LONDON METROPOLITAN UNIVERSITY

Revolutionizing Agriculture: A Broker-Free Marketplace for Farmers

Name: Mohammed Najimudeen Ahamed Sajee

London Met ID: 23038964

LONDON METROPOLITAN UNIVERSITY

Declaration

Module: CS6P05ES **Deadline:**

Module Leader: N.M.M NUHMAN **Student ID: 23038964**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

(signature:)_____M.N.A0190@myLondonmet.ac.uk_____ Date:10.06.2024

This header sheet should be attached to the work you submit. No work will be accepted without it.

Extracts from University *Regulations* on Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples…

(i) Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.

(ii) Falsifying data in experimental results.

(iii) Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.

(iv) Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.

(v) Collusion to present joint work as the work solely of one individual.

(vi) Plagiarism, where the work or ideas of another are presented as the candidate's own.

(vii) Other conduct calculated to secure an advantage on assessment.

(viii) Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. "$e = mc^2$ (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

LONDON METROPOLITAN UNIVERSITY

Dedication

Esoft Metro Campus, a place where innovation meets education. This coursework reflects my commitment to harnessing technology to create positive change, inspired by the dynamic learning environment fostered here.

To the dedicated faculty and staff of Esoft Metro Campus, whose commitment to excellence and student success have empowered me to reach my full potential. This project is a testament to their guidance and support.

Acknowledgement

I would like to express my sincere gratitude to [N.M.M Nuhman], my instructor for [Final Project], for their invaluable guidance and encouragement throughout this project. Their insightful feedback and expertise have helped me to develop a deeper understanding of the subject matter and to refine my research approach. I am also grateful to my classmates, , for their collaborative spirit and support during our group discussions and brainstorming sessions. Finally, I would like to thank for ESoft Metro Campus all staffs.

Abstract

This project proposes the development of a web and mobile application-based broker-free marketplace for farmers, aiming to address the persistent problem of limited market access and unfair pricing experienced by smallholder farmers. The project is motivated by the desire to empower farmers, enhance food security, and promote a more sustainable and equitable agricultural system.

The proposed solution leverages PHP for the backend(WEB) Html/CSS/JavaScript(Web Frontend) and using React for the Mobile Application Development, creating an intuitive and user-friendly platform. It will facilitate direct interaction between farmers and consumers,

eliminating the need for middlemen and empowering farmers to receive fair pricing for their produce.

The project includes a detailed analysis of user requirements, system architecture, technology stack selection, and implementation strategy. The proposed system will be evaluated based on its usability, effectiveness, scalability, and security. Comprehensive documentation, training, and ongoing support will be provided to ensure the successful adoption and long-term sustainability of the platform.

Introduction

The traditional agricultural supply chain often presents challenges for farmers, particularly smallholders, who struggle with limited market access and receive unfairly low prices for their produce. This situation contributes to food insecurity, poverty, and environmental degradation.

This project aims to revolutionize agriculture by creating a broker-free marketplace that directly connects farmers with consumers, removing the need for intermediaries and fostering a more equitable and transparent market. The project will leverage modern web and mobile technologies to create an intuitive and user-friendly platform accessible to farmers and consumers alike.

Background and Motivation

The problem of limited market access for smallholder farmers is a significant global issue. The Food and Agriculture Organization (FAO) estimates that smallholder farmers produce close to 80% of the world's food, yet they often receive only a small fraction of the final consumer price due to the inefficiencies and lack of transparency in the existing supply chain. This project is motivated by the desire to address this injustice and empower farmers by giving them direct control over their products and market access.

Problem in Brief

The specific problem we aim to solve is the lack of access to fair markets and equitable pricing for smallholder farmers. This is a pressing issue due to the following reasons:

- **Low Income:** Farmers are forced to sell their produce at low prices to middlemen, resulting in limited profit margins and perpetuating poverty.
- **Food insecurity:** inefficient market systems contribute to food waste and limit the availability of affordable, locally grown produce, impacting food security.
- **Environmental degradation:** Lack of control over market access and pricing can hinder the adoption of sustainable farming practices.

Aim:

The aim of this project is to develop a broker-free marketplace for farmers, using PHP and React Native, that connects them directly with consumers, enabling fair pricing, improved market access, and increased profitability.

Objectives:

The project will achieve the following objectives:

- **Conduct a critical review of the agricultural industry:** Understand the challenges faced by farmers, identify inefficiencies in the existing supply chain, and analyze the impact of intermediaries on farmer income.
- **Critically study technologies that can solve the problem:** Explore the potential of blockchain technology, cryptocurrencies, and smart contracts for enhancing transparency and security in agricultural transactions.
- **Design and develop a system for solving the problem:** Create a user-friendly web and mobile application that facilitates direct interactions between farmers and consumers.
- **Evaluate the proposed system:** Conduct thorough testing to ensure the system's usability, effectiveness, scalability, and security**.**
- **Prepare final documentation:** Develop comprehensive documentation for users, developers, and system administrators.

Proposed Solution:

The project will be implemented using a combination of PHP for the backend (Web) and HTML/CSS/JavaScript for frontend(web) and using React for the Mobile Application development. The system will feature the following:

- User Management: Both farmers and consumers will have dedicated accounts with distinct roles and permissions.
- Product Listing: Farmers will be able to create detailed listings for their produce, including descriptions, images, and pricing.
- Order Processing: Consumers can browse product listings, add items to their carts, and place orders directly with farmers.
- Communication: A messaging system will allow farmers and consumers to communicate about product inquiries, order status, and delivery details.
- Inventory Management: Farmers will be able to manage their inventory levels in real-time.
- Reporting and Analytics: The system will generate reports and analytics providing insights into sales patterns, customer behavior, and other key metrics.

Resource Requirements:

The project requires the following resources:

- Hardware: Development workstation, mobile devices for testing, and server infrastructure.
- Software: PHP development environment, React Native development environment, MySQL database, and AWS cloud platform.
- Other Requirements: Reliable internet connection, source code repository (e.g., GitHub), project management tools (e.g., Trello, Asana).

