FARMCONNECT

EMPOWERING AGRICULTURAL COMMERCE THROUGH DIGITAL PLATFORM

Project Report

Submitted by

N Amal Thomson

Reg. No.: AJC22MCA-2065

In Partial Fulfilment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS
(MCA TWO YEAR)
[Accredited by NBA]

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2022-2024

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**FARMCONNECT"** is the bona fide work of **N AMAL THOMSON (Regno: AJC22MCA-2065)** in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Ms. Meera Rose Mathew

Internal Guide

Ms. Meera Rose Mathew

Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

External Examiner

DECLARATION

I hereby declare that the project report **"FARMCONNECT"** is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Master of Computer Applications (MCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date: **N AMAL THOMSON**

KANJIRAPPALLY **Reg: AJC22MCA-2065**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I extend my sincere gratitude to **Rev. Fr. Dr. Rubin Thottupurathu Jose**, our Head of the Department, for the significant support and assistance provided. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Meera Rose Mathew** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

N AMAL THOMSON

ABSTRACT

FarmConnect is pioneering a groundbreaking digital platform revolutionizing traditional agricultural commerce, bridging the gap between farmers and buyers with unparalleled ease and efficiency. With its seamless integration of Flutter for front-end development and Firebase for back-end management, this platform offers an intuitive user experience while empowering users to access the freshest farm produce directly from growers and producers. At the core of FarmConnect lie key modules meticulously designed to streamline administrative tasks, ensuring smooth operations for both users and administrators. From managing user accounts to moderating product listings, facilitating secure transactions, and maintaining transparency throughout the process, the platform provides comprehensive tools for efficient management.

For farmers, FarmConnect represents a powerful ally, offering them a platform to showcase their products, receive valuable cultivation recommendations tailored to seasonal variations, manage inventory effectively, fulfill orders seamlessly, and access personalized guidance on sustainable farming practices. Simultaneously, buyers benefit from a user-friendly interface enabling them to explore a diverse array of products, place orders effortlessly, track deliveries in real-time, and engage directly with farmers for inquiries and product details. Moreover, FarmConnect takes innovation a step further with the integration of blockchain technology. By leveraging blockchain to store user data on a local blockchain network, the platform ensures decentralization and enhanced security. This integration not only safeguards user information but also reinforces transparency and trust within the ecosystem, establishing FarmConnect as a trailblazer in agricultural tech innovation.

Through its innovative approach, seamless user experience, and integration of cutting-edge technologies like Flutter, Firebase, React JS, Ethereum Blockchain, Ganache, Truffle and Web3, FarmConnect is poised to transform the agricultural sector, fostering a more efficient, transparent, and direct connection between producers and consumers.

CONTENT

LIST OF ABBREVIATION

UML	Unified Modelling Language
IDE	Integrated Development Environment
UI	User Interface
BaaS	Backend-as-a-Service
IoT	Internet of Things
URL	Uniform Resource Locator
APK	Android Package
iOS	iPhone Operating System
ML	Machine Learning
AI	Artificial Intelligence
GB	GigaByte
API	Application Programming Interface
RAM	Random Access Memory
FAQ	Frequently Asked Questions
JSON	JavaScript Object Notation
NoSQL	Not Only SQL
SQL	Structured Query Language
etc	et cetera
SDK	Software Development Kit

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

In the ever-changing world of agriculture, FarmConnect has emerged as a revolutionary solution that connects farmers and buyers directly through a sophisticated digital platform. Built with care and powered by Flutter for an intuitive user experience and Firebase for a strong back-end infrastructure, FarmConnect goes beyond traditional boundaries to offer a transformative journey for everyone involved in agriculture. FarmConnect also includes an Admin Dashboard for web powered by React JS.

The agricultural industry has long been characterized by complex supply chains and middlemen, creating inefficiencies that hurt both farmers and consumers. FarmConnect enters this arena as a catalyst for change, using cutting-edge technologies to redefine how fresh farm products are grown, shown, and finally delivered to consumers. This digital ecosystem envisions a future where transparency, efficiency, and sustainability are the cornerstones of agricultural commerce.

FarmConnect is more than just a marketplace; it's a dynamic hub that empowers farmers to expand their reach, adopt sustainable practices, and thrive in the digital age. At the same time, it gives buyers an easy-to-use interface, offering a wide variety of farm products while fostering direct connections with growers. Let's explore the details of FarmConnect's key modules and features that work together to reshape the agricultural landscape.

1.2 PROJECT SPECIFICATION

FarmConnect, an innovative digital platform powered by Flutter for the front-end and Firebase for the back-end, is a transformative force in the realm of agricultural commerce. By seamlessly connecting farmers and buyers, FarmConnect reshapes traditional supply chains, facilitating the direct purchase of fresh farm products. Let's delve into the key modules and features that make FarmConnect a comprehensive solution for both farmers and buyers.

- **Admin**
    - Authentication (Email/Google)
    - User Management (Enable/Disable)
    - Dashboard
    - Approve Farmers
    - Product Approval

- Email Notification
- Facilitate Payment
- Manage User Data on Blockchain
- Add Category and Products

- **Farmer**
  - Authentication (Email/Google)
  - Profile Management (Reset Password/Update Profile)
  - Dashboard
  - Add Products
  - Stock Management
  - Facilitate Payment
  - Bill Generation
  - Email Notification
  - Provide Latest Agricultural News
  - Section for Organic Products
- **Buyers**
  - Authentication (Email/Google)
  - Profile Management (Reset Password/Update Profile)
  - Dashboard
  - Product Search
  - Order Placement
  - Make Payment
  - Download Bills
  - Section for Organic Products
  - Wishlist

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

System analysis is an important part of developing any system. It involves gathering and studying data, solving problems, and making recommendations for changes. Communication between the people who will use the system and those who will develop it is key during this process.

Before making any changes, it's important to understand how the current system works. This includes looking at all the parts of the system and how they interact. System analysis helps us understand what the problem is, figure out what's important, and find the best solution.

Preliminary research is the process of gathering and analyzing data in order to use it for upcoming system investigations. Initial research requires strong collaboration between system users and developers since it involves problem-solving. It carries out several feasibility studies. These studies offer a rough idea of the system activities, which can be utilized to choose the methods to employ for effective system research and analysis.

2.2 EXISTING SYSTEM

The existing agricultural commerce system is characterized by a traditional, manual approach that heavily relies on intermediaries and lacks technological integration. Farmers, as primary producers, face challenges in reaching a wider market due to limited visibility and accessibility. Transactions are often opaque, leading to mistrust between buyers and sellers. Manual record-keeping processes contribute to delays and errors in order fulfillment, and communication between farmers and buyers is often inefficient. The dependency on intermediaries increases the overall cost of agricultural products for buyers and reduces profit margins for farmers.

2.2.1 NATURAL SYSTEM STUDIED

The natural system studied involves a comprehensive examination of the agricultural ecosystem, considering the seasonal and geographical aspects of crop cultivation, the inherent challenges faced by farmers, and the flow of produce through various stages of the supply chain. By understanding the natural system, including the interactions between farmers, buyers, and intermediaries, we gain insights into the complexities and opportunities within the agricultural sector.

2.2.2 DESIGNED SYSTEM STUDIED

The designed system, FarmConnect, introduces a paradigm shift in agricultural commerce. Utilizing the Flutter framework for the front-end and Firebase for the back-end, this digital platform

seamlessly connects farmers and buyers. The system is equipped with specialized modules for administrators, farmers, and buyers, each tailored to their specific needs. Key features include a direct connection between farmers and buyers, real-time order tracking, and personalized recommendations based on data-driven insights. Advanced functionalities, such as image recognition for product search, disease detection, and quality assessment tools, enhance the overall user experience.

2.3 DRAWBACKS OF EXISTING SYSTEM

- Limited Market Reach
- Opaque Transactions
- Inefficient Communication
- Manual Processes
- Dependency on Intermediaries:

2.4 PROPOSED SYSTEM

The proposed system, FarmConnect, emerges as a groundbreaking solution to revolutionize the landscape of agricultural commerce. Designed with precision and driven by advanced technologies such as Flutter for the front-end and Firebase for the back-end, FarmConnect aims to bridge the existing gaps in the traditional supply chain. The core of the system lies in fostering a direct connection between farmers and buyers, eliminating the need for intermediaries. This not only enhances transparency in transactions but also establishes a foundation of trust between producers and consumers. The platform's user-friendly interface provides a seamless experience for both farmers and buyers, empowering farmers to showcase their products effectively and buyers to explore a diverse range of fresh farm products. Real-time order tracking and secure payment processes ensure the efficiency of transactions, while advanced features such as image recognition for product search, disease detection, and quality assessment tools add a layer of sophistication to the user experience. FarmConnect goes beyond being a mere marketplace; it aspires to be a digital ecosystem that promotes sustainable farming practices, expands market reach for farmers, and sets new standards in the realm of agricultural commerce.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- Direct Farmer-Buyer Connection
- Efficient Transactions

- Enhanced User Experience
- Transparency and Trust
- Market Expansion for Farmers
- Simplified Administration
- Time Saving
- Additional Layer of Security using Blockchain

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A feasibility study helps decide if a project is worth doing. It looks at whether the project will meet the organization's goals considering the time, effort, and resources needed. This study helps predict how useful the project will be and what its future might look like. Before starting a new project, it's common to do a feasibility analysis. This looks at things like if the project will meet user needs, use resources well, and fit with the organization's plans. The study considers technical, economic, and operational factors to see if the project is viable. The following are its features: -

3.1.1 ECONOMICAL FEASIBILITY

Economically, FarmConnect demonstrates its viability through a well-structured financial analysis. Initial startup costs, encompassing development expenses, marketing efforts, operational setup, and administration, have been carefully estimated. Revenue projections are based on revenue models that include transaction fees, subscription plans, and other income streams, with income projections taking into account user adoption rates. Operating expenses, such as server hosting, maintenance, marketing, and customer support, have been forecasted to ensure sustainable operations. The project's break-even point, where total revenue equals total costs, has been identified, and sensitivity analysis has been conducted to understand how changes in key assumptions may impact financial viability. Potential funding sources, including investment, loans, grants, and partnerships, have been explored to secure the necessary capital for development and initial operations.

3.1.2 TECHNICAL FEASIBILITY

FarmConnect's technical setup is robust, incorporating a combination of technologies to ensure a smooth experience. For mobile app development, Flutter is utilized to create a single app that works on both Android and iOS devices. Additionally, React JS is employed for building the admin dashboard, offering a user-friendly interface and efficient data management. Firebase serves as the backend for both the mobile app and the admin dashboard, providing features like real-time database management and secure user authentication.

The development team is proficient in utilizing these technologies, ensuring effective implementation and seamless integration between the mobile app and the admin dashboard. The technical infrastructure is scalable to accommodate potential growth, with provisions for servers, databases, and hosting services in place. Security measures, including data encryption and robust user authentication, are implemented to protect user information and transactions.

3.1.3 BEHAVIORAL FEASIBILITY

Behaviorally, FarmConnect has been carefully designed to align with user preferences and practices. User acceptance has been assessed to ensure that the platform resonates with both farmers and buyers. User-friendliness and intuitiveness have been emphasized in the platform's design to provide a positive user experience. Continuous user feedback mechanisms are in place to understand user needs, expectations, and concerns, allowing for iterative improvements. The platform actively promotes desirable behavioral changes, such as supporting sustainable farming practices and fostering direct farmer-buyer interactions. User behaviors have been analyzed in the context of competition and market dynamics to adapt to changing conditions.

3.1.4 FEASIBILITY STUDY QUESTIONNAIRE

1. Project Overview:

FarmConnect represents a transformative digital platform poised to revolutionize the agricultural commerce landscape. Developed using the versatile Flutter framework for its front-end and the robust Firebase backend, this platform redefines how farmers and buyers interact in the agricultural sector. Its core objective is to seamlessly connect farmers, who are the primary producers of agricultural products, with buyers who seek fresh, locally sourced goods. FarmConnect strives to empower farmers by expanding their market reach and offers buyers a convenient, transparent, and direct way to access agricultural products.

2. To what extent the system is proposed for?

FarmConnect is an extensive system designed to encompass a broad spectrum of functionalities. It serves as an end-to-end solution for agricultural commerce, integrating tools and features that enhance the entire supply chain. From user management to product listings, order processing, and communication capabilities, it offers a comprehensive set of services. These include facilitating secure transactions, offering personalized recommendations based on user preferences, and promoting sustainable farming practices. The system's reach extends across the entire agriculture sector, from small-scale farmers to larger agricultural enterprises.

3. Specify the Viewers/Public which is to be involved in the System:

FarmConnect's primary audience comprises three key user groups:

- **Farmers**: These are the backbone of the agricultural sector, and FarmConnect empowers them by providing a platform to showcase their products, efficiently manage inventory,

and receive valuable seasonal cultivation recommendations. This not only enhances their market access but also supports sustainable farming practices.

- **Buyers:** Buyers use FarmConnect to access a diverse range of agricultural products. They can easily discover products, place orders, track deliveries in real-time, and engage directly with farmers for inquiries or additional product information. This direct communication fosters trust and transparency in the agricultural supply chain.
- **Admins:** The administrative team ensures the platform's smooth operation. They oversee user accounts, moderate product listings, provide support to users, and manage the overall functionality of the platform. Admins play a crucial role in maintaining the platform's integrity and efficiency.

4. List the Modules included in your System:

FarmConnect consists of three core modules:

- **Admin Module:** This module equips administrators with comprehensive tools to manage user accounts, review and approve product listings, facilitate secure transactions, and provide robust support to farmers and buyers.
- **Farmer Module:** Empowering farmers to present their products, efficiently manage inventory and orders, receive personalized cultivation recommendations, and engage in direct communication with both buyers and administrators.
- **Buyer Module:** Providing buyers with an intuitive interface for product discovery, order placement, real-time delivery tracking, and direct communication with farmers for inquiries and product details.

5. Identify the users in your project:

- **Admins:** Responsible for overseeing the entire platform, admins ensure smooth operations, manage user accounts, moderate product listings, and provide support to users.
- **Farmers:** These primary producers use FarmConnect to showcase their products, manage orders and inventory, receive valuable cultivation recommendations, and communicate directly with buyers and admins.
- **Buyers:** Buyers leverage FarmConnect for discovering, selecting, and purchasing agricultural products, with features including real-time order tracking, communication with farmers, and access to their order history.

6. Who owns the system?

The ownership details typically depend on the organization or individuals who initiated and funded the development of the platform.

- Entrepreneur or Founder
- Start-up Company
- Agricultural Cooperative
- Non-Profit Organization
- Government or Public Entity

7. System is related to which firm/industry/organization?

FarmConnect's direct connection is with the agricultural industry. It serves as a catalyst for improving the efficiency, transparency, and sustainability of agricultural commerce. By connecting farmers and buyers directly, FarmConnect contributes to the growth and prosperity of local agricultural ecosystems and empowers farmers to access broader markets. It aligns with the goals of agricultural organizations, government bodies, and industry associations committed to supporting sustainable farming practices and enhancing market access for growers.

8. Details of the person that you have contacted for data collection:

a. Mostly Online Sources
b. The process of data collection for FarmConnect involves collaboration with a diverse range of stakeholders. It encompasses collecting and curating essential data, such as product information, user profiles, and regional agricultural data. This data collection effort requires engagement with multiple parties, including:
    - Farmers and Agricultural Experts
    - Government Agencies
    - Farmers' Associations
    - Local Agricultural Communities.

9. Questionnaire to collect details about the project:

a. **What types of products do you list or purchase on FarmConnect?**
   I do not list or purchase products on the platform myself. My role primarily involves managing user accounts, monitoring platform activities, and facilitating payment processing.

b. **Are there specific features or functionalities you would like to see added to**

FarmConnect?

Adding additional features like real-time chat support for users, more comprehensive reporting and analytics tools, and tools for promoting sustainable farming practices would be valuable enhancements.

c. **Provide more information about your target user base? Are there any specific demographics or regions you are primarily focusing on?**

FarmConnect aims to cater to a diverse user base that includes:

Farmers: Both small-scale and large-scale farmers who want to expand their market reach.

Buyers: Individuals, restaurants, and businesses looking for quality farm products.

d. **Are there any unique features or functionalities you envision that will set FarmConnect apart from other agricultural commerce platforms?**

To set FarmConnect apart from other agricultural commerce platforms, we envision unique features such as:

- Personalized cultivation recommendations based on local weather data.
- Real-time order tracking for buyers.
- Direct communication channels between farmers and buyers.
- Robust image recognition for product search.
- Integrated disease detection and pest control guidance.

e. **How do you plan to ensure the security of user data and financial transactions on the platform?**

To guarantee the safety of user data and financial transactions, FarmConnect will incorporate:

- Strong encryption protocols.
- Regular security assessments.
- Adherence to industry-leading standards and practices.

f. **Are there any specific technologies or integrations you require for the image recognition, disease detection, and pest control modules?**

We plan to integrate machine learning models and image processing technologies for image recognition, disease detection, and pest control modules. Collaborations with specialized technology providers will be pursued.

g. How do you plan to gather and maintain information on organic products, and how will they be distinguished on the platform?

Information on organic products will be collected through a separate section on the platform. They will be distinguished by relevant certifications and labels, ensuring transparency for buyers.

h. What specific functionalities should the admin dashboard include?

The admin dashboard in the FarmConnect platform should provide comprehensive tools and functionalities to enable administrators to effectively manage user accounts, oversee platform operations, and ensure a smooth and secure experience for all users.

- User Management
- Product Approval
- Reports
- Facilitate Payment
- Manage Bills and Orders

i. Are there any legal or regulatory requirements that the platform must adhere to, especially in the agricultural sector?

Yes, there may be legal and regulatory requirements that the FarmConnect platform must adhere to, especially in the agricultural sector. Compliance with these regulations is essential to ensure the platform operates within the bounds of the law and maintains the trust of users. The specific regulations and requirements can vary by region and country, but some common considerations in the agricultural sector include:

- Data Privacy and Protection
- Food Safety and Quality Standards
- Organic Certification
- Pesticide and Chemical Regulations
- Intellectual Property

j. What types of notifications and alerts should users receive?

Users of the FarmConnect platform should receive various types of notifications and alerts to keep them informed about important events, updates, and interactions on the platform. These notifications are essential for providing a smooth and engaging user experience

- Farmer Approval
- Enable/Disable Updates
- Order Updates
- Payment Updates
- Product Updates

3.2 SYSTEM SPECIFICATION

3.2.1 HARDWARE SPECIFICATION

Device Processor	-	Apple iPhone (Apple Bionic) & Android (Qualcomm Snapdragon, Samsung Exynos)
Device RAM	-	4 GB or more
Device Storage	-	64 GB or more

3.2.2 SOFTWARE SPECIFICATION

Front End	-	Dart, Flutter, React JS
Back End	-	Firebase
Database	-	Firebase Firestore
Client on Device	-	iOS10 and above or Android 7 or above for mobile application & Windows 7 and above for web dashboard.
Technologies used	-	Dart, Flutter, Firebase Firestore, FirebaseStorage, Web3 Truffle, Ganache, React.JS, Ethereum, Blockchain, RazorPay & Firebase APIs

3.3 SOFTWARE DESCRIPTION

3.3.1 FLUTTER

In May 2017, Google introduced Flutter, an innovative mobile UI framework that is free and open-source. Flutter enables developers to create native mobile apps using a single codebase. This advancement allows developers to build separate applications for both iOS and Android platforms using the same programming language and codebase. Flutter comprises two key components:

- **SDK (Software Development Kit):** This toolkit equips you with a set of tools to create applications. It also provides the means to translate your code into native machine code, tailored for both Android and iOS platforms.

- **Framework (UI Library with Widgets):** This encompasses a versatile collection of reusable user interface (UI) elements that you can customize to align with your specific requirements. These elements encompass buttons, text input fields, sliders, and various other interactive objects.

Applications developed with Flutter utilize the Dart programming language. Although Google introduced Dart in October 2011, it has undergone significant advancements since its inception. Dart serves as a front-end coding framework that empowers the creation of programs for both web and mobile platforms.

3.3.2 FIREBASE

Firebase, a service that developers can use to build apps more easily. It offers services and tools to create great apps, grow the user base and hence make money. Firebase was made by Google and uses a technology platform developed by them. Data is stored in documents that look like JSON, which is a type of database called NoSQL.

Key features of firebase:

- **Authentication**: Firebase supports authentication. Passwords, Facebook, Google, Phone Number etc. can be used to authenticate. We can even integrate more than one sign-in method into our app using firebase.
- **Hosting:** Firebase offers the app quick hosting.
- **Test lab:** The app can be tested on physical devices located on Google's data center or even on virtual devices.
- **Performance Monitoring:** This feature helps developers gain insights into the performance of their app, allowing them to optimize it for better user experiences.
- **Cloud Firestore:** This is a flexible, scalable No-SQL cloud database that allows for seamless data storage and synchronization in real-time.
- **Cloud Messaging:** Firebase Cloud Messaging (FCM) enables reliable and efficient delivery of messages to target devices, including iOS, Android, and web.
- **Crashlytics:** Firebase Crashlytics provides detailed crash reports, allowing developers to quickly identify and fix issues in their apps.

3.3.3 REACT JS

React JS is a JavaScript library made by Facebook (META) for developing user interfaces. It helps developers create interactive and dynamic parts of websites easily. React organizes UI elements into reusable components, making code easier to manage. It uses a virtual DOM and one-way data binding to make websites faster and smoother for users. Many developers use React to build modern, responsive, and scalable websites.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System design, a critical phase in the process of application development, playing a pivotal role in shaping architecture and functionality of software. It encompasses the creation of a structured blueprint that outlines how various components, modules, and services within application will interact and collaborate to meet specific objectives. This process involves making crucial decisions regarding database architecture, server configuration, and overall system architecture to ensure scalability, efficiency, and robustness. Additionally, system design takes into account factors like user experience, security, and data management to create a well-rounded and effective application. It requires a deep understanding of the application's requirements, as well as proficiency in various technologies and programming paradigms to construct a robust foundation for the development process.

