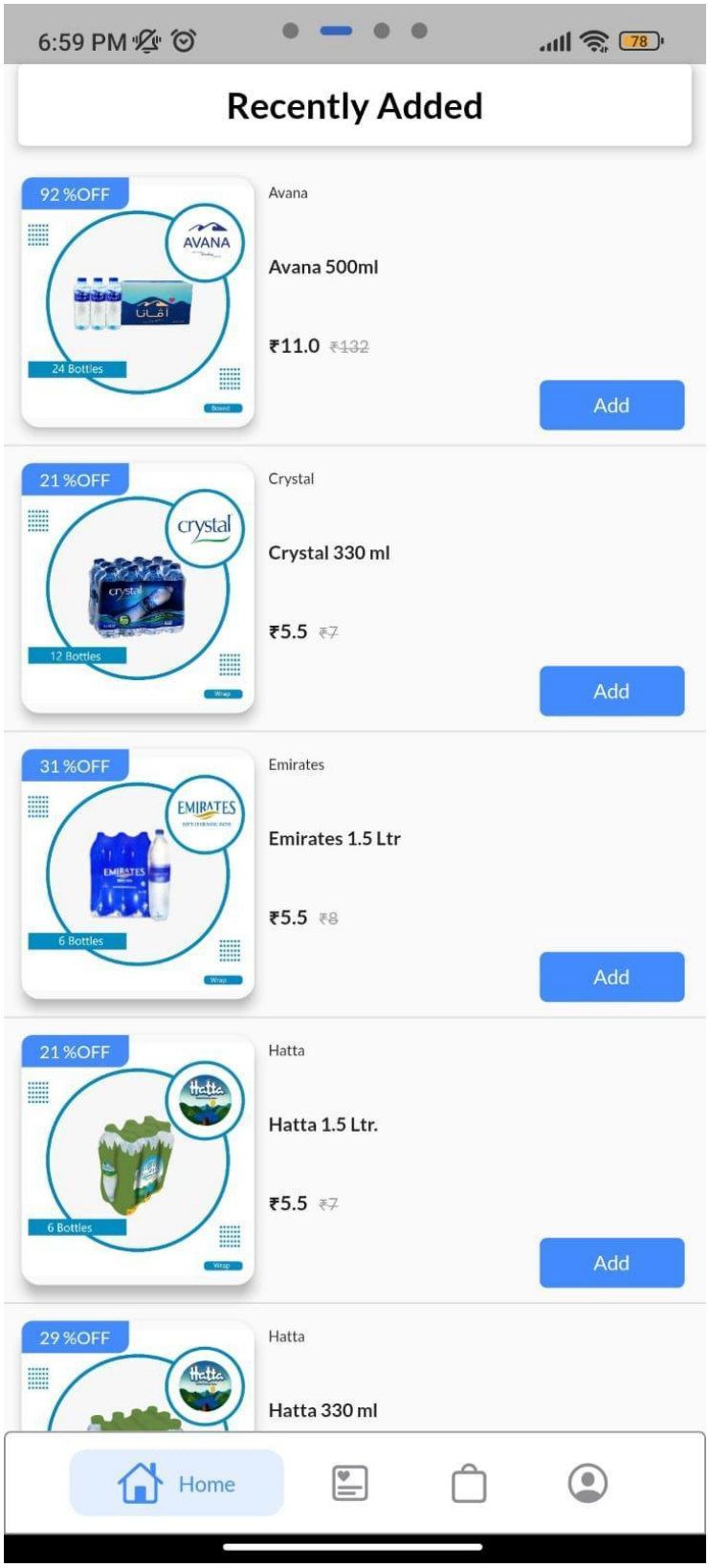


Figure 5.5.7 Recently Added Products page

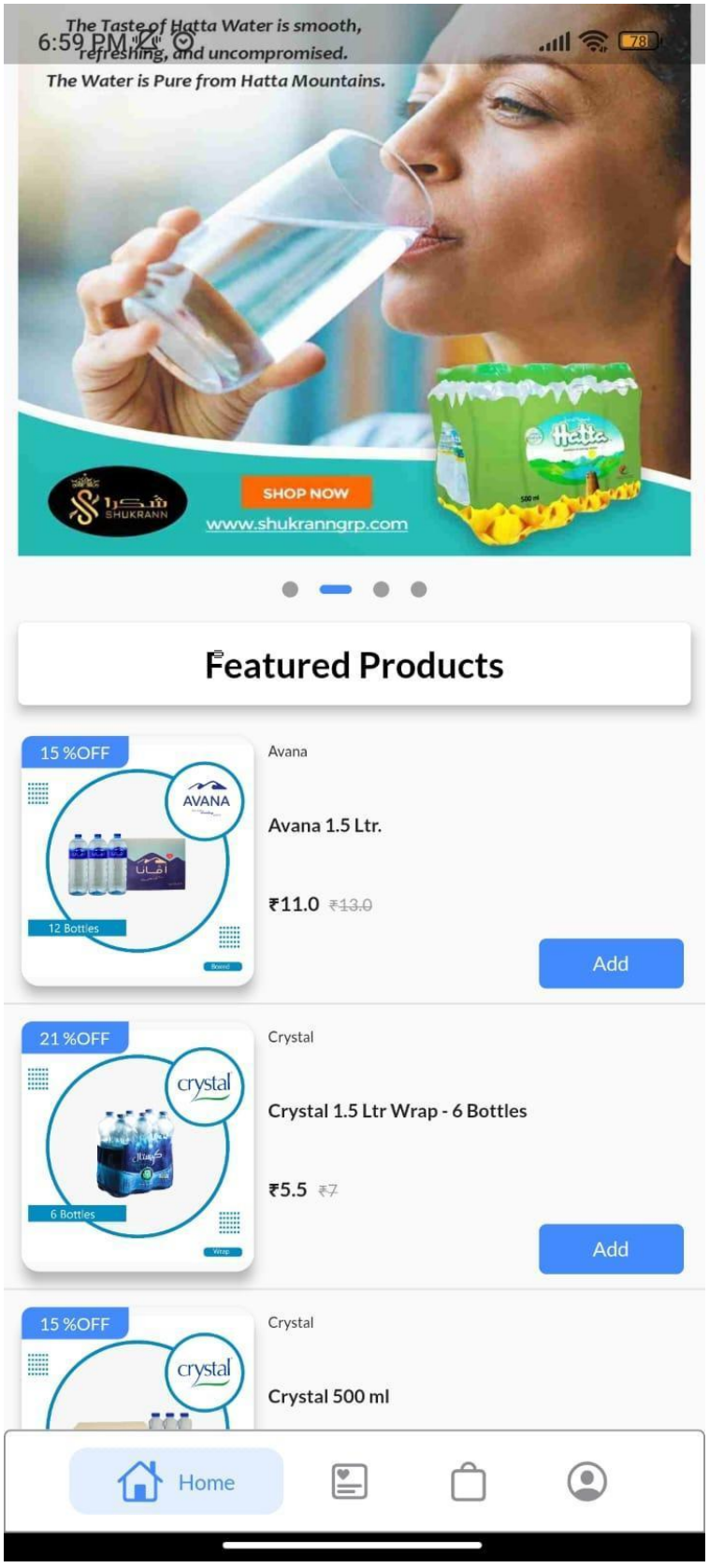

6:59 PM
The Taste of Hatta Water is smooth,
refreshing, and uncompromised.
The Water is Pure from Hatta Mountains.
SHUKRANN
SHOP NOW
www.shukranngrp.com
Featured Products
15 %OFF
AVANA
12 Bottles
Avana
Avana 1.5 Ltr.
₹11.0 ₹13.0
Add
21 %OFF
crystal
6 Bottles
Crystal
Crystal 1.5 Ltr Wrap - 6 Bottles
₹5.5 ₹7
Add
15 %OFF
crystal
Crystal
Crystal 500 ml
Home