Deliverables:

- **Web Application:** A fully functional web application that enables farmers to list their products, manage their inventory, and communicate with customers.
- **Mobile Application:** A user-friendly mobile application that provides farmers and consumers with on-the-go access to the marketplace.
- **Documentation:** Technical documentation, user manuals, and API reference guide.
- **Presentation:** A comprehensive presentation outlining project objectives, results, and future recommendations.

LONDON METROPOLITAN UNIVERSITY

Table of Contents

Dedication 3

Acknowledgement 3

Abstract 3

Chapter 1 Introduction 14

1.1 Goals 15

1.1.1 The Main Objectives of this project are Listed below 15

1.1.2 The Following are the objectives to consider during and after the development of this "Revolutionizing Agriculture" system 16

1.2 Motivation 18

1.1.1 Motivations for the Project 18

1.1.2 Market Drivers: 18

1.1.3 Core Value 18

1.3 Method 19

1.3.1 Methodologies Followed in Development Process 19

1.3.2 The Selected Methodology 21

1.4 Overview 22

1.4.1 Technologies used in Development Process 22

1.4.2 Integration Development Environment (IDE) 23

1.4.3 Reason for Choosing the Visual Studio Code for Revolutionizing Agriculture 25

1.4.4 Used Technologies in Revolutionizing Agriculture Mobile Application Development 25

1.4.5 Database Management Systems (DBMS) 26

Chapter 02 Background and Problem Statement 29

2.1 Introduction 29

2.2 Literature Review 29

2.2.1 Overview 29

2.2.2 Benefits of a Broker-Free Marketplace 29

2.2.3 Role of Mobile and Web Technologies 30

2.2.4 Challenges and Opportunities 30

2.2.4 Conclusion 30
2.3 Problem Statement 31
Chapter 3 Project Management 32
3.1 Approach 32
3.2 Initial project Plan 33
3.3 Problem and Changes to the initial Project plan: “Revolutionizing Agriculture” 35
3.4 Final Project Gantt Chart 37
Chapter 04 Feasibility Study 38
4.1 Time Feasibility 38
4.2 Technical Feasibility 39
4.2 Economic Feasibility 40
4.3 Scope Feasibility: 41
Chapter 05 System Design 42
5.1 Diagrams 42
5.1.1 Database Diagram 42
5.1.2 Use Case Diagram 47
5.2 User Interface Design 48
5.2.1 Web Applications 48
5.2.2 Mobile Application 57
5.3 Resource Requirements 66
5.3.1 Hardware Requirements 66
5.3.2 Software Requirements 66
5.4 Evaluation of Solutions 68
Chapter 06 Implementation 70
6.1 Customer side Mobile Application Implimentation 70
6.1.1 Create a main.js file to communicate MySQL Database 70
6.1.1 User Login Screen 73
6.1.2 User Registration Screen 80
6.1.3 Order Page Screen 83
6.1.4 Add to Cart Screen 85

6.1.5 Checkout Screen 87
6.1.6 Payment Screen 89
6.1.7 Order Complete Screen 91
6.2 Farmer and Customer Side Implementation 92
6.2.1 Database Connection Code 92
6.2.2 Admin Login 93
6.2.3 Add crops Page 97
6.2.4 View Crops Page 99
6.2.5 Update Crop Details 101
6.2.6 Add Harvest Page 104
6.2.7 View Harvest Details page 106
6.2.8 Update Harvest Page 108
Chapter 07 – Testing and Verification 109
7.1 Scope of Testing 109
7.2 Test Cases 110
Chapter 8 Evaluation and Conclusion 122
8.1 Evaluation 122
8.2 Conclusion 122
Chapter 09 User Guide 123
9.1 User Guide for mobile application 123
Chapter 10: References and Bibliography 124
10.1 References 124
10.2 Bibliography 125
Chapter 11 Appendices 126
11.1 Progress approval form and Project Commencement meeting Sheet 126

Figure 1 Gant Chart 37
Figure 2 Database Diagram 42
Figure 3 Use Case Diagram 47
Figure 4 Admin Login 48
Figure 5 Admin Dashboard 49
Figure 6Add Crops 49
Figure 7 View Crops 50
Figure 8 Delete Crops 51
Figure 9 Add a Harvest 52
Figure 10 View Harvest Details 53
Figure 11 Update Harvest 54
Figure 12 Add A New Farm 55
Figure 13 Update Farm 56
Figure 14 Start app screen 57
Figure 15 Customer Login Screen 58
Figure 16 Customer Registration Screen 59
Figure 17 Welcome Screen 60
Figure 18 Product List Screen 61
Figure 19 Add to Cart Screen 62
Figure 20 Checkout. Screen 63
Figure 21 Payment Screen 64
Figure 22 Order Complete Screen 65
Figure 23 Main.Js Code 70
Figure 24 Main.js Code 71
Figure 25 Check the server successfully running 72
Figure 26 Login Screen 73
Figure 27 Login Handling Code 74
Figure 28 Google Sign in 75
Figure 29 google Authentication Keys 75
Figure 30 Google Sign in Handling Code 76
Figure 31 Hidden Password Code 76
Figure 32 Enter Email Address and Password 77
Figure 33 Login Success full The page Redirect to Welcome page 78
Figure 34 Order Page 79
Figure 35 User Registration Screen 80
Figure 36 User Signup Handling Code 81
Figure 37 User Signup Set Submitting Code 82