In essence, system design serves as the architectural backbone of any software project. It involves breaking down the complex functionalities and requirements of the application into manageable components, each with a defined purpose and relationship with other elements. Through meticulous planning and consideration of various technical and user-centric aspects, system design lays the groundwork for developers to implement code efficiently and cohesively. A well-crafted system design not only ensures that the application meets performance and scalability requirements but also provides a framework for future enhancements and maintenance. It is a critical step that bridges the gap between conceptualizing an application and transforming it into a functional, reliable, and user-friendly software solution.

4.2 UML DIAGRAM

Unified Modeling Language (UML), a foundational tool in software engineering, renowned for its role in visually representing complex systems and processes. It provides a standardized set of graphical notations that facilitate the clear depiction of various aspects of a system's structure and behavior. Originating from the collaboration of industry experts, UML has gained widespread acceptance and adoption in both academia and industry. It serves as a powerful communication tool, enabling stakeholders, including developers, designers, and clients, to attain a shared understanding of system architecture, design, and functionality. UML diagrams act as a lingua franca, transcending language barriers and ensuring a consistent means of conveying intricate software concepts, ultimately enhancing the efficiency and effectiveness of the software development process.

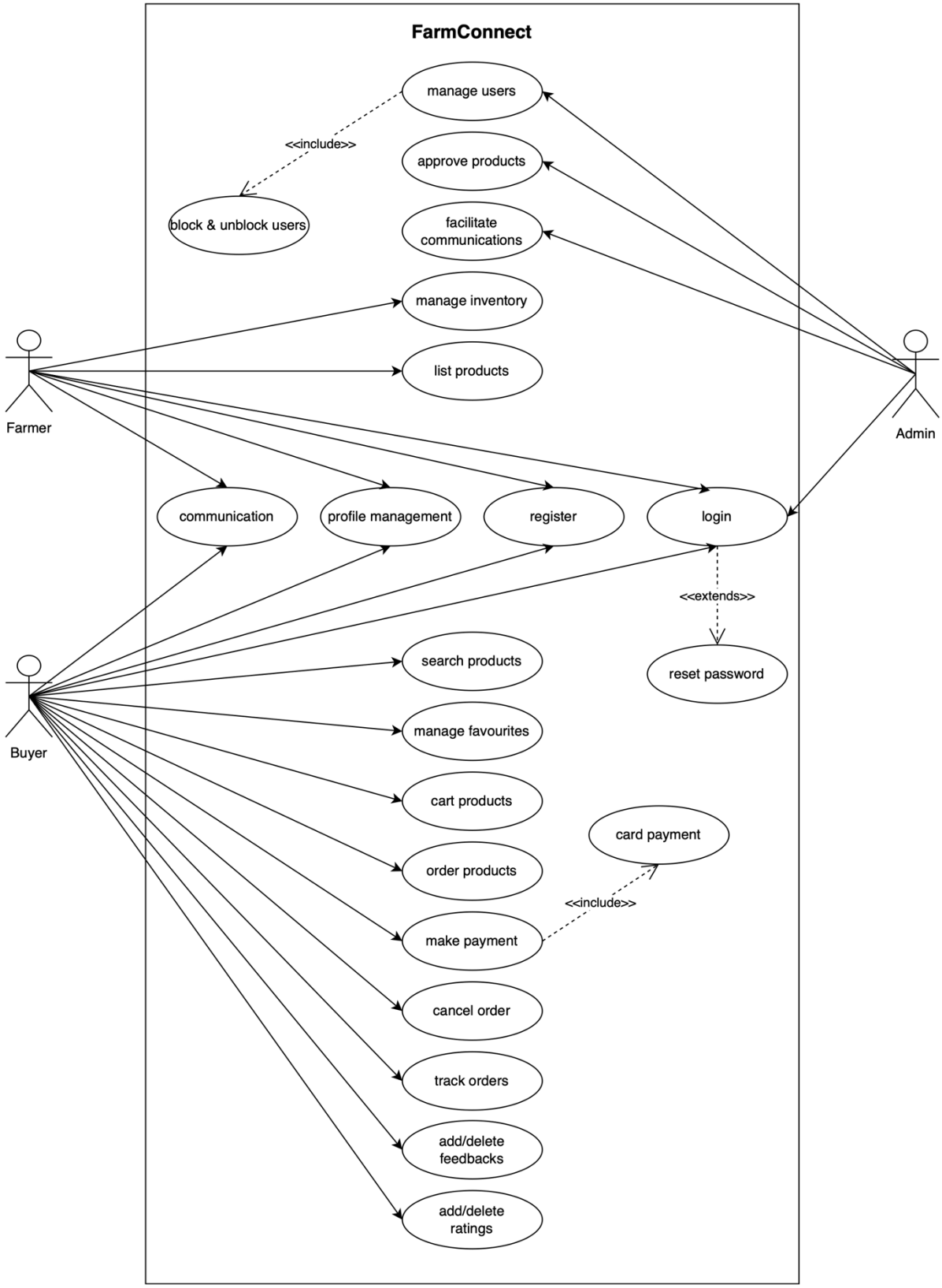
Types of UML diagrams

- Class diagram
- Object diagram
- Use Case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

Use Case Diagrams, a cornerstone in software engineering, serve as a visual representation of the interactions between a system and its external entities. At their core, they provide a structured means of identifying and defining the various functionalities a system offers and how these functionalities are accessed by different actors or entities. Actors, representing users, systems, or external entities, are depicted along with the specific use cases they engage with. Associations between actors and use cases elucidate the nature of these interactions, clarifying the roles and responsibilities of each entity within the system. This detailed visual representation not only enhances communication among stakeholders but also provides a clear blueprint for system functionality, laying the foundation for the subsequent stages of the software development process. Overall, Use Case Diagrams play a pivotal role in aligning development efforts with user expectations, ensuring that the resulting software system fulfills its intended purpose effectively and efficiently.

- **Actor Definition:** Clearly define and label all actors involved in the system. Actors represent external entities interacting with the system.
- **Use Case Naming:** Use descriptive names for use cases to accurately convey the functionality they represent.
- **Association Lines:** Use solid lines to represent associations between actors and use cases. This signifies the interaction between entities.
- **System Boundary:** Draw a box around the system to indicate its scope and boundaries. This defines what is inside the system and what is outside.
- **Include and Extend Relationships:** Use "include" relationships to represent common functionalities shared among multiple use cases. Use "extend" relationships to show optional or extended functionalities.

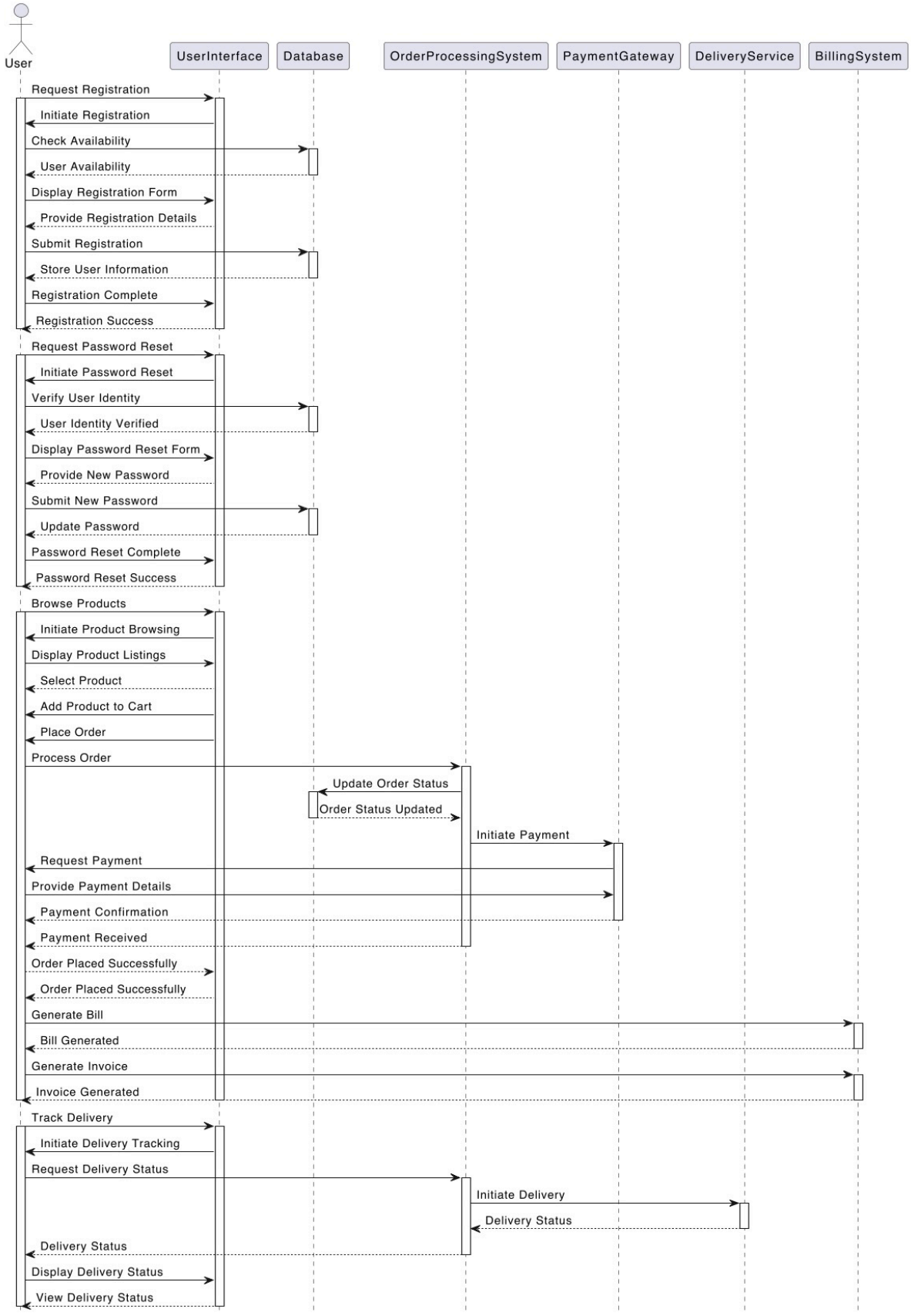


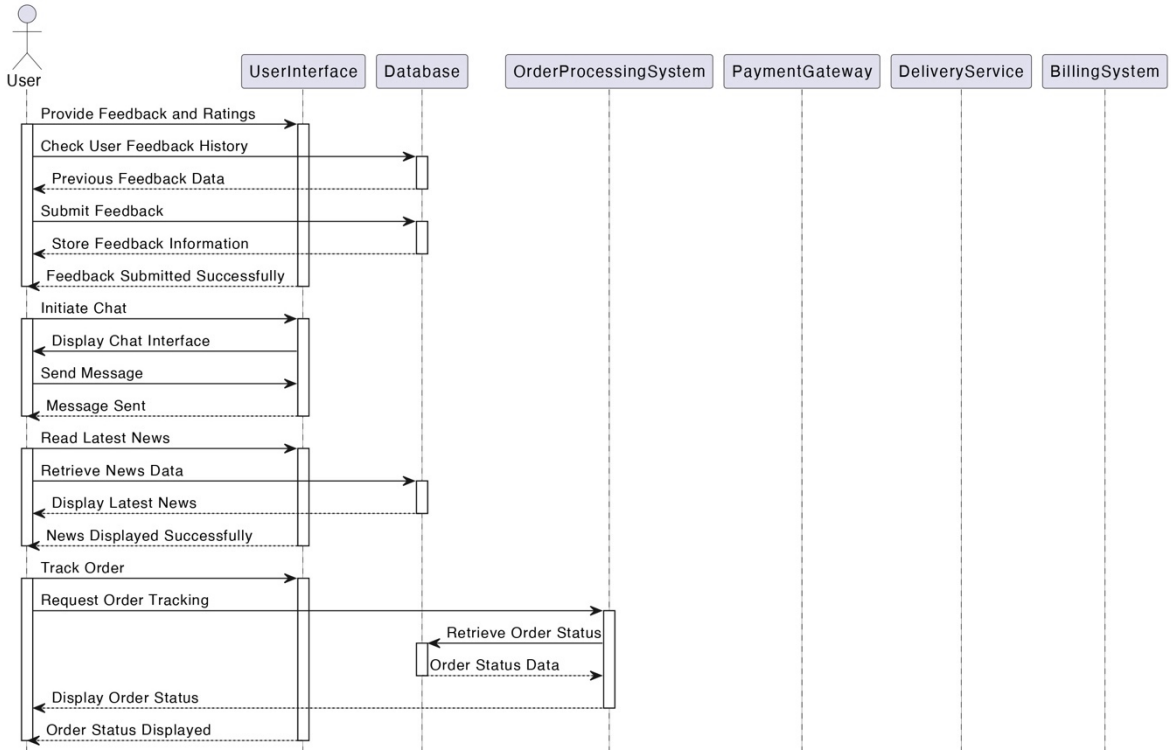
Use Case Diagram

4.2.2 SEQUENCE DIAGRAM

Sequence Diagrams stand as dynamic models in software engineering, portraying the chronological flow of interactions between various objects or components within a system. They spotlight the order in which messages are exchanged, revealing the behavior of the system over time. Actors and objects are represented along a vertical axis, with arrows indicating the sequence of messages and their direction. Lifelines, extending vertically from actors or objects, illustrate their existence over the duration of the interaction. These diagrams serve as a vital tool for visualizing system behaviour and understanding the temporal aspects of a software process. Through Sequence Diagrams, stakeholders gain valuable insights into how different elements collaborate to achieve specific functionalities, facilitating more effective communication among development teams and stakeholders alike. This detailed representation not only aids in detecting potential bottlenecks or inefficiencies but also provides a foundation for refining system performance in the later stages of software development.

- **Vertical Ordering:** Represent actors and objects along a vertical axis, indicating the order of interactions from top to bottom.
- **Lifelines:** Extend vertical lines from actors or objects to denote their existence and participation in the interaction.
- **Activation Bars:** Use horizontal bars along lifelines to show the period during which an object is active and processing a message.
- **Messages and Arrows:** Use arrows to indicate the flow of messages between objects, specifying the direction of communication.
- **Self-Invocation:** Use a looped arrow to represent self-invocation, when an object sends a message to itself.
- **Return Messages:** Indicate return messages with a dashed arrow, showing the response from the recipient.
- **Focus on Interaction:** Sequence Diagrams focus on the chronological order of interactions, avoiding implementation details.
- **Concise Notation:** Use clear and concise notation to represent messages and interactions, avoiding unnecessary complexity.
- **Consider System Boundaries:** Clearly define the boundaries of the system to indicate what is included in the interaction.
- **Feedback and Validation:** Seek feedback from stakeholders and team members to ensure the diagram accurately represents the system behavior.


User
UserInterface
Database
OrderProcessingSystem
PaymentGateway
DeliveryService
BillingSystem
Request Registration
Initiate Registration
Check Availability
User Availability
Display Registration Form
Provide Registration Details
Submit Registration
Store User Information
Registration Complete
Registration Success
Request Password Reset
Initiate Password Reset
Verify User Identity
User Identity Verified
Display Password Reset Form
Provide New Password
Submit New Password
Update Password
Password Reset Complete
Password Reset Success
Browse Products
Initiate Product Browsing
Display Product Listings
Select Product
Add Product to Cart
Place Order
Process Order
Update Order Status
Order Status Updated
Initiate Payment
Request Payment
Provide Payment Details
Payment Confirmation
Payment Received
Order Placed Successfully
Order Placed Successfully
Generate Bill
Bill Generated
Generate Invoice
Invoice Generated
Track Delivery
Initiate Delivery Tracking
Request Delivery Status
Initiate Delivery
Delivery Status
Delivery Status
Display Delivery Status
View Delivery Status



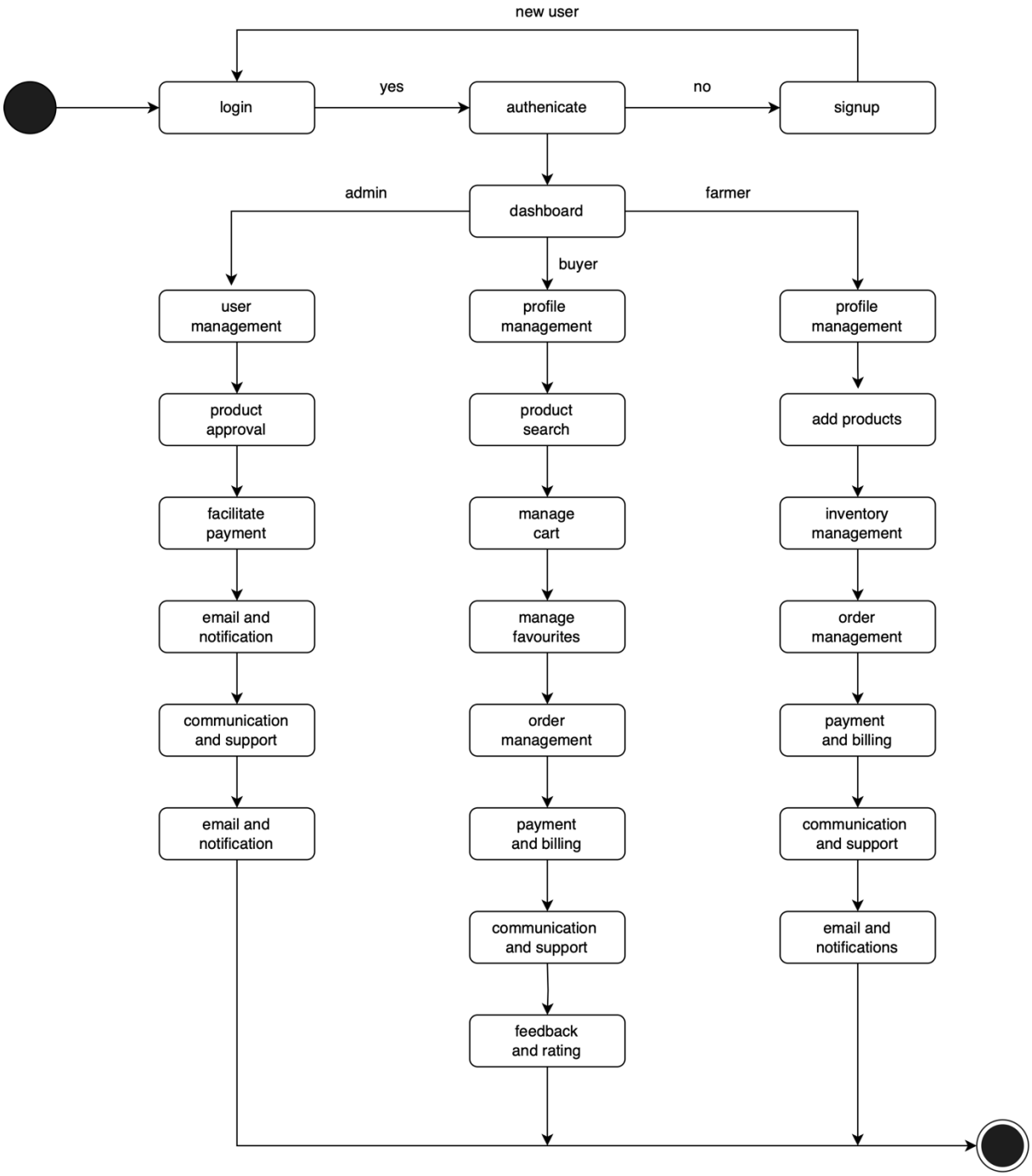
Sequence Diagram

4.2.3 STATE CHART DIAGRAM

A State Chart Diagram, a fundamental component of UML, provides a visual representation of an object's lifecycle states and the transitions between them. It depicts the dynamic behavior of an entity in response to events, showcasing how it transitions from one state to another. Each state represents a distinct phase in the object's existence, while transitions illustrate the conditions triggering state changes. Initial and final states mark the commencement and termination of the object's lifecycle. Orthogonal regions allow for concurrent states, capturing multiple aspects of the object's behavior simultaneously. Hierarchical states enable the representation of complex behaviors in a structured manner. Entry and exit actions depict activities occurring upon entering or leaving a state. Moreover, guard conditions ensure that transitions occur only under specified circumstances. State Chart Diagrams play a crucial role in understanding and designing the dynamic behavior of systems, aiding in the development of robust and responsive software applications.

Key notations for State Chart Diagrams:

- **Initial State:** Represented by a filled circle, it signifies the starting point of the object's lifecycle.
- **State:** Depicted by rounded rectangles, states represent distinct phases in an object's existence.
- **Transition Arrow:** Arrows denote transitions between states, indicating the conditions triggering a change.
- **Event:** Events, triggers for state changes, are labeled on transition arrows.
- **Guard Condition:** Shown in square brackets, guard conditions specify criteria for a transition to occur.
- **Final State:** Represented by a circle within a larger circle, it indicates the end of the object's lifecycle.
- **Concurrent State:** Represented by parallel lines within a state, it signifies concurrent behaviors.
- **Hierarchy:** States can be nested within other states to represent complex behavior.
- **Entry and Exit Actions:** Actions occurring upon entering or leaving a state are labeled within the state.
- **Transition Labels:** Labels on transition arrows may indicate actions or operations that accompany the transition.