Figure 5.5.8 Featured Products page

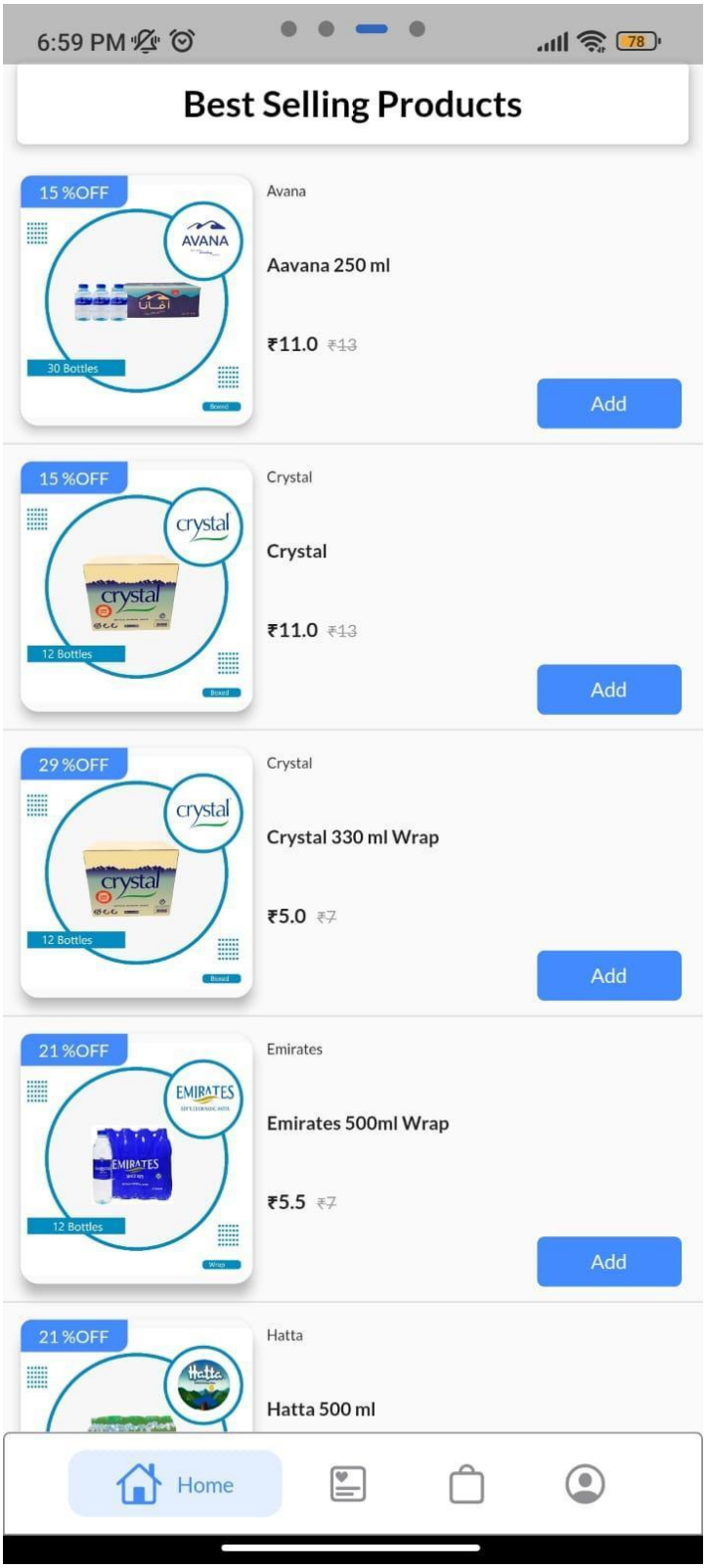


Figure 5.5.9 Best Selling Products page

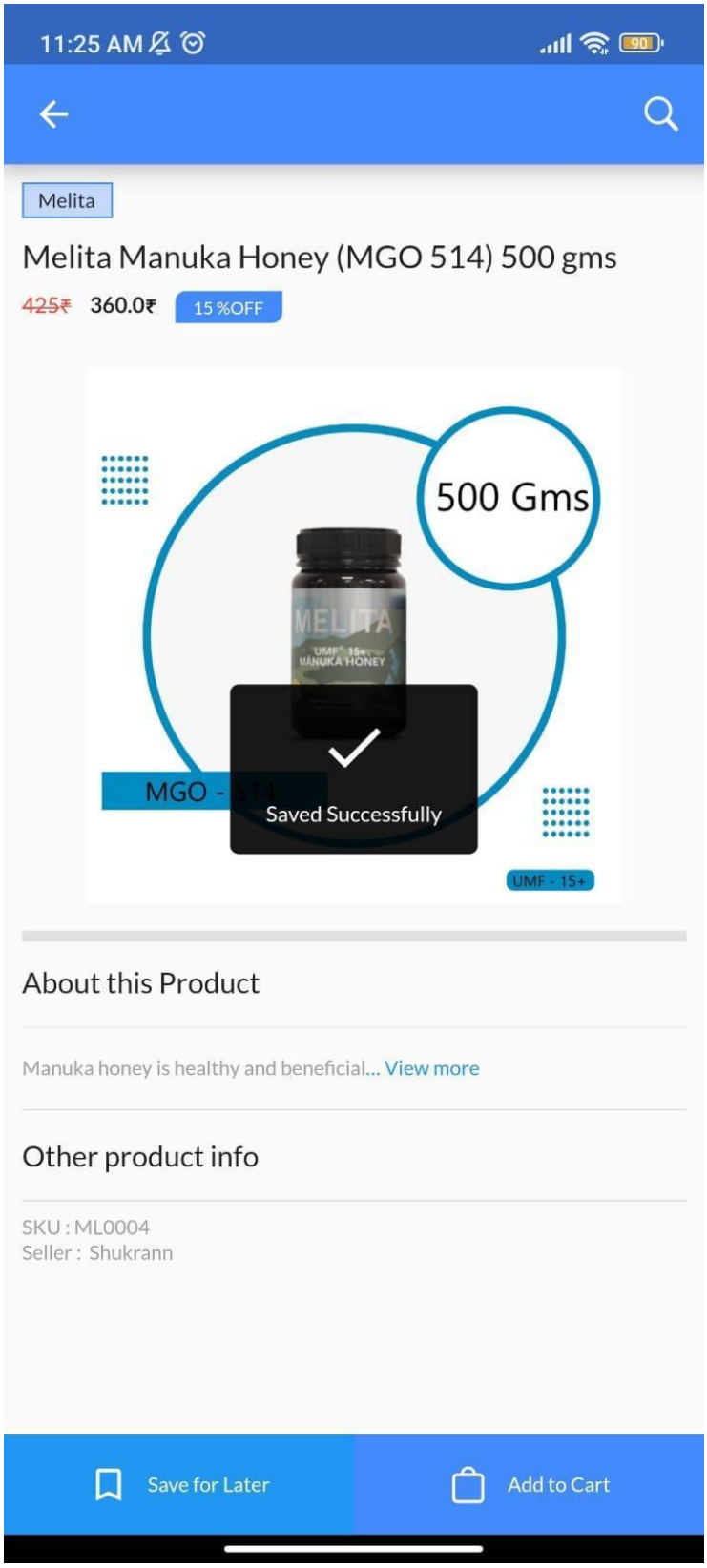

11:25 AM
Melita
Melita Manuka Honey (MGO 514) 500 gms
425₹ 360.0₹ 15 %OFF
500 Gms
MELITA
MANUKA HONEY
MGO - 514
Saved Successfully
UMF - 15+
About this Product
Manuka honey is healthy and beneficial... View more
Other product info
SKU : ML0004
Seller : Shukrann
Save for Later
Add to Cart

Figure 5.5.10 Add Products to Favourites page

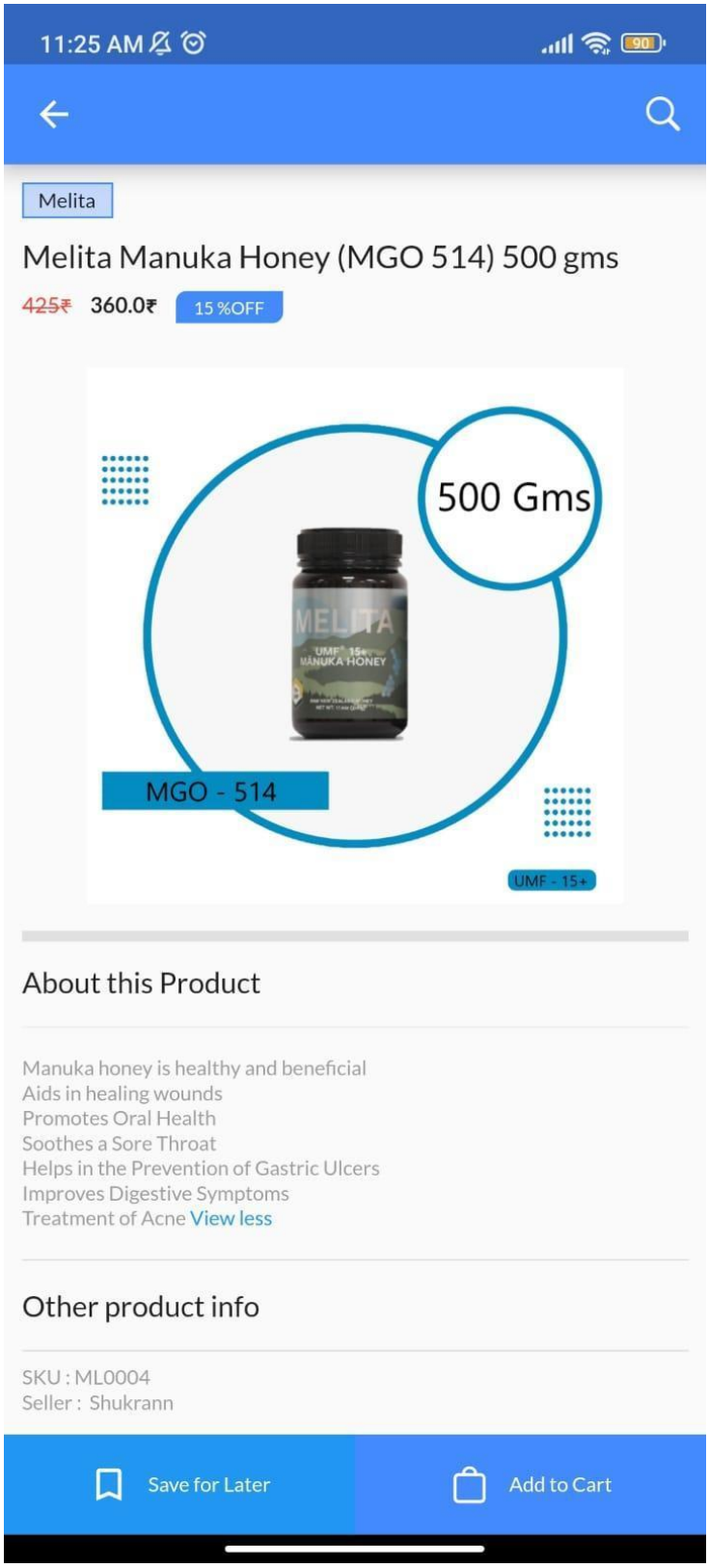


Figure 5.5.11 Product details page

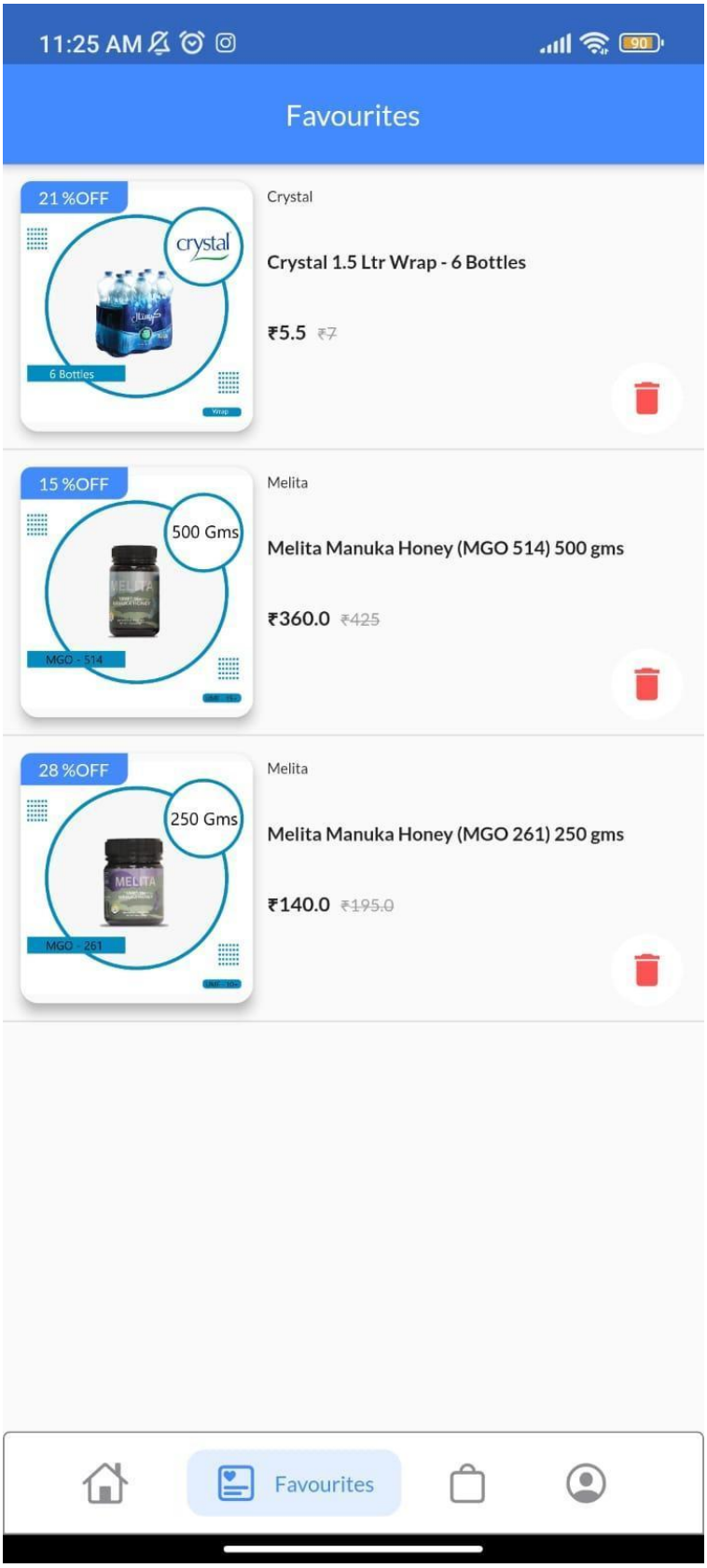
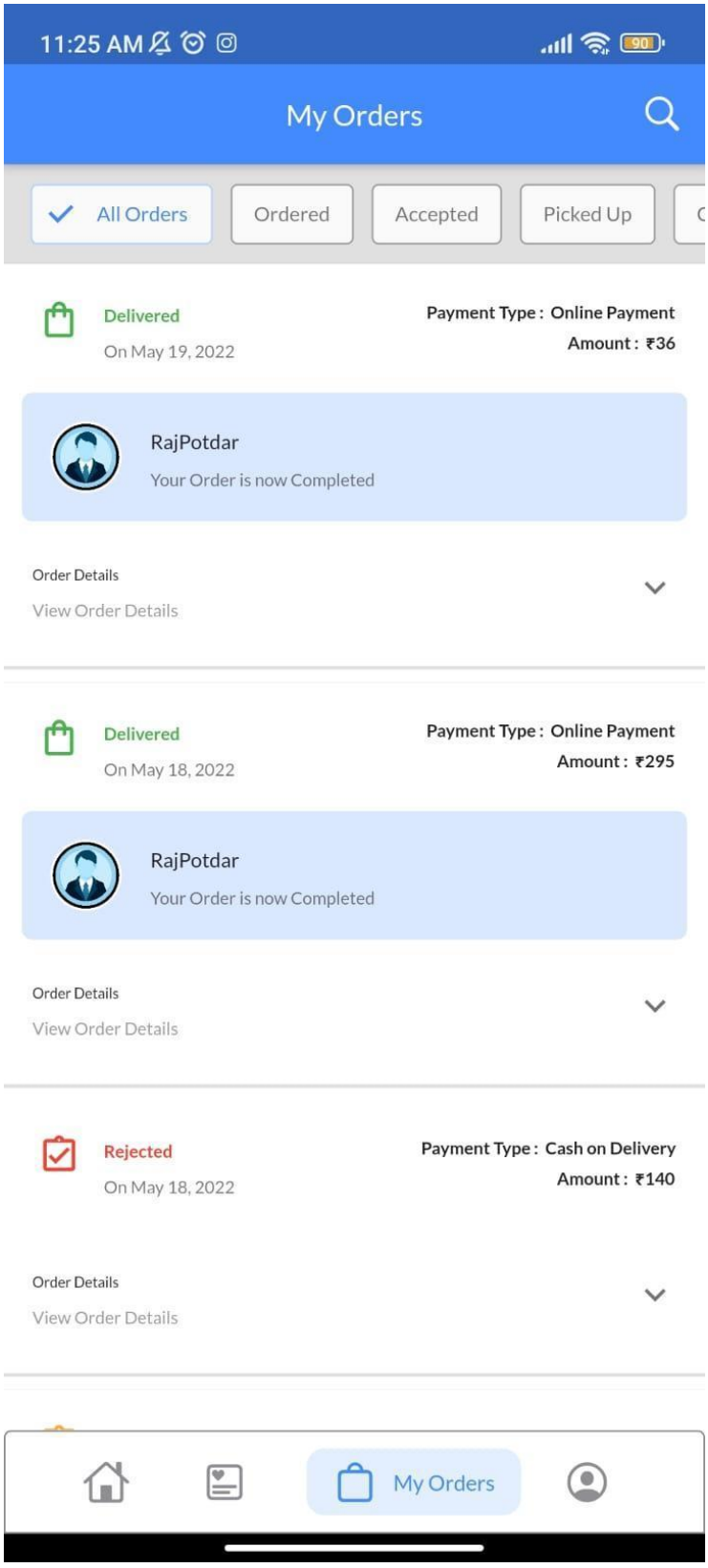

