"Design & Implementation Of Android Based Fruits Ordering Application : FruitBox"

A Project Report of Major Project Phase-II Submitted to

Rajiv Gandhi Proudyogiki Vishwavidhalaya, Bhopal

Towards Partial Fulfillment of the Degree of

Bachelor of Technology in Computer Engineering

Guided By:

Mr. Surendra Gupta
Associate Professor
Department of Computer Engg.

Ms. Priyanka Kokate
Assistant Professor
Department of Computer Engg.

Submitted By:

0801CS201022 - Ashi Bagedi
0801CS201036 - Gurdeep Singh
0801CS201091 - Simran Rathore
0801CS201098 - Umang Tiwari
0801CS201101 - Vikas Dahiya

Department Of Computer Engineering
Shri G.S. Institute Of Technology And Science, Indore(M.P.)
2023-2024

SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE, INDORE(M.P.)

Session 2023-24

RECOMMENDATION

The project report for Major Project Phase-II entitled **"Design & Implementation Of Android Based Fruits Ordering Application: FruitBox"** submitted by: **0801CS201022 - Ashi Bagedi**, **0801CS201036 - Gurdeep Singh, 0801CS201091 - Simran Rathore, 0801CS201098 - Umang Tiwari, 0801CS201101 - Vikas Dahiya** students of Bachelor of Technology IV year in the session 2023-2024, towards partial fulfillment of the degree of **Bachelor of Technology in Computer Science and Engineering** of Rajiv Gandhi Proudyogiki VishwaVidhyalaya, Bhopal is a satisfactory account of their work towards CO44498: Major Project (Phase-II) and is recommended for the award of degree.

Project Guide:

Mr. Surendra Gupta
Associate Professor
Department of Computer Engg.

Ms. Priyanka Kokate
Assistant Professor
Department of Computer Engg.

Dr. Vandan Tiwari
Head of Department
Department of Computer Engg.

SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE, INDORE(M.P.)

(A Govt. Aided Autonomous Institute, Affiliated to RGPV, Bhopal)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

The project report for CO44498: Major Project Phase-II entitled **"Design & Implementation Of Android Based Fruits Ordering Application: FruitBox"** submitted by: **0801CS201022 - Ashi Bagedi**, **0801CS201036 - Gurdeep Singh, 0801CS201091 - Simran Rathore, 0801CS201098 - Umang Tiwari, 0801CS201101 - Vikas Dahiya** students of Bachelor of Technology IV year in the session 2023-2024, towards partial fulfillment of the degree of Bachelor of Technology in Computer Engineering of Rajiv Gandhi Proudyogiki VishwaVidhyalaya, Bhopal is a satisfactory account of their work.

Internal Examiner **External Examiner**

Date

SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE, INDORE(M.P.)

DECLARATION

We **0801CS201022 - Ashi Bagedi , 0801CS201036 - Gurdeep Singh, 0801CS201091 - Simran Rathore, 0801CS201098 - Umang Tiwari, 0801CS201101 - Vikas Dahiya**, here by declare that the work presented in the Bachelor of Technology Major project report has been carried out by us. We further declare that the work submitted for the award of the degree doesn't contain any part of the work which has been submitted for the award of any degree either in this university or any other university without proper citation.

0801CS201022 - Ashi Bagedi
0801CS201036 - Gurdeep Singh
0801CS201091 - Simran Rathore
0801CS201098 - Umang Tiwari
0801CS201101 - Vikas Dahiya

ACKNOWLEDGEMENT

We express our profound sense of gratitude to our project guides Ms. Priyanka Kokate and Mr. Surendra Gupta who had advised us to do project work entitled "Design & Implementation Of Android Based Fruits Ordering Application : FruitBox". Their continuous support and motivation always made us deliver our best. Having such guidance has always been an amazing experience which is a valuable gift for an engineer to progress in his/her life.

We are also grateful to Dr. Vandan Tiwari, Head, Department of Computer Engineering and Dr. Rakesh Saxena, Director, S.G.S.I.T.S., Indore for providing us with numerous facilities and academic environment during the study.

We sincerely wish to express our gratefulness to all the members of staff of the Department of Computer Engineering, S.G.S.I.T.S. Indore, who have extended their cooperation at all times and have contributed in their own way in developing the project.

The successful completion of a project is not an individual effort. It is an outcome of the cumulative number of people, each having their importance to the objective. We express love and respect towards our parents and all family members who are our strength in everything we do. We are thankful to them for their constant support and motivation.

With a blend of gratitude, pleasure, and great satisfaction we convey our indebtedness to all those who have directly or indirectly contributed to the successful completion of our project work.

0801CS201022 - Ashi Bagedi _____

0801CS201036 - Gurdeep Singh _____

0801CS201091 - Simran Rathore _____

0801CS201098 - Umang Tiwari _____

0801CS201101 - Vikas Dahiya _____

ABSTRACT

In the bustling landscape of online shopping, the call for user-friendly mobile apps has reached an all-time high. This realization ignited our journey towards creating "FruitBox: Design & Implementation of Android-Based Fruits Ordering Application." We recognized a significant gap in the market—a lack of a seamless platform dedicated solely to ordering fresh fruits. Despite the proliferation of online grocery shopping platforms, none provided the specialized experience required for fruit procurement. The absence of tailored features for selecting and purchasing fresh produce left consumers navigating through cluttered interfaces, often resulting in frustration and uncertainty about the quality and freshness of the fruits they were ordering. Motivated by this clear void in the market, we seized the opportunity to innovate. Fueled by the desire to streamline fruit procurement and enhance the overall shopping experience, we committed ourselves to developing a tailored solution to bridge this gap effectively. With FruitBox, our aim is to provide users with a user-friendly and intuitive platform specifically designed for ordering fresh fruits, thus addressing the shortcomings of existing solutions and revolutionizing the way consumers access high-quality produce online.

While apps like Ondoor, Big Basket, and Blinkit have transformed the landscape of online grocery shopping, they often present challenges to users. These platforms offer a vast array of products, leading to overwhelming browsing experiences and making it difficult for users to find specific items, particularly fresh fruits. Moreover, they may lack comprehensive information about the fruits available for purchase, leaving consumers uncertain about their quality and freshness. Additionally, limitations such as limited delivery options and occasional stock unavailability further hinder the user experience.

That's where FruitBox comes in. We wanted to make ordering fruits super easy, so we made an app just for that. With FruitBox, you'll find a carefully picked selection of fresh fruits, complete with all the details about where they're from and how good they are. That way, you can shop with confidence, knowing exactly what you're getting.

Combining Flutter for the mobile platform and React with Material-UI for the admin panel, "FruitBox" offers intuitive interfaces. Firebase powers real-time database management, user authentication, and hosting, while Stripe integration ensures secure transactions. Git and GitHub streamline collaborative development, and Firestore handles dynamic data storage. These technologies redefine online food platforms, simplifying fruit accessibility and setting new standards for innovation.

CONTENTS

Recommendation
Certificate
Declaration
Acknowledgements
Abstract
1 Introduction
1.1 Preamble..... 1
1.2 Need of the Project..... 1
1.3 Problem Statement..... 2
1.4 Objectives..... 2
1.5 Proposed Approach..... 2
1.6 Organization of the Report. 3

2 Background
2.1 Tools & Technologies..... 4

3 Analysis
3.1 Existing Systems..... 7
3.1.1 Ondoor..... 7
3.1.2 Big Basket..... 7
3.1.3 Blinkit……...8
3.2 Requirement Analysis..... 8
3.2.1 Functional Requirements..... 8
3.2.2 Non Functional Requirements..... 10
3.3 Use Case Analysis……...10
3.4.1 Use Case Description..... 10
3.4 Activity Diagram ……14

4 Design
4.1 Design Diagrams..... 16
4.1.1 Class Diagram……16
4.1.2 Sequence Diagram..... 18

5 Implementation

5.1 Hardware & Software used..... 19

5.2 Code Description..... 19

6 Testing & Result

6.1 Test Cases

6.1 Test Case - I (User login)..... 30

6.2 Test case - II (Add to cart).....31

6.3 Test case - III (Proceed checkout process).....31

6.4 Test case - IV (Vendor create his/her profile).....32

6.5 Test case - V (Vendor add fruits).....33

6.6 Test case -VI (Vendor accept order).....33

6.7 Test case - VII (Vendor manage fruits).....34

6.2 Result……...…35

7 Conclusion

7.1 Conclusion.....36

7.2 Future Enhancement……36

Appendix

A User Manual……37

References…….50

LIST OF FIGURES

2.1 React architecture diagram……..……. 5

2.2 Firebase architecture diagram ….……5

2.3 Flutter architecture diagram..……6

3.1 Use case Diagram……...…..11

3.2 Activity Diagram……..14

4.1 Class Diagram……..16

4.2 Sequence diagram of the whole system……...17

A.1 Customer login & home screen....…….….36

A.2 Customer cart & profile screen…….....……..37

A.3 Customer orders screen & side bar……..38

A.4 Vendor Login & OTP screen..…….39

A.5 Create vendor profile & side drawer.....40

A.6 Vendor home screen & add fruits screen.……41

A.7 Vendor settings screen & orders screen.....42

A.8 Delivery app user details screen & home screen……...43

A.9 Admin welcome screen & dashboard screen.…… 44

A.10 Dashboard vendor & fruit categories screen…….……45

A.11 Admin add fruit & add category screen.....……46

Chapter 1

Introduction

In this chapter, there is a brief introduction about the project giving the need for the project, the problem being solved, objectives, and also the approach to be used.

1.1 Preamble

In a world where convenience and healthy living go hand in hand, we proudly present our latest venture. Inspired by the desire to bring the goodness of nature's bounty directly to your doorstep, we have crafted a seamless platform that connects you with a wide array of fresh, succulent fruits from around the world. Imagine a world where you can effortlessly browse through an extensive selection of fruits, from tangy citrus fruits to luscious berries, all at your fingertips. Gone are the days of fruit shopping in crowded supermarkets, struggling to find the ripest and juiciest produce. With our app, you can now experience the joy of virtual orchards at your home, where you can pick and choose from the finest fruits available.

1.2 Need of the Project

In today's fast-paced lifestyle, consumers seek hassle-free access to high-quality fruits, and FruitBox intends to bridge this gap by providing a user-friendly platform. The project's significance lies in fostering healthier dietary choices and supporting local farmers by creating a streamlined digital marketplace. Through an intuitive online interface, customers can easily browse, purchase, and have fresh fruits delivered to their doorstep. FruitBox not only capitalizes on the e-commerce trend but also contributes to promoting well-being and supporting local agriculture, making it a timely and impactful venture in the evolving market landscape.

1.3 Problem Statement

It has been observed that customers who are unable to purchase fresh fruits from the market are experiencing challenges like not having enough time to go to market and purchase fruits, unavailability of favorite or exotic fruits, and daily trips to market are inconvenient and time-consuming. Carrying fruits back from the market daily can be burdensome, especially for those who rely on public transportation or have no transportation. Lack of customization in the market based on their. preferences, dietary requirements etc.

1.4 Objectives

The main objective of this project is to make available following services on the customer's hand:

- Search for Fruits at nearby locations to buy and sell.
- Support local farmers by establishing a direct marketplace.
- Encourage healthier lifestyles by offering a diverse range of high-quality fruits.
- Implement a robust delivery system that ensures timely and efficient fruit deliveries.

1.5 Proposed Approach

Objective is to source fresh produce directly from farmers, promoting fair trade practices and supporting local agriculture. This approach not only guarantees a steady and diverse supply of high-quality fruits but also strengthens the community ties between the online platform and the vendors. Through effective communication and collaboration, FruitBox seeks to create a win-win scenario where vendors benefit from expanded market access, and customers gain access to a wider selection of fresh, locally sourced fruits. This vendor collaboration approach not only aligns with the project's objectives of supporting local agriculture but also enhances the overall reliability and quality of the FruitBox platform.

1.6 Organization of the Report

The brief summary about the organization of the report is given.

- Chapter 1 Introduction: A brief introduction about the project giving the need for the project, the problem it is going to solve, objectives, and also the approach to be used.
- Chapter 2 Background: It provides a background study of the project which includes Tools and Technologies used and the comparison of various document scanning applications.
- Chapter 3 Analysis: This Chapter describes the detailed problem statement of the project work. It deals with a detailed analysis of the project with the functional requirements and non-functional requirements, system components, use case analysis and the feasibility study.
- Chapter 4 Design: This Chapter focuses on design details consisting of the basic architecture of the system and working of various modules of the system discussed in this chapter.
- Chapter 5 Implementation: This Chapter shows the programming code of the project.
- Chapter 6 Testing & Result: This is the record of all the tests done on the application and its subsequent results.
- Chapter 7 Conclusion: This contains conclusion and future enhancement ideas for the project.

The appendix and references are listed in a separate section.

Chapter 2

Background

In this chapter, the background study of the project is presented. The first section contains the brief description of the Tools and Technologies to be used in the development process and the next sections contain information about the Existing Solutions.