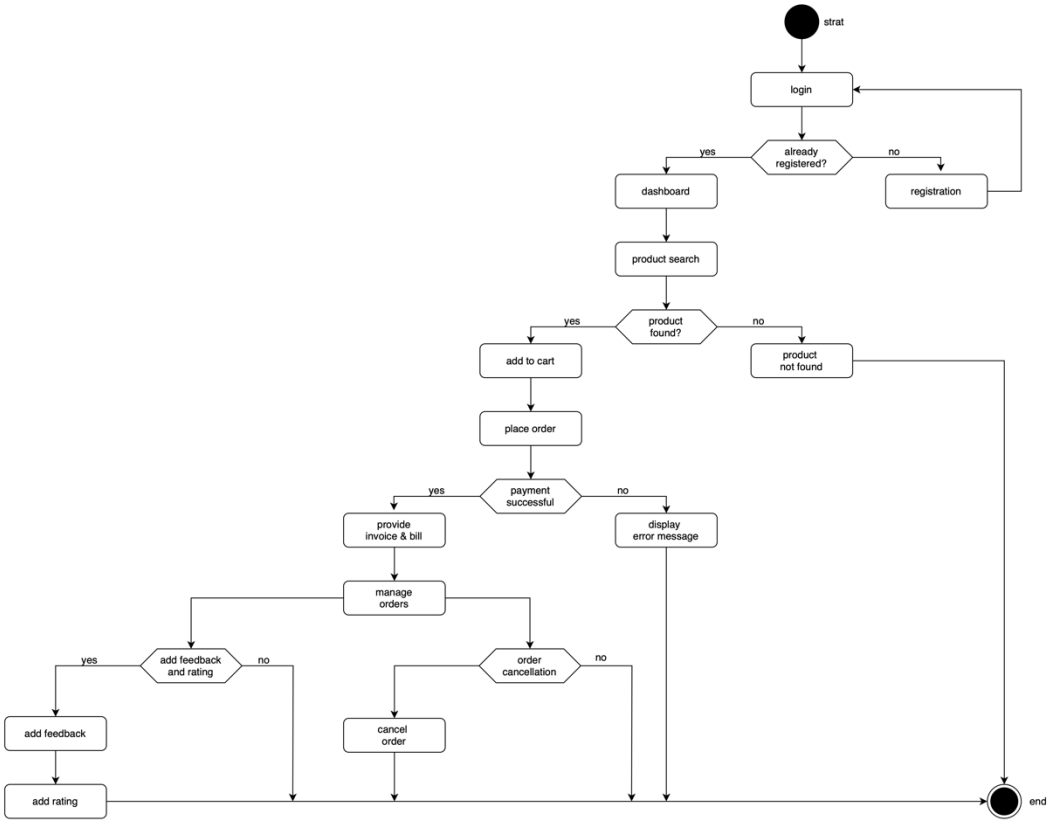
State Chart Diagram

4.2.4 ACTIVITY DIAGRAM

An Activity Diagram is a visual representation within UML that illustrates the flow of activities and actions in a system or process. It employs various symbols to depict tasks, decision points, concurrency, and control flows. Rectangles signify activities or tasks, while diamonds represent decision points, allowing for conditional branching. Arrows indicate the flow of control from one activity to another. Forks and joins denote concurrency, where multiple activities can occur simultaneously or in parallel. Swimlane segregate activities based on the responsible entity, facilitating clarity in complex processes. Initial and final nodes mark the commencement and completion points of the activity. Decision nodes use guards to determine the path taken based on conditions. Synchronization bars enable the coordination of parallel activities. Control flows direct the sequence of actions, while object flows depict the flow of objects between activities. Activity Diagrams serve as invaluable tools for understanding, modeling, and analyzing complex workflows in systems and processes. They offer a structured visual representation that aids in effective communication and system development.

Key notations for Activity Diagrams:

- **Initial Node:** Represented by a solid circle, it signifies the starting point of the activity.
- **Activity:** Shown as a rounded rectangle, it represents a task or action within the process.
- **Decision Node:** Depicted as a diamond shape, it indicates a point where the process flow can diverge based on a condition.
- **Merge Node:** Represented by a hollow diamond, it signifies a point where multiple flows converge.
- **Fork Node:** Shown as a horizontal bar, it denotes the start of concurrent activities.
- **Join Node:** Depicted as a vertical bar, it marks the point where parallel flows rejoin.
- **Final Node:** Represented by a solid circle with a border, it indicates the end of the activity.
- **Control Flow:** Arrows connecting activities, showing the sequence of actions.
- **Object Flow:** Lines with arrows representing the flow of objects between activities.
- **Swimlane:** A visual container that groups activities based on the responsible entity or system component.
- **Partition:** A horizontal or vertical area within a swimlane, further organizing activities.



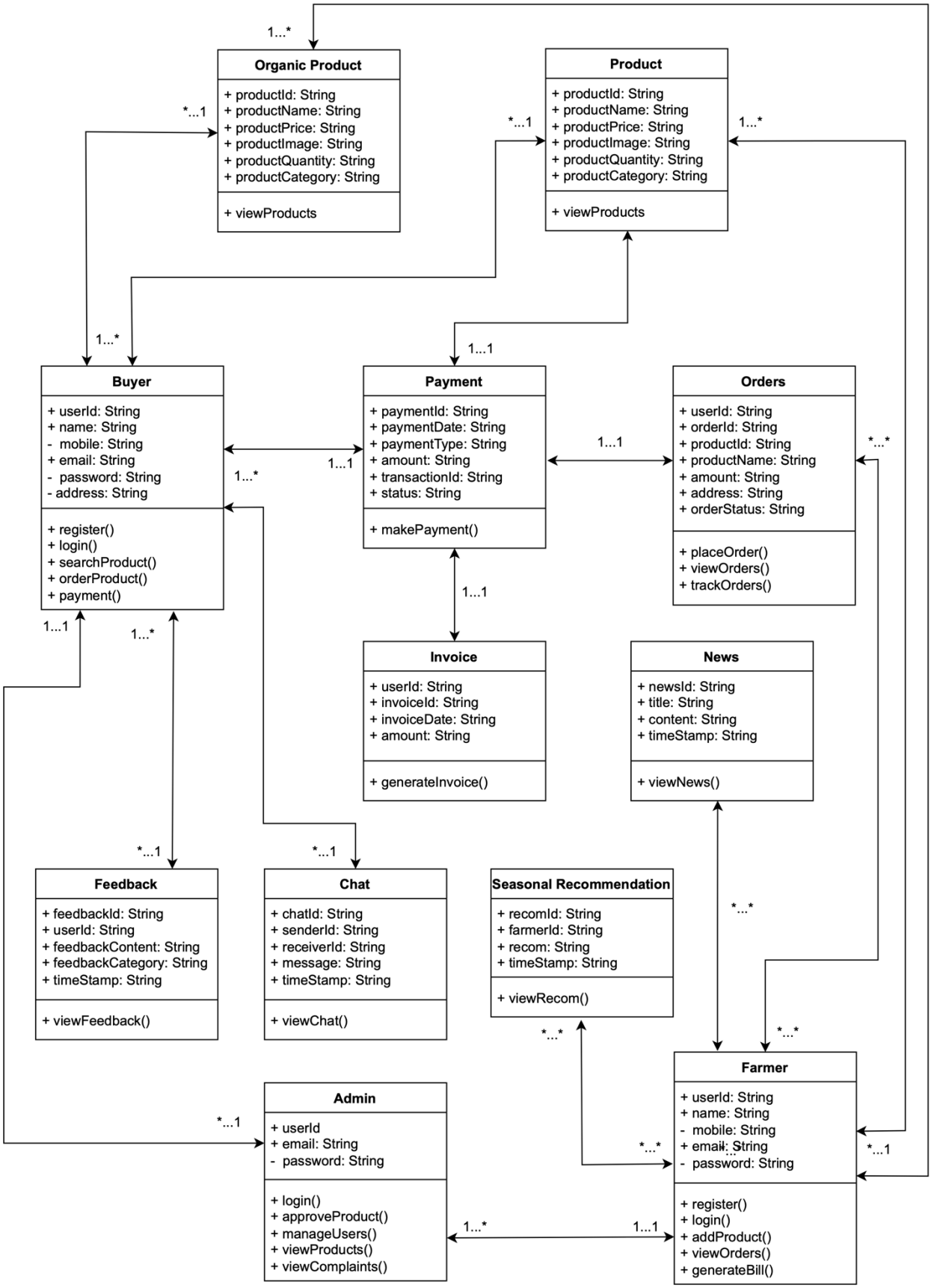
Activity Diagram

4.2.5 CLASS DIAGRAM

A Class Diagram, a fundamental tool in UML, visually represents the structure of a system by illustrating classes, their attributes, methods, and relationships. Classes, depicted as rectangles, encapsulate data and behavior within a system. Associations between classes indicate relationships, showcasing how they interact. Multiplicity notations specify the cardinality of associations. Inheritance is denoted by an arrow indicating the subclass inheriting from a super-class. Aggregation and composition illustrate whole-part relationships between classes. Interfaces, depicted as a circle, outline the contract of behavior a class must implement. Stereotypes provide additional information about a class's role or purpose. Dependencies highlight the reliance of one class on another. Association classes facilitate additional information about associations. Packages group related classes together, aiding in system organization. Class Diagrams play a pivotal role in system design, aiding in conceptualizing and planning software architectures. They serve as a blueprint for the development process, ensuring a clear and structured approach to building robust software systems.

Key notations for Class Diagrams:

- **Class:** Represented as a rectangle, it contains the class name, attributes, and methods.
- **Attributes:** Displayed as a list within the class, they describe the properties or characteristics of the class.
- **Methods:** Also listed within the class, they define the behaviors or operations of the class
- **Associations:** Lines connecting classes, indicating relationships and connections between them.
- **Multiplicity Notation:** Indicates the number of instances one class relates to another.
- **Inheritance:** Shown as an arrow, it signifies that one class inherits properties and behaviors from another.
- **Interfaces:** Represented by a dashed circle, they define a contract of behavior that implementing classes must follow.
- **Stereotypes:** Additional labels or annotations applied to classes to provide more information about their role or purpose.
- **Dependencies:** Shown as a dashed line with an arrow, they indicate that one class relies on another in some way.
- **Association Classes:** Represented as a class connected to an association, they provide additional information about the relationship.



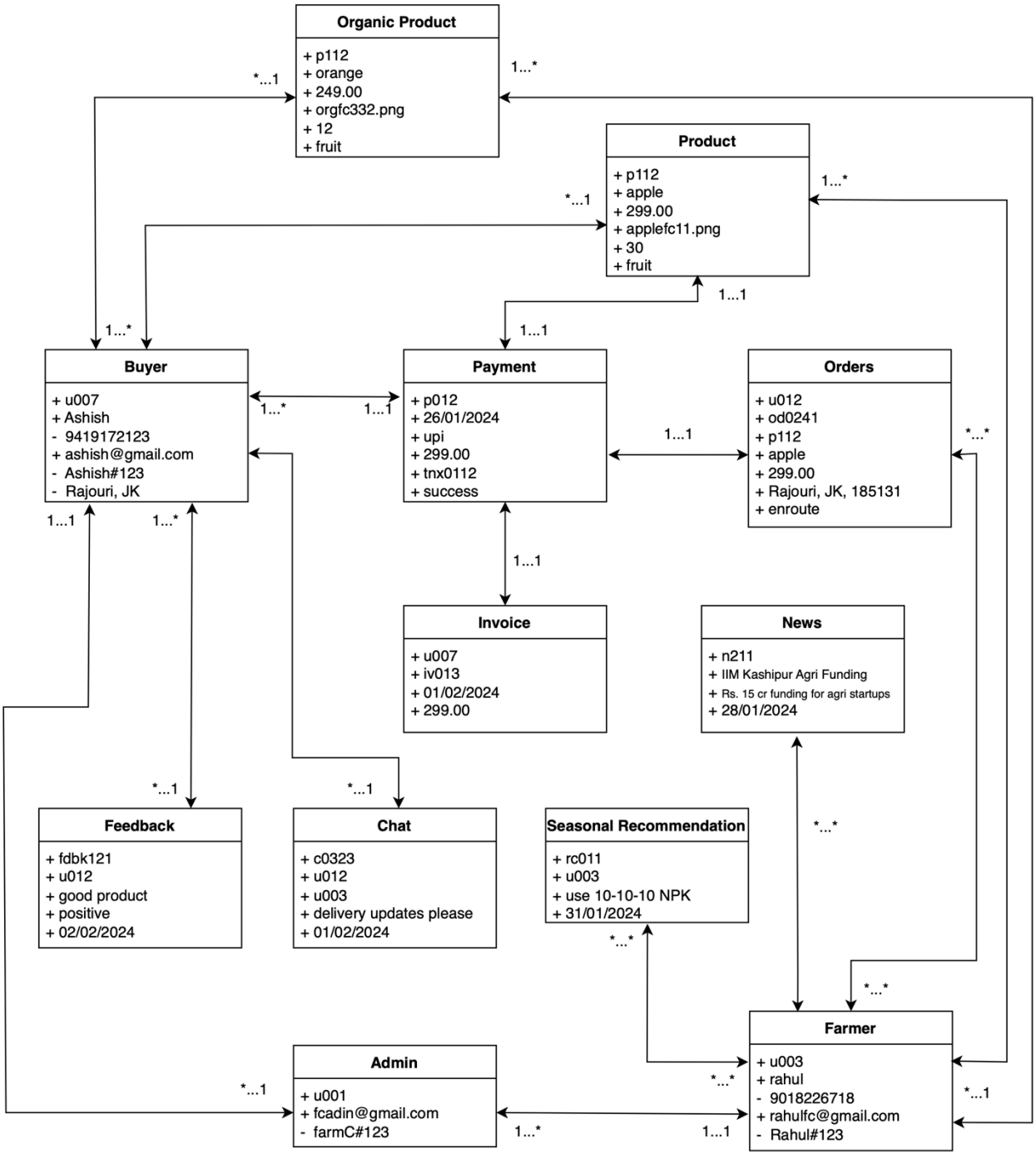
Class Diagram

4.2.6 OBJECT DIAGRAM

An Object Diagram in UML provides a snapshot of a system at a specific point in time, displaying the instances of classes and their relationships. Objects, represented as rectangles, showcase the state and behavior of specific instances. Links between objects depict associations, highlighting how they interact. Multiplicity notations indicate the number of instances involved in associations. The object's state is displayed through attributes and their corresponding values. Object Diagrams offer a detailed view of runtime interactions, aiding in system understanding and testing. They focus on real-world instances, providing a tangible representation of class relationships. While similar to Class Diagrams, Object Diagrams emphasize concrete instances rather than class definitions. They serve as valuable tools for validating system design and verifying that classes and associations work as intended in practice. Object Diagrams play a crucial role in system validation, ensuring that the system's components and their interactions align with the intended design and requirements.

Key notations for Object Diagrams:

- **Object**: Represented as a rectangle, it contains the object's name and attributes with their values.
- **Links**: Lines connecting objects, indicating associations or relationships between them.
- **Multiplicity Notation**: Indicates the number of instances involved in associations.
- **Attributes with Values**: Displayed within the object, they represent the state of the object at a specific point in time.
- **Role Names**: Labels applied to associations, providing additional information about the nature of the relationship.
- **Object Name**: Represents the name of the specific instance.
- **Association End**: Indicates the end of an association, often with a role name and multiplicity.
- **Dependency Arrow**: Indicates a dependency relationship, where one object relies on another.
- **Composition Diamond**: Represents a stronger form of ownership, where one object encapsulates another.
- **Aggregation Diamond**: Signifies a whole-part relationship between objects.



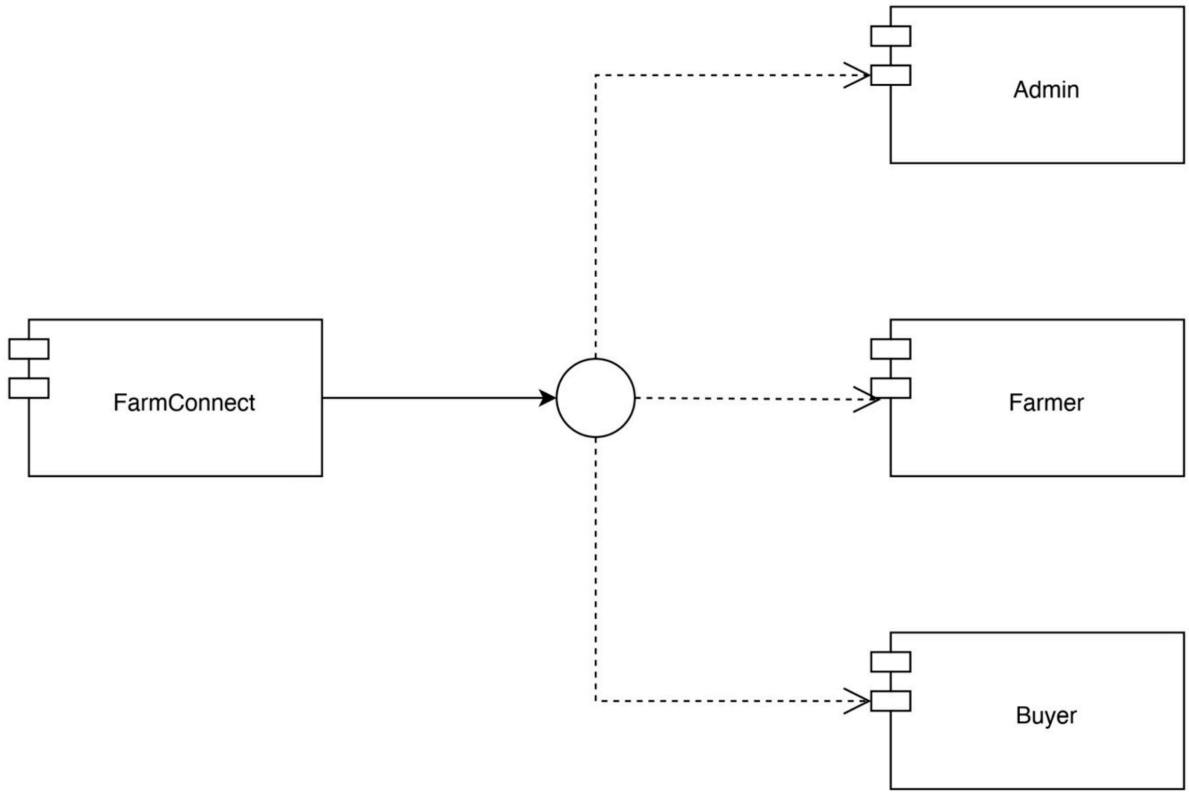
Object Diagram

4.2.7 COMPONENT DIAGRAM

A Component Diagram, a vital aspect of UML, offers a visual representation of a system's architecture by showcasing the high-level components and their connections. Components, depicted as rectangles, encapsulate modules, classes, or even entire systems. Dependencies between components are displayed through arrows, signifying the reliance of one component on another. Interfaces, represented by a small circle, outline the services a component offers or requires. Connectors link interfaces to denote the required or provided services. Ports, depicted as small squares, serve as connection points between a component and its interfaces. Stereotypes provide additional information about the role or purpose of a component. Deployment nodes indicate the physical location or environment in which components are deployed. Component Diagrams are instrumental in system design, aiding in the organization and visualization of system architecture. They emphasize the modular structure, facilitating ease of development, maintenance, and scalability of complex software systems. Overall, Component Diagrams play a pivotal role in planning and orchestrating the architecture of sophisticated software applications.

Key notations for Component Diagrams:

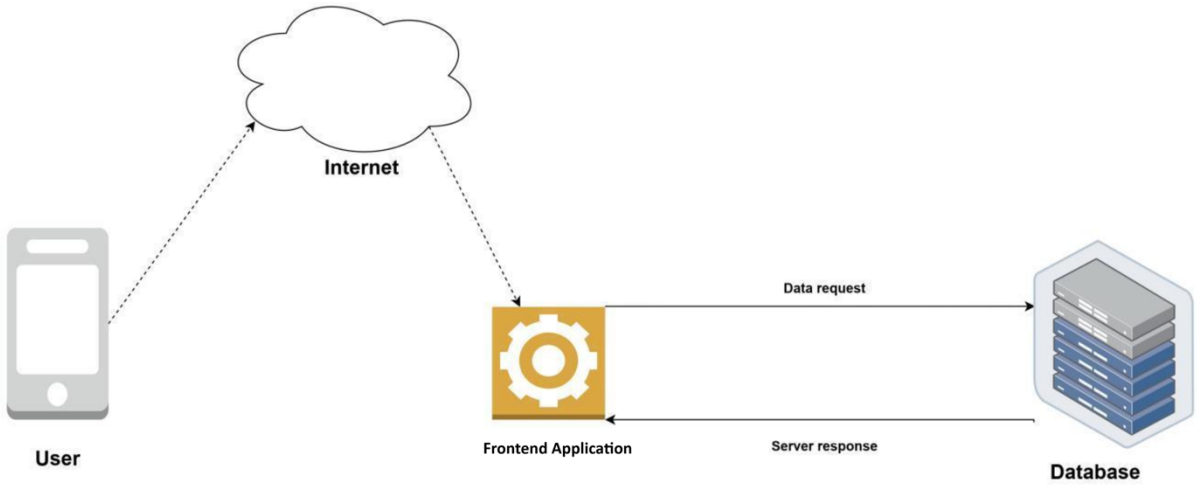
- **Component**: Represented as a rectangle, it encapsulates a module, class, or system.
- **Dependency Arrow**: Indicates that one component relies on or uses another.
- **Interface**: Depicted as a small circle, it outlines the services a component offers or requires.
- **Provided and Required Interfaces**: Connectors link provided interfaces to required interfaces.
- **Port**: Shown as a small square, it serves as a connection point between a component and its interfaces.
- **Stereotypes**: Additional labels or annotations applied to components to provide more information about their role or purpose.
- **Assembly Connector**: Represents the physical connection between two components.
- **Deployment Node**: Indicates the physical location or environment in which components are deployed.
- **Manifestation Arrow**: Indicates the implementation of an interface by a component



Component Diagram

4.2.8 DEPLOYMENT DIAGRAM

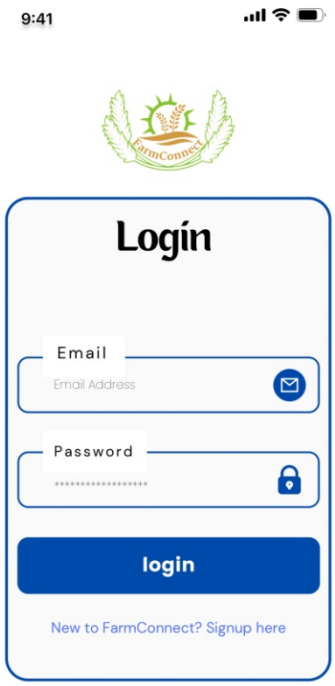
A Deployment Diagram, a crucial facet of UML, provides a visual representation of the physical architecture of a system, showcasing the hardware nodes and software components. Nodes, representing hardware entities like servers or devices, are depicted as rectangles. Artifacts, denoted by rectangles with a folded corner, represent software components or files deployed on nodes. Associations between nodes and artifacts indicate the deployment of software on specific hardware. Dependencies illustrate the reliance of one node on another. Communication paths, shown as dashed lines, represent network connections between nodes. Stereotypes provide additional information about the role or purpose of nodes and artifacts. Deployment Diagrams are instrumental in system planning, aiding in the visualization and organization of hardware and software components. They emphasize the allocation of software modules to specific hardware nodes, ensuring efficient utilization of resources. Overall, Deployment Diagrams play a pivotal role in orchestrating the physical infrastructure of complex software applications.