11:25 AM
Favourites
21 %OFF
crystal
6 Bottles
Crystal
Crystal 1.5 Ltr Wrap - 6 Bottles
₹5.5 ₹7
15 %OFF
500 Gms
MGO - 514
Melita
Melita Manuka Honey (MGO 514) 500 gms
₹360.0 ₹425
28 %OFF
250 Gms
MELITA
MGO - 261
Melita
Melita Manuka Honey (MGO 261) 250 gms
₹140.0 ₹195.0
Favourites

Figure 5.5.12 Favourites page



Fihure 5.5.13 My Orders page 1

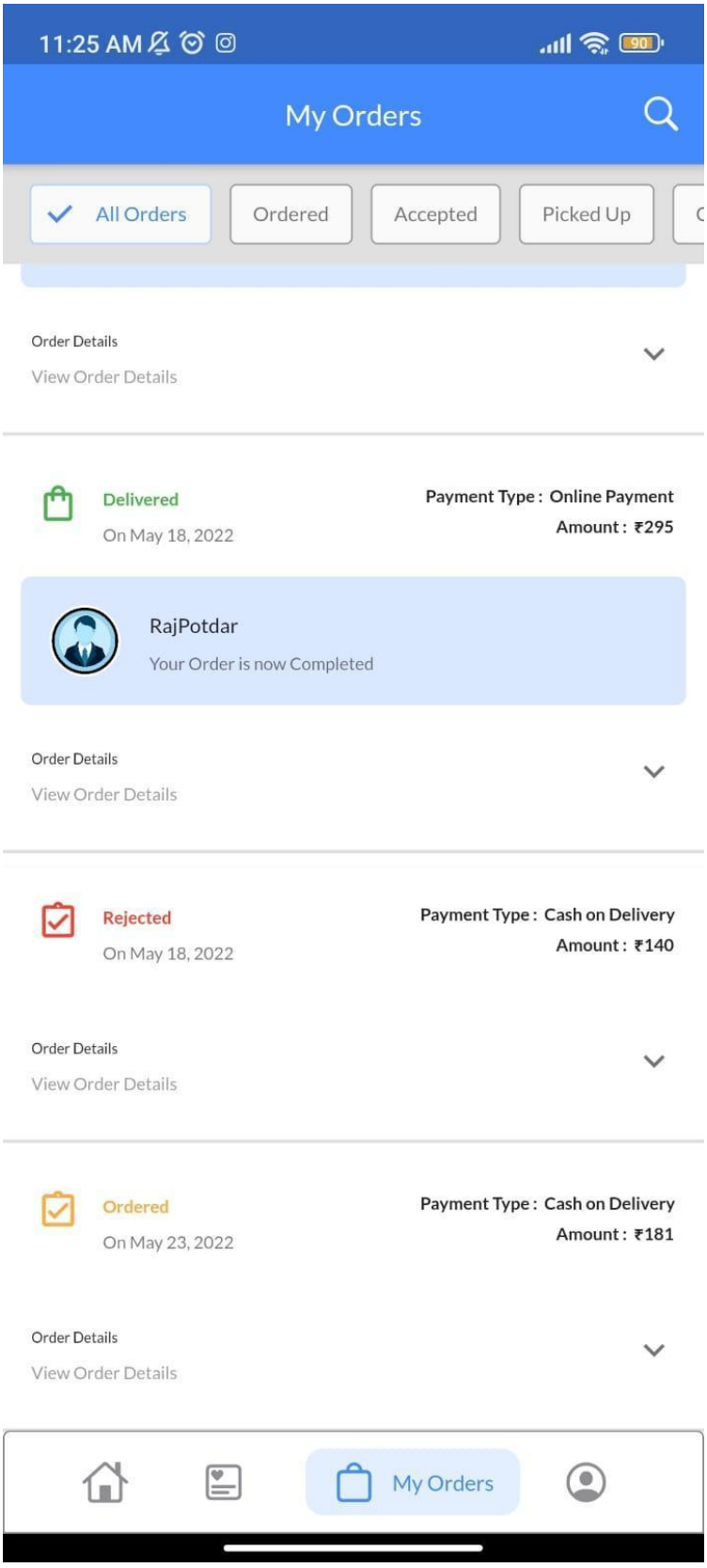