Figure 38 Hide and show Password box Values code 82
Figure 39 Order Screen 83
Figure 40 Order Screen Code 84
Figure 41 Add to Cart Screen 85
Figure 42Add to Cart Implementation codes 86
Figure 43 Checkout Screen 88
Figure 44Checkout Screen Implementation 88
Figure 45 Payment screen With using strip Api 89
Figure 46 Payment Screen implementation 90
Figure 47 Order Complete Screen 91
Figure 48 Database Connection Code 92
Figure 49Admin Login 93
Figure 50 Admin Login Codes 94
Figure 51 Enter Admin Email and Password 95
Figure 52 Login Success Page Redirect to Admin Dashboard Page 96
Figure 53 Use Session to get Current Login Admin email 96
Figure 54 Add corps Page 97
Figure 55 Add crops page implementation 98
Figure 56 Add Crops page Implementation 98
Figure 57 View Crops Details 99
Figure 58 View Crops Details Code 100
Figure 59 Update Crops 101
Figure 60 Update Crops Code 102
Figure 61 Update crop Codes 103
Figure 62 Add Harvest Page 104
Figure 63 Add Harvest Page Codes 105
Figure 64 View Harvest Details Code 106
Figure 65 View Harvest Codes 107
Figure 66 Update Harvest Page 108
Figure 67 Update Harvest Page Codes 109

Table 1 Requirements and Design 33
Table 2 Development 34
Table 3 Testing and Deployment 34
Table 4 maintenance and Updates 35
Table 5 Testcase 01 110
Table 6 Testcase 02 110
Table 7 Test Case 03 111
Table 8 Testcase 04 112
Table 9 Testcase 05 112
Table 10 Testcase06 113
Table 11 Testcase 07 114
Table 12 Testcase 08 114
Table 13 Testcase 09 114
Table 14 Test Case 10 115
Table 15 Testcase 11 115
Table 16 Testcase 12 116
Table 17 Testcase 13 116
Table 18 Test case14 117
Table 19 Testcase 15 117
Table 20 Testcase 16 118
Table 21 Test case 17 119
Table 22 Test case 18 119
Table 23 TEst case 19 120
Table 24 Test case 20 121

Chapter 1 Introduction

Technological innovations and the increasing need for fairness, sustainability, and openness in the food supply chain are causing a major upheaval in the agriculture sector. A major obstacle to this change is that smallholder farmers frequently do not have access to fair markets, which results in low prices, little income, and food insecurity.

The agricultural industry is undergoing a significant upheaval due to technological advancements and the growing need for transparency, sustainability, and fairness in the food supply chain. Smallholder farmers usually lack access to fair markets, which leads to low prices, limited income, and food insecurity. This is a fundamental barrier to this shift.

Through a user-friendly web application and a convenient mobile app, we aim to create a dynamic ecosystem where:

- Farmers Can list and sell their products directly to consumers.
- Consumers can browse, search, and purchase fresh, locally grown produce.
- Transactions are secure, transparent, and efficient.

this project will promote a more sustainable and equitable agricultural system that benefits both farmers and consumers by tearing down the obstacles found in traditional marketplaces. The project's goals, methodology, development status, and possible effects on the agricultural environment are all covered in length in this study.

1.1 Goals

1.1.1 The Main Objectives of this project are Listed below

1. Empowering Farmers:
   - giving farmers a means of selling their produce to customers directly, doing away with middlemen and brokers who collect a commission.
   - granting farmers more authority over their pricing and a more equitable portion of the value they produce.

2. Enhancing Market Access:
   - introducing farmers to a larger pool of prospective customers, particularly in cities, in order to increase their market share and consumer base.
   - assisting smallholder farmers in connecting with customers and finding buyers for their produce, as they frequently struggle with limited market access.

3. Promoting Transparency and Fairness:
   - establishing an open market where price information is directly and readily visible to both buyers and sellers.
   - Eliminating opaque transactions and hidden costs in the agricultural supply chain will promote accountability and confidence.

4. Supporting Sustainable Practices:
   - encouraging a more sustainable food system, lowering the need for long-distance transportation, and facilitating the sale of fresh, locally grown goods.
   - establishing a connection between customers and farmers who are dedicated to high-quality, sustainable agriculture in order to promote ethical farming practices.

1.1.2 The Following are the objectives to consider during and after the development of this "Revolutionizing Agriculture" system

1. **Development & implementation**
   - **Technical Objectives:**
     - **Develop a Robust and secure Platform:** This consists of an easy-to-use mobile app, an intuitive web application, and a safe backend system that manages user, product, and transaction data through a database.
     - **Implement secure payment gateway:** Integrate your business with reputable payment processing companies to guarantee secure and easy transactions for buyers and farmers alike.
     - **Integrate mapping and location services:** Allow users to easily locate farms, markets, and delivery locations using maps and geo-location features.
     - **Develop a robust communication system:** Build in features like messaging, chat, and notifications to facilitate communication between farmers, buyers, and platform administrators.
     - **Implement a user-friendly interface:** Ensure that the platform is easy to navigate and use for both farmers and consumers, regardless of their tech-savviness.
   - **Business Objectives:**
     - **Attract a significant number of farmers:** Develop a marketing and outreach strategy to onboard a substantial number of farmers onto the platform.
     - **Onboard a critical mass of consumers:** Reach out to consumers through various channels (e.g., social media, online advertising, partnerships with local businesses) to create a steady demand for the products available on the platform.
     - **Develop strong relationships with farmers:** Provide excellent support, resources, and training to help farmers utilize the platform effectively and build confidence in the system.

- **Ensure fair pricing:** Collaborate with farmers to establish pricing that is both buyer-pleasing and competitive, reflecting the value of their produce.
- **Control operating costs:** Effectively manage platform development, maintenance, and marketing costs to guarantee long-term viability.