Deployment Diagram

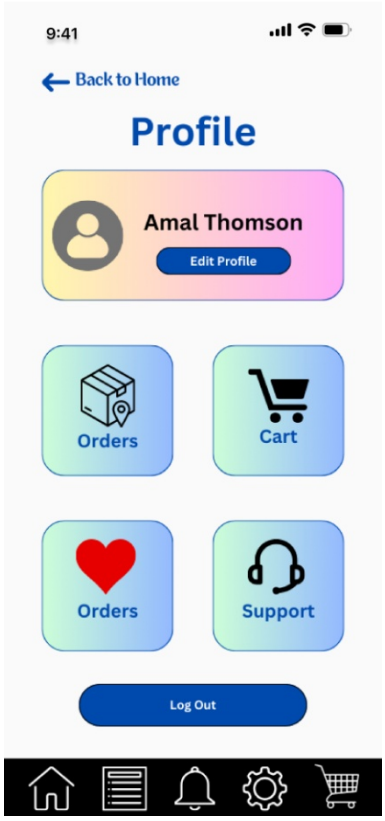
4.3 USER INTERFACE DESIGN USING FIGMA

4.3.1 Splash Screen

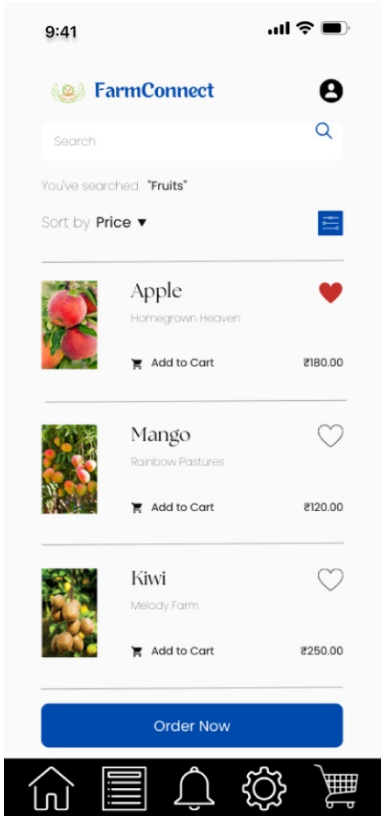


4.3.2 Login Screen

4.3.3 SignUp Page

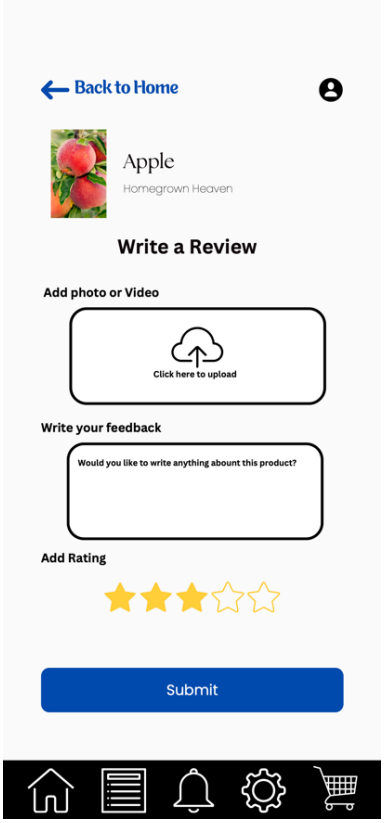


4.3.4 Profile Page

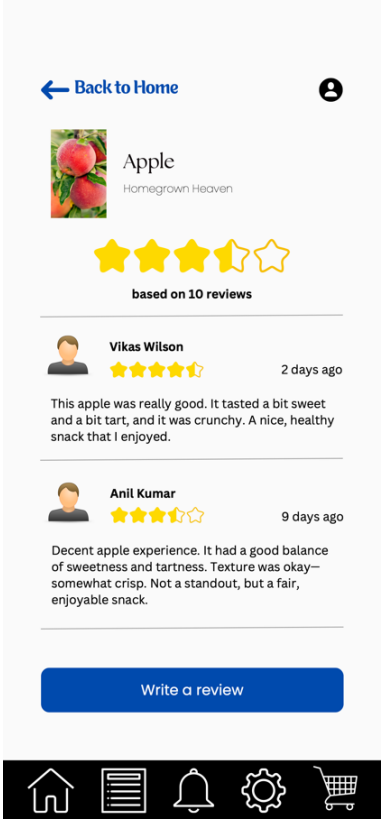


4.3.5 Products Listing

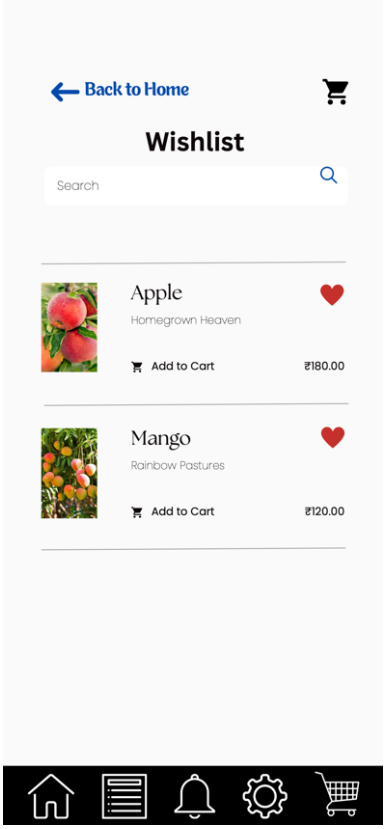
9.3.6 About Page



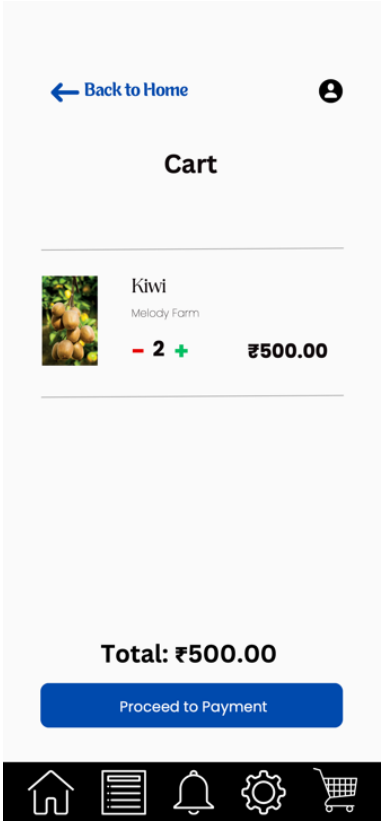
4.3.7 Add Feedback



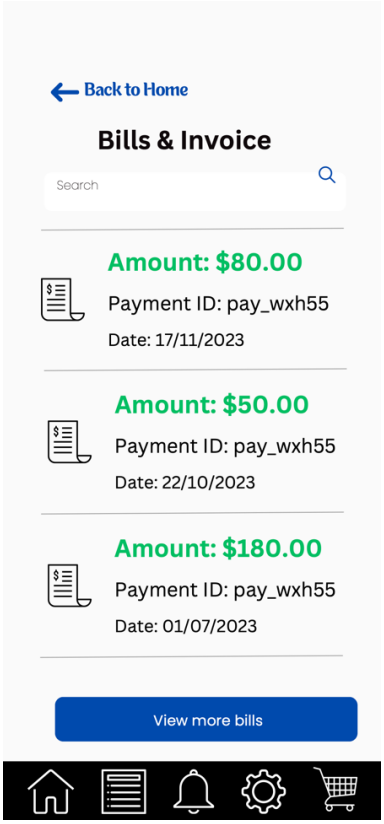
4.3.8 View Feedback



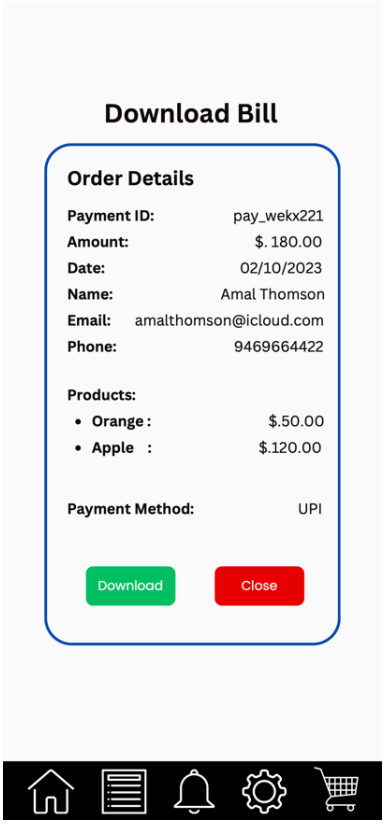
4.3.9 Wishlist



4.3.10 Cart



4.3.11 View Bills



4.3.12 Download Bills

4.4 DATABASE DESIGN

Database Design is a critical component in the realm of information management and software development. It involves the thoughtful and systematic organization of data to ensure efficient storage, retrieval, and manipulation. A well-designed database serves as the backbone of applications, enabling them to handle large volumes of information with speed and accuracy. This process encompasses defining the structure, relationships, and constraints of data entities, optimizing for performance and scalability. Effective database design is pivotal in minimizing redundancy, ensuring data integrity, and providing a foundation for robust data analytics. It involves a deep understanding of business requirements and user needs, translating them into a coherent and logical data model. The goal of a sound database design is to create a reliable, scalable, and maintainable system that supports the organization's objectives and facilitates seamless information flow.

4.4.1 NOSQL DATABASES (NOSQL)

NoSQL database, also popularly known as "Not Only SQL," offer a modern approach to managing data. Unlike traditional relational databases, they specialize in handling large amount of unstructured or semi structured data. NoSQL databases are designed to scale easily, making them suitable for handling rapidly growing data volumes in web applications and Big Data environments. They usually comes in different types, including document-oriented, key-value stores, wide-column stores, and graph databases, each serving specific purposes. Document-oriented databases, like MongoDB, are particularly useful for storing JSON-like documents, making them popular choices for content management systems and real-time analytics.

One key advantage of NoSQL databases is their schema flexibility. They allow for dynamic or semi-structured schemas, enabling data to be added or modified on the fly. This characteristic is invaluable in projects where data structures are likely to evolve over time. NoSQL databases have found wide adoption in domains like social media, IoT applications, gaming, and real-time analytics. However, it's crucial to choose between NoSQL and traditional relational databases based on the unique requirements of the application. In essence, NoSQL databases offer a potent alternative, providing scalability, flexibility, and high performance in scenarios demanding the handling of large volumes of diverse data. Their diverse types make them indispensable tools in modern data management.

4.4.2 INDEXING

Indexing in NoSQL databases is a crucial technique to enhance query performance and retrieval speed. Unlike relational databases, NoSQL databases utilize a variety of indexing methods tailored to different data models. In document-oriented databases like MongoDB, B-tree indexes are commonly used to accelerate search operations based on keys or fields within documents. Hash indexes, on the other hand, are prevalent in key-value stores like Redis, enabling swift retrieval of values associated with specific keys. Wide-column stores like Cassandra utilize techniques like row-level indexing to swiftly locate specific columns within a wide row. Graph databases like Neo4j employ specialized index structures optimized for graph traversal operations, allowing for rapid traversal of connected nodes. While indexing significantly improves read performance, it's important to weigh the trade-offs, as indexes can increase storage requirements and potentially slow down write operations. In summary, indexing plays a vital role in optimizing query performance and is a key aspect of designing efficient data retrieval systems in NoSQL databases.

4.4.3 TABLE/COLLECTION DESIGN

4.4.3.1 Collection Name: users

Field Name	Data Type	Field Description
uid	String	Unique identifier for user
email	String	Email address of user
name	String	Full name of user
phone	String	Phone number of user
ftl	String	Flag indicating whether the user is First Time Login
isActive	String	Indicates whether the user account is active
isAdminApproved	String	Indicates whether the user is approved as an admin
gender	String	Gender
role	String	Role
profileImageUrl	String	URL pointing to user's profile image

street	String	street details of user
town	String	Town of user's location
district	String	District of user's location
state	String	State of the user's location
pincode	String	Pincode or postal code of user's location

Collection: users

4.4.3.2 Collection Name: products

Field Name	Data Type	Field Description
userId	String	Unique identifier for the user who added product
productId	String	Unique identifier for product
productName	String	Name of product
category	String	Category of product
productPrice	String	Price of product
productDescription	String	Description of product
productImage	String	URL pointing to product image
isApproved	String	Approval status of product,
stock	String	Stock quantity of product
remark	String	Additional remarks or comments from Admin

Collection: products

4.4.3.3 Collection Name: cart

Field Name	Data Type	Field Description
userId	String	Unique identifier for user who added the product
productId	String	Unique identifier for product
productName	String	Name of product
productDescription	String	Description of product
productImage	String	URL pointing to image of the product
productPrice	String	Price of product
quantity	String	Quantity of the product in the cart

Collection: cart

4.4.3.4 Collection Name: payment

Field Name	Data Type	Field Description
userUid	String	Unique identifier for user
paymentId	String	Unique identifier for payment
customerName	String	Name of customer
customerEmail	String	Email address of customer
customerPhone	String	Phone number of customer
amount	Number	The total amount of transaction
timestamp	Timestamp	Timestamp of transaction
products	Array	Array of products included in transaction
products[i]	Map	Map representing the details of the first product in the array

products[i].productId	String	ID of first product
products[i].productName	String	Name of first product
products[i].quantity	Number	Quantity of first product
products[i].totalPrice	Number	Total price of first product
products[i].unitPrice	Number	Unit price of first product

Collection: payment

4.4.3.5 Collection Name: orders

Field Name	Data Type	Field Description
orderid	String	unique order id
userid	String	unique ID generated by firebase
orderdate	String	date of order
orderstatus	String	status of order
totalamount	String	total price of ordered items
invoiceid	String	unique invoice id
paymentmethod	Stirng	method of payment
paymentstatus	String	status of payment
email	String	email of user

Collection: orders

4.4.3.6 Collection Name: reviews

Field Name	Data Type	Field Description
reviewid	String	unique id for each review
userid	String	unique id generated by firebase
reviewcontent	String	content of review
timestamp	String	time of review creation
ratings	String	ratings of particular product

Collection: reviews

4.4.3.7 Collection Name: reviews

Field Name	Data Type	Field Description
userId	String	Unique identifier for user who added product
productId	String	Unique identifier for product
productName	String	Name of product
productDescription	String	Description of product
productImage	String	URL pointing to image of the product
productPrice	String	Price of product

Collection: wishlist

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

System Testing is an important step in making software. It's when we check if the whole system works like it's supposed to. We look at everything to make sure it meets the requirements and does what it's supposed to do. This includes testing how all the parts of the system work together. We check if it does what it's supposed to and if it's easy to use. System Testing happens in an environment that's like the real world, so we can see how it works in practice. The main goal is to find and fix any problems before people start using the software. We do this by testing it a lot and keeping good records. This helps make sure the software we make is reliable and works well.

Testing is the systematic process of running a program to uncover potential errors or flaws. An effective test case possesses a high likelihood of revealing previously unnoticed issues. A test is considered successful when it reveals a previously unidentified error. If a test functions as intended and aligns with its objectives, it can detect flaws in the software. The test demonstrates that the computer program is operating in accordance with its intended functionality and performing optimally. There are three primary approaches to assessing a computer program: evaluating for accuracy, assessing implementation efficiency, and analyzing computational complexity.

5.2 TEST PLAN

A test plan is a thorough document that delineates the strategy, scope, objectives, resources, schedule, and expected outcomes for a specific testing endeavor. It functions as a guiding framework for carrying out testing activities, guaranteeing that every facet of the testing process is methodically organized and executed. Additionally, the test plan establishes the roles and responsibilities of team members, outlines the required testing environment, and sets forth the criteria for the successful completion of testing activities. This document plays a pivotal role in ensuring that the testing phase is conducted in a structured and effective manner, ultimately contributing to the overall success of the project.

The levels of testing include:

- Integration Testing
- Unit testing
- Validation Testing or System Testing
- Output Testing or User Acceptance Testing
- Automation Testing
- Widget Testing

5.2.1 UNIT TESTING

Unit Testing is not only a meticulous examination of discrete units or components within a software system but also an indispensable quality assurance measure. This phase serves as a crucial foundation for the entire software testing process, where the focus lies on isolating and scrutinizing individual units of code. The objective remains unwavering: to verify that each unit performs its designated function accurately, yielding precise outputs for predefined inputs.

Moreover, Unit Testing operates independently, detached from other components, and any external dependencies are either emulated or replaced by "mock" objects, ensuring controlled evaluation. This meticulous process establishes a robust foundation for the software, confirming that each unit functions reliably and adheres meticulously to its predefined behavior.

The significance of Unit Testing cannot be overstated, as it acts as a vanguard against potential discrepancies or errors early in the development cycle. This proactive approach not only fortifies the integrity and reliability of the software but also lays the groundwork for subsequent testing phases, thereby fostering a robust and dependable software solution. This meticulous process ensures that each unit functions reliably and adheres precisely to its defined behavior. By subjecting individual code units to rigorous scrutiny, any discrepancies or errors are identified and rectified early in the development cycle, bolstering the overall integrity and reliability of the software.

5.2.2 INTEGRATION TESTING

Integration Testing stands as a pivotal phase in the software testing process, dedicated to scrutinizing the interactions and interfaces among diverse modules or components within a software system. Its primary objective is to ascertain that individual units of code seamlessly converge to create a unified and functional system. In stark contrast to unit testing, which assesses individual units in isolation, integration testing delves into the interplay between these units, with a keen eye for any disparities, communication glitches, or integration hurdles. By subjecting the integrated components to rigorous testing, development teams aim to affirm that these elements function cohesively, addressing any potential issues before deployment. This systematic evaluation is instrumental in ensuring that the software operates as an integrated whole, free from any unforeseen conflicts or errors that may arise from the convergence of individual modules.

5.2.3 VALIDATION TESTING OR SYSTEM TESTING

Validation Testing places the end-users at the forefront of evaluation, ensuring that the software aligns precisely with their anticipated needs and expectations. This phase stands distinct from other testing methodologies, as its primary objective is to authenticate that the software, in its final form, serves its intended purpose seamlessly within the real-world scenarios it was designed for. As a culmination of the testing process, Validation Testing carries the responsibility of confirming that the software not only meets the defined technical specifications but also delivers genuine value to its users. It does so by scrutinizing the software against the backdrop of actual usage, thereby fortifying its readiness for deployment. Moreover, in Validation Testing, user stories and acceptance criteria form the cornerstone of assessment. Stakeholders' expectations are meticulously validated, ensuring that every specified requirement is met. Additionally, beta testing, a common practice in this phase, involves a select group of end-users testing the software in a live environment, providing invaluable feedback that can inform potential refinements.

5.2.4 OUTPUT TESTING OR USER ACCEPTANCE TESTING.

Output Testing, also known as Results Validation, is a critical phase in the software testing process. Its primary focus is to verify the correctness and accuracy of the output generated by a software application. The goal is to ensure that the system produces the expected results for a given set of inputs and conditions.

Key aspects of Output Testing include:

- **Comparison with Expected Results**: This phase involves comparing the actual output of the software with the expected or predefined results.
- **Test Case Design:** Test cases are designed to cover various scenarios and conditions to thoroughly evaluate the accuracy of the output.
- **Validation Criteria**: The criteria for validating the output are typically defined during the requirements and design phase of the software development process.
- **Regression Testing**: Output Testing often includes regression testing to ensure that changes or updates to the software do not affect the correctness of the output.
- **Data Integrity**: It verifies that data is processed and displayed correctly, without any corruption or loss.
- **Precision and Completeness**: Output Testing assesses not only the precision of the results but also their completeness in addressing the requirements.

- **Error Handling**: It evaluates how the system handles errors or exceptions and ensures that appropriate error messages are displayed.

5.2.5 AUTOMATION TESTING

Automation Testing stands as a cornerstone in the software testing process, harnessing the power of automated tools and scripts to meticulously execute test cases. In stark contrast to manual testing, which hinges on human intervention, automation testing brings forth a streamlined approach, employing software to conduct repetitive, intricate, and time-consuming tests. This methodology not only heightens operational efficiency but also significantly diminishes the likelihood of human error, ensuring precise and reliable results. Moreover, it empowers thorough testing across a diverse array of scenarios and configurations, from browser compatibility to load and performance assessments.