Figure 5.5.14 My Orders page 2

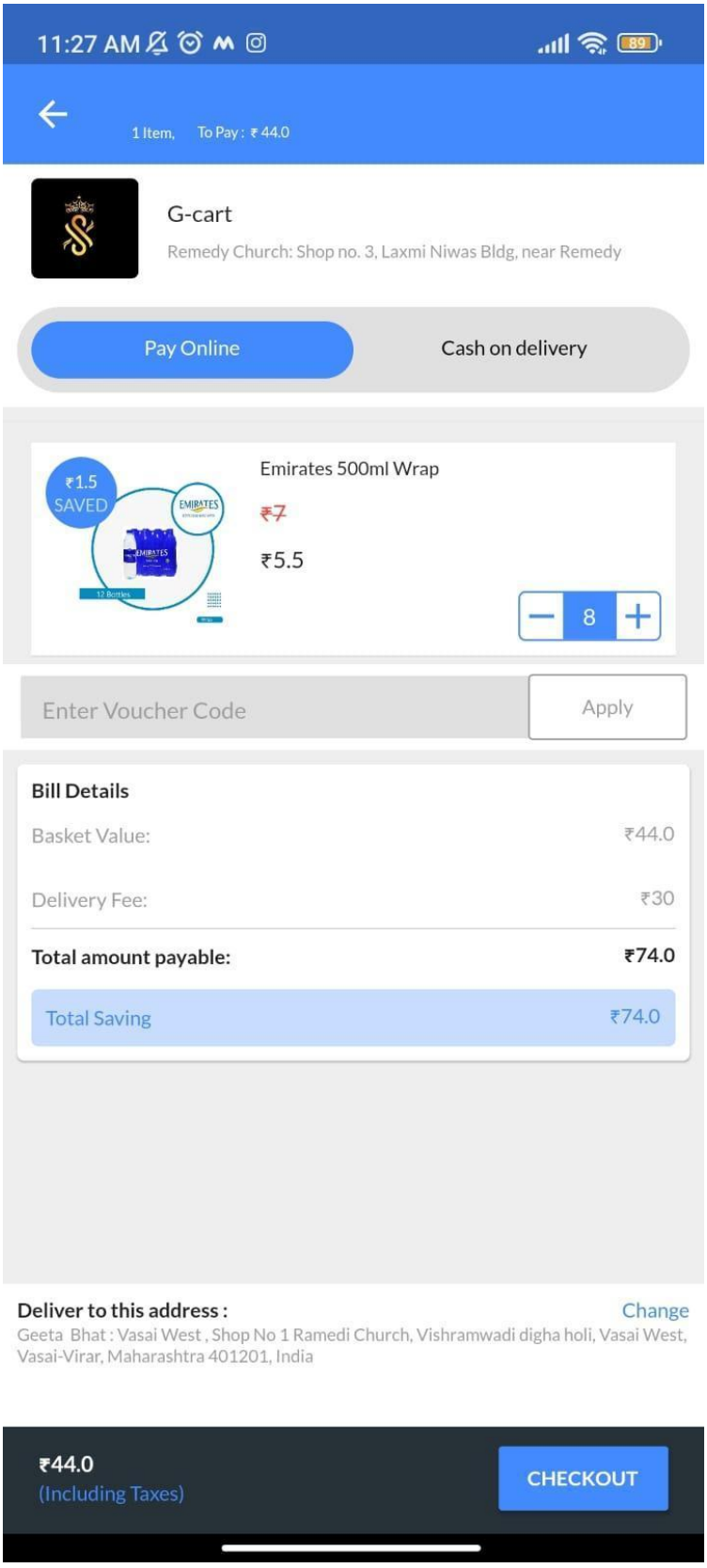


Figure 5.5.15 Cart Checkout page

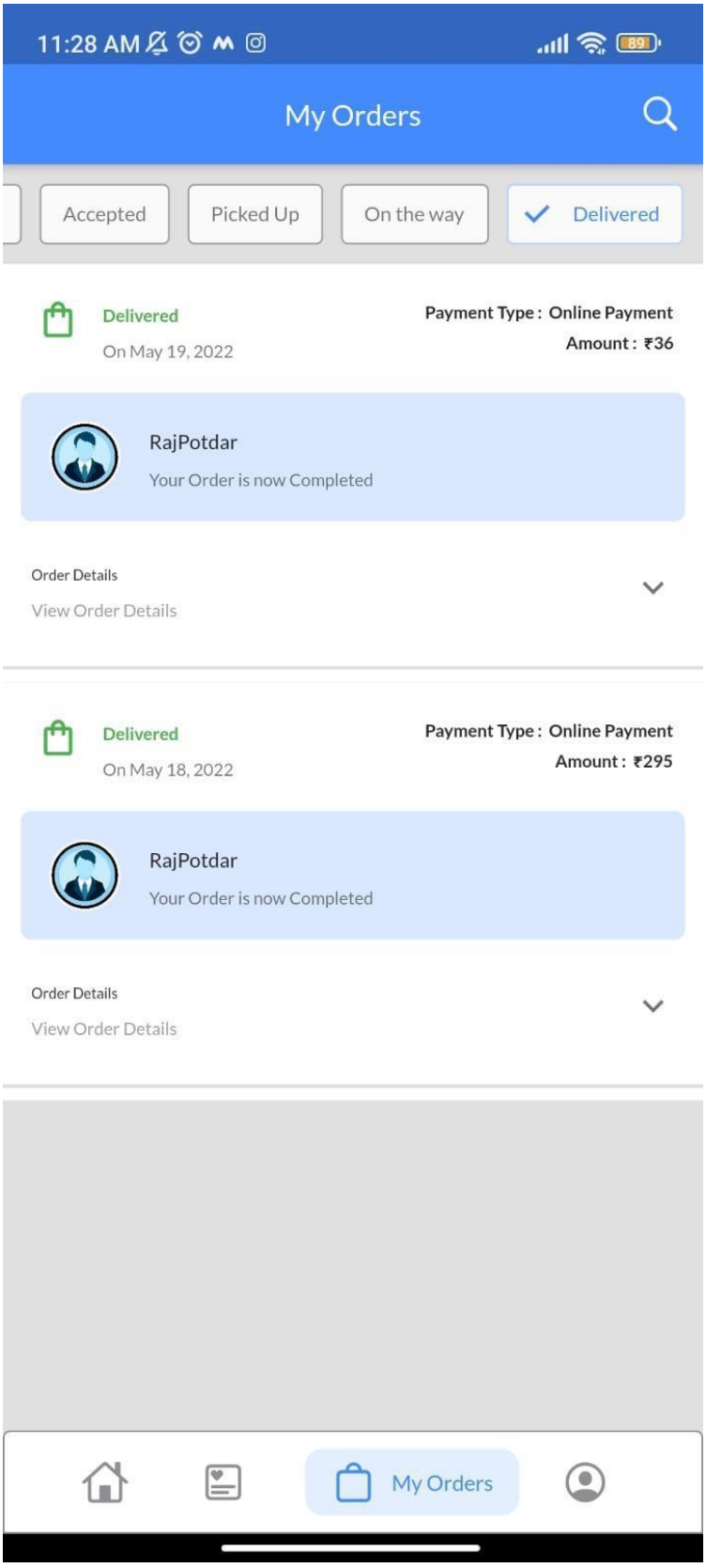


Figure 5.5.16 My Orders page 3

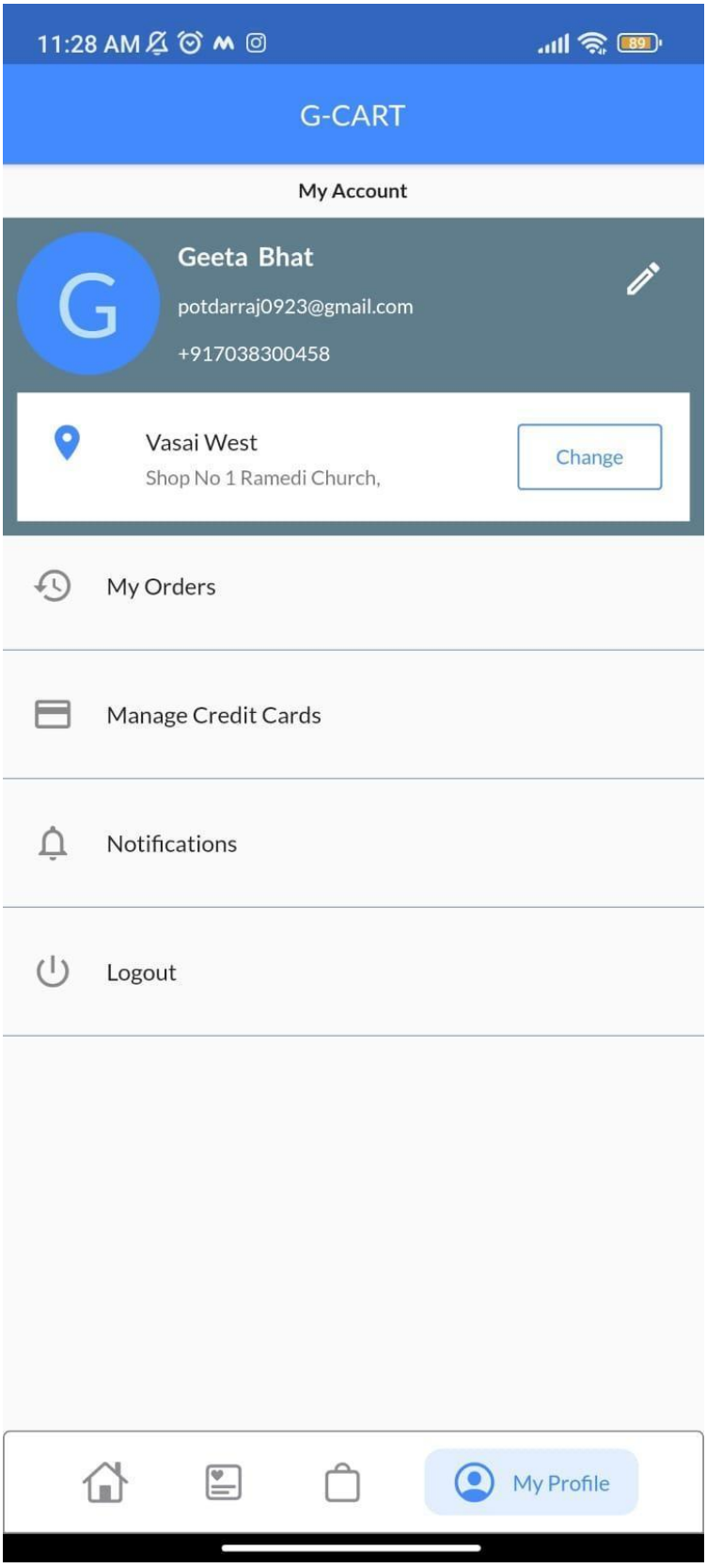


Figure 5.5.17 My Accounts page

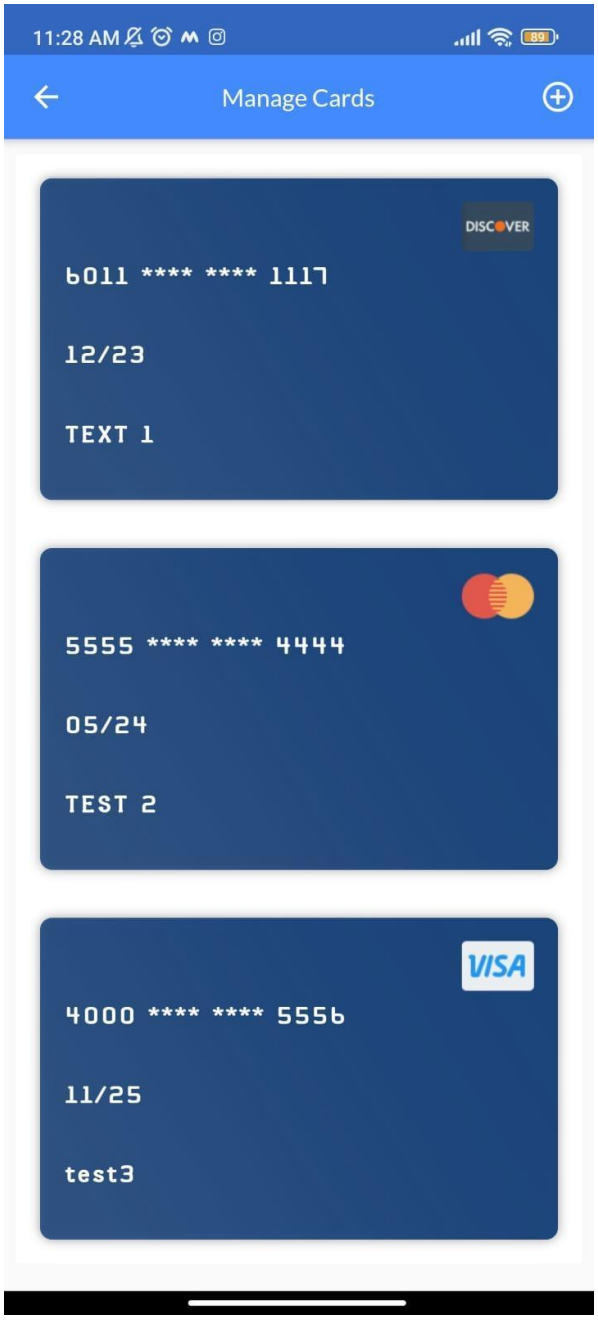


Figure 5.5.18 Manage Cards page

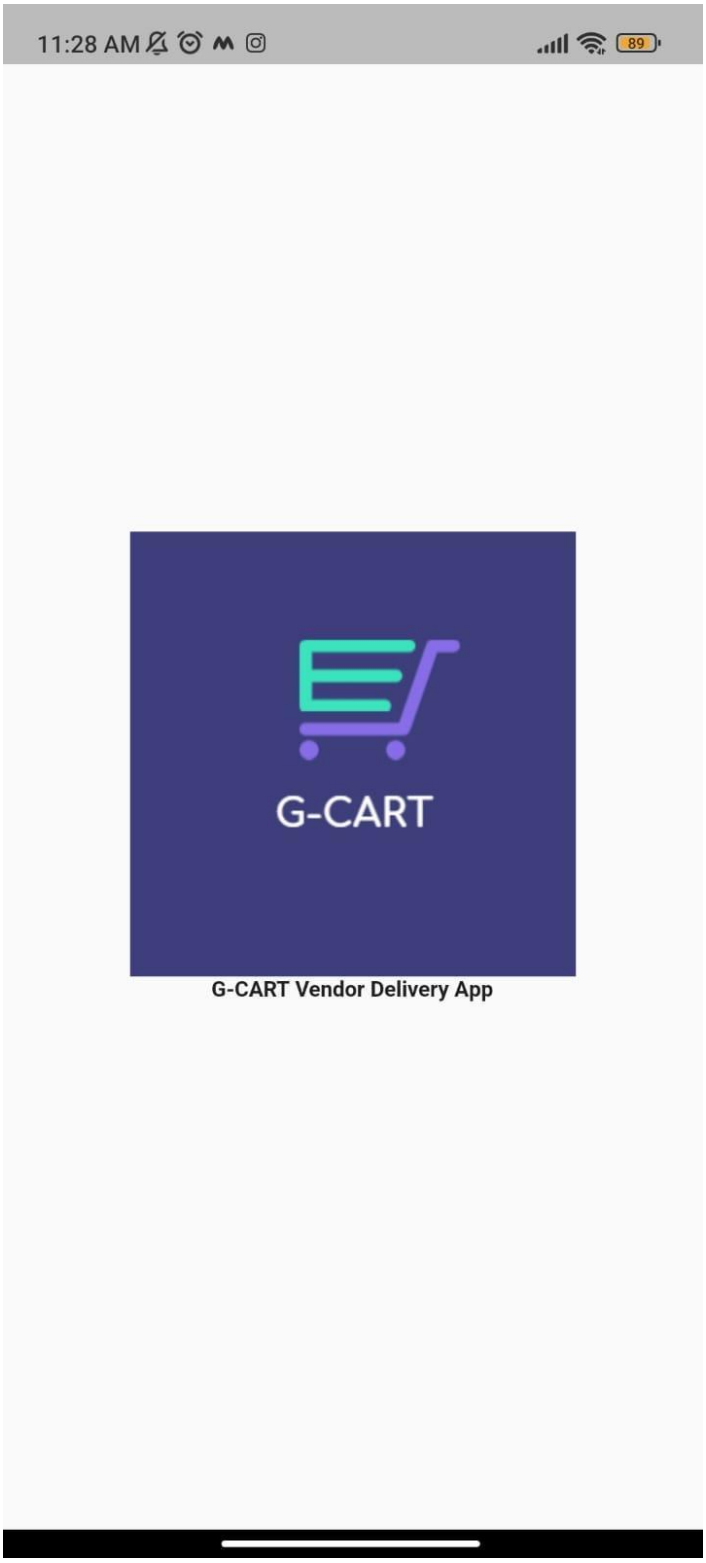