2. **Post-launch & Expansion**

- **Market Expansion Objectives:**
  - **Expand into new geographical areas:** Make a deliberate effort to locate new areas with high levels of consumer demand or agricultural production.
  - **Increase product diversity:** To accommodate a range of consumer preferences, encourage farmers to list a greater variety of crops, livestock, and agricultural products.
  - **Develop partnerships:** Collaborate with local governments, agricultural organizations, and food retailers to increase awareness and adoption of the platform.
- **Sustainability & Impact Objectives:**
  - **Reduce food waste:** Enable farmers to sell more of their produce, minimizing waste and maximizing the utilization of agricultural resources.
  - **Promote sustainable farming practices:** Assist farmers in implementing environmentally friendly farming practices and providing knowledge on sustainable farming.
  - **Empower farmers financially:** Demonstrate a clear positive impact on farmer income by comparing pre-platform and post-platform earnings.
  - **Contribute to local communities:** Highlight the platform's positive effects on rural economies and communities.
- **Data and Analysis Objectives:**
  - **Gather user data:** Track key metrics like user engagement, product sales, and customer feedback to understand platform usage and identify areas for improvement.

- **Implement analytics dashboards:** Provide farmers with insights into their sales performance, customer demographics, and market trends.
- **Use data to refine the platform:** Continuously improve the platform based on user feedback, sales data, and market trends.

1.2 Motivation

1.1.1 Motivations for the Project

- **A Family Tradition:** The initiative is motivated by the commitment of many generations of farmers and their desire to secure agricultural success while also improving their standard of living.
- **Passion for Fresh, Local Food:** There is a growing desire for access to healthy, delicious food that is locally grown and sustainably produced. A broker-free marketplace enables this connection between farmers and consumers.
- **Fairness and Equity:** Farmers frequently face difficulties in the agricultural industry, such as unjust pricing and restricted market access. By establishing a more egalitarian structure, the project hopes to ensure that farmers are fairly compensated for their labor.

1.1.2 Market Drivers:

- **Growing Consumer Demand:** The demand from consumers for sustainably derived, locally sourced food items is rising significantly.
- **Economic Challenges in the Agricultural Industry:** The agricultural industry faces numerous economic challenges, including low prices, rising input costs, and market volatility. A platform that connects farmers directly with consumers can offer a solution to these issues.

1.1.3 Core Value

The idea behind the project is the conviction that we need a food system that is more egalitarian, sustainable, and just. A better future for agriculture can result from direct relationships between

farmers and consumers, which will benefit both the people who grow and the people who eat our food.

1.3 Method

1.3.1 Methodologies Followed in Development Process

Software Development methodology

1. **Agile Development Methodology**
   - **Iterative and Incremental:** Agile emphasizes building the project in small, manageable increments called "sprints." This allows for flexibility, feedback integration, and continuous improvement throughout the development cycle.
   - **User-Centric Focus:** Agile methodologies prioritize user feedback and input. You'll engage farmers and consumers throughout the development process to gather requirements, test features, and refine the platform based on their needs.
   - **Cross-Functional Teams:** You'll work with a team of developers, designers, testers, and project managers who collaborate closely to deliver features and address issues promptly.
   - **Scrum Framework:** Scrum is a popular framework within agile development. It involves daily stand-up meetings, sprint planning, sprint reviews, and retrospectives to track progress and make adjustments.
2. **Waterfall Development Methodology**
   - **Linear and Sequential:** The waterfall method follows a linear approach, moving from one stage to the next in a fixed order.
   - **Well-Defined Requirements:** You'll spend considerable time upfront gathering and documenting detailed user requirements and specifications.
   - **Detailed Design and Planning:** Before coding begins, you'll create a comprehensive system design and a detailed project plan with clear milestones.
   - **Testing at the End:** Testing is performed at the end of the development cycle, rather than throughout the process.

3. **Lean Development Methodology**
    - **Minimal Viable Product (MVP):** Lean focuses on building a basic, functional version of the platform as quickly as possible (the MVP) and then iterating based on user feedback.
    - **Experimentation and Learning:** You'll conduct experiments, gather data, and make adjustments based on what you learn about user behavior and market demands.
    - **Eliminating Waste:** Lean emphasizes minimizing unnecessary effort and focusing on delivering value to users.
    - **Customer Validation:** You'll prioritize getting your platform in front of real users early in the development process to validate your assumptions and gather feedback.
4. **Design Development Methodology**
    - **Human-Centered Approach:** Design thinking places user needs and experiences at the center of the development process.
    - **Empathy:** You'll spend time understanding the needs, challenges, and motivations of farmers and consumers.
    - **Ideation and Prototyping:** You'll generate creative solutions and build prototypes to test and refine your ideas**.**
    - **Testing and Iteration:** You'll continuously test and iterate on your designs to ensure that they meet user needs and are effective.

1.3.2 The Selected Methodology

Agile methodology offers a framework that enables the development of a robust, user-friendly, and successful "Revolutionizing Agriculture" platform. Its core principles of flexibility, user-centricity, and iterative development align perfectly with the project's goals of empowering farmers and meeting the needs of a dynamic agricultural market.

1.3.2.1 Benefits of Agile Methodology for “Revolutionizing Agriculture”

1. **Flexibility and Adaptability**: Agile methodology is well-suited for projects with evolving requirements and dynamic environments. In the agricultural sector, farmers have unique needs, and market trends are constantly changing. Agile's iterative nature allows for adjustments to the platform based on feedback from farmers and consumers, ensuring its continuous relevance
2. **User-Centric Focus:** Agile prioritizes user needs and feedback. This is crucial for a project aimed at empowering farmers. The iterative development process encourages gathering feedback from farmers and consumers throughout the project. Features can be tested, refined, and improved based on real-world user input, leading to a platform that truly meets their needs.
3. **Faster Time to Market:** Agile encourages the rapid development and deployment of a Minimum Viable Product (MVP). This allows the project to get the platform into the hands of users early on, gather valuable insights, and validate assumptions. The iterative nature of Agile allows for continuous improvement and the addition of new features based on user feedback and market trends.
4. **Team Collaboration:** Agile promotes cross-functional team collaboration, bringing together developers, designers, testers, and other stakeholders. This fosters efficient communication and problem-solving. Agile methodologies emphasize open communication, frequent updates, and transparent feedback, ensuring alignment and preventing misunderstandings.
5. **Risk Management:** Agile development cycles allow for the early detection and mitigation of potential risks. This reduces the likelihood of costly mistakes or delays,

ensuring the project stays on track. Agile's iterative nature allows for adaptive planning, where adjustments can be made throughout the project to address unexpected challenges or changes in market conditions.