By automating the testing process, organizations can realize a myriad of benefits. It enables the seamless execution of regression tests, providing confidence that existing functionalities remain intact after each round of enhancements or modifications. Furthermore, automation facilitates the concurrent execution of multiple tests, thereby expediting the overall testing cycle. This approach is particularly invaluable in environments characterized by rapid development and frequent software updates, such as Agile and DevOps setups.

5.2.6 WIDGET TESTING

Widget Testing in Flutter is a critical step towards building robust and reliable user interfaces. By isolating and scrutinizing individual widgets, developers gain confidence in the functionality and appearance of each component. This level of granularity allows for precise testing, ensuring that widgets respond correctly to various user interactions and scenarios.

One of the key advantages of Widget Testing is the ability to mock dependencies. This means that external services or resources that the widget relies on can be simulated, allowing for controlled and predictable testing environments. Additionally, developers can set expectations and employ assertions to validate the widget's behavior under different conditions. Furthermore, Widget Testing is seamlessly integrated with popular testing frameworks like `flutter_test`. This ensures that tests can be organized efficiently and run as part of the automated testing process, providing rapid feedback on the status of widgets.

Another important facet of Widget Testing is the inclusion of Golden Tests. These tests capture screenshots of the widget's visual appearance and compare them against reference images. This helps maintain visual consistency, making sure that UI elements render consistently across different devices and screen sizes.

TEST CASE 1

CODE:

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:farmconnect/main.dart' as app;
void main() {
  group('User Login Test', () {
    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
    testWidgets("Login Scenarios", (WidgetTester widgetTester) async {
      app.main();
      await widgetTester.pumpAndSettle(Duration(seconds: 2));
      final getStartedBtn = find.byKey(Key('getStartedBtn'));
      await widgetTester.tap(getStartedBtn);
      await widgetTester.pumpAndSettle();
      final emailField = find.byKey(const ValueKey('email'));
      final passwordField = find.byKey(const ValueKey('password'));
      final loginButton = find.byKey(const ValueKey('login'));
      print("Scenario 1: Press login button without entering anything");
      await widgetTester.tap(loginButton);
      await widgetTester.pumpAndSettle(Duration(seconds: 5));
      expect(find.text("TEST PASS"), findsNothing);
      print("Scenario 2: Enter wrong password and press login");
      await widgetTester.enterText(emailField, "amalthomson@gmail.com");
      await widgetTester.enterText(passwordField, "amal123");
      await widgetTester.tap(loginButton);
      await widgetTester.pumpAndSettle(Duration(seconds: 5));
      expect(find.text("TEST PASS"), findsNothing);
      print("Scenario 3: Enter correct email and password");
      await widgetTester.enterText(emailField, "amalthomson@icloud.com");
      await widgetTester.enterText(passwordField, "Amal#123");
      await widgetTester.tap(loginButton);
      await widgetTester.pumpAndSettle(Duration(seconds: 5));
      expect(find.text("TEST PASS"), findsNothing);
      print("All scenarios passed!");
    });
  });
}
```

SCREENSHOT TEST CASE 1

User Login Functionality

TEST REPORT

Test Case 1

Project Name: FarmConnect					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 14/03/2024**		
Module Name: Login			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Login Scenario			**Test Execution Date: 21/03/2024**		
Description: Login Functionality Test - User					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Navigate to login screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login without any credentials		Error message should display	displayed error message	Pass
3	Enter wrong password then press login	amalthomson@icloud.com Amal123	Invalid credential message should shown	displayed error message	Pass
4	Enter valid Email and password	amalthomson@icloud.com Amal#123	No error should displayed	No error occurred	Pass
Post-Condition: After successful validation with the database, the user is able to login into their account. The session details associated with the account are then appropriately logged into the database for future reference and security purposes. This confirms that the user's interaction with system is authenticated and systematically tracked, ensuring a secure and reliable user experience.					

TEST CASE 2

CODE:

```
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:farmconnect/main.dart' as app;
void main() {
  group('Password Update Test', () {
    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
    testWidgets("Password Update Scenarios", (WidgetTester tester) async {
      try {
        await tester.pumpAndSettle(Duration(seconds: 2));
        expect(find.byType(SplashScreen), findsOneWidget);
        print("Step 1: Splash screen shown");
        final emailField = find.byKey(const ValueKey('email'));
        final passwordField = find.byKey(const ValueKey('password'));
        final loginButton = find.byKey(const ValueKey('login'));
        await tester.tap(emailField);
        await tester.enterText(emailField, "amalthomson007@gmail.com");
        await tester.tap(passwordField);
        await tester.enterText(passwordField, "Amal#123");
        print("Step 2: Entered email and password");
        await tester.tap(loginButton);
        print("Step 3: Tapped 'Login' button");
        final profileButton = find.byKey(const ValueKey('profileButton'));
        await tester.tap(profileButton);
        print("Step 4: Tapped on profile");
        final resetPasswordButton = find.byKey(const ValueKey('resetPasswordButton'));
        await tester.tap(resetPasswordButton);
        print("Step 5: Tapped on reset password");
        final oldPasswordField = find.byKey(const ValueKey('oldPassword'));
        final newPasswordField = find.byKey(const ValueKey('newPassword'));
        final confirmPasswordField = find.byKey(const ValueKey('confirmPassword'));
        await tester.enterText(oldPasswordField, "old_password");
        await tester.enterText(newPasswordField, "new_password");
        await tester.enterText(confirmPasswordField, "new_password");
        print("Step 6: User validated & Entered new password");
        final updatePasswordButton = find.byKey(const ValueKey('updatePasswordButton'));
        await tester.tap(updatePasswordButton);
        print("Step 7: Tapped on entered password");
        expect(find.text("Password updated successfully"), findsOneWidget);
        print("Step 8: Password updated successfully");
        bool testPassed = true;
        printResult(testPassed);
    });
  });
}
```

SCREENSHOT TEST CASE 2

```
Step 1: Splash screen shown
Step 2: Entered email and password
Some possible finders for the widgets at Offset(418.4, 489.4):
  find.byType(GlowingOverscrollIndicator)
  find.byType(SingleChildScrollView)
  find.byType(KeyedSubtree)
  find.byType(CustomMultiChildLayout)
  find.byType(PhysicalModel)
  find.byElementType(InheritedModelElement<Object>)
Some possible finders for the widgets at Offset(387.1, 472.9):
  find.byType(InkResponse)
  find.byType(IconButton)
  find.byType(IconButtonTheme)
  find.byKey(const Key('LoginPassword'))
  find.byType(Column)
  find.byType(WillPopScope)
  find.byElementType(InheritedModelElement<Object>)
Some possible finders for the widgets at Offset(418.4, 524.7):
  find.byType(GlowingOverscrollIndicator)
  find.byType(SingleChildScrollView)
  find.byType(KeyedSubtree)
  find.byType(CustomMultiChildLayout)
  find.byType(PhysicalModel)
  find.byElementType(InheritedModelElement<Object>)
Some possible finders for the widgets at Offset(331.4, 535.7):
  find.byType(GlowingOverscrollIndicator)
  find.byType(SingleChildScrollView)
  find.byType(KeyedSubtree)
  find.byType(CustomMultiChildLayout)
  find.byType(PhysicalModel)
  find.byElementType(InheritedModelElement<Object>)
┌─────────────────────────────────────────────────────────────────────────────────────────────
│ #0   _LoginScreenState.signInWithEmailAndPasswordAndCheckEmailVerification
│ #1   <asynchronous suspension>
├┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄
│ 🐛 active
└─────────────────────────────────────────────────────────────────────────────────────────────
Error: uid is null or empty


Step 3: Tapped 'Login' button
Step 4: Tapped on profile
Step 5: Tapped on reset password'
```

```
Step 6:User validated & Entered new password


Step 7: Tapped on entered password'
Step 8: Password updated successfully'
Test Passed
√ User Test Password Update
Exited
```

Reset Password Functionality

TEST REPORT

Test Case 2

Project Name: FarmConnect					
Reset Password Test Case					
Test Case ID: Test_2			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): Medium			**Test Designed Date: 14/03/2024**		
Module Name: Buyer			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Reset Password			**Test Execution Date: 21/03/2024**		
Description: Test Reset Password Functionality					
Pre-Condition: User holds a valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Splash Screen Shown		Splash Screen Shown	Splash Screen Shown	Pass
2	Navigation from splash screen		Navigate to Login Screen	Navigated to Login Screen	Pass
3	Entered correct email and password	amalthomson007@gmail.com Amal#123	No error message to be displayed	Error message not displayed	Pass
4	Tap 'Login' button		Navigate to buyer dashboard	Navigated to buyer dashboard	Pass
5	Tap profile button		Navigate to buyer profile	Navigated to buyer profile	Pass
6	Tap reset password button		Navigate to password reset page	Navigated to reset password page	Pass
7	Enter new password	Amal@1234	Password changed without any error	Password changed without any error	Pass
Post-Condition: After successful validation with the database, the user is able to log into their account. User can interact with the buttons and functionalities and reset the password.					

TEST CASE 3

CODE:

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:farmconnect/main.dart' as app;
void main() {
  group('Farmer Approval Test', () {
    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
    testWidgets("Farmer Approval Scenarios", (WidgetTester tester) async {
      try {
        app.main();
        Color green = Color.fromARGB(255, 0, 255, 0);
        Color red = Color.fromARGB(255, 255, 0, 0);
        await tester.pumpAndSettle(Duration(seconds: 2));
        expect(find.byType(SplashScreen), findsOneWidget);
        print("Step 1: Splash screen shown");
        final emailField = find.byKey(const ValueKey('email'));
        final passwordField = find.byKey(const ValueKey('password'));
        final loginButton = find.byKey(const ValueKey('login'));
        await tester.tap(emailField);
        await tester.enterText(emailField, "amalthomson007@gmail.com");
        await tester.tap(passwordField);
        await tester.enterText(passwordField, "Amal#123");
        print("Step 2: Entered email and password");
        await tester.tap(loginButton);
        print("Step 3: Tapped 'Login' button");
        final pendingApprovalButton = find.byKey(const ValueKey('pendingApprovalButton'));
        await tester.tap(pendingApprovalButton);
        await tester.pumpAndSettle();
        print("Step 4: Tapped on pending farmer approval");
        final approveFarmerButton = find.byKey(const ValueKey('approveFarmerButton'));
        await tester.tap(approveFarmerButton);
        await tester.pumpAndSettle();
        print("Step 5: Tapped on approve Farmer");
        expect(find.text("Farmer approved successfully"), findsOneWidget);
        print("Step 6: Farmer approved successfully");
        expect(find.text("All details saved successfully"), findsOneWidget);
        print("Step 7: All details saved successfully");
        bool testPassed = true;
        printResult(testPassed);
    });
  });
}
```

SCREENSHOT TEST CASE 3

```
Step 1: Splash screen shown
Step 2: Entered email and password
Some possible finders for the widgets at Offset(414.9, 534.5):
  find.byType(GlowingOverscrollIndicator)
  find.byType(SingleChildScrollView)
  find.byType(KeyedSubtree)
  find.byType(CustomMultiChildLayout)
  find.byType(PhysicalModel)
  find.byElementType(InheritedModelElement<Object>)
```

```
Step 3: Tapped 'Login' button
Step 4: Tapped pending farmer approvals button
Step 5: Tapped on approve farmer'
Step 6: Farmer approved successfully


Step 7: All details saved successfully'
Test Passed
✓ User Test Admin test
Exited
```

Farmer Approval Functionality

TEST REPORT

Test Case 3

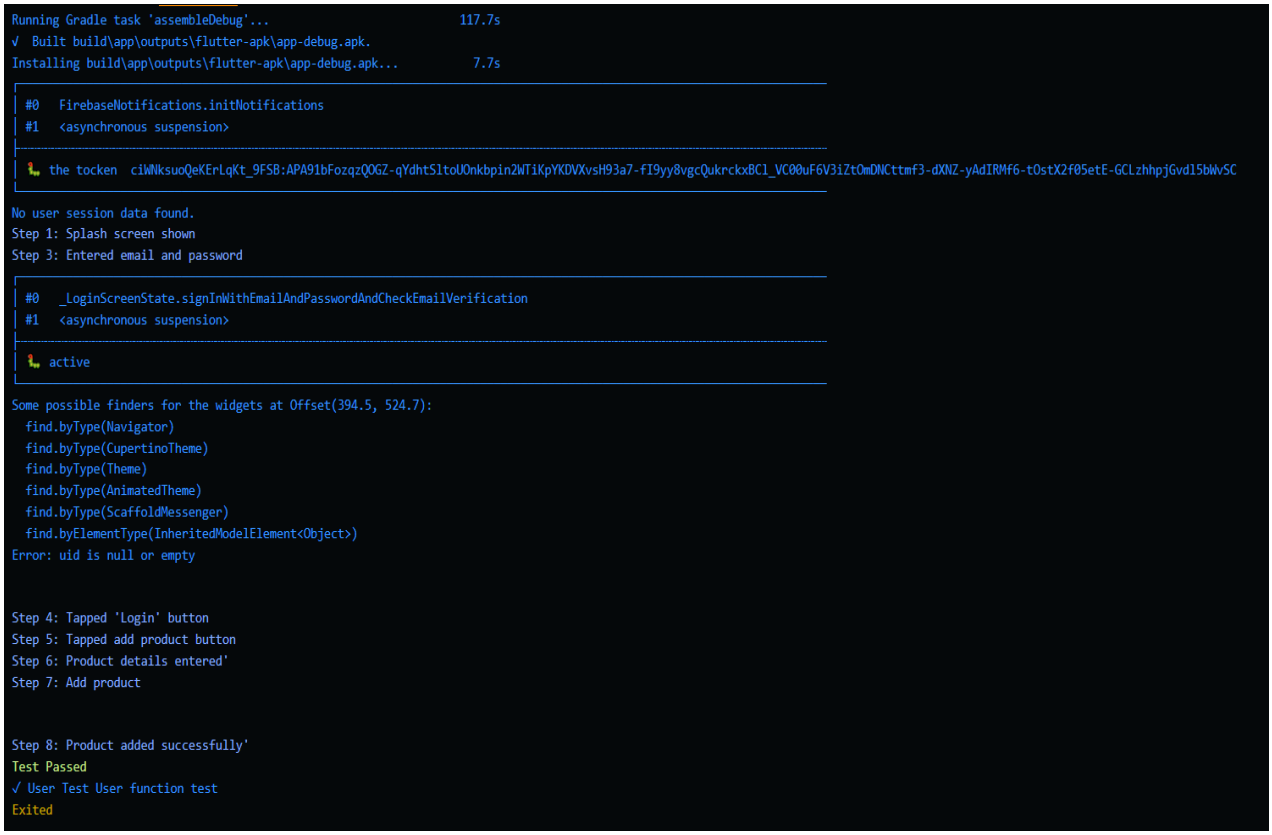
Project Name: FarmConnect					
Farmer Approval Test Case					
Test Case ID: Test_3			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 14/03/2024**		
Module Name: Admin			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Farmer Approval			**Test Execution Date: 21/03/2024**		
Description: Test Pending Farmer Approval Functionality					
Pre-Condition: User holds a valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Splash Screen Shown		Splash Screen Shown	Splash Screen Shown	Pass
2	Navigation from splash screen		Navigate to Login Screen	Navigated to Login Screen	Pass
3	Entered correct email and password	amalthomson@icloud.com Amal#123	No error message to be displayed	Error message not displayed	Pass
4	Tap 'Login' button		Navigate to admin dashboard	Navigated to admin dashboard	Pass
5	Tap pending farmer approvals button		Navigate to pending farmer approvals page	Navigated to pending farmer approvals page	Pass
6	Tap approve button	Sreerag K U sreeragku@gmail.com	Successfully Approve Farmer	Successfully Approve Farmer	Pass
7	Display SnackBar Farmer Approved Successfully	Successfully Approved	Display SnackBar Farmer Approved Successfully	Displayed SnackBar Farmer Approved Successfully	Pass
Post-Condition: After successful validation with the database, the user is able to login into their account. User can interact with the buttons and functionalities and approve the pending farmer approval requests.					

TEST CASE 4

CODE:

```
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:farmconnect/main.dart' as app;
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
  group('Product Listing Test', () {
    testWidgets("Product Listing Test", (WidgetTester tester) async {
      try {
        await tester.pumpAndSettle(Duration(seconds: 2));
        print("Step 1: Splash screen shown");
        final emailField = find.byKey(const ValueKey('email'));
        final passwordField = find.byKey(const ValueKey('password'));
        final loginButton = find.byKey(const ValueKey('login'));
        await tester.tap(emailField);
        await tester.enterText(emailField, "amalthomson007@gmail.com");
        await tester.tap(passwordField);
        await tester.enterText(passwordField, "Amal#123");
        print("Step 3: Entered email and password");
        await tester.tap(loginButton);
        print("Step 4: Tapped 'Login' button");
        final addProductsButton = find.byKey(Key('addProductsButton'));
        await tester.tap(addProductsButton);
        await tester.pumpAndSettle();
        print("Step 5: Tapped add product button");
        final productPriceField = find.byKey(const ValueKey('productPrice'));
        final stockField = find.byKey(const ValueKey('stock'));
        final productDescriptionField = find.byKey(const ValueKey('productDescription'));
        await tester.enterText(productPriceField, "10.00");
        await tester.enterText(stockField, "50");
        await tester.enterText(productDescriptionField, "Fresh and organic Apples");
        print("Step 6: Product details enetered");
        final uploadProductButton = find.byKey(const ValueKey('uploadProductButton'));
        await tester.tap(uploadProductButton);
        print("Step 7: Add product");
        print("Product added successfully");
        expect(find.text("Product added successfully"), findsOneWidget);
        expect(find.byType(ProductItemWidget), findsOneWidget);
        bool testPassed = true;
        printResult(testPassed);
        }
      }
    });
  });
}
```

SCREENSHOT TEST CASE 4



Add Products Functionality

TEST REPORT

Test Case 4

Project Name: FarmConnect					
Add Products Test Case					
Test Case ID: Test_4			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 14/03/2024**		
Module Name: Farmer			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Add Products			**Test Execution Date: 21/03/2024**		
Description: Test Add Products Functionality					
Pre-Condition: User has a correct username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Splash Screen Shown		Splash Screen Shown	Splash Screen Shown	Pass
2	Navigation from splash screen		Navigate to Login Screen	Navigated to Login Screen	Pass
3	Entered valid email and password	thomsonamal@gmail.com Amal#1234	No error message to be displayed	Error message not displayed	Pass
4	Tap 'Login' button		Navigate to admin dashboard	Navigated to admin dashboard	Pass
5	Tap add products button		Navigate to pending farmer approvals page	Navigated to pending farmer approvals page	Pass
5	Enter Products Details	10.00 50 Fresh and organic Apples	No error message to be displayed	Error message not displayed	
7	Tap add product button		Successfully Approve Farmer	Successfully Approve Farmer	Pass
8	Display SnackBar Product Added Successfully	Successfully Added	Display SnackBar Product Added Successfully	Displayed SnackBar Product Added Successfully	Pass
Post-Condition: After successful validation with the database, the user is able to login into their account. User can interact with the buttons and functionalities and add the product.					

TEST CASE 5

CODE:

```
const { Given, When, Then, After } = require('cucumber');
const assert = require('assert');
const { Builder, By, Key, until } = require('selenium-webdriver');
console.log("Scenario: Login Test");
Given('I am on the login page', async function () {
    this.driver = await new Builder().forBrowser('chrome').build();
    await this.driver.get('http://localhost:3000/login');
    console.log('Navigated to the login page');
});
When('I enter valid credentials', async function () {
    await this.driver.findElement(By.id('username')).sendKeys('admin');
    await this.driver.findElement(By.id('password')).sendKeys('adminpassword');
    console.log('Entered valid credentials');
});
When('click on the login button', async function () {
    await this.driver.findElement(By.id('login_button')).click();
    console.log('Clicked on the login button');
});
Then('I should be redirected to the admin dashboard', async function () {
    await this.driver.wait(until.urlIs('http://localhost:3000/admin_dashboard'), 5000);
    const currentUrl = await this.driver.getCurrentUrl();
    assert.strictEqual(currentUrl, 'http://localhost:3000/admin_dashboard');
    console.log('Successfully redirected to the admin dashboard');
    console.log('Login test passed!');
});
After(async function () {
    if (this.driver) {
        await this.driver.quit();
    }
});
```

SCREENSHOT TEST CASE 5

```
amalthomson@amalthomsons-MacBook-Air Main Project % cd farmconnect_web
amalthomson@amalthomsons-MacBook-Air farmconnect_web % cd farmconnect_web_testing
amalthomson@amalthomsons-MacBook-Air farmconnect_web_testing % npm test

> testing@1.0.0 test
> cucumber-js features/*.feature

Scenario: Login Test

✔ Navigated to the Admin Login Page Successfully
.Entered Valid Credentials
.Clicked on the Login Button
.Successfully redirected to the Admin Dashboard
✔ Login Test Passed!
..

1 scenario (1 passed)
4 steps (4 passed)
```

Admin Login Functionality

TEST REPORT

Test Case 5

Project Name: FarmConnect					
Admin Login Test					
Test Case ID: Test_5			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 14/03/2024**		
Module Name: Web Dashboard			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Login Scenario			**Test Execution Date: 21/03/2024**		
Description: Test Web Admin Login Page					
Pre-Condition: Admin has a username and password which is valid					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Navigate to Login Screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login with valid credentials	amalthomson@icloud.com Amal#123	Admin should be navigated to the dashboard	Dashboard displayed	Pass
Post-Condition: After successful validation with the database, the admin is able to login into their account.					