2.1 Tools & Technologies

For collaboration and version control, Github has been used, and vscode editor has been used as a default editor .Their descriptions are as follows:

- GitHub is a version management and collaboration tool for programming. It allows us and others to collaborate on projects from any location. It offers the distributed version control and Source Code Management(SCM) functionality of Git, plus its own features.
- VS Code Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

Firebase will be used as our database. Front-end will be developed using Flutter. Following are the description of libraries and framework used:

- React (https://www.react.dev/) React is a JavaScript library for building user interfaces, developed by Facebook. React follows creation of interactive and dynamic web applications by allowing developers to build a declarative approach, making it efficient for handling changes in application state and automatically updating the user interface accordingly.

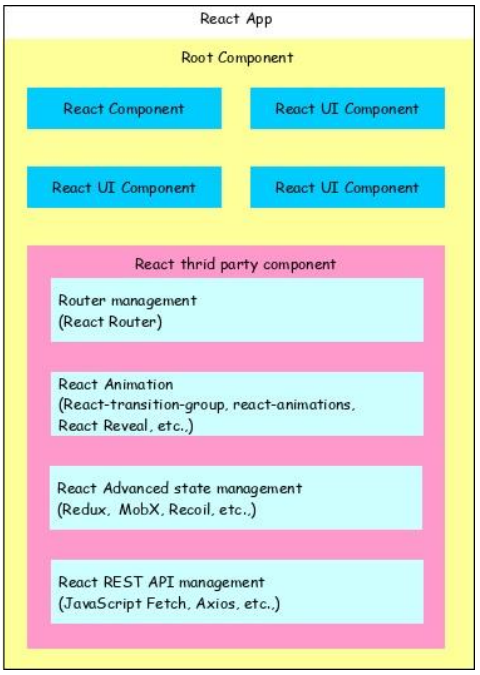


Figure 2.1: React-Architecture Diagram

- Firebase (https://www.firebase.org/) is a powerful, open source realtime database system. It is suitable for a wide range of applications, from small projects to large-scale.

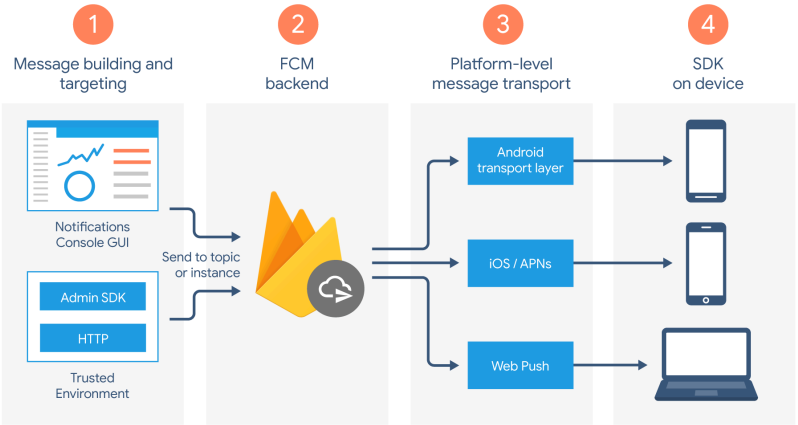


Figure 2.2: Firebase-Architecture Diagram

- Flutter (https://flutter.org/) Flutter is an open-source UI software development toolkit created by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It is designed to help developers create high-performance, visually appealing, and responsive applications with a consistent user interface across different platforms.

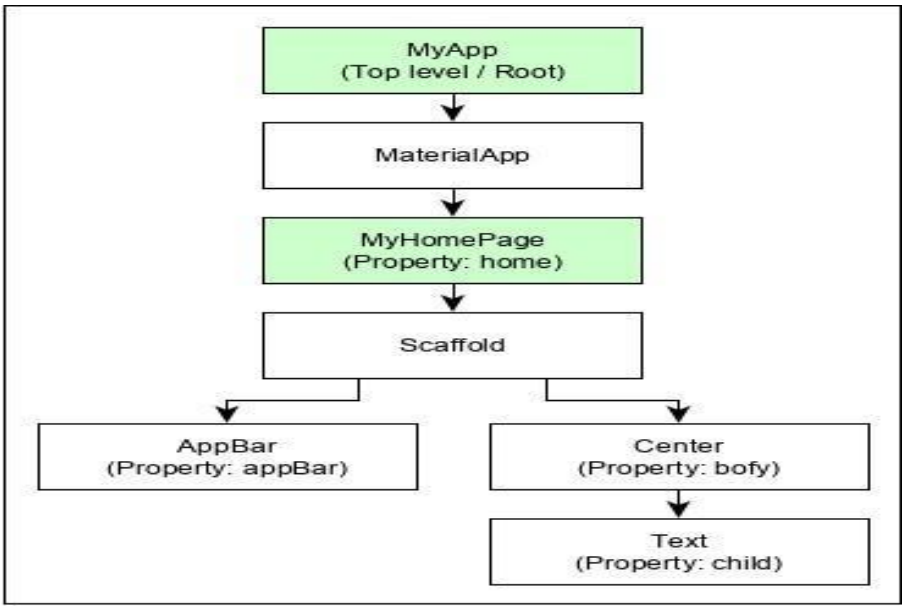


Figure 2.3: Flutter-Architecture Application Diagram

.

Chapter 3

Analysis

In this chapter, a detailed analysis of the project has been discussed. It includes the existing systems, detailed problem statement, and requirement analysis. i.e. functional requirements and non-functional requirements of the proposed system, and the use case analysis.

3.1 Existing Systems

This section enlists the various online platforms which are providing helpers for a particular task.

3.1.1 Ondoor

- Indian online grocery store and delivery platform, which offers a wide range of grocery products.
- Ondoor's commitment to sourcing locally supports regional farmers, fostering sustainable practices and contributing to the community's economy.

Drawbacks

- Common issues faced by their customers include incorrect or missing items.
- They provide poor fruit quality, billing/payment problems.

3.1.2 Big Basket

- Big basket is an Indian e-commerce platform selling groceries and household essentials online. It provides services in domains like grocery, Home essential, laundry products, frozen items, etc

Drawbacks

- The availability of convenient delivery time slots.
- Another potential limitation is related to product availability.

3.1.3 Blinkit

- Blinkit, is India's last minute app selling online milk products, fresh fruits & vegetables, daily grocery, kitchen, home & office items etc to their users.

Drawbacks

- Occasional delays in delivery schedules
- Accuracy issues with product descriptions

3.2 Requirement Analysis

In this section the requirements of the system are specified, Functional and Non-Functional requirements.

3.2.1 Functional Requirements

A functional requirement specifies the functionality of a system or one of its subsystems. It also depends upon the type of software, expected users, and the type of system where the software is used. Functional user requirements may be high-level statements of what the system should do but functional system requirements should also describe clearly the system services in detail.

1. Functional requirements of Customer

1. User Registration with phone number : Users will be able to log in with phone number.
2. Search fruits : Customers are able to search for specified fruits.
3. Search Vendor : Customers can search specific vendors.
4. Add to cart : The user can add desired fruits to cart.
5. Place order : Customers can place orders.
6. Remove from cart : Can be able to remove fruits from cart.
7. Payments : The user can pay once the order is completed.
8. Track order : Customers can track the order.

2. Functional requirements of Vendor

1. Vendor Registration with phone number : Will be able to log in with phone number.
2. Add fruits : Vendors are able to add fruits.
3. Accept orders : Can be able to accept orders from customers.
4. Manage items : Listing & Managing the available fruits listed.
5. Remove fruits : Vendors can remove fruits as per the availability of the fruits.
6. Payments : Able to receive payments.
7. Fruit Listing : Updating daily fruits lists in application.

3. Functional requirements of Delivery

1. Delivery Person Registration with phone number : Will be able to log in with phone number.
2. Delivery notification : Person will be able to receive an order notification.
3. Accept/Reject orders : Able to accept or reject the order.
4. Navigation and Routing : Integration with navigation services like Google Maps to provide optimal routes for deliveries.
5. Update order status : Ability to update the status of each order in real-time

3.2.2 Non Functional Requirements

Requirements, which are not related to the functional aspect of the software, fall into this category.

They are implicit or expected characteristics of software.

1. The system should ensure a high level of availability, with an average system downtime not exceeding 1 hour per month, ensuring that users can access the system reliably and consistently.
2. The system can be accessed from anywhere globally via app.
3. System should function properly on multiple devices to improve portability.
4. The system shall maintain data integrity by keeping backups of all updates to the database.

3.3 Use Case Analysis

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams to model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a website. The "actors" are people or entities operating under defined roles within the system.

3.3.1 Use Case Description

- Register
  - **Actor : Individual**
  - **Basic Flow :** This use case starts when the individual does not have an existing account The system asks for profile information like phone number for registering a new account. The individual enters the information , after validation, the account is created successfully.
  - **Precondition :** The user does not have an account.
  - **Postcondition :** The user registers successfully.

- Login
  - **Actor : User**
  - **Basic Flow :** This use case starts when an individual is not logged in. The system asks for login credentials and after validation, logs the individual in and redirects to the home page.
  - **Precondition :** The login exists.
  - **Postcondition :** The individual logs in successfully.
- Place order and add to cart
  - **Actor : User**
  - **Basic Flow :** The user will search and select the fruits through images and add to cart, then the user will place an order.
  - **Precondition :** The user must be logged in and the items must be available.
  - **Postcondition :** The vendor will receive the order and the customer will pay.
- Create and view profile
  - **Actor : Vendor**
  - **Basic Flow :** Vendors will be able to create and view their profile.
  - **Precondition :** Vendor must be logged in.
  - **Postcondition :** Profile will be visible to the vendor.
- Select Fruits
  - **Actor : Customers**
  - **Basic Flow :** Customers will see available fruits, select desired fruits and add to cart.
  - **Precondition :** Customers must be logged in and they must be at the home page.
  - **Postcondition :** Fruits will be selected.
- Accept Orders
  - **Actor : Vendor**
  - **Basic Flow :** The vendor will receive orders and accept/reject the order.
  - **Precondition :** Vendor must be logged in and online.
  - **Postcondition :** Vendor will accept orders.

- Feedback
  - **Actor : Customer**
  - **Basic Flow :** The customers can rate the vendor and application as well.
  - **Precondition :** Both users must have logged in and the user must have used the app and services.
  - **Postcondition :** Vendors will be notified for their services.
- Remove/Update cart
  - **Actor : Customer**
  - **Basic Flow :** If a customer wants to remove an item or update the cart can do so.
  - **Precondition :** Customers must be logged in and must add fruits to the cart.
  - **Postcondition :** Customer will successfully remove/update the cart.

- Payment
  - **Actor : Customers**
  - **Basic Flow :** On completion the customer will complete payment to the vendor.
  - **Precondition :** Customer must have selected the fruits and placed order.
  - **Postcondition :** Payment will be done.

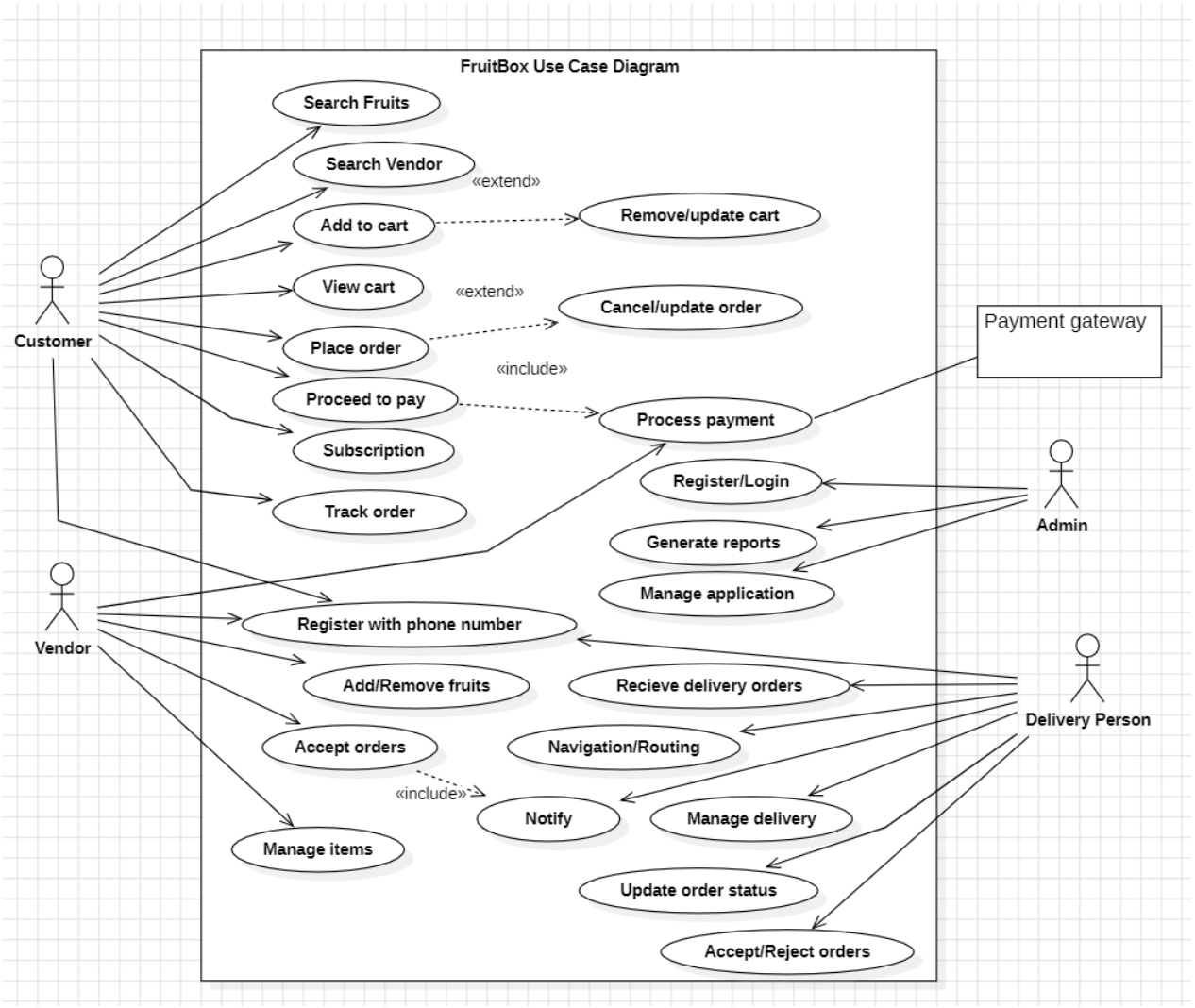


Figure 3.1 : Use case diagram of whole system

3.4 Activity Diagram

An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. It depicts both sequential processing and concurrent processing of activities.They are used in business and process modeling where their primary use is to depict the dynamic aspects of a system. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system but are also used to construct the executable system by using forward and reverse engineering techniques.