1.4 Overview

1.4.1 Technologies used in Development Process

1.4.1.1 used Technologies in Revolutionizing Agriculture

1. **Web Application**
    - **Frontend(HTML,CSS,Javascript):**
        - **Explain:** the core technologies needed to create user interfaces for websites. JavaScript provides interaction and dynamic capabilities, whereas HTML supplies the structure and CSS specifies the styling.
        - **Why chosen:** This classic combination is widely adopted for building modern web applications. It's flexible, adaptable, and has a vast community and resources available.
    - **Backend (PHP):**
        - **Explain:** A widely-used server-side scripting language for web development. It handles data logic, interacts with databases, and manages user authentication.
        - **Why chosen:** PHP's flexibility, scalability, and mature ecosystem make it a strong choice for managing the backend processes of the marketplace.

2. **Mobile Application**
    - **React native:**

      Explain: A cross-platform framework for building native mobile apps (iOS and Android) using JavaScript. It enables developers to create a single codebase for both platforms, saving time and resources while maintaining a native-like user experience.

      Why Chosen: React Native is popular for its performance, ease of use, and large developer community. It allows for faster development cycles and ensures a smooth user experience on both iOS and Android devices.

    - **Node.js:**

      Explain: A JavaScript runtime environment built on Chrome's V8 JavaScript engine. It's ideal for creating scalable server-side applications, APIs, and real-time applications.

      Why Choose: Node.js is used to build the mobile application's backend API, ensuring efficient communication between the frontend (React Native) and the database. Its asynchronous, non-blocking nature makes it well-suited for handling real-time updates and interactions within the marketplace.

1.4.2 Integration Development Environment (IDE)

This section outlines the integrated development environment (IDE) employed in the development of the "Revolutionizing Agriculture: A Broker-Free Marketplace for Farmers" project. IDEs provide developers with a comprehensive set of tools and features to streamline the coding process, enhance productivity, and ensure code quality.

1. Visual Studio Code (VS Code):
    - Strengths: Lightweight, fast, customizable, excellent for multiple languages (including PHP, Python, JavaScript, etc.), vast extension library, cross-platform compatibility, excellent debugging features.
    - Why it might be ideal for your Project:

- **Versatile**: your project involves various coding tasks (backend work, data analysis, scripting, etc.), VS Code's versatility and extensibility make it a great choice.
- **Community and Resources**: It has a huge and active community of developers, providing plenty of support, extensions, and tutorials.
- **Cross-platform Compatibility:** If you're working on a team with different operating systems, VS Code ensures everyone can use the same IDE seamlessly.

2. PyCharm (by JetBrains):
    - **Strengths:** Powerful IDE specifically for Python development, excellent for web development (with Django and Flask support), advanced debugging and refactoring tools.
    - **Why You Might Choose** It: If your project is heavily focused on Python development (e.g., data analysis, machine learning), PyCharm is a great choice.
3. IntelliJ IDEA (by JetBrains)
    - Strengths: Known for its powerful features, intelligent code completion, great debugging capabilities, support for multiple languages (including Java, Kotlin, Scala, etc.).
    - Why You Might Choose It: If your project uses Java or other JVM-based languages, IntelliJ IDEA is an excellent option.
4. Atom (by GitHub):
    - Strengths: Open-source, highly customizable, extensive package library, great for web development (especially with JavaScript).
    - Why You Might Choose It: If you want a highly customizable, open-source IDE that's very flexible, Atom might be a good choice.

1.4.3 Reason for Choosing the Visual Studio Code for Revolutionizing Agriculture

- **Flexibility:** VS Code's versatility to handle multiple languages is ideal if your project involves various coding tasks (e.g., database interactions, backend logic, data analysis).
- **Community:** The huge VS Code community provides a wealth of extensions, tutorials, and support, making it easier to find help and expand your capabilities.
- **Focus on Core Functionality:** VS Code is not as feature heavy as some IDEs (like PyCharm or IntelliJ), but its core functionality is highly efficient and intuitive, making it a productive development environment.

1.4.4 Used Technologies in Revolutionizing Agriculture Mobile Application Development

This section highlights the specific technologies employed for developing the mobile application component of the "Revolutionizing Agriculture" project. These technologies ensure a robust, scalable, and user-friendly mobile experience for both farmers and consumers.

- React Native:
  - Explain: A JavaScript framework for building native mobile applications for iOS and Android platforms using a single codebase. This significantly reduces development time and ensures a consistent user experience across both operating systems.
  - Why chosen: React Native is a popular choice for cross-platform mobile development due to its performance, ease of use, and large developer community. It's particularly suitable for projects where speed and efficiency are crucial.
- Node.js:
  - Explain: A JavaScript runtime environment built on Chrome's V8 JavaScript engine. It's ideal for creating scalable server-side applications, APIs, and real-time applications.
  - Why chosen: Node.js provides a robust and scalable backend for the mobile application, ensuring efficient communication between the React Native front end and the database. Its asynchronous, non-blocking nature makes it well-suited for handling real-time updates and interactions within the marketplace.

1.4.5 Database Management Systems (DBMS)

1.4.5.1 DBMS Options for Your Project

- MySQL: A popular open-source relational database, well-suited for structured data, known for its reliability, maturity, and support for large datasets. It's a good choice for managing data like product information, user profiles, and transactions.

- PostgreSQL: Another open-source relational database, known for its advanced features (like data integrity, complex queries, and object-relational mapping). It could be a good choice if your project requires complex data relationships or specific data types.