Figure 5.5.19 G-Cart Vendor Delivery App Logo

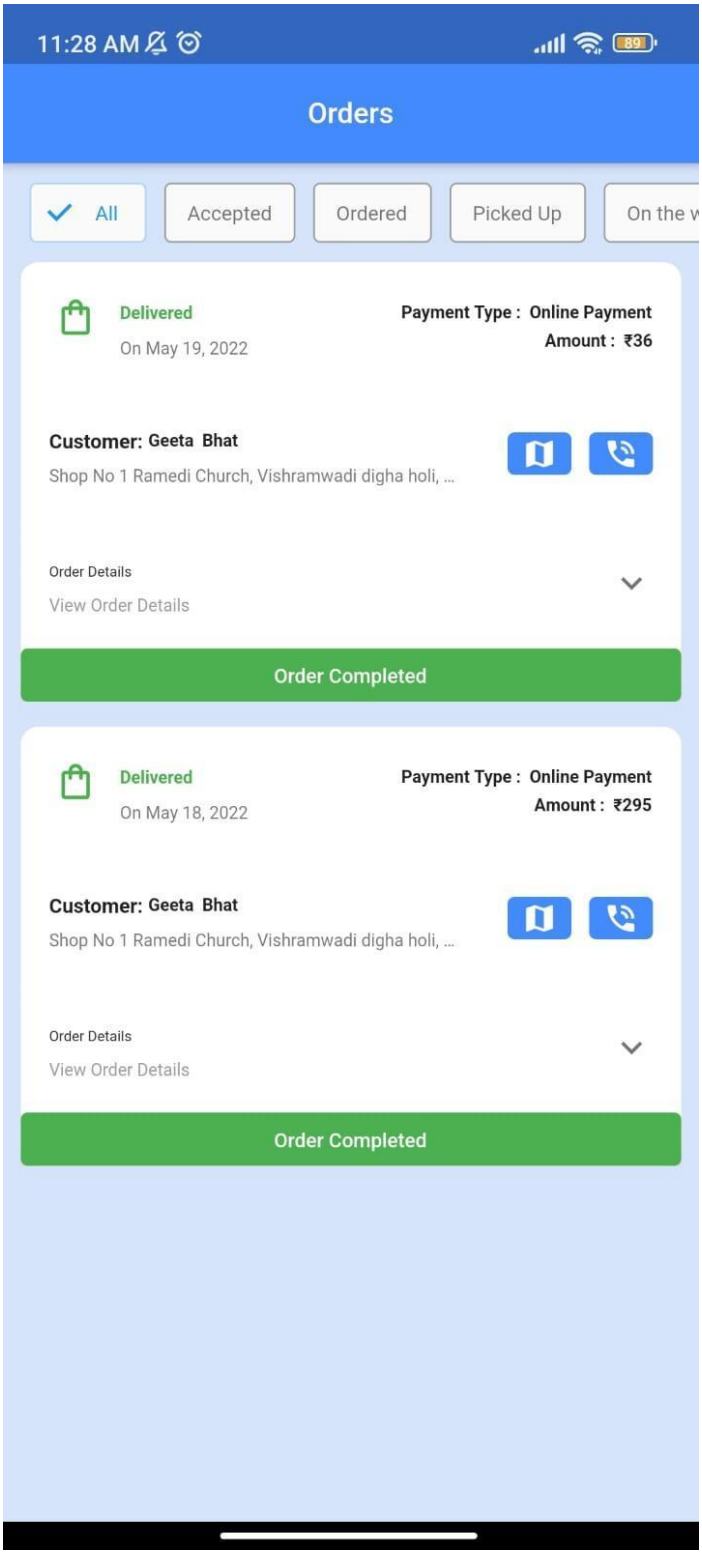


Figure 5.5.20 All Orders Page

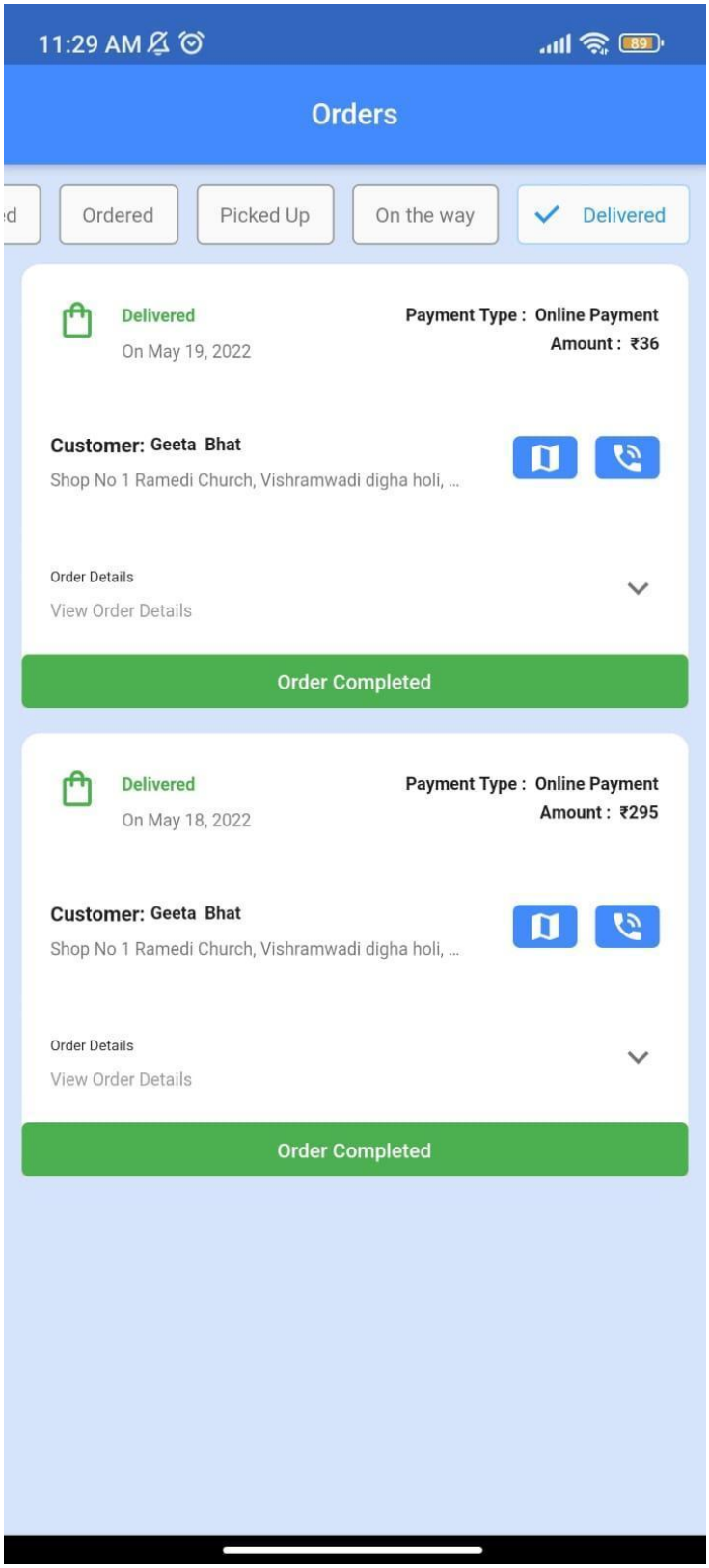


Figure 5.5.21 Delivered Orders Page

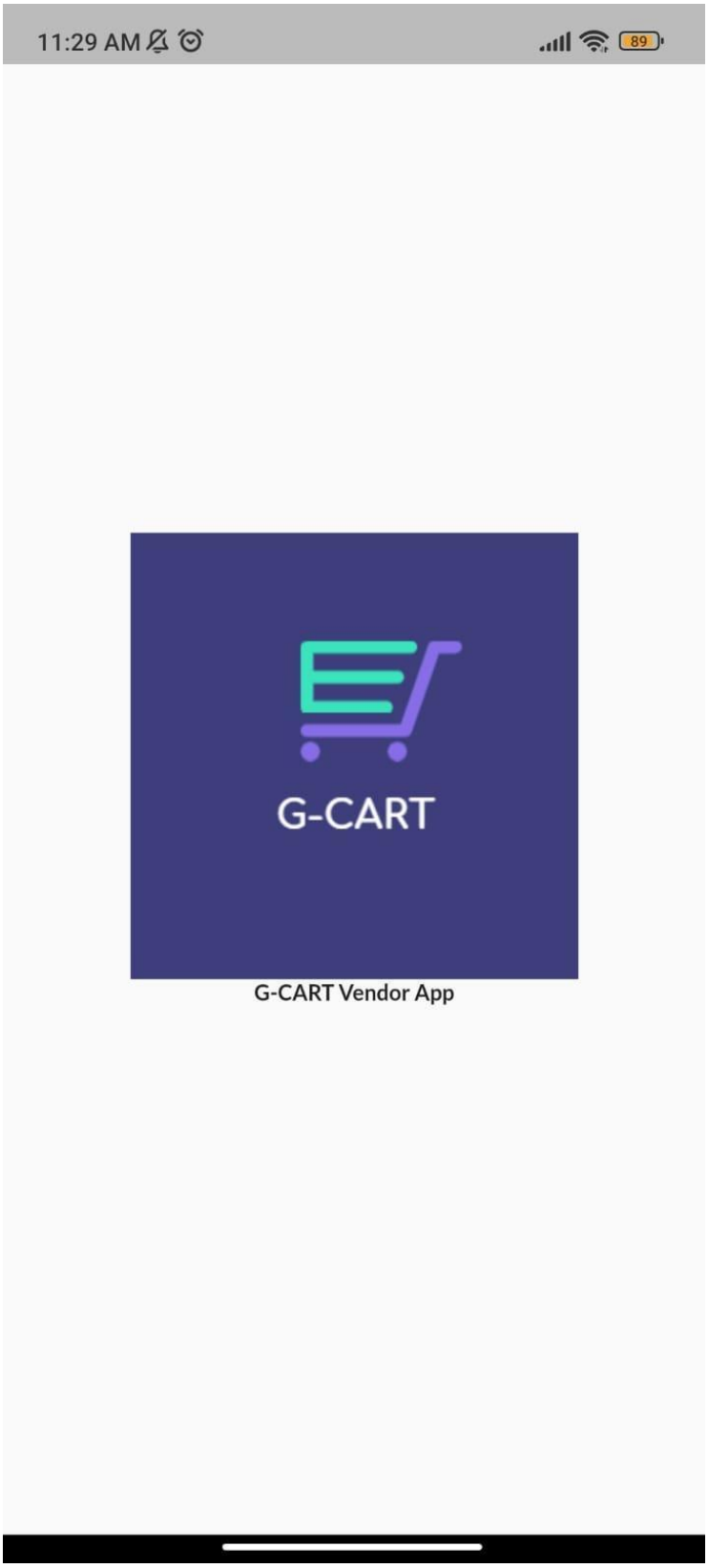


Figure 5.5.22 G-Cart Vendor App Logo

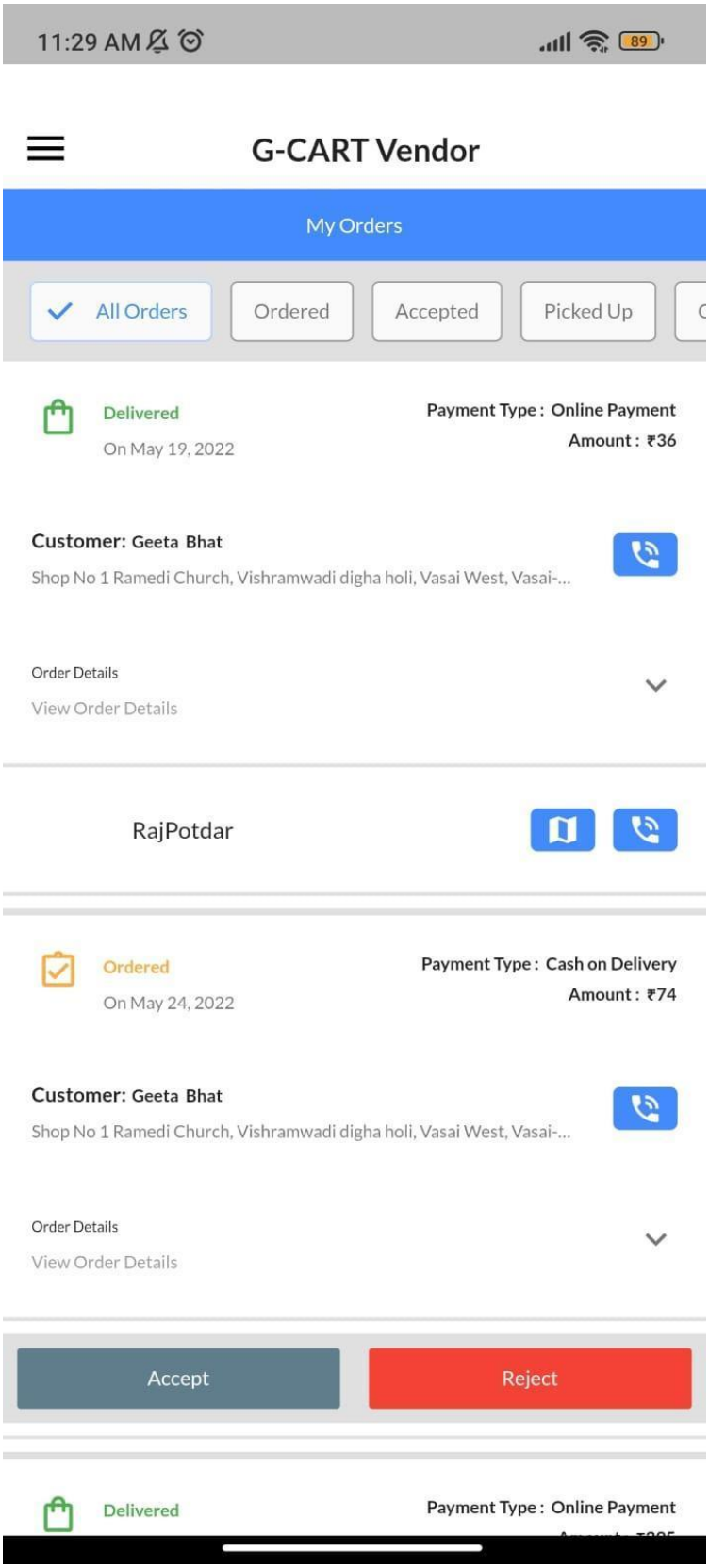
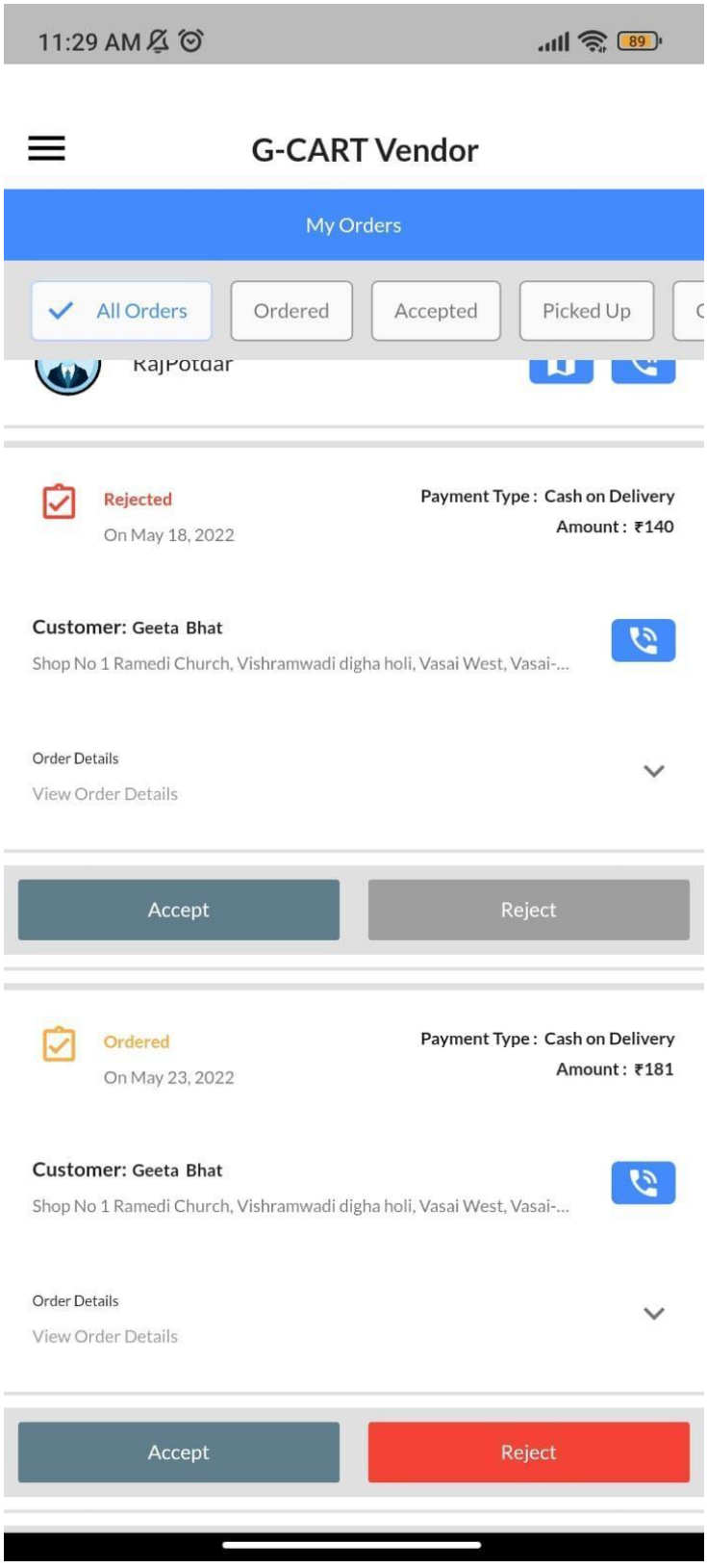