- The purpose of an activity diagram can be described as
- Modeling workflow by using activities.
- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched, and concurrent flow of the system.
- High-level understanding of the system’ s functionalities.

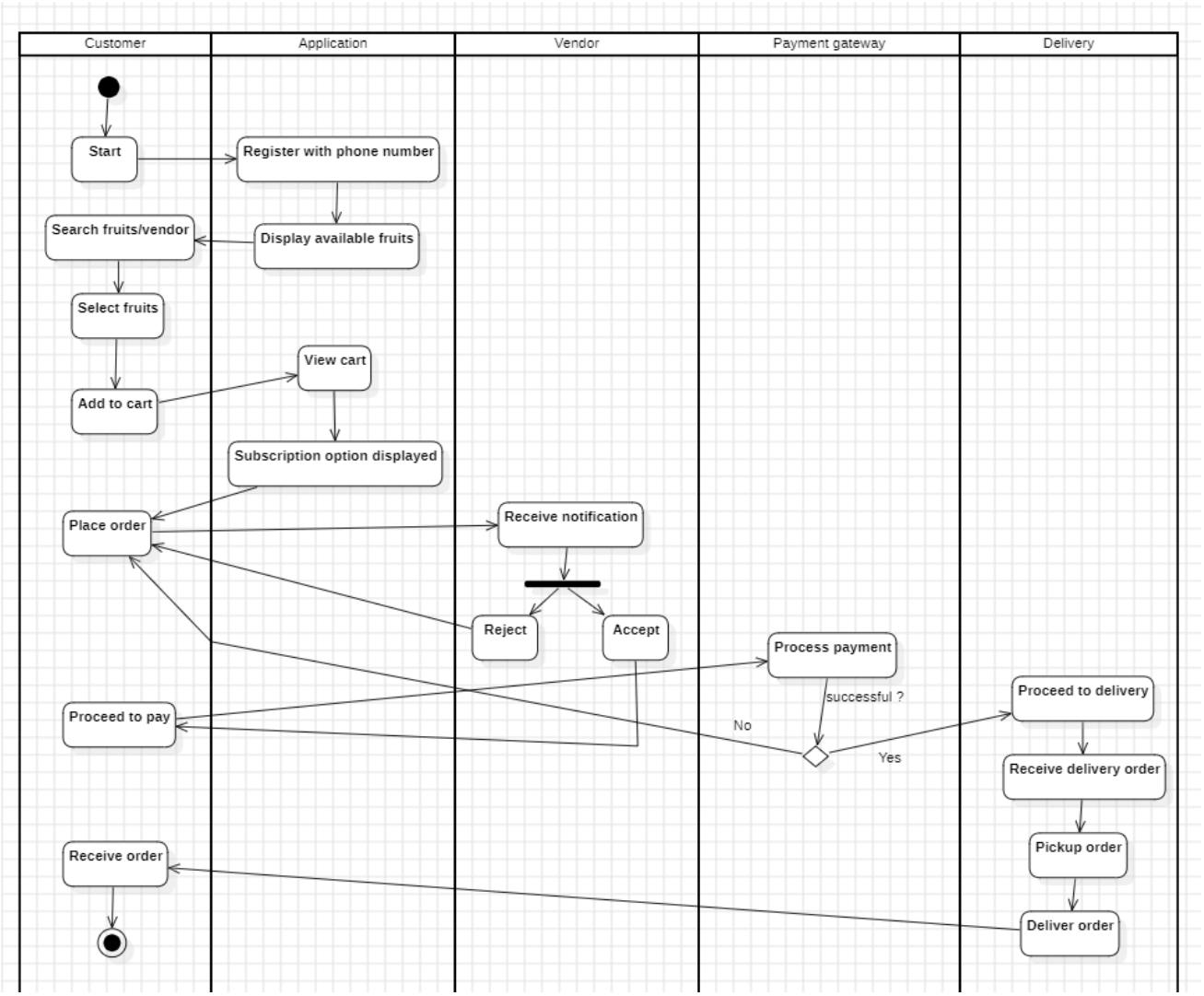


Figure 3.2 : Activity diagram of whole system

Chapter 4

Design

This chapter specifies the working of the system and various UML diagrams.

4.1 Design Diagram

In this section Class and Sequence diagrams have been drawn for a better understanding of the system.

4.1.1 Class Diagram

A class diagram is a visual representation of the structure and relationships within a system. It showcases classes, their attributes, and the methods they use, providing a clear blueprint for understanding how different components of a software system interact. In simple terms, it's like a map that helps developers and stakeholders grasp the organization of the code, making it an essential tool in the design and documentation of software projects.

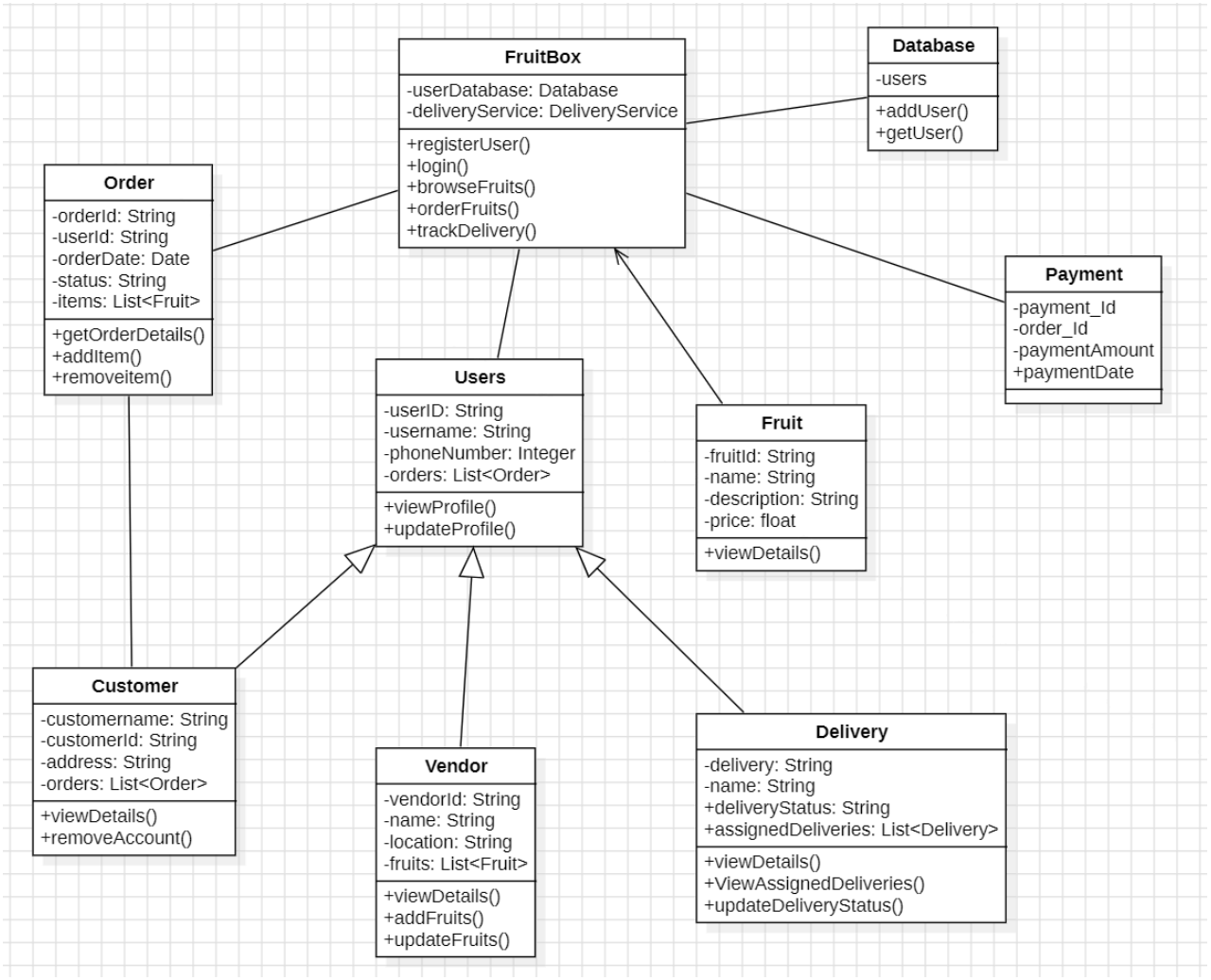


Figure 4.2: Class Diagram

4.1.2 Sequence Diagram

A sequence diagram simply depicts the interaction between objects in sequential order i.e. the order in which these interactions take place. The purpose of the Sequence Diagram is to model high-level interaction among active objects within a system and to model interaction among objects inside a collaboration realizing a use case.

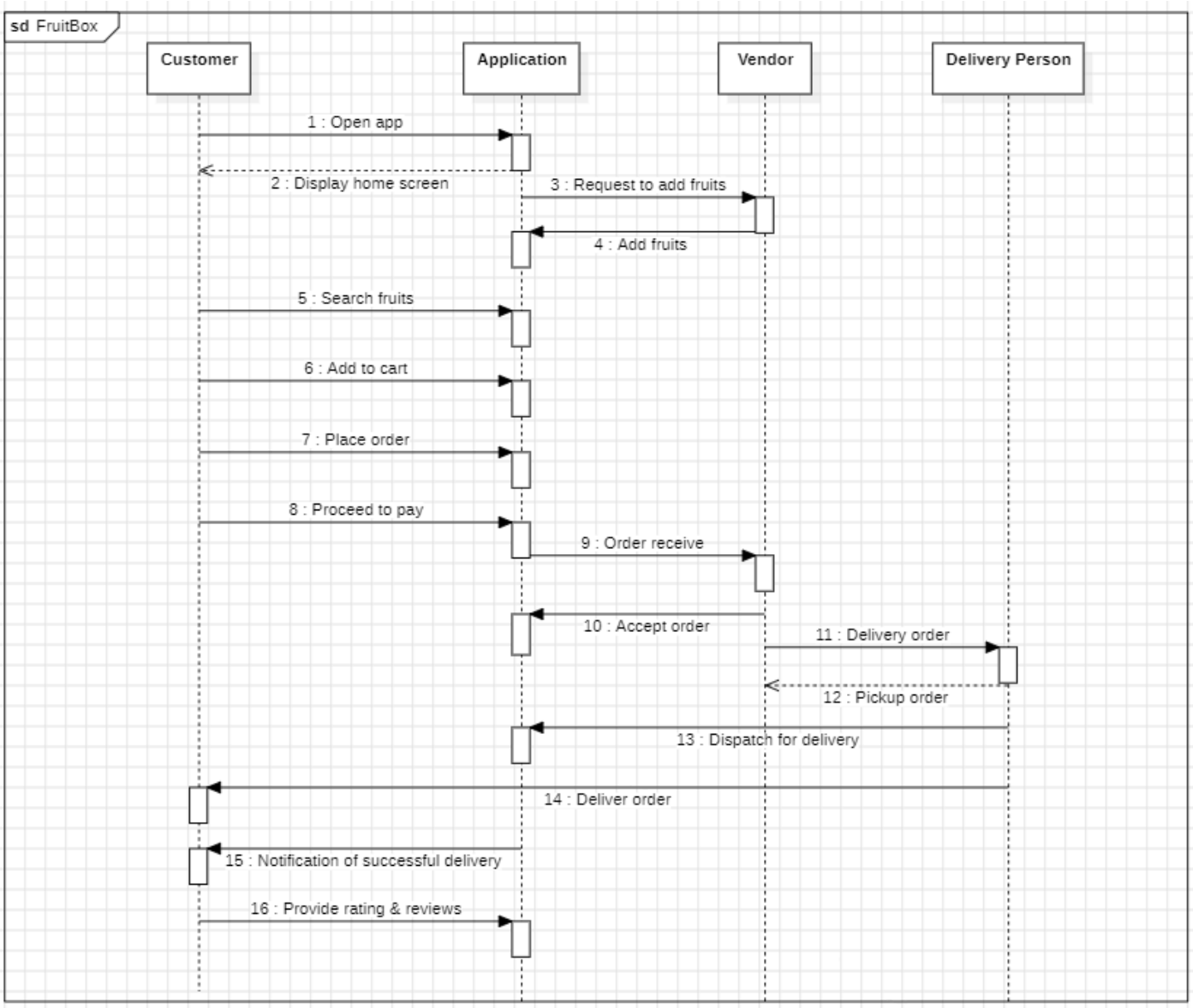


Figure 4.3: Sequence Diagram Of whole system

Chapter 5

Implementation

5.1 Hardware & Software used

The various hardware and software used to train and deploy our project are :

1. OS - Windows, Linux (Ubuntu)
2. Processor - Intel Core i5
3. RAM - 8 GB, GPUs
4. Softwares: React, Visual Studio Code, Github, android studio, Dart, Firebase.