- MongoDB: A popular NoSQL database, good for handling semi-structured data like user preferences or product reviews. It offers high scalability and real-time updates, which could be beneficial for a growing marketplace.

- AWS DynamoDB: A fully managed, serverless NoSQL database offered by Amazon Web Services. It's known for its scalability, performance, and reliability. It can be a good choice if you need a highly scalable, cloud-based solution.

How to Choose:

- Evaluate your project's specific needs: Data model, volume, scalability, performance, security.
- Research and compare the options: Look at features, cost, and community support.
- Consider the experience of your team: Choose a DBMS that your team is familiar with to ensure smooth development and maintenance.

1.4.5.2 Reason For choosing My SQL Database Management System

MySQL is a widely used, open-source relational database management system (RDBMS). It stores data in tables with structured relationships, making it ideal for organizing and managing structured information.

Why Chosen:

- **Reliability and Maturity:** MySQL is a well-established and reliable DBMS with a large community and extensive support resources.
- **Scalability:** MySQL is designed to handle large datasets and can be scaled horizontally (adding more servers) to accommodate growth.
- **Cost-Effectiveness:** Being open-source, MySQL is free to use, making it an attractive option for projects with budget constraints.
- **Suitability for Project:** The relational nature of MySQL makes it a good fit for managing structured data within the marketplace, such as:
    - Product information (name, description, price, quantity, etc.)
    - Farmer profiles (contact details, location, specialties)
    - User accounts (login credentials, purchase history, preferences)
    - Transaction records (order details, payment information)

1.4.5.3 UI Designing Tools

Figma and Adobe XD are popular vector-based design tools widely used for UI/UX design, prototyping, and collaboration. They offer features for creating high-fidelity mockups, interactive prototypes, and sharing designs with stakeholders.

Why Chosen Figma

- **Visualizing the User Experience:** Even though you may not be building a front-end application, using these tools helps to visualize how users will interact with the system, such as the flow of the marketplace, the information displayed to farmers and consumers, and the overall user interface.
- **Prototyping:** Prototyping with Figma or Adobe XD allows you to test and iterate on the design of your system without writing code. This helps identify usability issues early on, ensuring a smoother user experience.
- **Communication:** Sharing prototypes with stakeholders and developers allows for clearer communication and collaboration on the project's design and functionality.

Chapter 02 Background and Problem Statement

2.1 Introduction

The agricultural sector, a critical pillar of global economies, faces numerous challenges that impact both farmers and consumers. One pressing issue is the lack of direct access to markets for smallholder farmers, often leading to unfair pricing and limited profitability. This problem is exacerbated by the presence of intermediaries who control distribution channels, profiting at the expense of farmers and driving up costs for consumers. This creates an unequal and unsustainable food system that hinders the growth of the industry and impacts food security

2.2 Literature Review

2.2.1 Overview

The agricultural sector is facing numerous challenges, including inefficiencies in the supply chain, price volatility, and limited market access for smallholder farmers, which contribute to food insecurity and poverty (World Bank, 2020). A broker-free marketplace for farmers, facilitated by web and mobile technologies, has the potential to address these challenges and transform the agricultural landscape, promoting inclusive and sustainable food systems (FAO, 2021).

2.2.2 Benefits of a Broker-Free Marketplace

By eliminating intermediaries, a broker-free marketplace reduces transaction costs and increases farmers' profits, empowering them economically (UNDP, 2019). It also improves price discovery, leading to fairer and more transparent pricing, benefiting both farmers and consumers (IFPRI, 2020). Furthermore, it expands market access for smallholder farmers, connecting them with buyers from various regions and reducing food waste, contributing to more efficient and sustainable food systems (World Economic Forum, 2018).

2.2.3 Role of Mobile and Web Technologies

Mobile and web technologies play a crucial role in the success of a broker-free marketplace for farmers. Mobile apps allow farmers to access the marketplace, manage their listings, receive notifications, and communicate with buyers, regardless of their location, bridging the digital divide and empowering them to participate in the digital economy (UNDP, 2019). Web applications provide a comprehensive platform for buyers to browse products, compare prices, and place orders, facilitating payment processing, logistics coordination, and customer support, enhancing the overall user experience and efficiency of the marketplace (World Economic Forum, 2018).

2.2.4 Challenges and Opportunities

While a broker-free marketplace offers numerous benefits, it also presents challenges. These include digital literacy limitations among some farmers, poor internet connectivity in rural areas, and establishing trust and security in an online environment (World Bank, 2020; FAO, 2021). Despite these challenges, the opportunities presented by a broker-free marketplace are substantial, and governments, industry stakeholders, and technology companies can collaborate to unlock its full potential, contributing to the achievement of the Sustainable Development Goals (IFPRI, 2020).

2.2.4 Conclusion

A broker-free marketplace for farmers, powered by web and mobile technologies, has the potential to revolutionize the agricultural sector by reducing costs, improving price discovery, increasing market access, reducing food waste, and empowering farmers, contributing to more inclusive and sustainable food systems (World Economic Forum, 2018). Overcoming challenges related to digital literacy, infrastructure limitations, and trust and security will be crucial to ensuring the widespread adoption and success of such a marketplace, enabling it to transform the lives of farmers and consumers alike (FAO, 2021).

2.3 Problem Statement

Smallholder farmers face significant challenges in accessing fair markets and earning a sustainable income, leading to poverty, food insecurity, and a less equitable agricultural system.

Specifically:

- **Lack of access to markets**: Farmers often lack access to transportation, infrastructure, and information about potential buyers, forcing them to sell their produce at low prices to intermediaries or let their crops spoil in the fields.
- **Exploitation by intermediaries:** Middlemen and brokers often take a large share of the profit, leaving farmers with limited income and reducing their ability to invest in improving their farms and production.
- **Lack of transparency and price volatility:** Farmers have limited control over the pricing of their products, leaving them vulnerable to market fluctuations and unfair practices.