Figure 5.5.23 G-Cart Vendor App All orders Page-1



5.5.24 G-Cart Vendor App All Orders Page-2

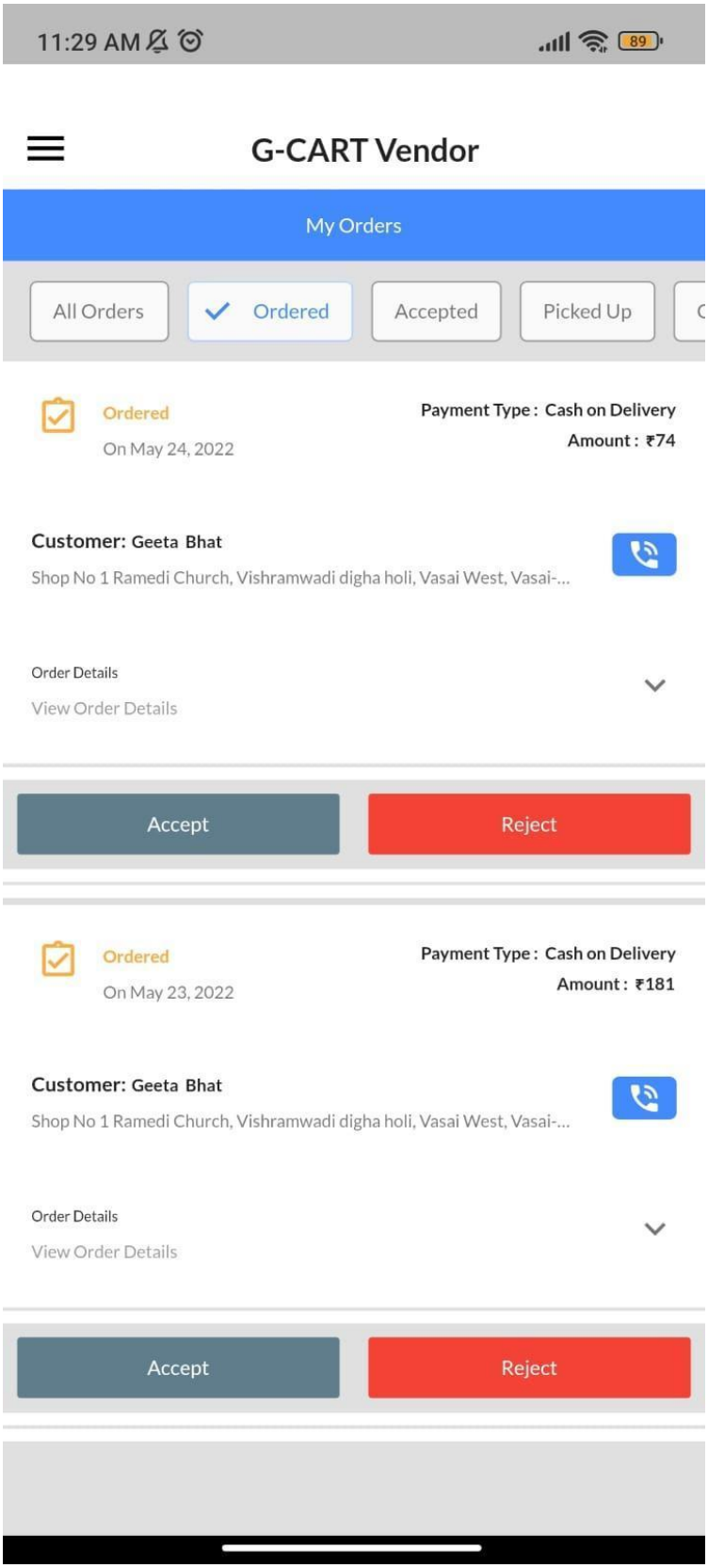


Figure 5.5.25 G-Cart Vendor App Ordered Orders Page

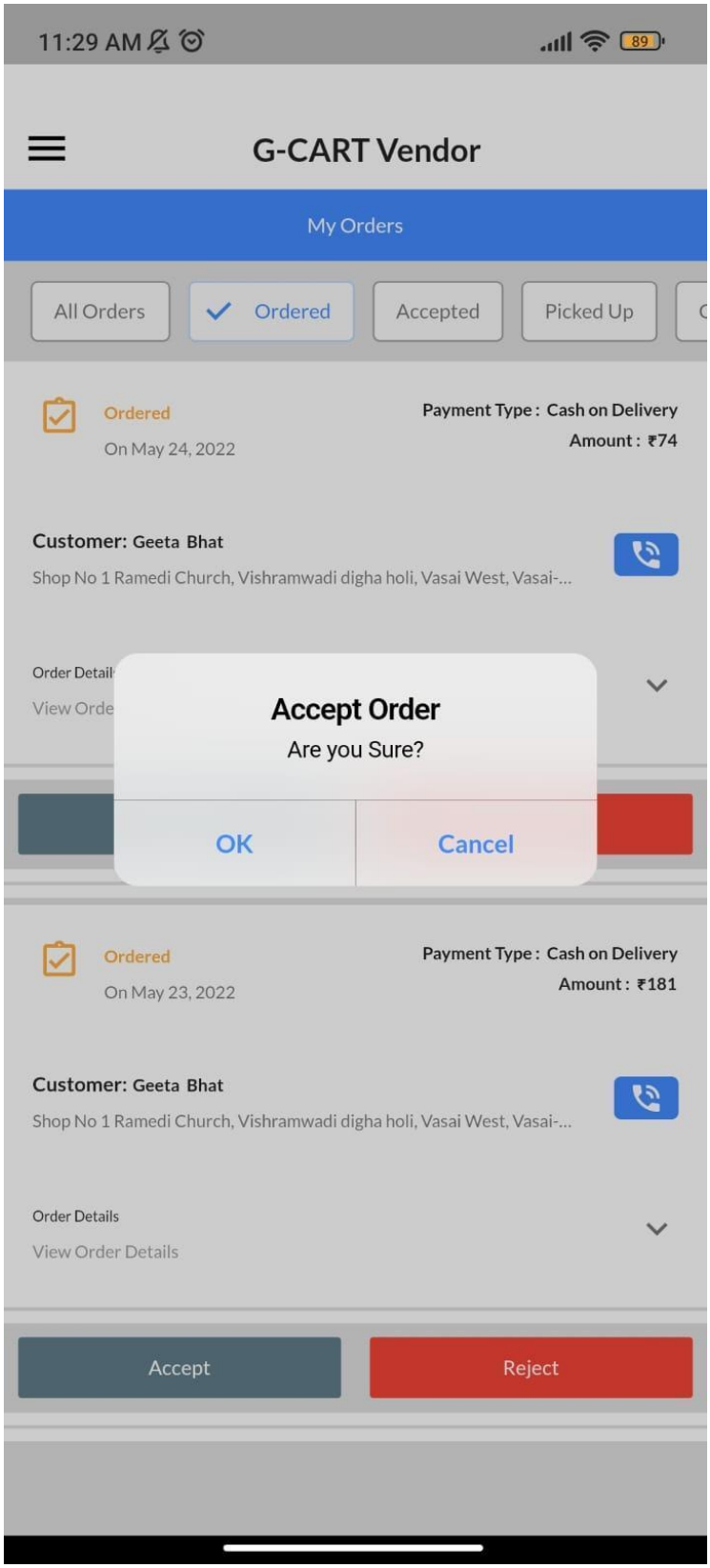


Figure 5.5.26 G-Cart Vendor App Accept Order Page

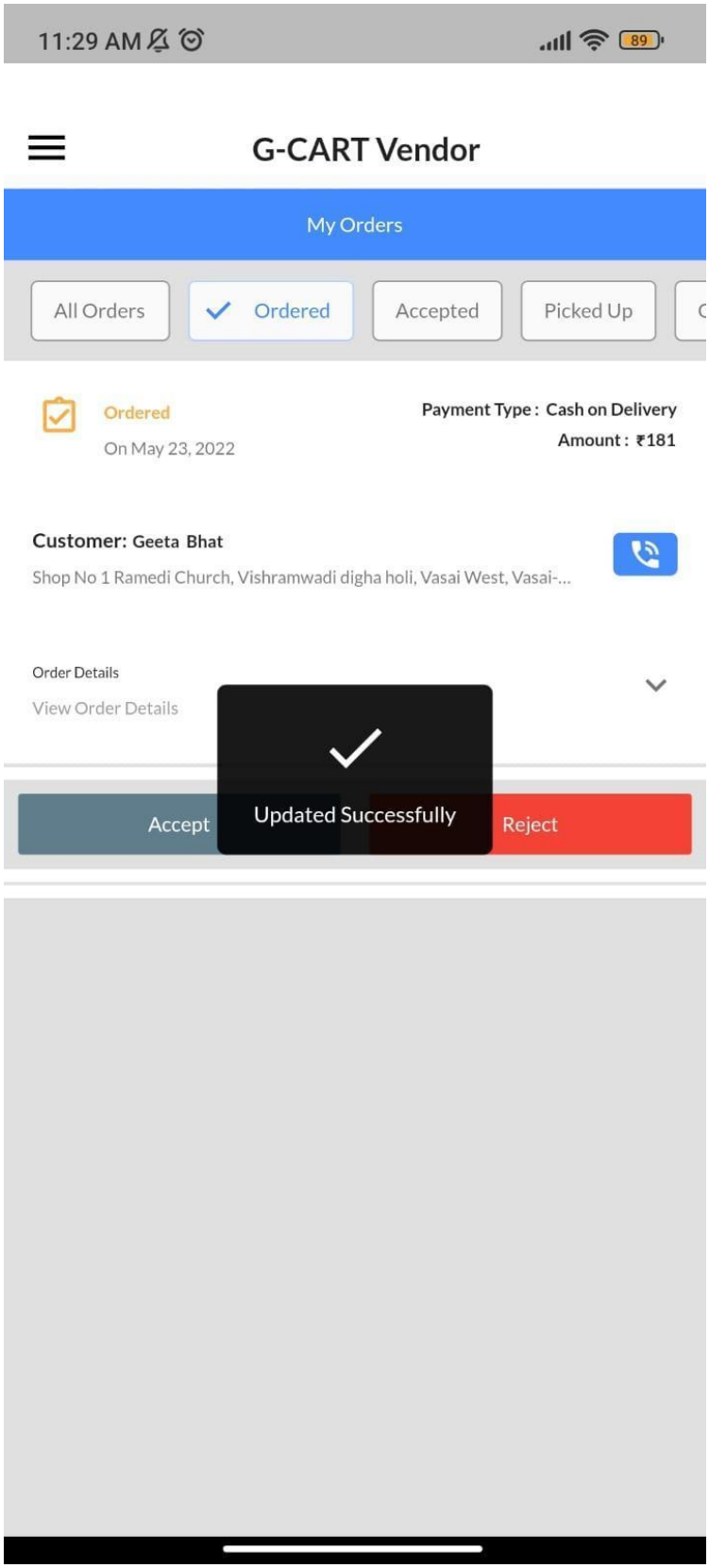


Figure 5.5.27 G-Cart Vendor App Orders Accepted Page 1

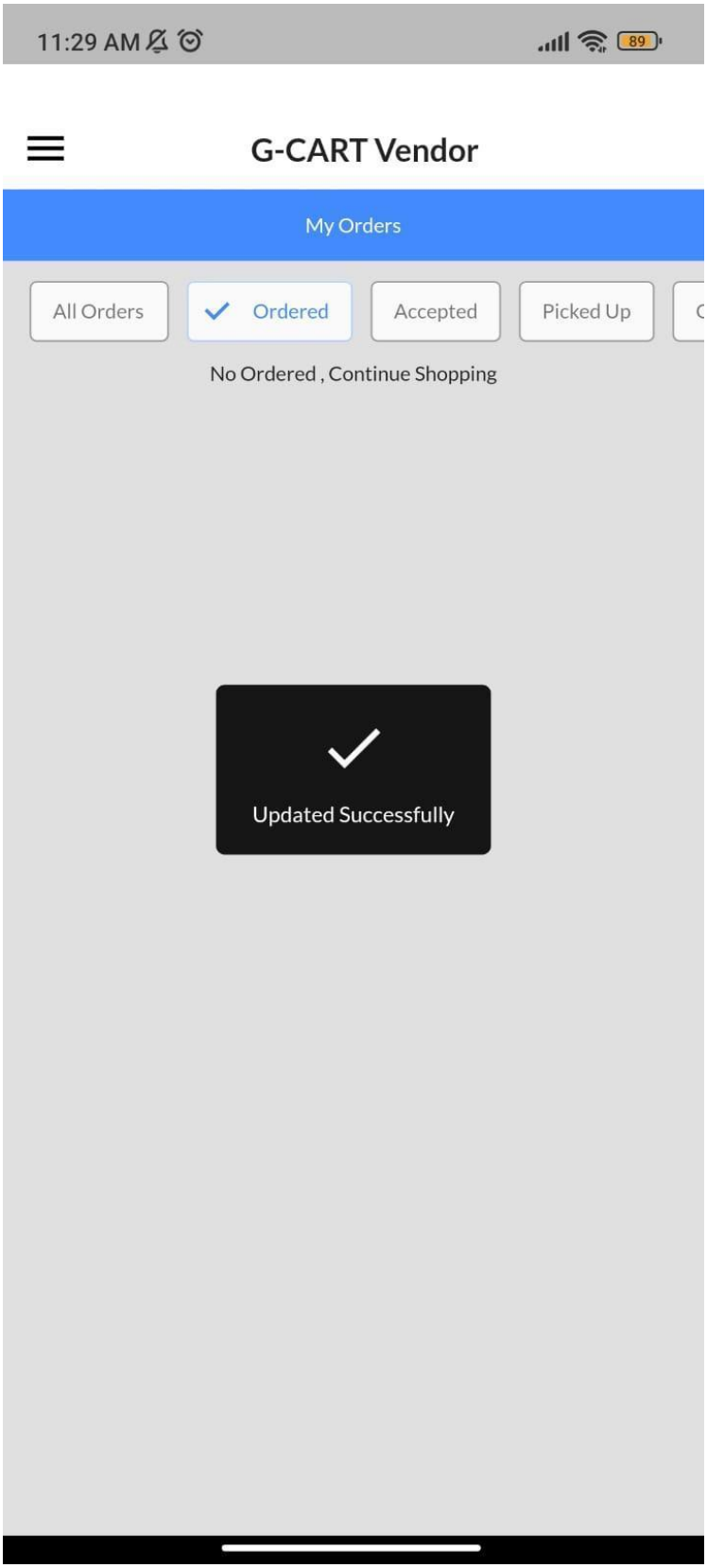


Figure 5.5.28 G-Cart Vendor App Order Accepted Page-2

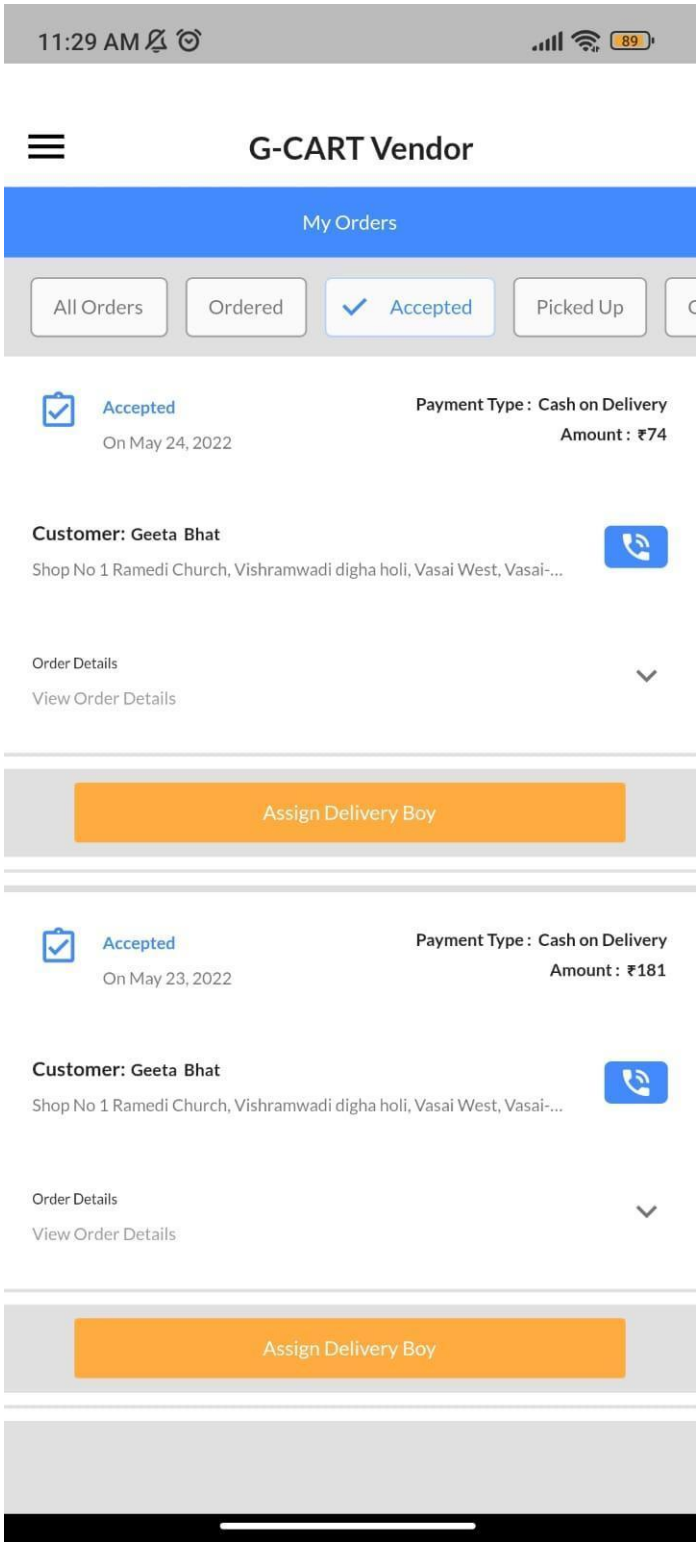


Figure 5.5.29 G-Cart Vendor App Accepted Orders Page

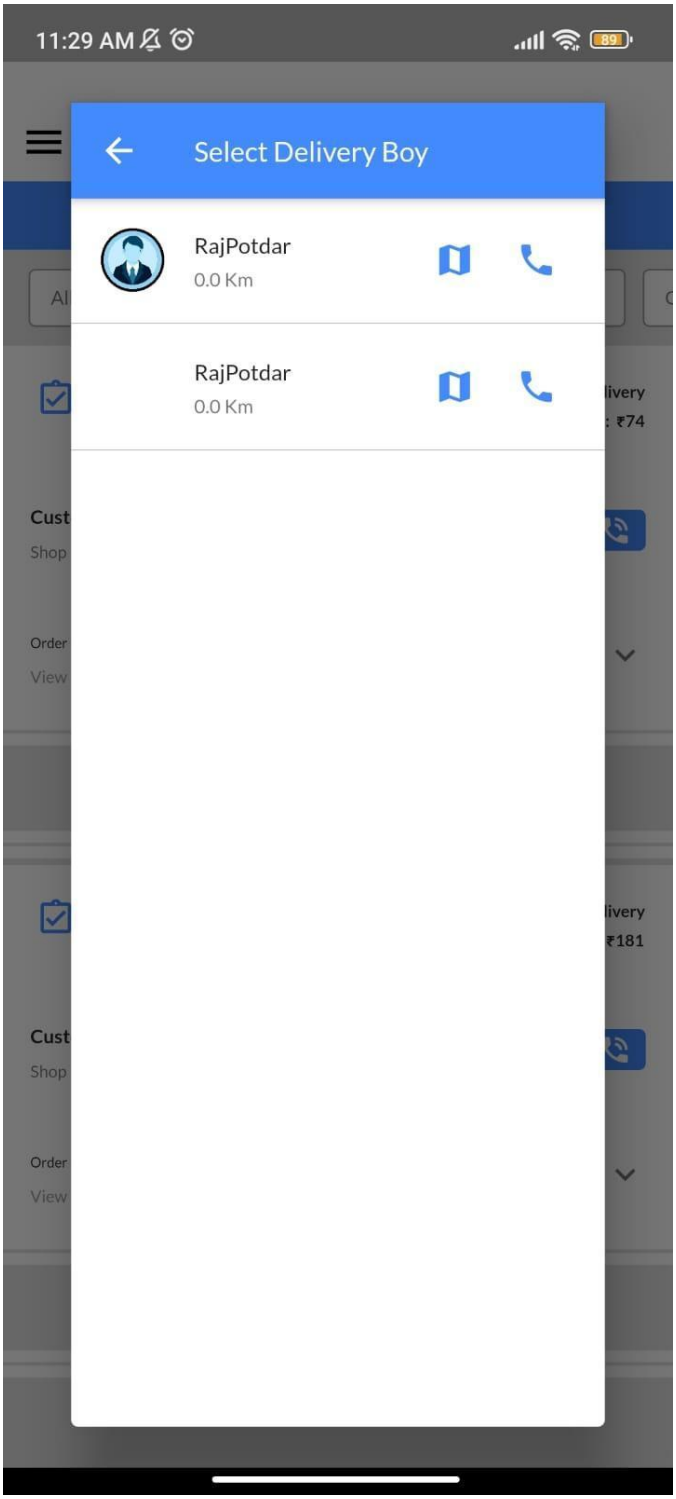


Figure 5.5.30 G-Cart Vendor App Delivery Boy Assigning Page

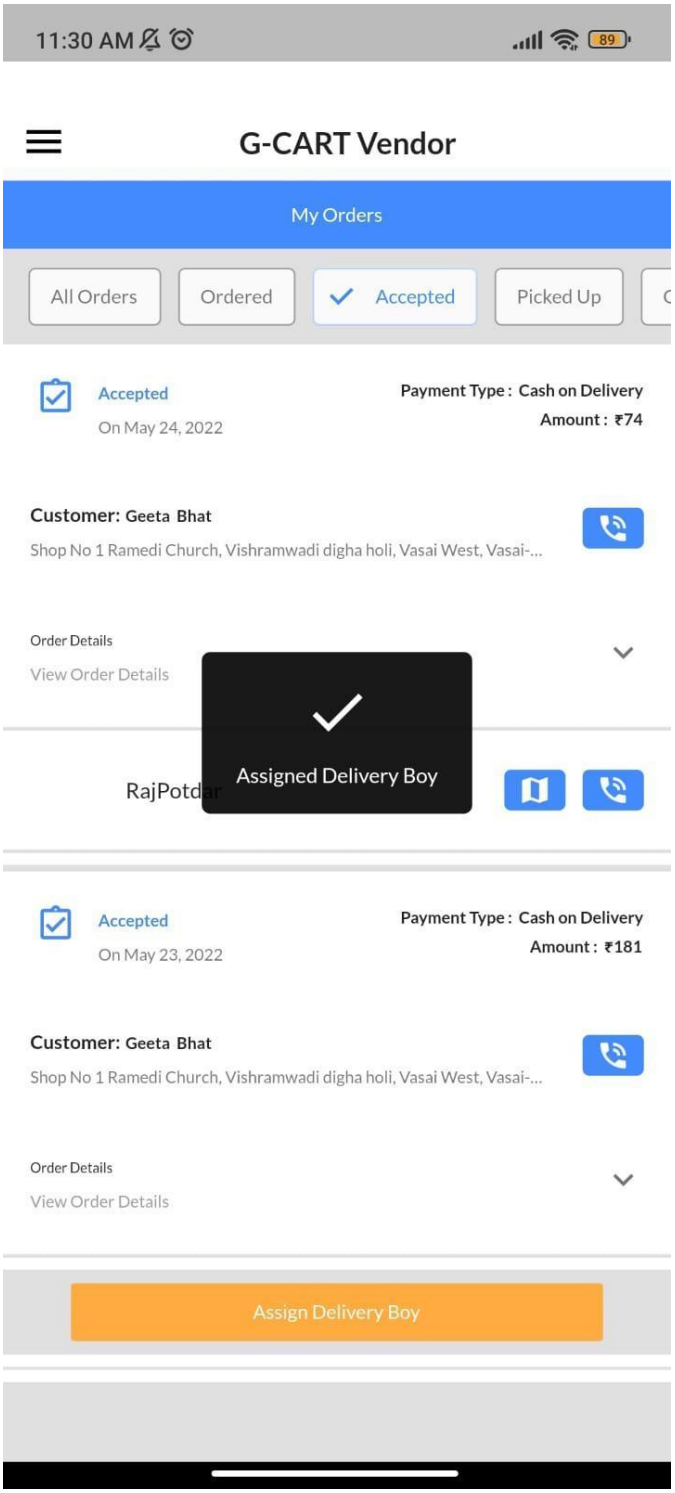


Figure 5.5.31 G-Cart Vendor App Delivery Boy Assigned Page

5.6 Test Cases & Report

5.6.1 Types of Testing

5.6.1.1 White Box Testing

White-box testing is a method of software testing that tests internal structures or workings of an application, as opposed to its functionality. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.

White box testing techniques analyses the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

Working process of white box testing:

- Input: Requirements, Functional specifications, design documents, source code.
- Processing: Performing risk analysis for guiding through the entire process.
- Proper test planning: Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- Output: Preparing the final report of the entire testing process.

Advantages:

- White box testing is very thorough as the entire code and structures are tested.
- It results in the optimization of code removing error and helps in removing extra lines of code.
- It can start at an earlier stage as it doesn't require any interface as in case of black box testing.
- Easy to automate.

5.6.1.2 Black Box Testing

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance.

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

There are many types of Black Box Testing but the following are the prominent ones –

Functional testing – This black box testing type is related to the functional requirements of a system; it is done by software testers.

Non-functional testing – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

Regression testing – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black box testing largely depends on the type of black box testing you are doing.

For Functional/ Regression Tests you can use – QTP, Selenium

For Non-Functional Tests, you can use – LoadRunner, Jmeter

Black box testing has its own life cycle called Software Testing Life Cycle (STLC) and it is relative to every stage of Software Development Life Cycle of Software Engineering.

Requirement – This is the initial stage of STLC and in this stage, a requirement is gathered. Software testers also take part in this stage.

Test Planning & Analysis – Testing Types applicable to the project are determined. A Test Plan is created which determines possible project risks and their mitigation.