5.2 Code Description

1. Customer Application code:-

Function Name: proceedToCheckout

Input: cart items and details

Output: a new pending order being added to the orders list

Code:

```
void proceedToCheckout() async {
   try {
     final userId = FirebaseAuth.instance.currentUser?.uid;
     if (userId != null) {
```

```
final userCartRef = FirebaseFirestore.instance.collection('users')
    .doc(userId)
    .collection('cart');
final currentItems = await userCartRef.get();
if (currentItems.docs.isNotEmpty) {
  final orderDate = DateTime.now().toIso8601String();
  final totalAmount = calculateTotalCost().toString();
  final newOrder = UserOrder(
    orderDate: orderDate,
    status: 'Pending',
    totalAmount: totalAmount,
    userId: userId,
    orderItems: cartItems.map((cartItem) {
      return OrderItem(
        name: cartItem.productName,
        productId: cartItem.productId,
        boxId: '',
        quantity: cartItem.quantity,
      );
    }).toList(),
  );
  await FirebaseFirestore.instance.collection('orders').add({
    'orderDate': newOrder.orderDate,
    'status': newOrder.status,
    'totalAmount': newOrder.totalAmount,
    'userId': newOrder.userId,
    'orderItems': newOrder.orderItems.map((orderItem) {
      return {
        'name': orderItem.name,
        'productId': orderItem.productId,
        'boxId': orderItem.boxId,
        'quantity': orderItem.quantity,
      };
```

```
            }).toList(),
          });
          clearCart();
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
              content: Text('Order placed successfully!'),
            ),
          );
        } else {
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
              content: Text('Your cart is empty. Add items before proceeding.'),
            ),
          );
        }
      }
    } catch (e) {
      print('Error processing checkout: $e');
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Failed to proceed to checkout'),
        ),
      );
    }
  }
```

2. Vendor Application code:-

Function Name: PhoneAuthCredential

Input: Phone Number, OTP Received

Output: Login Successful or Error message

Code:

```
try {
                    PhoneAuthCredential credential = PhoneAuthProvider.credential(
                      verificationId: MyLogin.verify,
                      smsCode: code,
                    );
                    await auth.signInWithCredential(credential);
                    User? user = auth.currentUser;
                    if (user != null) {
                      // Save user information to Firestore under the "vendors" collection
                      DocumentReference userDoc = FirebaseFirestore.instance
                          .collection('vendors')
                          .doc(user.uid);
                      userDoc.set({
                        'address': '', // Add the address field
                        'contact': user.phoneNumber, // Save the phone number as contact
                        'email': '', // Initialize email as empty
                        'name': '', // Initialize name as empty
                      }, SetOptions(merge: true));

                      Navigator.pushNamedAndRemoveUntil(
                        context,
                        'create_profile',
                            (route) => false,
                      );
                    }
                  } catch (e) {
                    print("Error during authentication: $e");
                    // Handle specific error cases (e.g., incorrect OTP, network issues)
                    // Provide user feedback such as a SnackBar or an alert dialog
                    if (e is FirebaseAuthException && e.code == 'invalid-verification-code') {
                      print("Invalid OTP provided");
```

```
// Show an alert to the user that the OTP is invalid
ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
              content: Text("The OTP is incorrect. Please try again."),
                    backgroundColor: Colors.red,
                  ),
                );
               }
              }
             },
```

Function Name: fruitCollection

Input: Name, Price, Description, Image

Output: Fruits added to firebase successfully

Code:

```
// Add the fruit data to Firestore with the image URL
   try {
    final fruitCollection = FirebaseFirestore.instance.collection('fruits');
    await fruitCollection.add({
     'name': name,
     'price': price,
     'description': description,
     'category': selectedCategory,
     'image_url': imageUrl,
     // Add other fields as needed
    });
    // Navigate back after successful upload
    Navigator.pop(context);
   } catch (e) {
    print('Error: $e');}}
```

Function Name: fetchFruitsFromDatabase

Input:

Output: Fruits successfully fetched from database

Code:

```
// Function to fetch fruits data from Firestore
  Future<List<Fruit>> fetchFruitsFromDatabase() async {
    // Perform a database query to fetch fruits data
    QuerySnapshot querySnapshot = await FirebaseFirestore.instance.collection('fruits').get();
    // Convert the fetched data into a list of Fruit objects
    List<Fruit> fruits = [];
    querySnapshot.docs.forEach((doc) {
      var data = doc.data() as Map<String, dynamic>;
      fruits.add(Fruit(
        name: data['name'] ?? '',
        price: (data['price'] ?? 0.0).toDouble(),
        imageUrl: data['imageUrl'] ?? '',
      ));
    });
    return fruits;
  }
}
```

3. Delivery Application code:-

Function Name: pickAndUploadImage

Input: user pick an image

Output: image upload to the database

Code:

```
  Future<void> pickAndUploadImage() async {
```

```
  final picker = ImagePicker();
 final pickedFile = await picker.pickImage(source: ImageSource.gallery);
 if (pickedFile != null) {
  final file = File(pickedFile.path);
  final userId = FirebaseAuth.instance.currentUser?.uid ?? '';
   if (userId.isEmpty) {
    return;
   }

   try {
    final storageRef = FirebaseStorage.instance.ref().child('drivers_profile_image/$userId.jpg');
    final uploadTask = storageRef.putFile(file);
    final taskSnapshot = await uploadTask.whenComplete(() => {});
    final downloadUrl = await taskSnapshot.ref.getDownloadURL();

    await saveUserDetails('profilePhotoUrl', downloadUrl);

    setState(() {
     profilePhotoUrl = downloadUrl;
    });

    ScaffoldMessenger.of(context).showSnackBar(
     const SnackBar(
      content: Text('Profile photo updated successfully!'),
      backgroundColor: Colors.green,
     ),
    );
   } catch (e) {
    print('Error uploading profile photo: $e');
```

```
ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Error uploading profile photo. Please try again.'),
          backgroundColor: Colors.red,
        ),
      );
    }
  }
}
```

Function Name: checkIfNewUser

Input: user

Output: return false if user exist and true if not exist checks

Code:

```
Future<bool> checkIfNewUser(User user) async {
  try {
    // Assuming you have a 'drivers' collection in Firestore
    DocumentSnapshot userDoc = await FirebaseFirestore.instance.collection(
        'drivers').doc(user.uid).get();
    if (userDoc.exists) {
      // User exists in the 'drivers' collection
      return false; // Existing user
    } else {
      // User does not exist in the 'drivers' collection
      return true; // New user
    }
```

```
    } catch (e) {
      // Handle exceptions, e.g., if there is an issue connecting to Firestore
      print("Error checking if the user is new: $e");
      return false; // Assume existing user in case of an error
    }
  }
```

4. Admin Panel code:-

Function Name: loginWithEmailAndPassword

Input: Email and Password

Output: access userCredential.user

Code:

```
export const loginWithEmailAndPassword = async (email, password) => {
  try {
    const userCredential = await signInWithEmailAndPassword(auth, email, password);
    // User is logged in, you can access userCredential.user
    return userCredential.user;
  } catch (error) {
    // Handle login error
    return { error: error.message };
  }
};
```

Function Name: fetchUserCount

Input: usersCollection

Output: userCount

Code:

```
export const fetchUserCount = async () => {
  try {
    const usersCollection = collection(firestore, 'users');
    const querySnapshot = await getDocs(usersCollection);
    const userCount = querySnapshot.size; // Number of documents in the collection

    // Set the user count in your state or variable
    setUserCount(userCount); // You can use the state to display it in your component
  } catch (error) {
    console.error('Error fetching user count:', error);
  }
};
```

Function Name: fetchVendorData

Input: Name, Address, Contact, Email

Output: Vendor data fetched successfully

Code:

```
const fetchVendorData = async () => {
  try {
    const vendorsCollection = collection(firestore, 'vendors');
    const querySnapshot = await getDocs(vendorsCollection);

    const vendorsData = querySnapshot.docs.map((doc) => ({
```

```
 name: doc.data().name,
      address: doc.data().address,
      contact: doc.data().contact,
email: doc.data().email,
     id: doc.id,
    }));
    setData(vendorsData);
  } catch (error) {
    console.error('Error fetching vendor data from Firestore:', error);
  }
};
```

Chapter 6

Testing and Result

6.1 Testing

Testing is a process, to evaluate the functionality of a system or software application with an intent to find whether the developed system or software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product. The subsequent test cases of our system are as follows:

6.1.1 Test Case - I

Test Case Name: User Registration

System: Fruit ordering application (customer & vendor app)

Description: Test registration with user phone number on android application.

Precondition: System of the end user should be connected to the internet.

Table 6.1: **Specification of Test Case - I**

Step	Action	Expected System	Pass/Fail
1	Login with user phone number	Login requested will be rejected	Pass
2	Enter valid phone number	User will be prompted to enter OTP	Pass
3	Click on submit button	Validated & OTP send to user phone number	Pass
4	Enter valid OTP & submit	Successful registration of user	Pass

Post condition: User now able to login into fruitbox customer application.

6.1.2 Test Case - II (Customer App)

Test Case Name: Add to cart

System: Fruit ordering application (customer app)

Description: This test case verifies the functionality of adding fruits to the cart.

Precondition:

1. The user is logged into the customer app.
2. The user is on the homepage of the customer app.

Table 6.2: **Specification of Test Case - II**

Step	Action	Expected System	Pass/Fail
1	Select/Search desired fruits	Fruits successfully selected/searched	Pass
2	Adjust the quantity of selected fruits.	Fruits successfully adjusted	Pass
3	Click on add to cart	Fruits successfully added to the cart	Pass

Post condition: The selected fruit is added to the cart.

6.1.3 Test Case - III (Customer App)

Test Case Name: Proceed checkout process

System: Fruit ordering application (customer app)

Description: Verify that a user can successfully complete the checkout process with items in the cart.

Precondition:

1. User is logged in.
2. User has added fruits to the cart.

Table 6.3: **Specification of Test Case - III**

Step	Action	Expected System	Pass/Fail
1	User navigates to the cart page	Successfully navigated	Pass
2	User verifies required fruits in the cart	Successfully display selected fruits	Pass
3	Click on 'Proceed checkout process'	System response successfully	Pass

Post condition:

1. The fruits were then removed from the cart.
2. The order is processed at the payment gateway.

6.1.4 Test Case - IV (Vendor App)

Test Case Name: Vendor create his/her profile

System: Fruit ordering application (vendor app)

Description: This test case checks the functionality of creating a vendor profile.

Precondition: Vendor must be logged in to the vendor app.

Table 6.4: **Specification of Test Case - IV**

Step	Action	Expected System	Pass/Fail
1	Profile page displayed	Profile page successfully displayed	Pass
2	Enter the valid credentials	Credentials entered successfully	Pass
3	Click on 'Submit' & proceed to next screen	Worked successfully	Pass

Post condition: User is logged in with a valid phone number.

6.1.5 Test Case - V (Vendor App)

Test Case Name: Vendor add fruits

System: Fruit ordering application (vendor app)

Description: This test case checks the functionality of adding fruits to the cart.

Precondition: Vendor must be logged in to the app.

Table 6.5: **Specification of Test Case - V**

Step	Action	Expected System	Pass/Fail
1	User logged in to the app	Successfully logged in	Pass
2	User at the home page	Home page displayed successfully	Pass
3	Click on the add fruits button	Successfully responded	Pass

Post condition: User proceeds to the next screen.

6.1.6 Test Case - VI (Vendor App)

Test Case Name: Vendor accept orders

System: Fruit ordering application (vendor app)

Description: This test case verifies the functionality of the vendor accepting orders placed by customers.

Precondition:

1. Vendors must be logged into the app.
2. Vendor must be at the order page.

Post condition: The status of the accepted orders is updated.

Table 6.6: **Specification of Test Case - VI**

Step	Action	Expected System	Pass/Fail
1	Vendor receive orders from customer	Order received from customer	Pass
2	Select an order to accept	Any valid order must be accepted	Pass
3	Click on the ‘Accept order’ button	Status of the order updated	Pass

6.1.7 Test Case - VII (Vendor App)

Test Case Name: Vendor manage fruits

System: Fruit ordering application (vendor app)

Description: Vendor manages fruits by adding, updating, and deleting them from the system.

Precondition:

1. Vendors must be logged into the app & must be validated.

Table 6.7: **Specification of Test Case - VII**

Step	Action	Expected System	Pass/Fail
1	Add/remove/update the fruits listed	Display the lists	Pass
2	Action performed whichever user need	Successfully worked	Pass

Post condition: The changes made by the vendor are reflected accurately in the system.