This situation creates a vicious cycle of poverty, food insecurity, and environmental degradation, as farmers struggle to make a living and invest in sustainable agricultural practices.

This project aims to address this problem by creating a broker-free marketplace that directly connects farmers with consumers, empowering farmers to earn fair prices for their produce and promoting a more sustainable and equitable agricultural system.

Chapter 3 Project Management

3.1 Approach

The approach to this project was meticulously planned to ensure a successful and impactful outcome. Here's a breakdown of the key elements and the rationale behind each:

1. **Problem Identification and Research:**

- **Understanding the Pain Points: Initial research focused on identifying the specific problems faced by farmers in accessing fair markets and receiving competitive prices for their produce. This involved:**
  - Studying existing agricultural supply chains.
  - Analyzing the role of intermediaries and their impact on farmer income.
  - Gathering insights from farmers themselves through surveys and interviews.
- **Literature Review: A thorough literature review explored existing solutions and technologies in the field of agricultural marketplaces, focusing on:**
  - Success stories of existing platforms.
  - The potential and limitations of blockchain technology in agriculture.
  - The role of mobile and web technologies in empowering farmers.
  - The challenges and opportunities of creating a broker-free

2. **Feasibility Analysis:**

- **Technical Feasibility: Evaluating the availability and compatibility of technologies required for building the web and mobile applications. This involved analyzing:**
  - The suitability of PHP, React Native, and MySQL for the project's needs.
  - The availability of resources and expertise for development.
  - The potential challenges in scaling the platform to handle future growth.
- **Economic Feasibility: Assessing the potential for profitability and return on investment. This involved:**
  - Analyzing the target market size and potential user base.
  - Considering the cost of development and operation.

- Examining the potential for revenue generation through commission fees or subscription models.

- **Operational Feasibility: Ensuring the project can be implemented effectively and managed efficiently. This involved:**
  - Creating a realistic timeline and budget.
  - Identifying potential risks and mitigation strategies.
  - Assessing the availability of resources like manpower and infrastructure

3.2 Initial project Plan

Phase 1: Requirements and Design

Task	Start Date	End Date	Duration
1.1 Market Research and Analysis	**2023-10-20**	**2023-11-10**	**3 Weeks**
1.2 User Interviews and Feedback	**2023-11-11**	**2023-11-25**	**2 Weeks**
1.3 System Architecture Design	**2023-11-26**	**2023-12-10**	**2 weeks**
1.4 Database Design	**2023-12-11**	**2023-12-25**	**2 weeks**
1.5 UI/UX Design(web and Mobile)	**2023-12-26**	**2024-01-10**	**2 weeks**

Table 1 Requirements and Design

Phase 2: Development

Task	Start Date	End Date	Duration
2.1 Front-End Development (web)	2024-01-11	2024-02-10	4 Weeks
2.2 back-End Development	2024-01-11	2024-02-10	4 weeks
2.3 Mobile App Development (React Native)	2024-01-11	2024-02-10	4 weeks
Database integration	2024-02-11	2024-02-25	2 Weeks

Table 2 Development

Phase 3: Testing and Deployment

Task	Start Date	End Date	Duration
3.1 Unit Testing	2024-02-26	2024-03-10	2 Weeks
3.2 integration Testing	2014-03-11	2024-03-25	2 Weeks
3.3 user Acceptance Testing	2024-03-26	2024-04-10	2 Weeks
3.4 Deployment	2024-04-11	2024-04-25	2 Weeks

Table 3 Testing and Deployment

Phase 4: Maintenance and Updates

Task	Start Date	End Date	Duration
Monitoring and Bug Fixing	**2024-04-26**	**Ongoing**	**Ongoing**
Feature Updates and Enhancement's	**2024-05-01**	**Ongoing**	**Ongoing**

Table 4 maintenance and Updates

3.3 Problem and Changes to the initial Project plan: "Revolutionizing Agriculture"

While the initial project plan provided a solid framework, several challenges arose that required adjustments to ensure the project's success. Here are the key problems encountered and the corresponding changes made to the plan:

1. **Problem: Underestimation of Development Complexity**
    - **Initial Plan:** The initial timeline for development (Weeks 11-16) was based on a somewhat optimistic assessment of the project's complexity.
    - **Changes:**
        - **Extended Development Timeline:** The development phase was extended by two weeks (Weeks 11-18) to accommodate the more complex functionalities, particularly in the mobile app development with React Native.
        - **Task Refinement:** The development tasks were further broken down into smaller, more manageable units to improve progress tracking and resource allocation.

Justification: The initial plan underestimated the time required for complex tasks like integrating various features, ensuring smooth communication between web and mobile applications, and developing secure payment gateways.

2. **Problem: Limited User Feedback in Early Stages**
    - **Initial Plan:** User feedback was primarily planned for the User Acceptance Testing phase (Week 20).
    - **Changes:**
    - **Early User Feedback:** The plan was modified to incorporate user feedback earlier in the design and development phases (Weeks 4-8). This involved:
        - Conducting user testing of wireframes and prototypes.
        - Gathering feedback from farmers and consumers on key features and user flow.
        - Utilizing online surveys and focus groups for broader user input.

Justification: Collecting feedback early in the project allowed for addressing potential issues and user needs before committing to significant development efforts.

3. **Problem: unexpected Database Integration Issues**
    - **Initial Plan:** Database integration was allocated a short timeframe (Week 15).
    - **Changes:**
        - **Extended Database Integration Phase:** The timeframe for database integration was extended to two weeks (Week 15-16) to accommodate potential challenges and ensure a robust integration process.
        - **Dedicated Database Developer:** The plan incorporated the addition of a dedicated database developer to the team to focus solely on the complex integration tasks and optimize performance.

Justification: The database integration process proved to be more complex than anticipated due to the need to ensure data security, handle large volumes of transactions, and optimize performance.