TEST CASE 6

CODE:

```
const { Given, When, Then } = require('cucumber');
const assert = require('assert');
const { Builder, By, Key, until } = require('selenium-webdriver');
const GREEN = '\x1b[32m';
const RESET = '\x1b[0m';
Given('I am logged in to the admin dashboard', async function () {
    this.driver = await new Builder().forBrowser('chrome').build();
    await this.driver.get('http://localhost:3000/admin_dashboard');
    console.log(`${GREEN}✔ Given I am logged in to the admin dashboard${RESET}`);
});
When('I navigate to the product approval page', async function () {
    await this.driver.findElement(By.id('product_approval_link')).click();
    console.log(`${GREEN}✔ When I navigate to the product approval page${RESET}`);
});
When('I select a product to approve', async function () {
    await this.driver.findElement(By.className('product_checkbox')).click();
    console.log(`${GREEN}✔ When I select a product to approve${RESET}`);
});
When('I click on the approve button', async function () {
    await this.driver.findElement(By.id('approve_button')).click();
    console.log(`${GREEN}✔ When I click on the approve button${RESET}`);
});
Then('the product should be approved and made available for users', async function () {
    const productStatus = await this.driver.findElement(By.id('product_status')).getText();
    assert.strictEqual(productStatus, 'Approved');
    console.log(`${GREEN}✔ The product should be approved and made available for users`);
    console.log(`${GREEN}Product approval test passed!${RESET}`);
});
After(async function () {
    if (this.driver) {
        await this.driver.quit();
    }
});
```

SCREENSHOT TEST CASE 6

```
amalthomson@amalthomsons-MacBook-Air farmconnect_web % cd farmconnect_web_testing
amalthomson@amalthomsons-MacBook-Air farmconnect_web_testing % npm test

> testing@1.0.0 test
> cucumber-js features/*.feature

Scenario: Approve Product

✔ Admin Logged in to the Admin Dashboard Successfully
.✔ Navigate to the Product Approval Page
.✔ Successfully Select a Product to Approve
.✔ Click on the Approve Button Success
.✔ Then the Product is Approved and made Available for Users
Product Approval Test Passed!
.

1 scenario (1 passed)
5 steps (5 passed)
```

Approve Product Functionality

TEST REPORT

Test Case 6

Project Name: FarmConnect					
Approve Product Test Case					
Test Case ID: Test_6			**Test Designed By: N Amal Thomson**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 14/03/2024**		
Module Name: Admin			**Test Executed By: Ms. Meera Rose Mathew**		
Test Title : Product Approval			**Test Execution Date: 21/03/2024**		
Description: Approve Product through the Web Dashboard					
Pre-Condition: Admin holds a username and password that is valid					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Navigate to Login Screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login with valid credentials	amalthomson@icloud.com Amal#123	Admin should be navigated to the dashboard	Dashboard displayed	Pass
3	Navigate to the product approval page		Navigate to the product approval page	Navigated to the product approval page	Pass
4	Tap pending product approvals	Apple $3.00 Homegrown Heaven	Navigate to pending products approvals page	Navigated to pending products approvals page	Pass
5	Tap approve button		Successfully Approve Product	Successfully Approve Products	Pass
Post-Condition: After successful validation with the database, the admin is able to login into their account, successfully navigate to product approval page and Approve a product and hence the product is visible to the users.					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

Project implementation is the phase where plans and strategies transform into tangible actions and outcomes. It marks the transition from theoretical concepts to practical application. This pivotal stage requires meticulous planning, resource allocation, and a dedicated team to execute tasks according to the established timeline and objectives. In this phase, the project team translates the project's blueprints into real-world activities, ensuring that each step aligns with the overarching goals. Effective project implementation demands clear communication, robust leadership, and a keen eye for detail. This introduction sets the stage for a comprehensive understanding of the project implementation process, emphasizing its significance in achieving the envisioned goals. As we delve deeper, we will explore key components, strategies, and best practices that contribute to successful project implementation.

The crux of successful project implementation lies not just in technical proficiency, but also in the art of effective communication. Clear channels of dialogue serve as the lifeblood that sustains the project's momentum, fostering synergy among team members and stakeholders alike. A bedrock of robust leadership provides the necessary guidance and inspiration, steering the ship through uncharted waters with confidence and purpose. Additionally, an unyielding commitment to detail acts as the linchpin that secures the integrity of each executed task.

This introduction sets the stage for a profound comprehension of the project implementation process, underlining its indomitable significance in realizing the envisioned goals. As we embark on this journey of exploration, we will unfurl the tapestry of key components, unveil strategies, and illuminate best practices that form the crucible of triumphant project implementation.

The implementation state involves the following tasks:

- Careful planning.
- Investigation of system and constraints.
- Design of methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

6.2.1 USER TRAINING

User training is one of the critical component of ensuring the effective utilization of any application software. It involves imparting the necessary knowledge and skills to end-users, enabling them to

navigate and utilize the software efficiently. This training equips users with a comprehensive understanding of the software's features, functions, and capabilities. Through hands-on sessions and guided tutorials, users learn how to perform tasks, customize settings, and troubleshoot common issues. Moreover, user training fosters confidence and proficiency, empowering individuals to maximize their productivity while using the application. Regular updates and refresher sessions further enhance user competence, ensuring they stay abreast of new features and functionalities.

6.2.2 TRAINING ON THE APPLICATION SOFTWARE

Training on the application software is a structured program designed to familiarize individuals with the intricacies and functionalities of a specific software application. It encompasses a range of topics, from basic navigation to advanced features, tailored to meet the diverse needs of users. This training often includes interactive demonstrations, hands-on exercises, and Question and Answer sessions to facilitate effective learning. Trainers may also provide supplemental resources such as user manuals or online guides for reference. By the end of the training, participants are equipped skills and knowledge required to proficiently utilize the application software in their respective contexts.

6.2.3 SYSTEM MAINTENANCE

System maintenance is a crucial aspect of ensuring the seamless operation and longevity of any software application. It encompasses a series of tasks aimed at monitoring, optimizing, and troubleshooting the underlying infrastructure on which the application runs. This includes activities such as regular software updates, performance monitoring, and data backups. Additionally, system maintenance involves identifying and rectifying any potential vulnerabilities or inefficiencies that may impede the software's performance. Proactive maintenance measures contribute to a stable and secure environment, minimizing the risk of unexpected downtime or data loss.

6.2.4 INSTALLATION

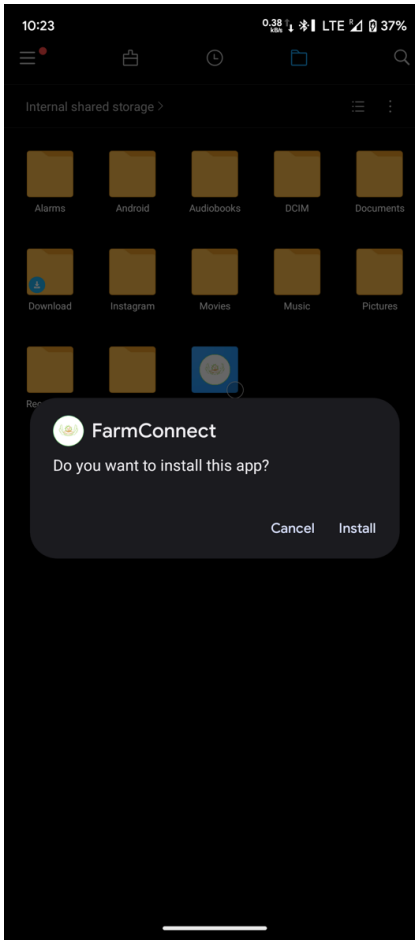
Installing this application is a straightforward process that grants accessibility on mobile devices supporting API versions from 29 to 34. Here's a concise guide:

- **Compatibility Verification**: Ensure your mobile device runs on API version 29 to 30 for seamless installation and operation.
- **Download the APK**: Obtain the APK file from a trusted source, ensuring it matches the application's official version.
- **Adjust Security Settings**: If needed, enable 'Unknown Sources' in your device's settings

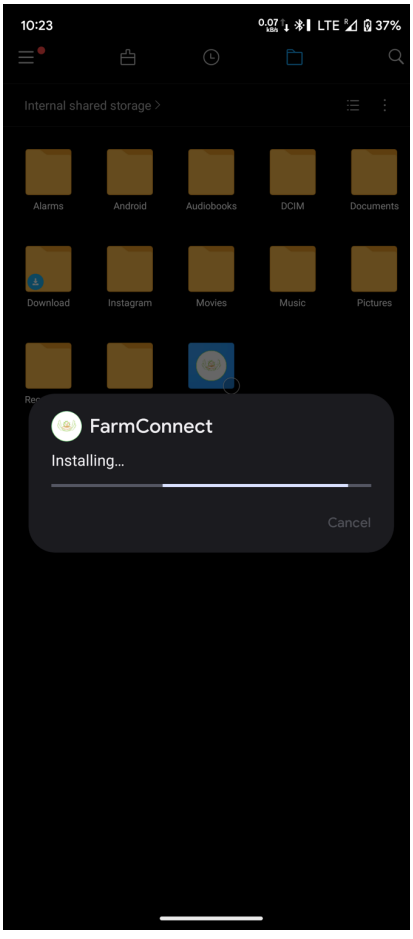
to allow APK installations.

- **Locate and Run the APK**: Access the downloaded APK file, typically found in the 'Downloads' folder, and tap to initiate installation.
- **Permissions and Installation**: Grant necessary permissions for the application to function optimally and proceed with the installation process.
- **Quick Installation**: APKs often install swiftly, bypassing the need for extensive downloads from official app stores.
- **Independence from App Stores**: APKs offer the advantage of being downloadable from various sources, granting users flexibility in obtaining apps.
- **Beta and Unreleased Versions**: Users can access beta or unreleased versions of applications via APKs, providing a preview of upcoming features.
- **Offline Installation**: APKs can be shared and installed without an active internet connection, enhancing accessibility.

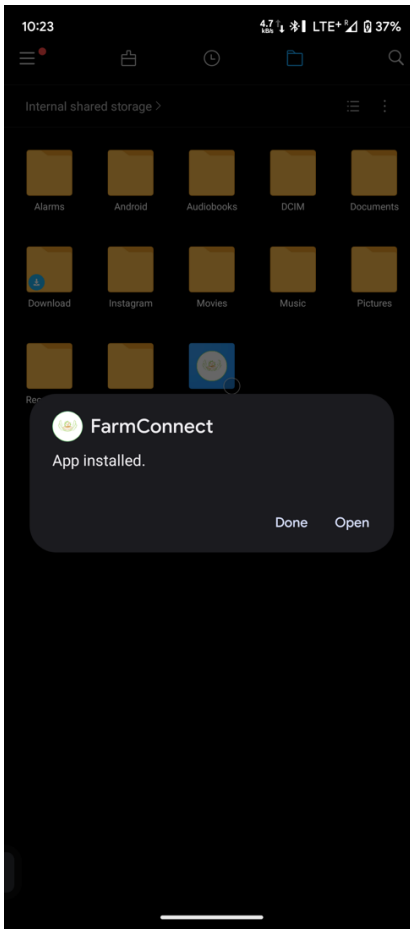
SCREENSHOTS



6.2.4.1



6.2.4.2

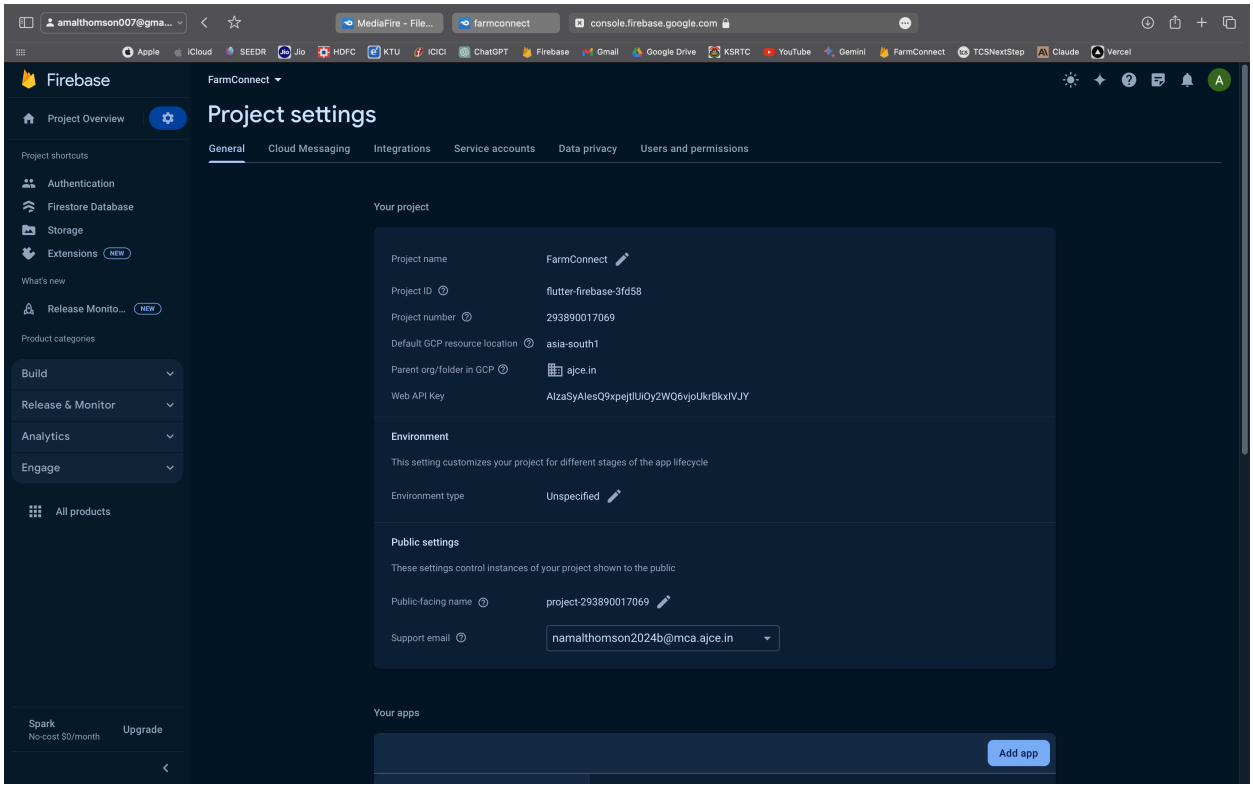


6.2.4.3

6.2.5 HOSTING (Application)

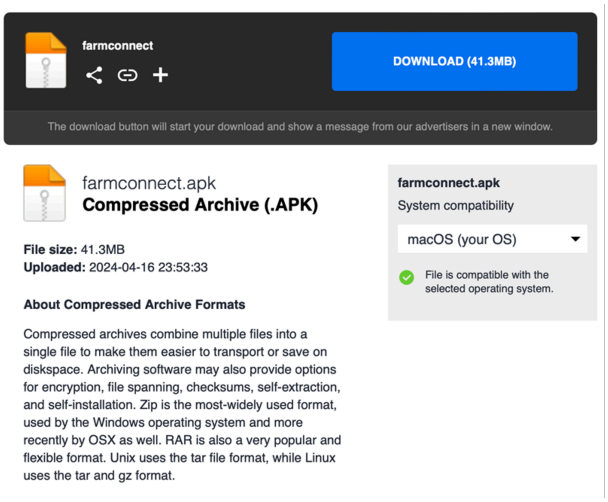
Hosting within Firebase is the process of providing a platform or infrastructure for an application or website to be accessible over the internet. It involves allocating server space, configuring network settings, and ensuring optimal performance and reliability within the Firebase hosting environment. Hosting services in Firebase come with Firebase-specific features like Firebase Hosting deployment, security measures, scalability options, and technical support tailored to meet the diverse needs of applications and businesses within Firebase. Choosing the right hosting solution within Firebase is crucial for ensuring a seamless and accessible online presence integrated with Firebase services.

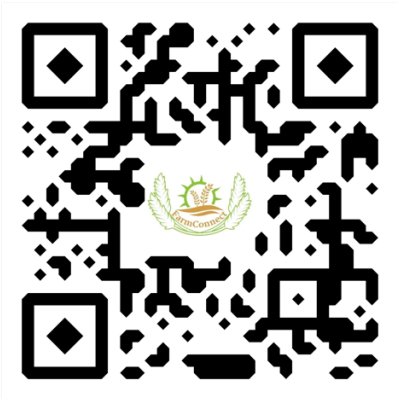
Firebase Cloud Server



6.2.5.1 Cloud Web Server

Download Link: ***https://www.mediafire.com/file/vowrrd64cn16sn7/farmconnect.apk/file***





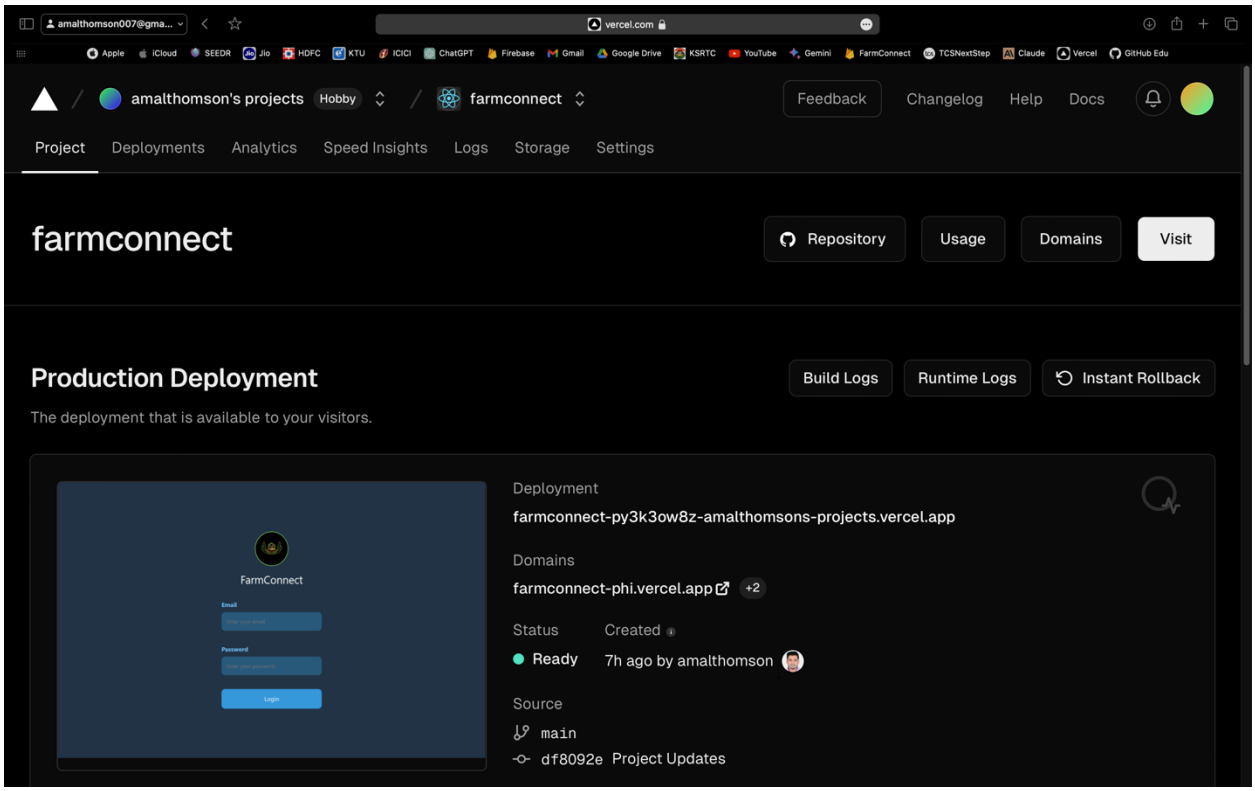
6.2.5.2 Share & Download Application

6.2.6 HOSTNG (Web)

Hosting in Firebase means giving space on the internet for a website made with React and using Firebase as the back end. It involves setting up server space, adjusting network settings, and making sure everything runs smoothly. Firebase hosting offers specific features like easy deployment, security, and support to help businesses.

However, if you're making a website using React with Firebase as the back end and hosting it on Vercel while following CI/CD practices, Vercel provides the hosting environment. This includes putting the React part and Firebase part together, making sure they work well, and using Vercel's features for deployment, scaling, and CI/CD pipelines. It's important to pick the right hosting solution in Vercel to have a smooth online presence connected with Firebase services and use Vercel's tools for managing the front end.

Hosting Link: *https://farmconnect-phi.vercel.app*



6.2.5.3 Deployement

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

FarmConnect emerges as a pivotal solution in the agricultural landscape, introducing a digital platform that seamlessly connects farmers and buyers. With a robust technology stack comprising Flutter for the front-end and Firebase for the back-end, FarmConnect transcends traditional supply chain models, providing a user-friendly interface for farmers to showcase their products and buyers to access the freshest farm produce directly.

The key modules, ranging from the comprehensive Admin functionalities to the specific tools tailored for Farmers and Buyers, underscore the platform's commitment to transparency, efficiency, and sustainability. Admin functionalities empower platform administrators to ensure a smooth and secure user experience, from managing user accounts to facilitating secure transactions. Farmers benefit from tools for efficient inventory management, order fulfillment, and personalized guidance for sustainable farming. Buyers, on the other hand, enjoy a user-friendly interface for seamless product discovery, order placement, and real-time order tracking.

FarmConnect's functionalities extend beyond mere commerce. The inclusion of advanced features such as image recognition for search, disease detection, pest control guidance, and fertilizer recommendations showcase a commitment to leveraging technology for the holistic improvement of farming practices. The platform not only facilitates transactions but also fosters a community where farmers and buyers can engage directly, inquire about products, and provide feedback.

7.2 FUTURE SCOPE

As FarmConnect progresses into future stages of development, the integration of additional functionalities promises to enhance the platform's capabilities even further. In the upcoming stages, it is envisioned that:

- **Integration of Artificial Intelligence (AI) and Machine Learning (ML):** Implementing AI and ML algorithms can enhance user experience by providing personalized service recommendations based on user preferences and behavior patterns
- **Expansion of Categories:** Introducing additional categories based on user feedback and evolving market trends can further diversify the offerings and meet a wider range of user needs.
- **Chatbot Integration:** Enabling voice assistants or chatbots can streamline user interactions.