6.2 Result

App Functionality :

Customer App: The customer app allows users to browse fruits, place orders, and view order history. The app's user-friendly interface ensures a smooth ordering process.

Delivery App: The delivery app enables delivery personnel to view deliveries.

Vendor App: The vendor app helps vendors manage inventory.

Integration and Coordination :

The three apps work together to facilitate the entire fruit ordering process, from placing orders to delivery and vendor management.

Real-time data synchronization between apps ensures smooth coordination and efficiency using firebase.

Chapter 7

Conclusion

7.1 Conclusion

In conclusion, the FruitBox application offers a user-friendly Android platform for customers to easily select and order fresh fruits, enhancing accessibility and convenience. However, challenges such as reliance on consistent internet connectivity, potential limitations in fruit availability, and dependence on suppliers' reliability exist. Despite these drawbacks, FruitBox effectively addresses common issues associated with traditional fruit shopping, providing a modern solution for consumers seeking convenience and quality in their fruit procurement process.

Advantages:

- User-friendly android platform for selecting and ordering fresh fruits
- Enhanced accessibility and convenience for customers
- Resolves common challenges associated with traditional fruit shopping

Limitations:

- Reliance on consistent internet connectivity, which may be challenging in areas with poor connectivity
- Potential limitations in terms of the types of fruits available, depending on the supplier's inventory
- Dependence on the supplier's ability to deliver fresh and high-quality fruits consistently

7.2 Future Enhancement

1. Creating & Managing inventory.
2. Integration of Mobile Wallets.
3. Localized Partnerships with farmers for a connected and region-specific fruit-buying community.

User Manual

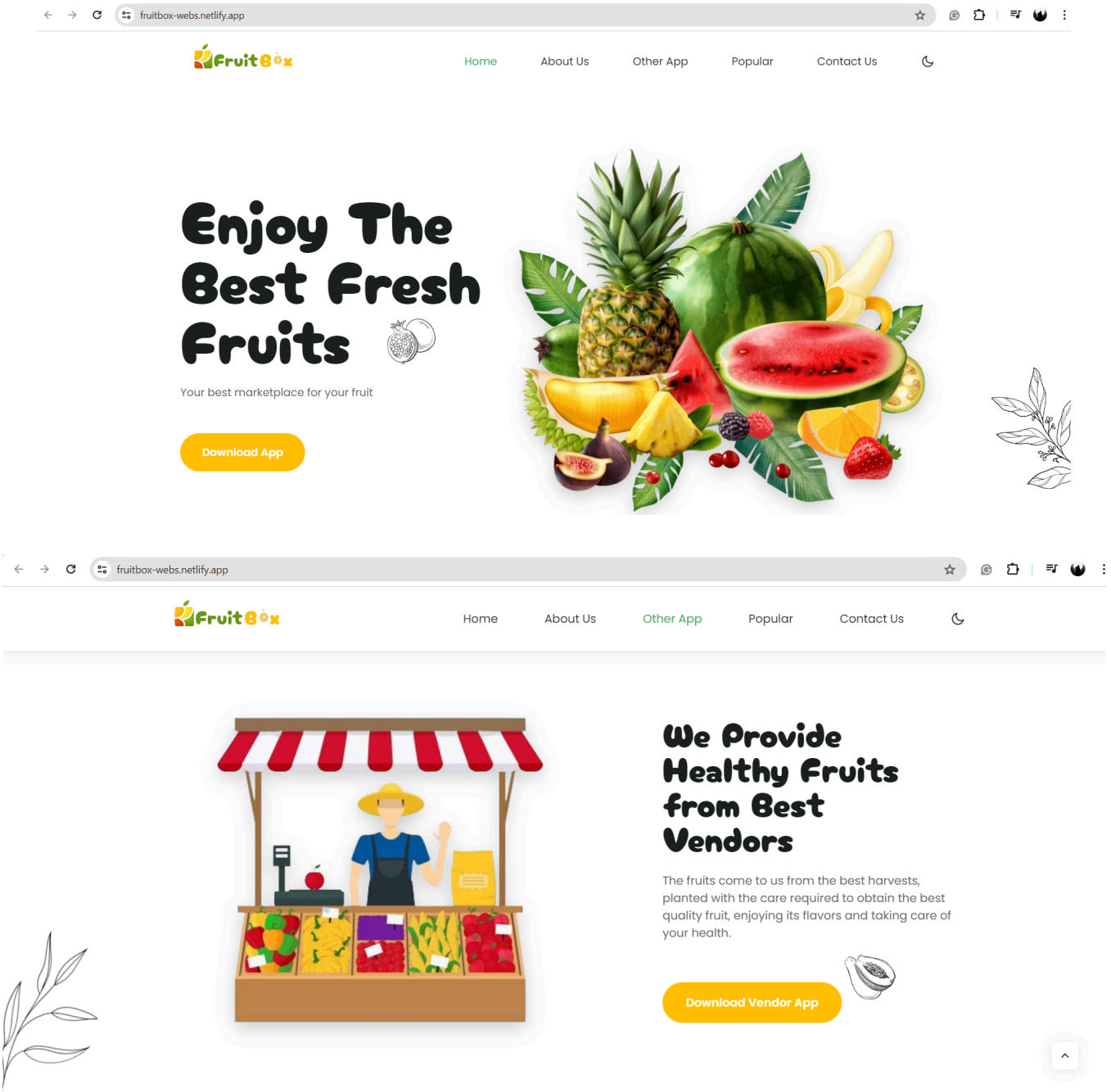


Figure A.1 FruitBox Website (Download Customer & Vendor Application)

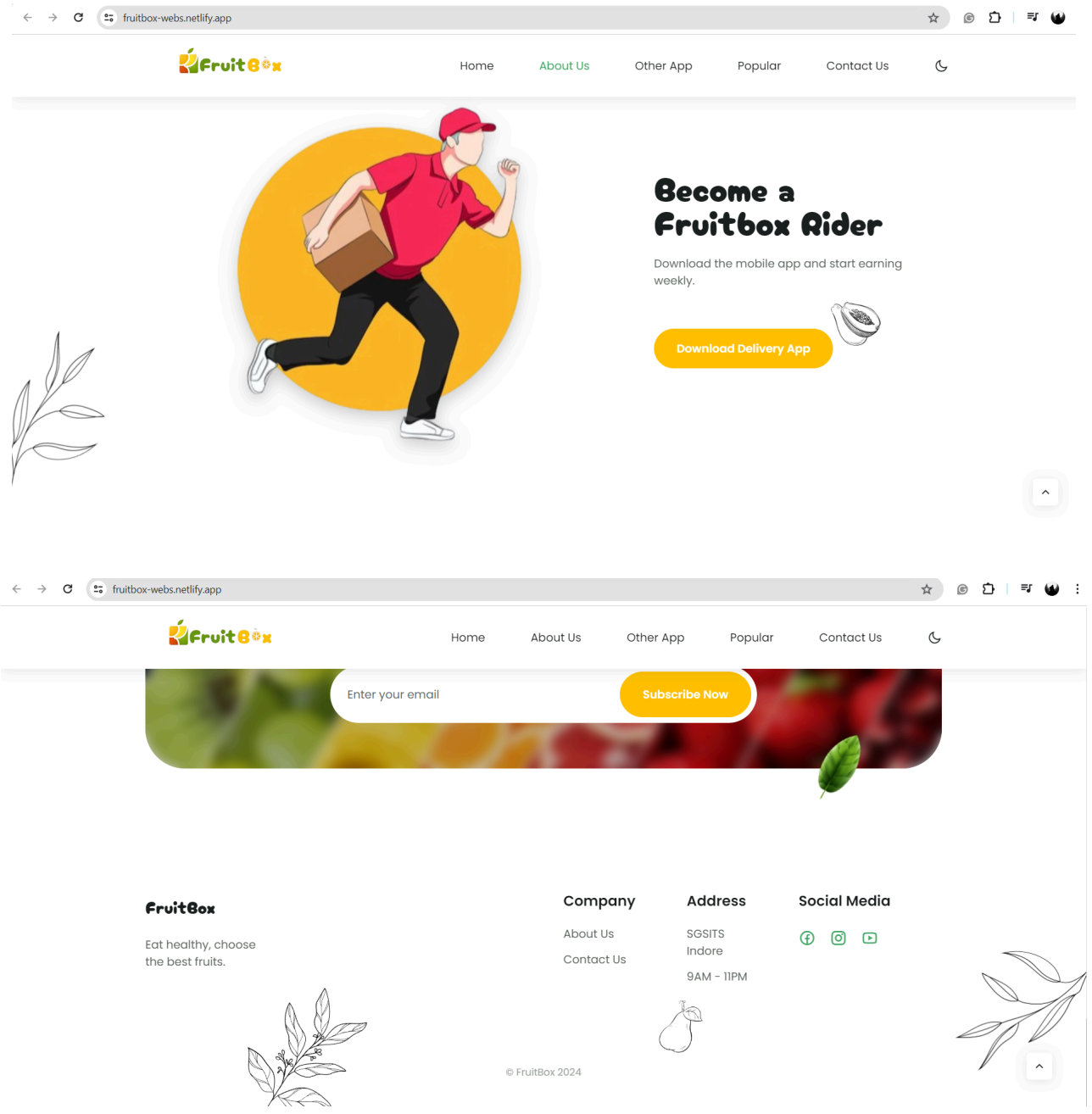


Figure A.2 FruitBox Website (Download Delivery Or Subscribe)