Design – In this stage Test cases/scripts are created on the basis of software requirement documents

Test Execution– In this stage Test Cases prepared are executed. Bugs if any are fixed and re-tested.

5.6.2 Test Report

Table 5.1 Test Report

Project Name:	G-CART, G-CART VENDOR, G-CART DELIVERY
Module Name:	Register, Online Ordering, Delivery & Management
Test Title:	To verify order and delivery
Test Designed By:	Geeta Bhat
Test Executed By:	Geeta Bhat

Table 5.2 Test Plan

Test ID	Test Name	Test Description	Expected Output	Actual Output	Result
GC1	Mobile Registration	Register with valid mobile number	Registration Successful	Registration Successful	PASS
GC2		Register with invalid mobile number	Registration Unsuccessful	Registration Unsuccessful	PASS
GC3	Shop by category	Category results are filtered properly	Results displayed according to filter	Results displayed according to filter	PASS

GC4	Approve Order	Accept or Reject User Order	Order is accepted or rejected	Order is accepted or rejected	PASS
GC5	Assign Delivery Associate	Assigning delivery associate to the accepted order	Delivery associate assigned	Delivery associate assigned	PASS
GC6	Pickup Order	Pickup Order from Shop	Update the status	Update the status	PASS
GC7	Deliver Order	Deliver Order to Customer	Update the status	Update the status	PASS
GC8	Receive Payment	Take payment from Customer	Update the status	Update the status	PASS

6. System Maintenance & Evaluation

6.1 Maintenance

System maintenance is an ongoing activity, which covers a wide variety of activities, including removing program and design errors, updating documentation and test data and updating user support. For the purpose of convenience, maintenance may be categorized into three classes, namely:

i) Corrective Maintenance: This type of maintenance implies removing errors in a program, which might have crept in the system due to faulty design or wrong assumptions. Thus, corrective maintenance, processing or performance failures are repaired.

ii) Adaptive Maintenance: In adaptive maintenance, program functions are changed to enable the information system to satisfy the information needs of the user. This type of maintenance may become necessary because of organizational changes which may include:

a) Change in the organizational procedures,

b) Change in organizational objectives, goals, policies, etc.

c) Change in forms,

d) Change in information needs of managers.

e) Change in system controls and security needs, etc.

iii) Perfective Maintenance: Perfective maintenance means adding new programs or modifying the existing programs to enhance the performance of the information system. This type of maintenance undertaken to respond to user's additional needs which may be due to the changes within or outside of the organization. Outside changes are primarily environmental changes, which may in the absence of system maintenance, render the information system ineffective and inefficient. These environmental changes include:

a) Changes in governmental policies, laws, etc.,

b) Economic and competitive conditions, and

c) New technology.

6.2 Evaluation

System Evaluation is an assessment process of the complete system performance that is compared with its requirements and determines the possibilities for growth and improvement.

Depending on the requirements for system evaluation, a system can be evaluated at the application level, functional level or the technological level. The application determines the systems´ boundaries, the degree of abstraction and the suitable methods.

Evaluation provides a systematic method to study a program, practice, intervention, or initiative to understand how well it achieves its goals. Evaluations help determine what works well and what could be improved in a program or initiative.

7. Future Enhancements

The future scope of this Multi-Vendor Flutter Ecommerce Application is that it will be added with more products and shops in the menu. The payment gateway will be added for online payments through bank transfer via mobile banking and net banking, card payments and UPI payments.

Some features which can be added for enhancements are:

- Adapt Advanced Recommendation for enhancing user experience.
- Improve and enhance the GUI.
- Improve the Tracing options for Delivery App.
- Implement more secure or advanced payment gateways for supporting all kinds of payment.
- Adding a secure domain.

8. Limitations

● The recommendation system designed for the system is not compatible with the firebase database as of now.

● The developed versions might need more enhanced database that is compatible with machine learning algorithms.

9. CONCLUSION

In conclusion, The Multi-Vendor Flutter Ecommerce Application created is user friendly for the users. The user has to login through his mobile number and he/she will get authenticated and get access to the application. The multi-vendor Flutter Ecommerce Application is very time saving in such a way that the user doesn't need to go to multiple shops to buy essential healthy food products. It helps to search products in an easy manner. The application has a logout option too.

Nowadays, big brands are selling lifestyle products which target specific customers. However, these products are available at local shops made in various small manufacturing local companies that are of high quality and affordable to all customers. Giving a platform for small shopkeepers will help these small and large local manufacturers and shopkeepers to sell ethically sourced and healthy food and drinks directly to customers. The app aims to build a huge network for all healthy and organic raw material for food as well as nutritious packed food materials.

10. Bibliography

Following is the list of websites which were used while researching for this project and developing it:

- https://firebase.flutter.dev/docs/auth/start
- https://pub.dev/
- https://docs.flutter.dev/