- **Filtering Options:** Further refinement of filtering options, allowing buyers to tailor searches based on specific criteria such as product attributes, farm practices, and certifications.
- **Communications and Support Enhancements**: Introduction of features for efficient communication and support, including ticketing systems, real-time chat support, and comprehensive FAQs.
- **AI-Powered Disease Detection**: Advancement of disease detection algorithms using artificial intelligence, enabling more accurate and early identification of crop diseases.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009.
- Roger S Pressman, “Software Engineering”, 1994.
- Pankaj Jalote, “Software engineering: a precise approach”, 2006.
- James lee and Brent ware Addison, “Open source web development with LAMP”, 2003
- IEEE Std 1016 Recommended Practice for Software Design Descriptions.

WEBSITES:

- https://pub.dev/
- https://console.firebase.google.com
- https://docs.flutter.dev/development/ui/widgets
- https://fluttergems.dev/
- https://chat.openai.com/
- https://www.github.com/
- https://www.youtube.com/
- https://draw.io/
- https://medium.com/

CHAPTER 9

APPENDIX

9.1 SAMPLE CODE

Application Entry Point – main.dart

```
import 'dart:async';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:farmconnect/blockchain/web3client.dart';
import 'package:farmconnect/features/splashScreen/splash_screen.dart';
import 'package:farmconnect/pages/Admin/dashboard.dart';
import 'package:farmconnect/pages/Buyer/buyer_dashboard.dart';
import 'package:farmconnect/pages/Buyer/buyer_ftl.dart';
import 'package:farmconnect/pages/Buyer/buyer_profile.dart';
import 'package:farmconnect/pages/Buyer/dairy.dart';
import 'package:farmconnect/pages/Buyer/fruits.dart';
import 'package:farmconnect/pages/Buyer/poultry.dart';
import 'package:farmconnect/pages/Buyer/vegetable.dart';
import 'package:farmconnect/pages/Cart/cart_provider.dart';
import 'package:farmconnect/pages/Common/login_page.dart';
import 'package:farmconnect/pages/Common/sign_up_page.dart';
import 'package:farmconnect/pages/Farmer/farmer_dashboard.dart';
import 'package:farmconnect/pages/Farmer/farmer_ftl.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  final cartProvider = CartProvider();
  final user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    cartProvider.setUserId(user.uid);
    await cartProvider.initializeCartFromFirestore();
  }
  await checkAndMarkExpiredProducts();
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider.value(
          value: cartProvider,
        ),
        ChangeNotifierProvider(
          create: (context) => UserServices(),
        ),
      ],
      child: MyApp(),
    ),
  );
```

```
  Timer.periodic(Duration(hours: 6), (Timer timer) async {
    await checkAndMarkExpiredProducts();
  });
}
Future<void> checkAndMarkExpiredProducts() async {
  final DateTime now = DateTime.now();
  final QuerySnapshot<Map<String, dynamic>> productsSnapshot =
  await FirebaseFirestore.instance.collection('products').get();
  final List<QueryDocumentSnapshot<Map<String, dynamic>>> expiredProducts = [];
  for (final QueryDocumentSnapshot<Map<String, dynamic>> product
  in productsSnapshot.docs) {
    final DateTime expiryDate = DateTime.parse(product['expiryDate']);
    if (expiryDate.isBefore(now) && product['isApproved'] != 'Expired') {
      expiredProducts.add(product);
    }
  }
  final WriteBatch batch = FirebaseFirestore.instance.batch();
  for (final QueryDocumentSnapshot<Map<String, dynamic>> product
  in expiredProducts) {
    final DocumentReference productRef = FirebaseFirestore.instance
        .collection('products')
        .doc(product.id);
    batch.update(productRef, {'isApproved': 'Expired'});
  }
  await batch.commit();
  print('Expired Products:');
  print('Total Expired Products: ${expiredProducts.length}');
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final cartProvider = Provider.of<CartProvider>(context);
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      cartProvider.setUserId(user.uid);
    }
    return MaterialApp(
      title: 'FarmConnect',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      debugShowCheckedModeBanner: false,
      initialRoute: '/',
      routes: {
```

```
        '/': (context) => SplashScreen(
          child: LoginPage(),
        ),
        '/login': (context) => LoginPage(),
        '/signUp': (context) => SignUpPage(),
        '/buyer_profile': (context) => BuyerProfile(),
        '/buyer_home': (context) => BuyerDashboard(),
        '/update_details': (context) => UpdateDetailsPage(),
        '/update_password': (context) => UpdatePasswordPage(),
        '/farmer_dash': (context) => FarmerDashboard(),
        '/buyer_ftl': (context) => BuyerFTLPage(),
        '/farmer_ftl': (context) => FarmerFTLPage(),
        '/admin_dashboard': (context) => AdminDashboard(),
        '/email_verification_pending': (context) => EmailVerificationPendingPage(),
        '/terms': (context) => TermsPage(),
        '/poultry_page': (context) => PoultryProductsPage(),
        '/dairy_page': (context) => DairyProductsPage(),
        '/fruits_page': (context) => FruitsProductsPage(),
        '/vegetables_page': (context) => VegetableProductsPage(),
        '/added_product': (context) => FarmerDashboard(),
      },
    );
  }
}
```

Connect Flutter App to Blockchain – web3client.dart

```
import 'dart:convert';
import 'dart:io';
import 'package:farmconnect/blockchain/user_datastructure.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:http/http.dart' as http;
import 'package:web3dart/web3dart.dart';
import 'package:web_socket_channel/io.dart';
class UserServices extends ChangeNotifier {
  List<User> users = [];
  final String _rpcUrl =
  Platform.isAndroid ? 'http://10.0.2.2:7545' : 'http://127.0.0.1:7545';
  final String _wsUrl =
  Platform.isAndroid ? 'http://10.0.2.2:7545' : 'ws://127.0.0.1:7545';
  bool isLoading = true;
  final String _privatekey =
      '0x8d8ee8e432ef1a171013770c3e0dcaafabf6dfe6e97714521f190a7bea5447d7';
  late Web3Client _web3Client;
```

```
UserServices() {
  init();
}
Future<void> init() async {
  _web3Client = Web3Client(
    _rpcUrl,
    http.Client(),
    socketConnector: () {
      return IOWebSocketChannel.connect(_wsUrl).cast<String>();
    },
  );
  await getABI();
  await getCredentials();
  await getDeployedContract();
}
late ContractAbi _abiCode;
late EthereumAddress _contractAddress;
Future<void> getABI() async {
  String abiFile = await rootBundle
      .loadString('build/contracts/UserDetailsContract.json');
  var jsonABI = jsonDecode(abiFile);
  _abiCode = ContractAbi.fromJson(
      jsonEncode(jsonABI['abi']), 'UserDetailsContract');
  _contractAddress = EthereumAddress.fromHex(
      jsonABI["networks"]["5777"]["address"]);
}
late EthPrivateKey _credentials;
Future<void> getCredentials() async {
  _credentials = EthPrivateKey.fromHex(_privatekey);
}
late DeployedContract _deployedContract;
late ContractFunction _createUserDetails;
Future<void> getDeployedContract() async {
  _deployedContract = DeployedContract(_abiCode, _contractAddress);
  _createUserDetails = _deployedContract.function('createUserDetails');
  await fetchUsers();
}
Future<void> fetchUsers() async {
  List userCountList = await _web3Client.call(
    contract: _deployedContract,
    function: _deployedContract.function('userCount'),
    params: [],
  );
  int userCount = userCountList[0].toInt();
```

```
    users.clear();
    for (var i = 0; i < userCount; i++) {
      var user = await _web3Client.call(
        contract: _deployedContract,
        function: _deployedContract.function('userDetails'),
        params: [BigInt.from(i)],
      );
    isLoading = false;
    notifyListeners();
  }
  Future<void> addUser({
    required String fuid,
    required String name,
    required String farmname,
    required String email,
    required String phone,
    required String aadhar,
    required String address,
  }) async {
    await _web3Client.sendTransaction(
      _credentials,
      Transaction.callContract(
        contract: _deployedContract,
        function: _createUserDetails,
        parameters: [
          fuid,
          name,
          farmname,
          email,
          phone,
          aadhar,
          address,
        ],
      ),
      chainId: 1337,
    );
    isLoading = true;
    fetchUsers();
  }
}
```

Cart Provider Page for Cart Model – cart_provider.dart

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
class CartProvider extends ChangeNotifier {
  String? _userId;
  List<Map<String, dynamic>> _cartItems = [];
  List<Map<String, dynamic>> get cartItems => _cartItems;
  void setUserId(String userId) {
    _userId = userId;
  }
  Future<void> fetchCartFromFirestore() async {
    try {
      final userCartCollection = FirebaseFirestore.instance.collection('cart').doc(_userId);
      final cartData = await userCartCollection.get();
      if (cartData.exists) {
        _cartItems = List.from(cartData['cartItems']);
        notifyListeners();
      }
    } catch (error) {
      print('Error fetching cart from Firestore: $error');
    }
  }
  Future<void> addToCart(Map<String, dynamic> item) async {
    final userCartCollection = FirebaseFirestore.instance.collection('cart').doc(_userId);
    item['quantity'] = item['quantity'] ?? 1;
    if (!_cartItems.contains(item)) {
      final productCollection = FirebaseFirestore.instance.collection('products');
      final productData = await productCollection.doc(item['productId']).get();
      int productStock = productData['stock'] ?? 0;
      if (productStock >= item['quantity']) {
        await _updateStock(item['productId'], item['quantity'], false);
        _cartItems.add(item);
        notifyListeners();
        await userCartCollection.set({'cartItems': _cartItems});
      } else {
        print('Insufficient stock for ${item['productName']}');
      }
    } else {
      print('${item['productName']} is already in the cart');
    }
  }
  double totalAmount() {
    double total = 0.0;
    for (var item in _cartItems) {
      double productPrice = double.tryParse(item['productPrice'].toString()) ?? 0.0;
```

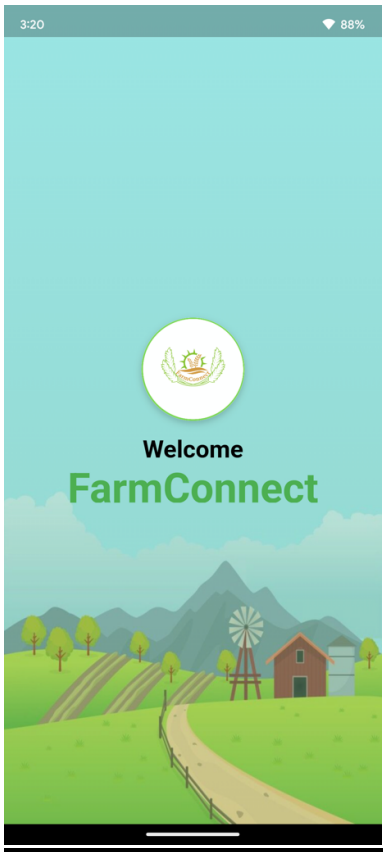
```
    total += productPrice * (item['quantity'] ?? 1);
  }
  return total;
}
Future<void> removeFromCart(String productId) async {
  var removedItem = _cartItems.firstWhere(
        (item) => item['productId'] == productId,
    orElse: () => {},
  );
  if (removedItem.isNotEmpty) {
    await _updateStock(productId, removedItem['quantity'], true);
    _cartItems.remove(removedItem);
    notifyListeners();
    await _updateFirestoreCart();
  }
}
Future<void> clearCart() async {
  _cartItems.clear();
  notifyListeners();
  await _updateFirestoreCart();
}
Future<void> _updateFirestoreCart() async {
  final userCartCollection = FirebaseFirestore.instance.collection('cart').doc(_userId);
  await userCartCollection.set({'cartItems': _cartItems});
}
int productCount(String productId) {
  return _cartItems.where((item) => item['productId'] == productId).length;
}
List<Map<String, dynamic>> get uniqueProducts {
  Set<String> uniqueProductIds = Set<String>();
  List<Map<String, dynamic>> uniqueProductsList = [];
  for (var item in _cartItems) {
    if (uniqueProductIds.add(item['productId'])) {
      uniqueProductsList.add(item);
    }
  }
  return uniqueProductsList;
}
int get uniqueProductCount {
  return uniqueProducts.length;
}
void decreaseQuantity(String productId) async {
  var productIndex = _cartItems.indexWhere((item) => item['productId'] == productId);
  if (productIndex != -1) {
```

```
      _cartItems[productIndex]['quantity'] = (_cartItems[productIndex]['quantity'] ?? 1) - 1;
      if (_cartItems[productIndex]['quantity'] == 0) {
        await removeFromCart(productId);
      } else {
        await _updateStock(productId, 1, true);
        notifyListeners();
        await _updateFirestoreCart();
      }
    }
  }
  void increaseQuantity(String productId) async {
    var productIndex = _cartItems.indexWhere((item) => item['productId'] == productId);
    if (productIndex != -1) {
      _cartItems[productIndex]['quantity'] = (_cartItems[productIndex]['quantity'] ?? 0) + 1;
      await _updateStock(productId, 1, false);
      notifyListeners();
      await _updateFirestoreCart();
    } else {
      addToCart({
        'productId': productId,
        'quantity': 1,
      });
      notifyListeners();
    }
  }
  Future<void> initializeCartFromFirestore() async {
    await fetchCartFromFirestore();
  }
  Future<void> _updateStock(String productId, int quantity, bool increase) async {
    final productCollection = FirebaseFirestore.instance.collection('products');
    final productData = await productCollection.doc(productId).get();
    int productStock = productData['stock'] ?? 0;
    if (increase) {
      productCollection.doc(productId).update({
        'stock': productStock + quantity,
      });
    } else {
      productCollection.doc(productId).update({
        'stock': productStock - quantity,
      });
    }
  }
}
```

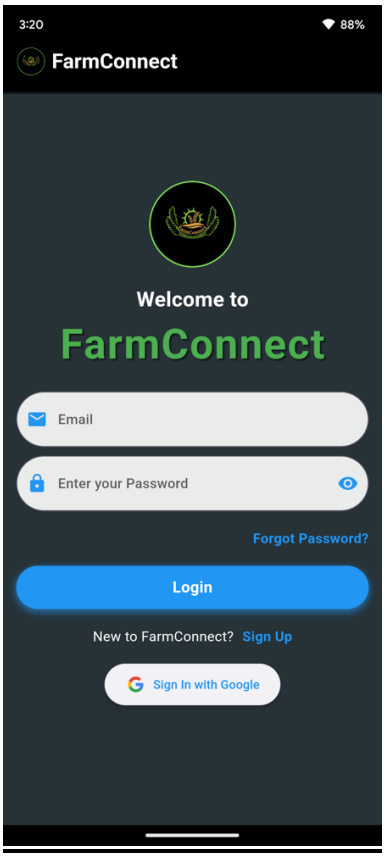
Web Application Entry – App.jsx

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import AdminDashboard from './components/AdminDashboard';
import FarmerDetailsReactPage from './components/FarmerDetails';
import BuyerDetailsReactPage from './components/BuyerDetails';
import LoginPage from './components/LoginPage';
import FarmerPendingReactPage from './components/FarmerPending';
import FarmerRejectedReactPage from './components/FarmerRejected';
import StockDetails from './components/StockDetails';
import PaymentDetails from './components/PaymentDetails';
import OrderDetails from './components/OrderDetails';
import ProductApproved from './components/ProductsApproved';
import ProductRejected from './components/ProductsRejected';
import ProductPending from './components/ProductsPending';
import ReviewDetails from './components/ReviewDetails';
import AllProducts from './components/AllProducts';
import Farmers from './components/Farmers';
import Products from './components/Products';
const App = () => {
  return (
    <Router>
      <Routes>
      <Route path="/" element={<LoginPage/>} />
      <Route path="/admin-dashboard" element={<AdminDashboard/>} />
      <Route path="/farmer-details" element={<FarmerDetailsReactPage/>} />
      <Route path="/buyer-details" element={<BuyerDetailsReactPage/>} />
      <Route path="/farmer-pending" element={<FarmerPendingReactPage/>} />
      <Route path="/farmer-rejected" element={<FarmerRejectedReactPage/>} />
      <Route path="/all-products" element={<AllProducts/>} />
      <Route path="/stock-details" element={<StockDetails/>} />
      <Route path="/payment-details" element={<PaymentDetails/>} />
      <Route path="/order-details" element={<OrderDetails/>} />
      <Route path="/products-approved" element={<ProductApproved/>} />
      <Route path="/products-rejected" element={<ProductRejected/>} />
      <Route path="/products-pending" element={<ProductPending/>} />
      <Route path="/category-details" element={<CategoryDetails/>} />
      <Route path="/review-details" element={<ReviewDetails/>} />
      <Route path="/farmers" element={<Farmers/>} />
      <Route path="/products" element={<Products/>} />
      </Routes>
    </Router>
  );
};
export default App;
```