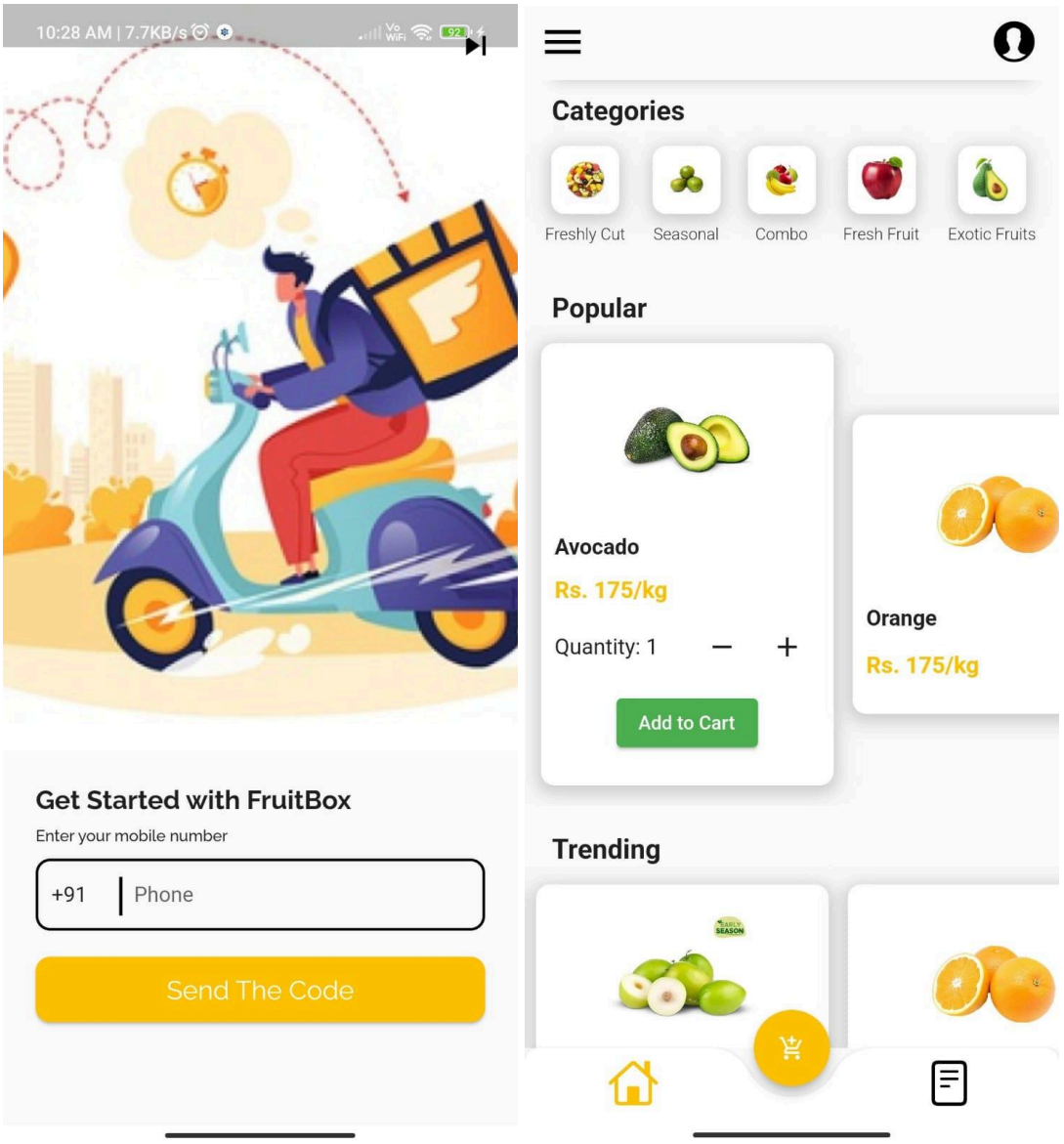


Figure A.3 Customer login & home screen

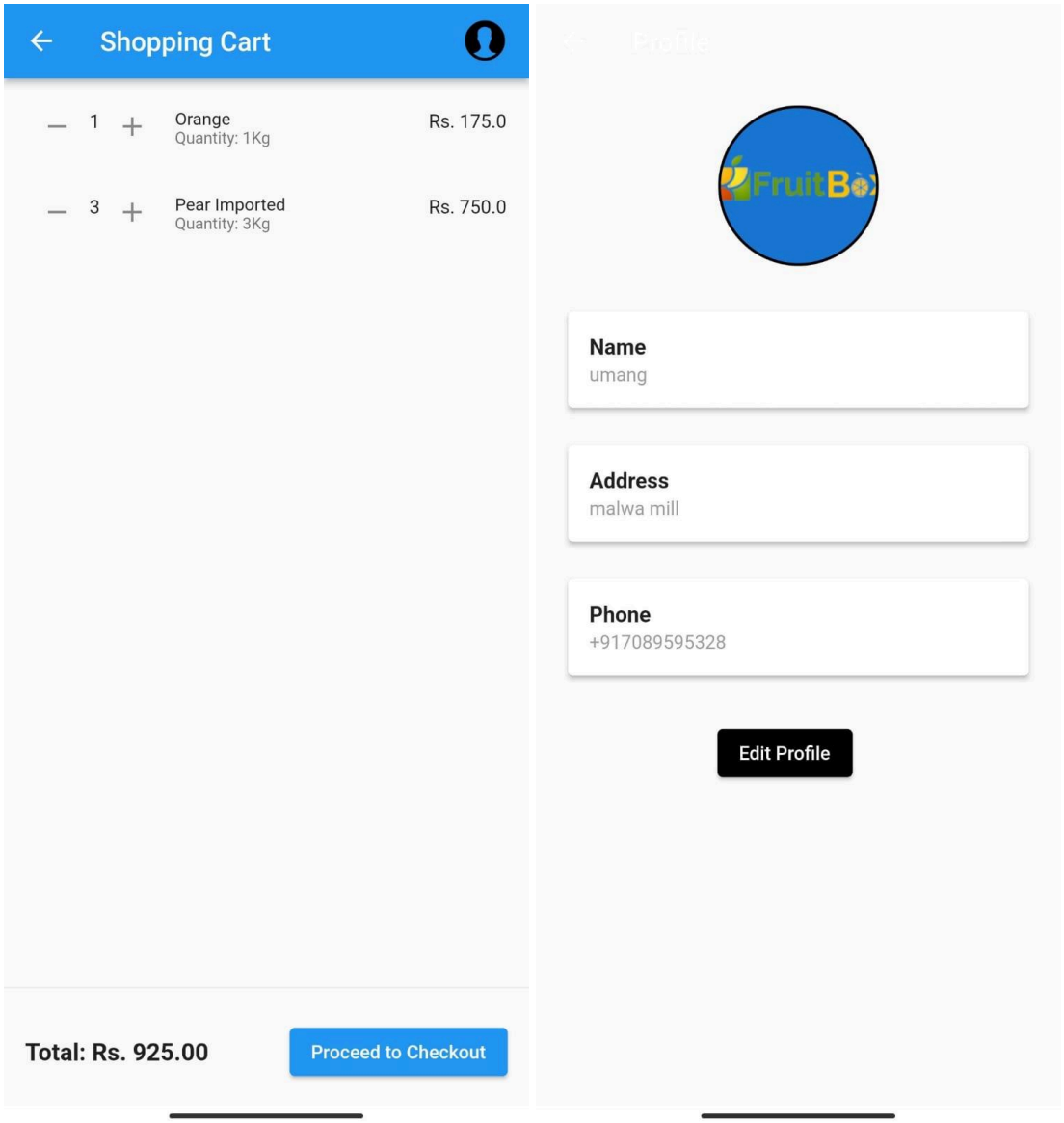


Figure A.4 Customer cart & profile screen

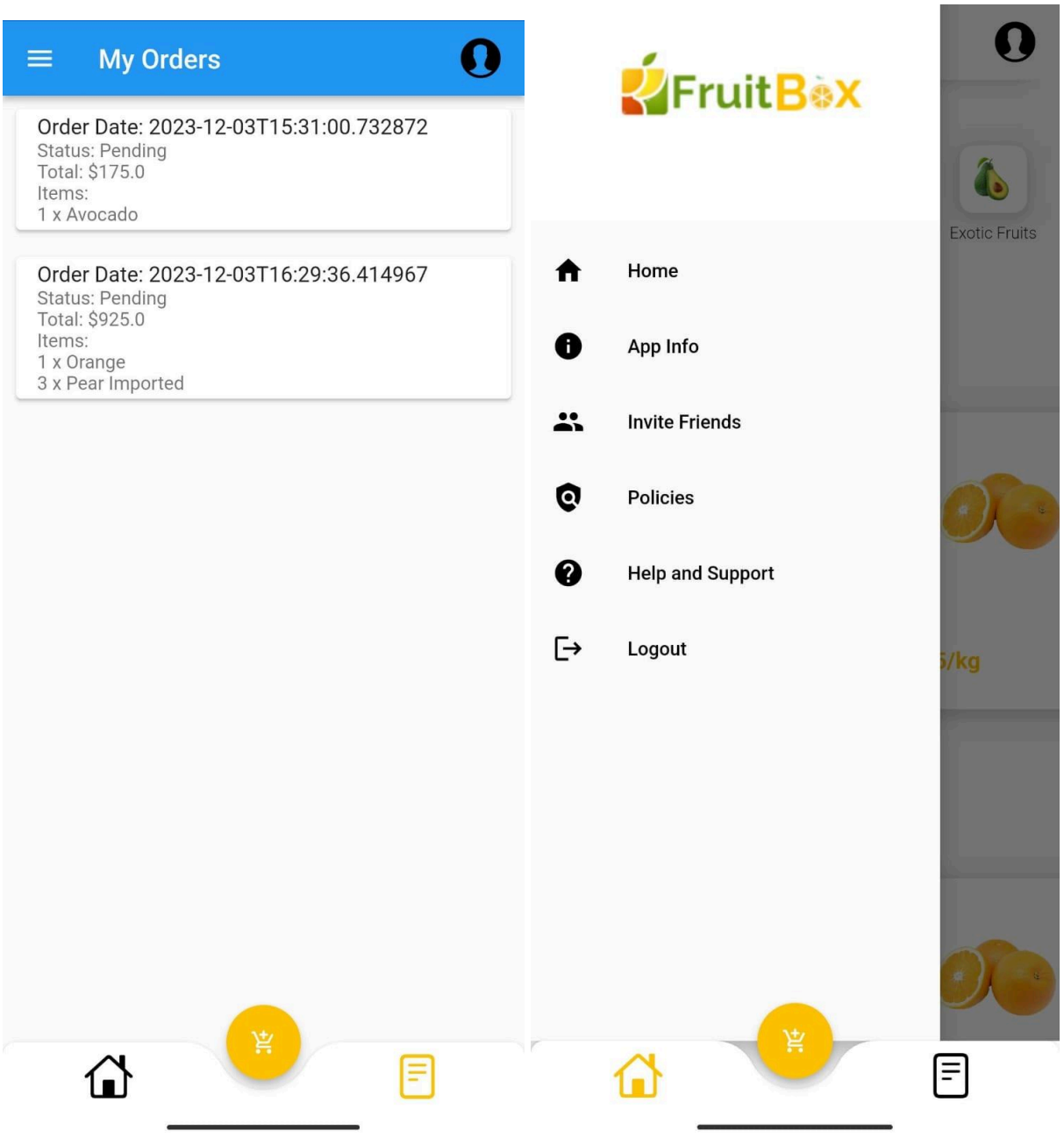


Figure A.5 Customer orders screen & side bar

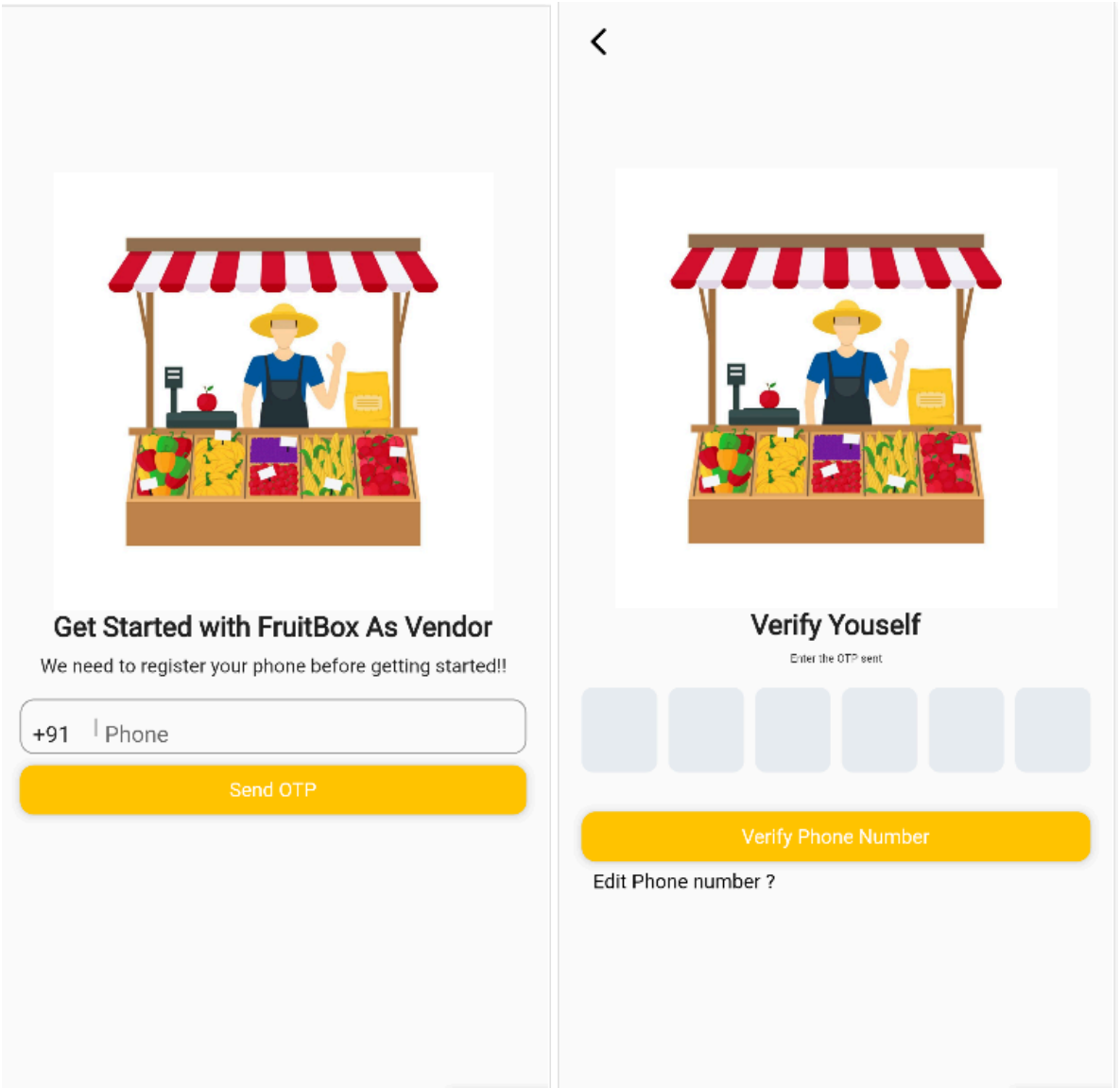


Figure A.6 Vendor Login & OTP screen