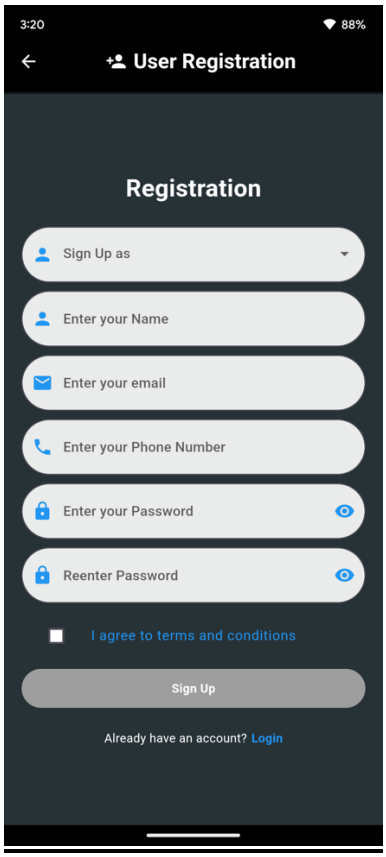
9.2 SCREEN SHOTS



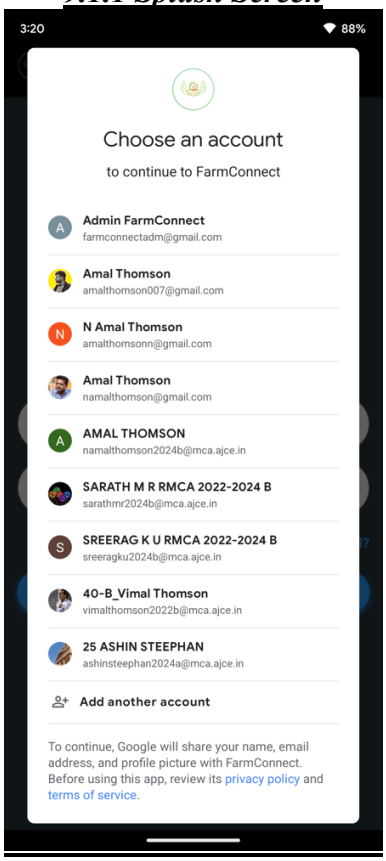
9.1.1 Splash Screen



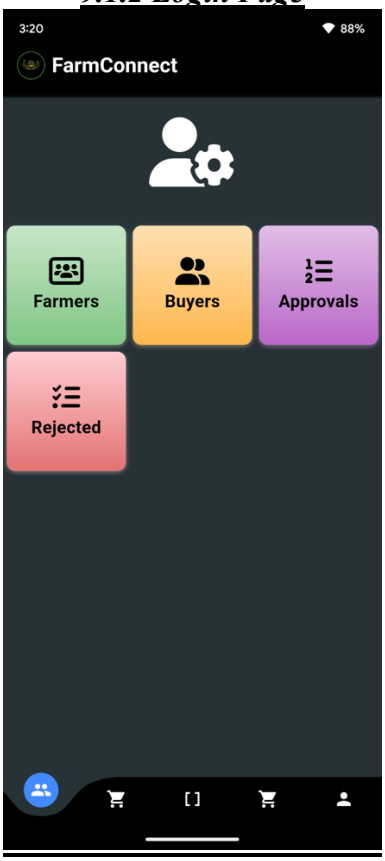
9.1.2 Login Page



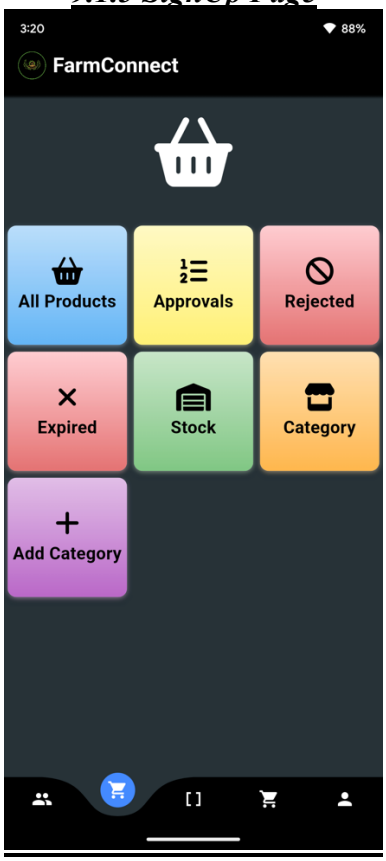
9.1.3 SignUp Page



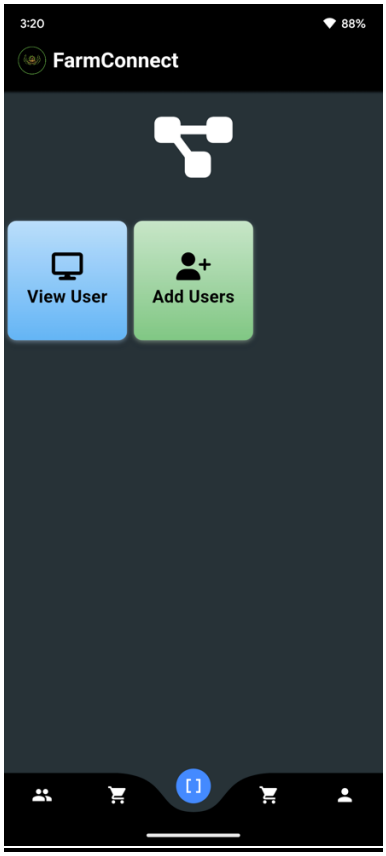
9.1.4 Google Authentication



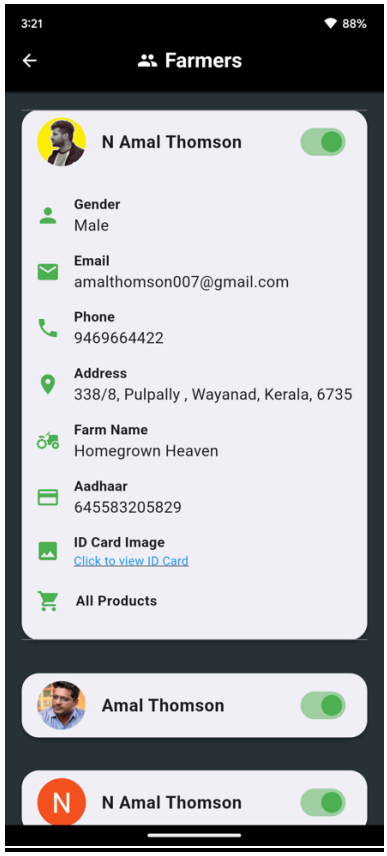
9.1.5 Display Users



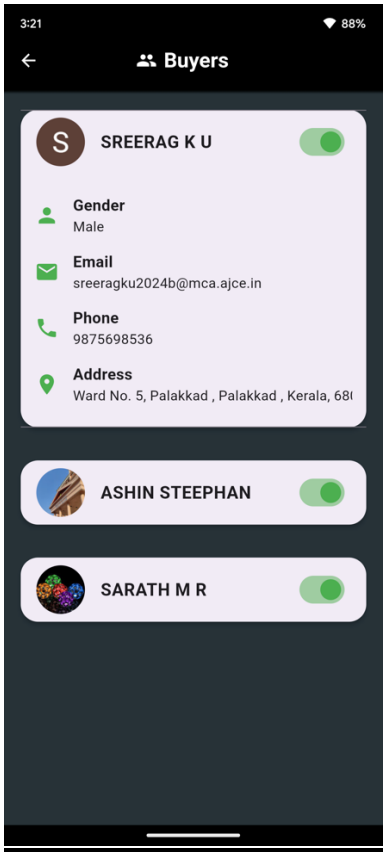
9.1.6 Display Products



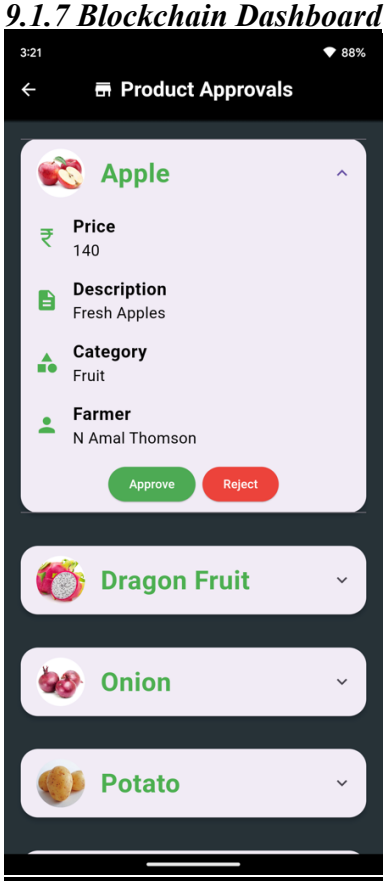
9.1.7 Blockchain Dashboard



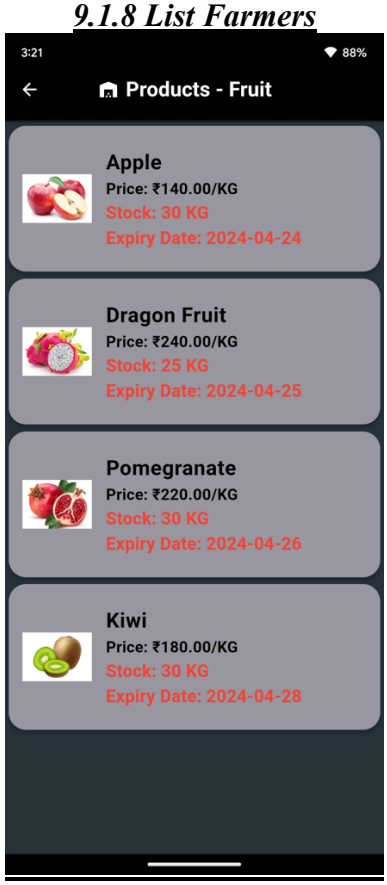
9.1.8 List Farmers



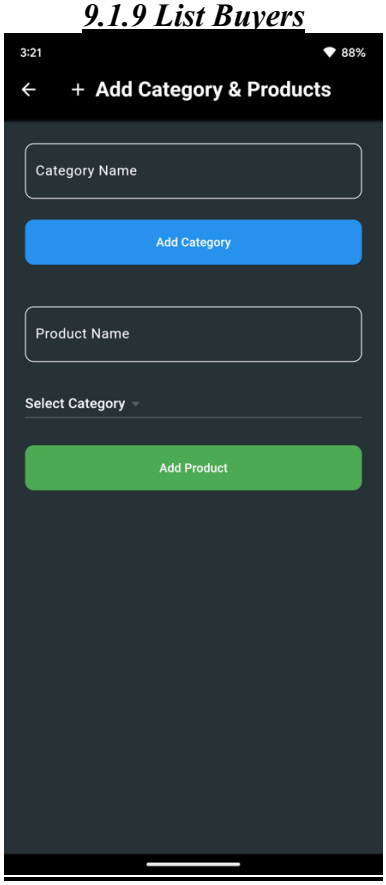
9.1.9 List Buyers



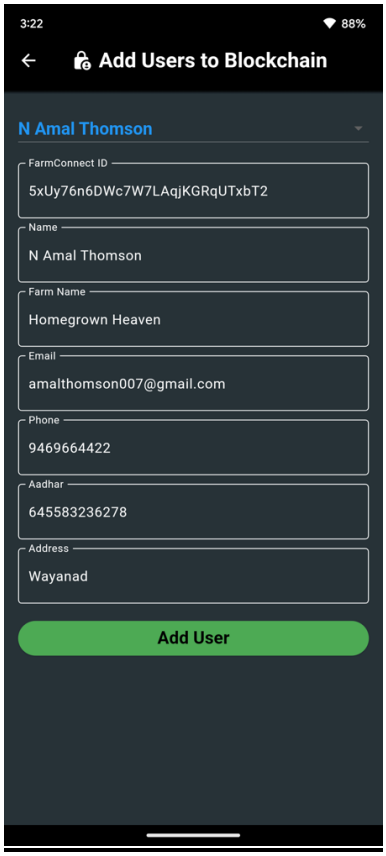
9.1.10 List Products



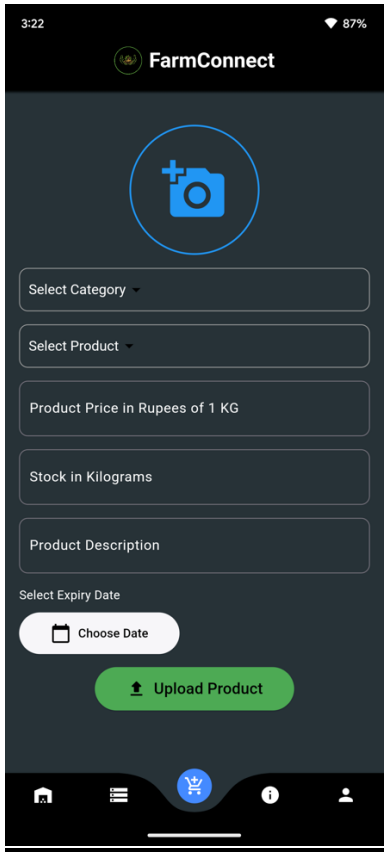
9.1.11 List Fruits



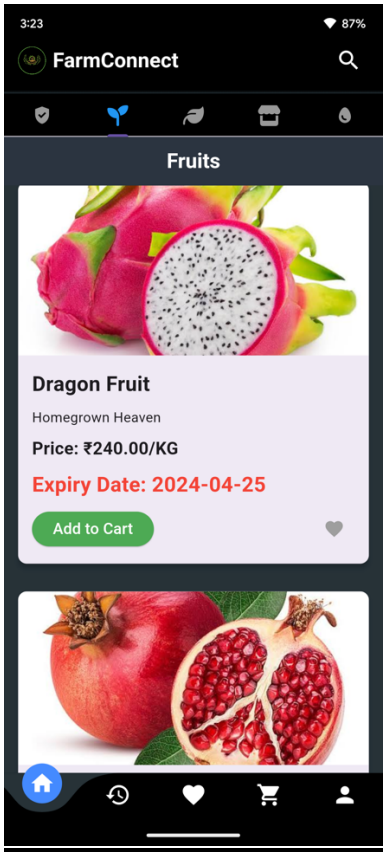
9.1.12 Add Category



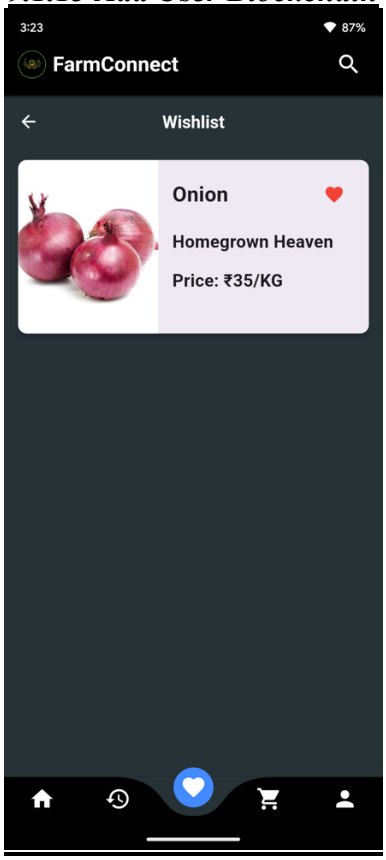
9.1.13 Add User Blockchain



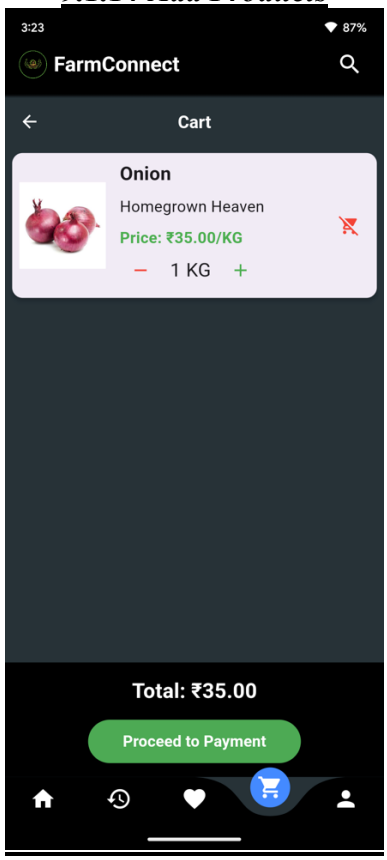
9.1.14 Add Products



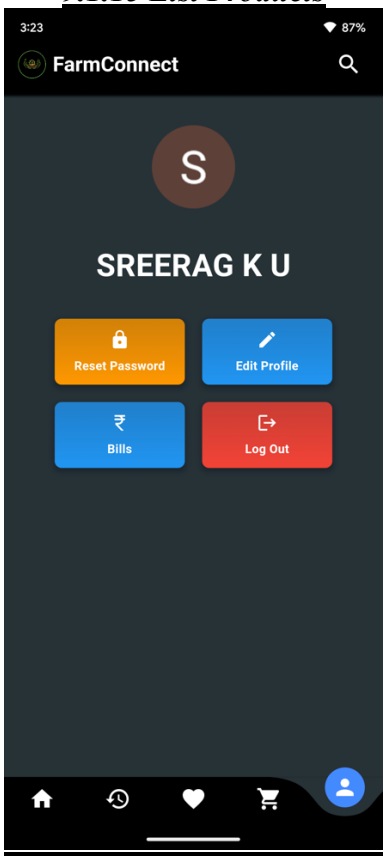
9.1.15 List Products



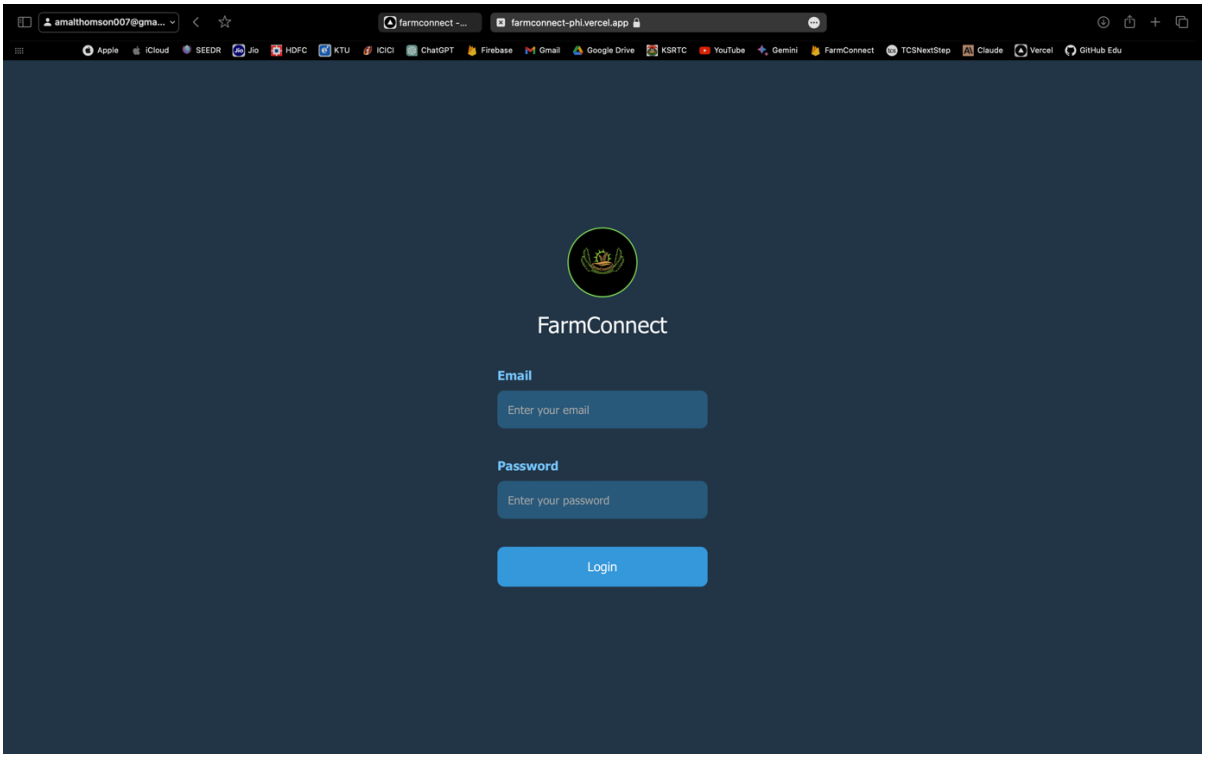
9.1.16 Wishlist



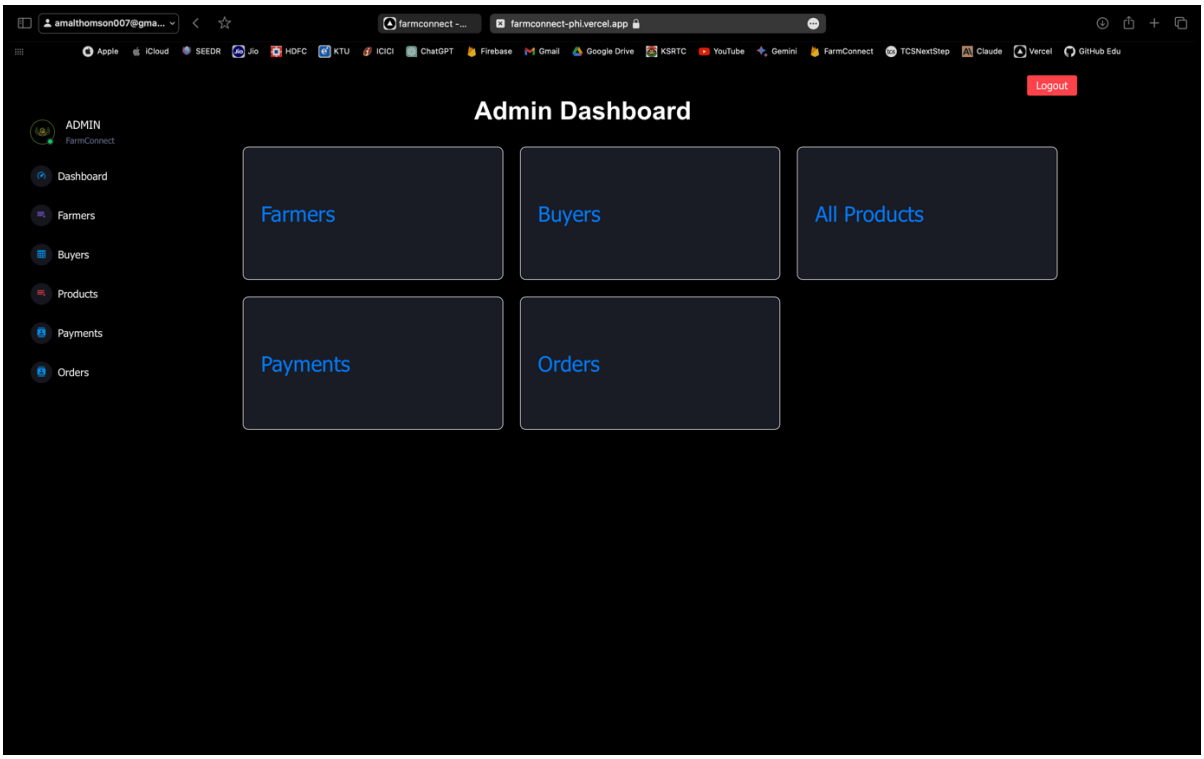
9.1.17 Cart



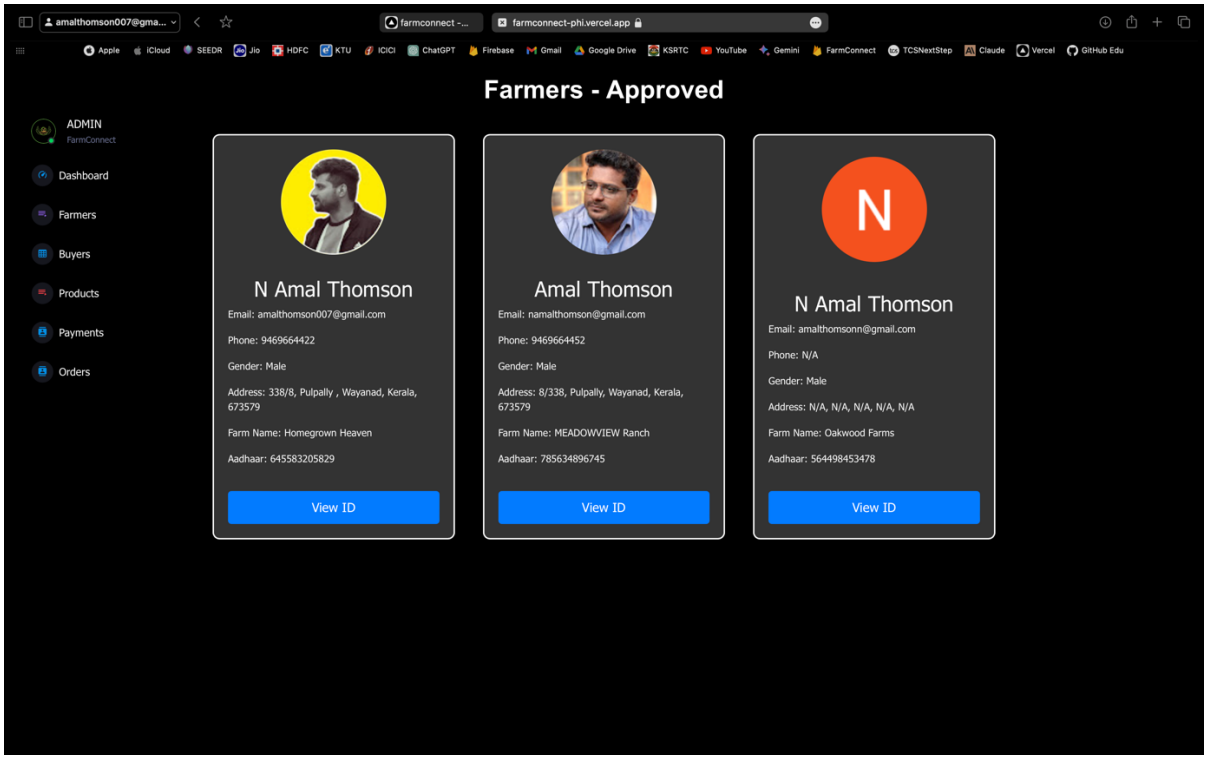
9.1.18 Profile



9.1.19 Login Screen



9.1.19 Admin Dashboard



9.1.19 Display Farmers