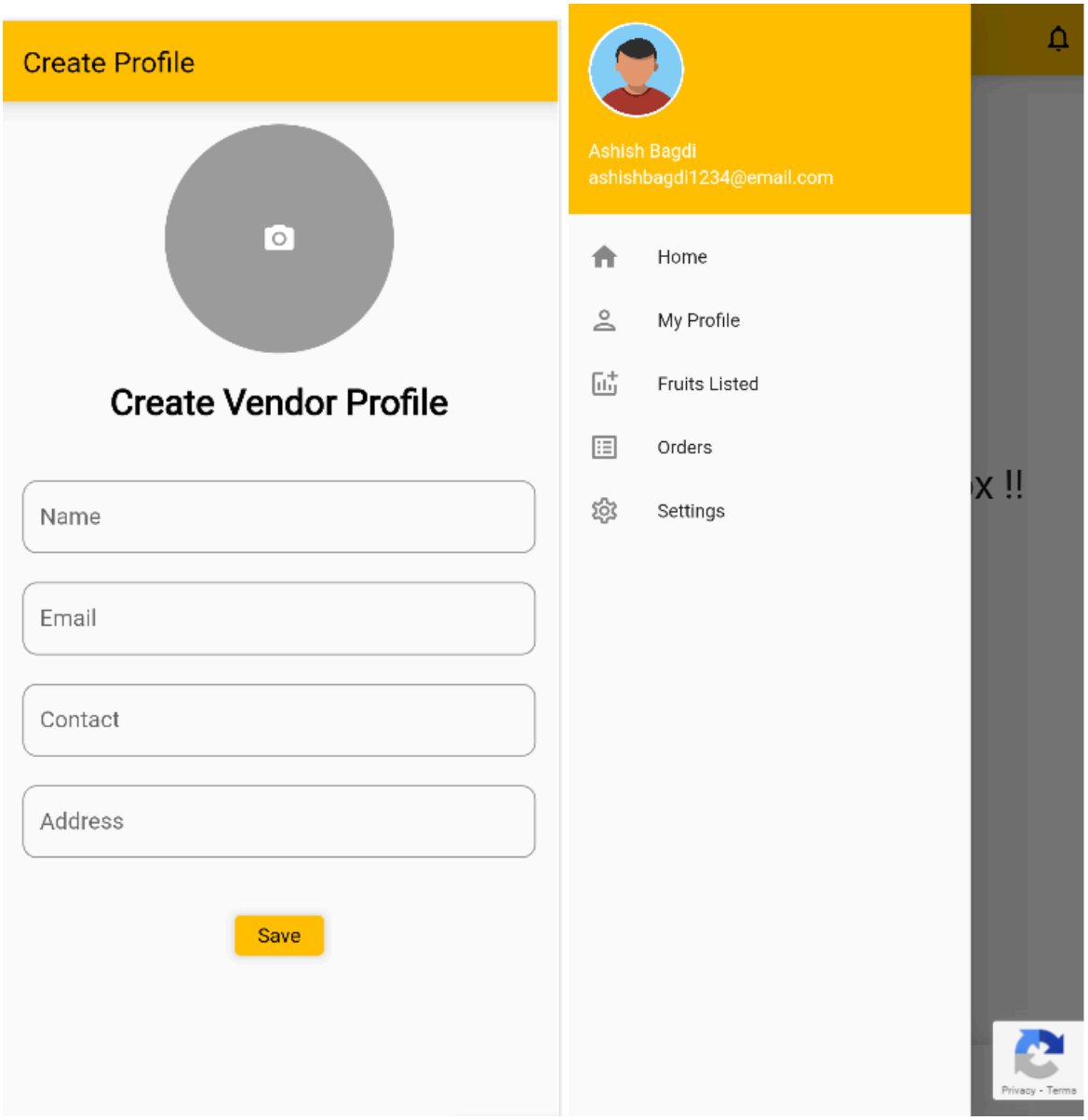


Figure A.7 Create vendor profile & side drawer

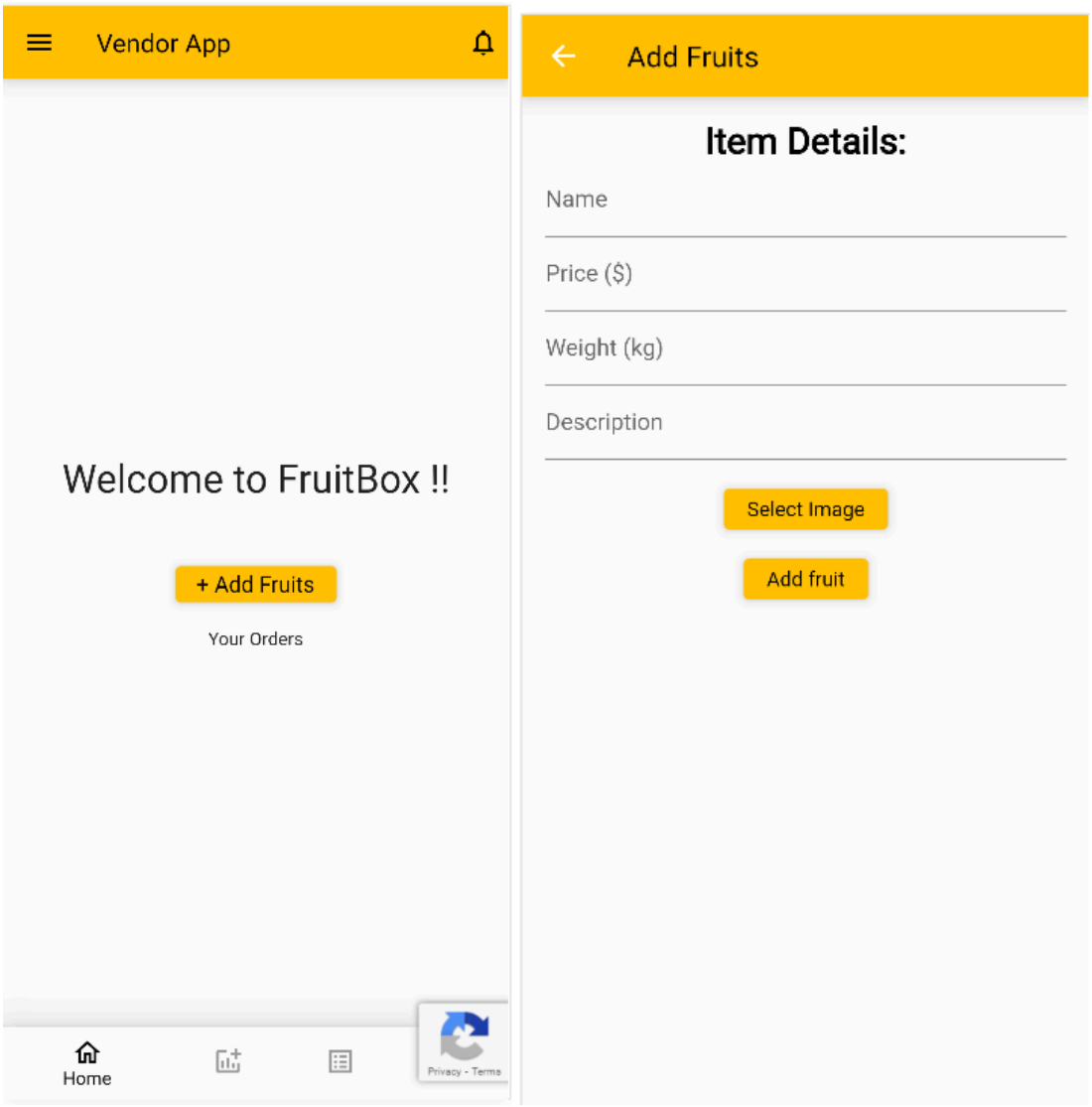


Figure A.8 Vendor home screen & add fruits screen

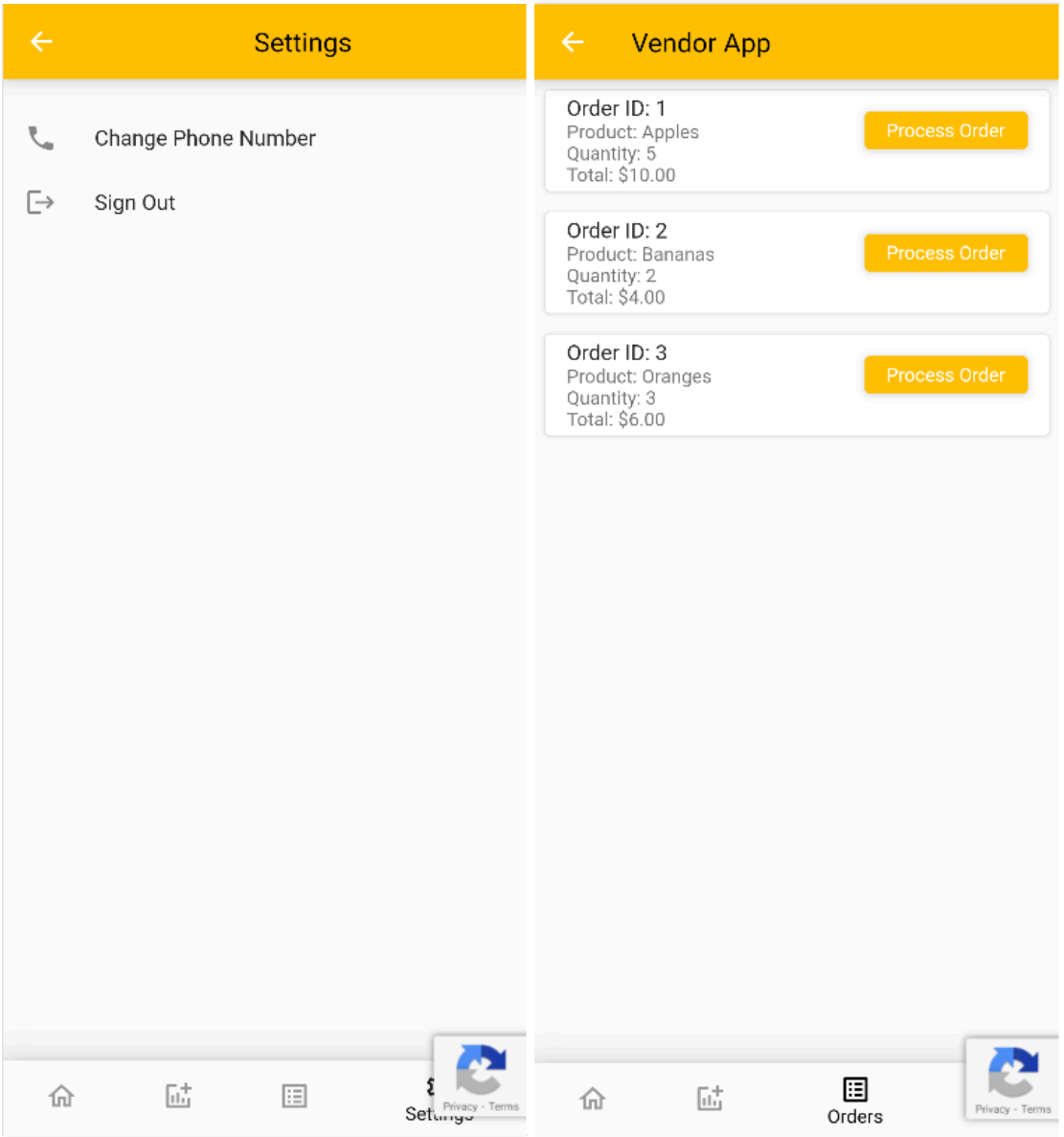


Figure A.9 Vendor settings screen & orders screen

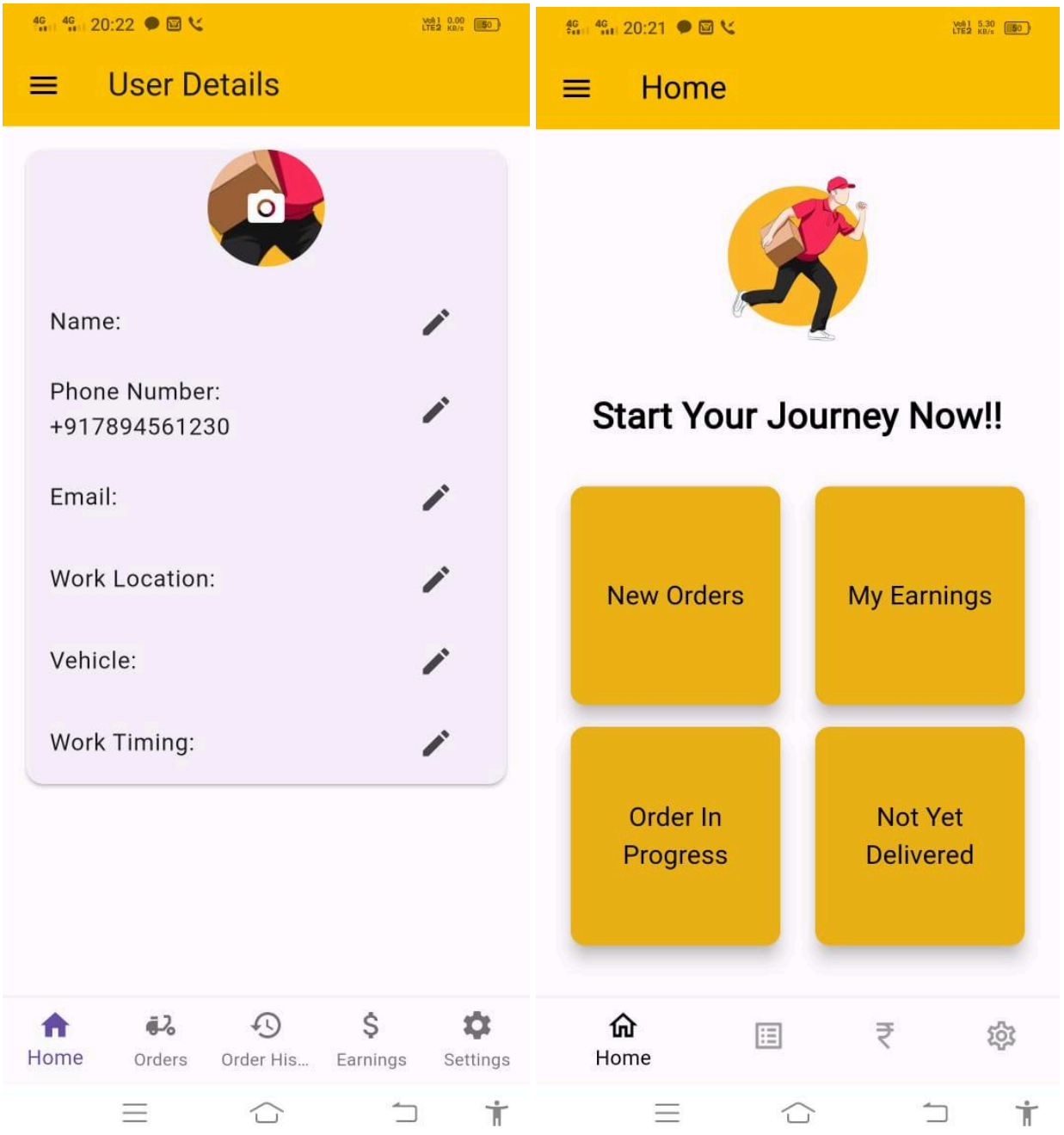
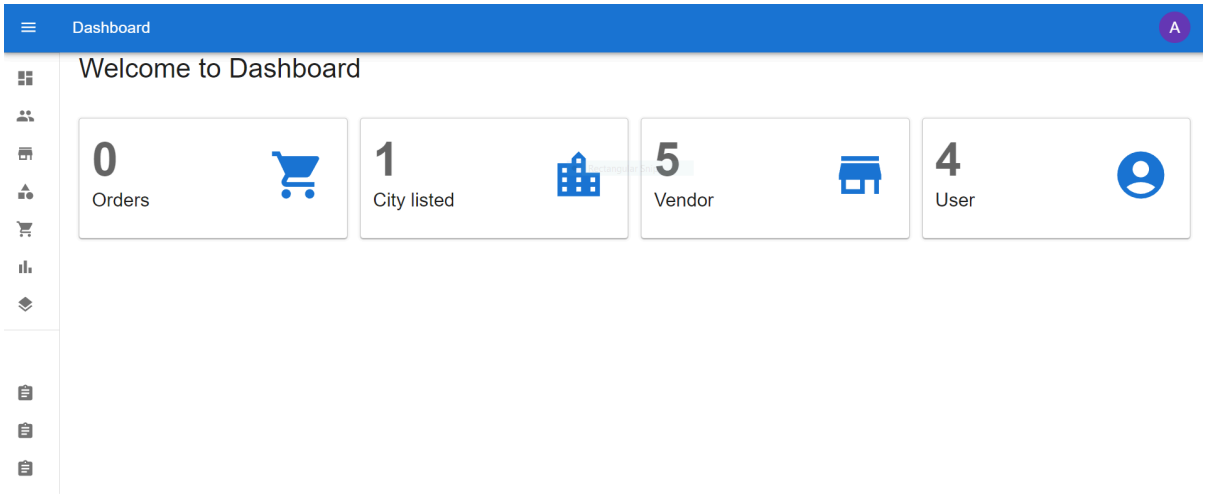


Figure A.10 Delivery app user details screen & home screen



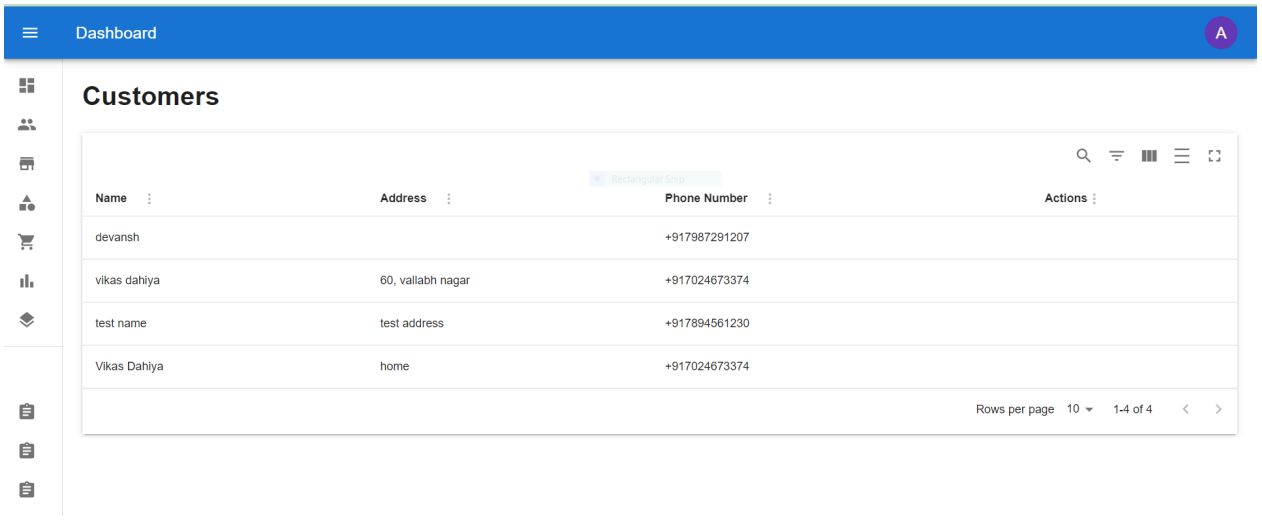


Figure A.11 Admin welcome screen & dashboard screen

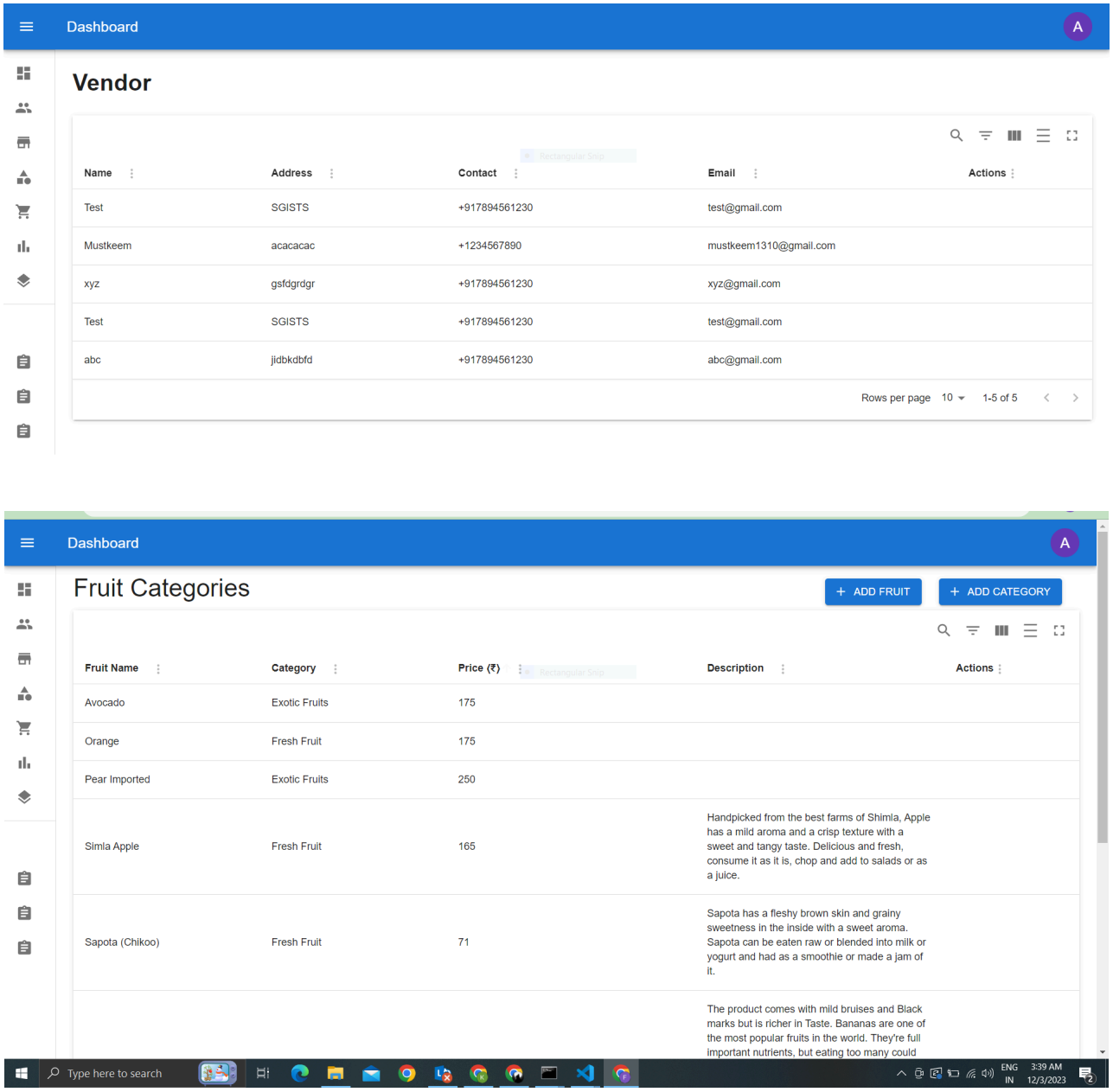


Figure A.12 Dashboard vendor & fruit categories screen

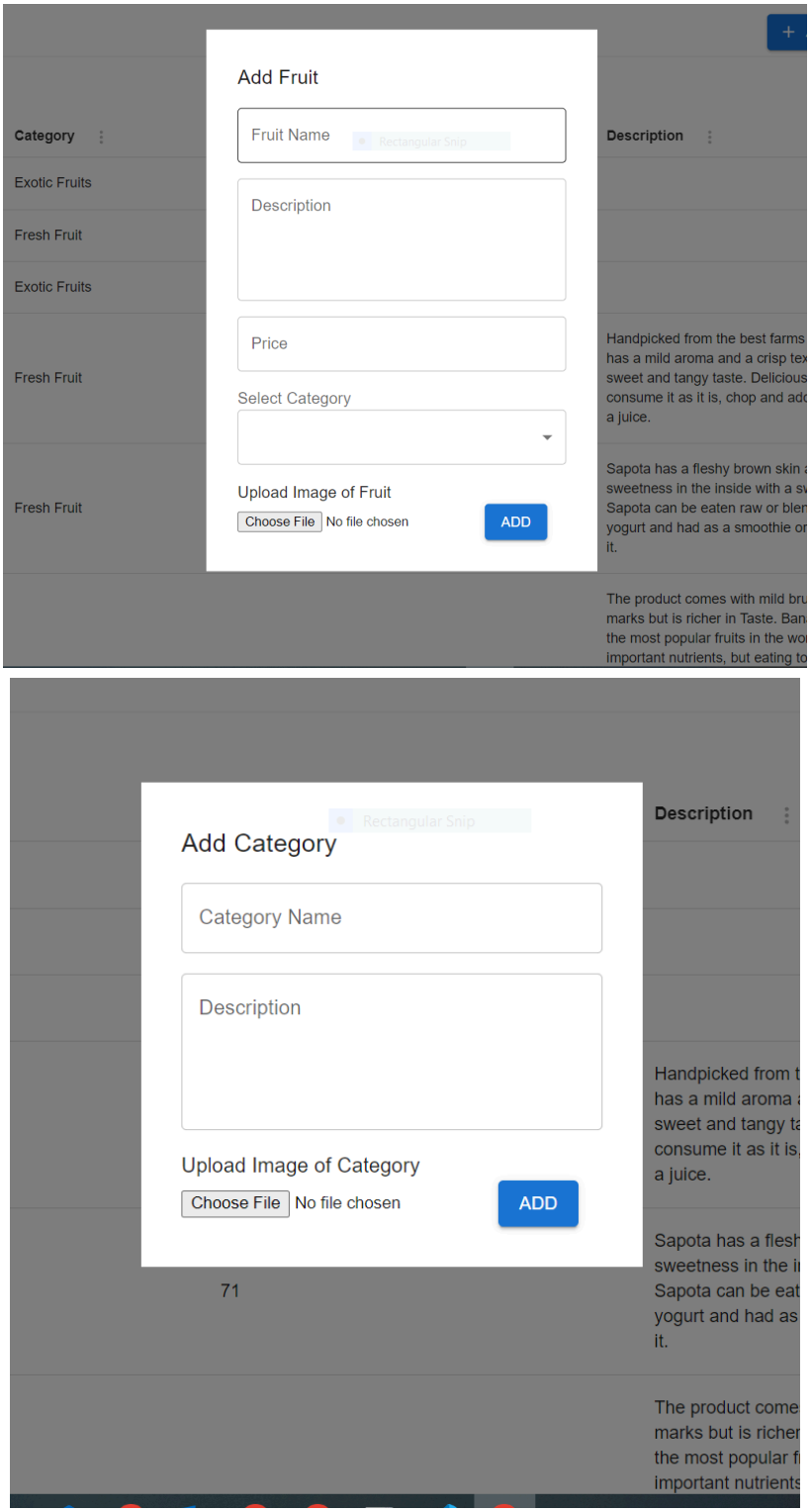


Figure A.13 Admin add fruit & add category screen

References

1. “Flutter Documentation” https://docs.flutter.dev/get-started/install
2. “Github” https://github.com/GurdeepSingh-767/admin_panel
3. Fruits Deilvery Survey by news channel https://www.news18.com/india/only-7-of-indian-households-buying-fruits-and-vegetables-online-finds-localcircles-survey-8674711.html
4. UML Diagram Tool https://docs.staruml.io/working-with-uml-diagrams/
5. Ondoor Application https://www.ondoor.com/
6. Big Basket Application https://g.co/kgs/S8oFwcp