4. **Problem: Lack of Emphasis on Security Measures**
    - **Initial Plan:** Security measures were primarily planned for the deployment phase (Week 22).
    - **Changes:**
        - **Early Security Integration:** Security considerations were integrated into the development process from the outset. This involved:
            - Conducting security audits of code during development.
            - Implementing encryption and authentication measures for user data and transactions.
            - Consulting with security experts for best practices

Justification: Integrating security measures early in the project ensured a more secure platform from the ground up and reduced the risk of vulnerabilities during deployment.

3.4 Final Project Gantt Chart

ID	Name	Start Date	End Date	Duration
1	Collecting User Reqirements	Sep 28, 2023	Oct 04, 2023	7 days
2	Get Ready For Proposal	Oct 07, 2023	Oct 10, 2023	4 days
3	Creating User Interface	Oct 11, 2023	Oct 26, 2023	16 days
4	Get Feedback for UI	Oct 27, 2023	Nov 06, 2023	11 days
5	Frontend Development for Web Application	Nov 07, 2023	Nov 30, 2023	24 days
6	Backend Development For Web Application	Dec 01, 2023	Jan 21, 2024	52 days
7	Testing	Jan 22, 2024	Jan 24, 2024	3 days
8	Bug Fixing	Jan 25, 2024	Jan 31, 2024	7 days
9	Develop Mobile Application	Feb 01, 2024	Mar 09, 2024	38 days
10	Testing Mobile App	Mar 10, 2024	Mar 17, 2024	8 days
11	Implementing the Mobile app	Mar 18, 2024	May 21, 2024	65 days
12	Get feedback all about the system	May 22, 2024	Jun 02, 2024	12 days

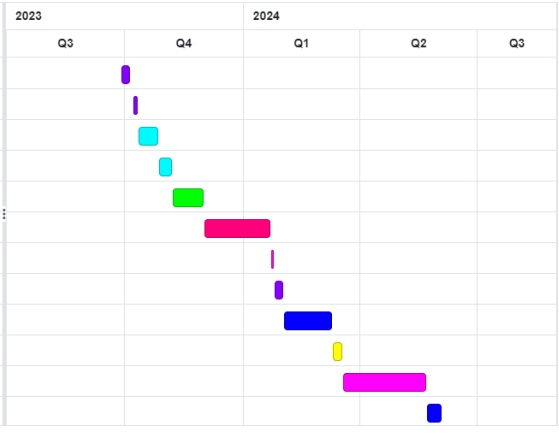


Figure 1 Gant Chart

Chapter 04 Feasibility Study

This feasibility study assesses the viability of creating a broker-free online marketplace for farmers, connecting them directly with consumers. The study considers technical, economic, operational, and social aspects.

4.1 Time Feasibility

Time feasibility is indeed crucial for the success of any project, including "Revolutionizing Agriculture." Here's an analysis of the time feasibility of this project, considering potential challenges and mitigation strategies:

Initial Timeline:

The initial project plan outlined a timeline of 24 weeks, aiming for a deployment by week 24. However, as discussed earlier, the initial timeline proved to be overly optimistic due to underestimating the complexity of certain tasks.

Revised Timeline:

The revised timeline, after incorporating feedback and addressing challenges, extends the project duration to 28 weeks. This revised plan, while longer, provides a more realistic timeframe to ensure a robust and feature-rich platform.

Key Factors Impacting Time Feasibility:

- Development Complexity: Building a comprehensive platform with features like secure payment gateways, advanced search functionalities, and mobile app compatibility requires significant development time.
- Integration of Multiple Systems: Connecting different systems like payment gateways, logistics providers, and data analysis tools can be complex and time-consuming.

- User Testing and Feedback: Iterating on designs based on user feedback and ensuring a user-friendly experience necessitates additional time for testing and adjustments.
- Team Availability and Skillset: A dedicated and skilled team with expertise in web development, mobile app development, and database management is essential for efficient execution.
- Unexpected Challenges: Unforeseen issues, bugs, and technical difficulties can arise, adding to the overall project duration.

Strategies to Enhance Time Feasibility:

- Agile Development: Adopting an agile development methodology allows for flexibility, quicker iterations, and continuous progress monitoring.
- Prioritization of Core Features: Focusing on essential features first, while prioritizing non-essential features for later releases, can expedite the initial launch.
- Team Collaboration and Communication: Efficient communication and clear task delegation among team members minimize delays caused by miscommunication or lack of coordination.
- Resource Allocation: Ensuring the right resources, including skilled developers, testers, and project managers, are assigned to each task enhances efficiency.
- Risk Management: Identifying potential risks early on and developing contingency plans for addressing them reduces the impact of unforeseen challenges.
- Regular Monitoring and Adjustments: Periodically reviewing progress, identifying bottlenecks, and making necessary adjustments to the timeline keeps the project on track.

4.2 Technical Feasibility

- Existing Technologies: The project leverages readily available and mature technologies:
  - **Web Development:** HTML, CSS, JavaScript, PHP, and related frameworks are widely used and well-documented.
  - **Mobile App Development:** React Native allows for building cross-platform mobile apps for iOS and Android with a single codebase.

  - **Database Management:** MySQL is a robust and scalable relational database management system.
  - **Cloud Hosting:** Cloud providers like AWS, Azure, and Google Cloud offer reliable and scalable infrastructure for hosting the web and mobile applications.
- **Development Expertise:** Skilled developers are readily available with expertise in the required technologies.
- **Scalability:** The chosen technologies can handle a significant number of users and transactions, making the platform scalable for future growth.
- **Security:** Implementing secure authentication, encryption, and other security measures is essential and feasible using established best practices.
- **Integration:** The platform can integrate with existing payment gateways and logistics providers for seamless transactions and deliveries.

4.2 Economic Feasibility

- Market Demand: There is a growing demand for locally sourced and sustainably produced food, creating a large potential market for the platform.
- Revenue Streams: Potential revenue streams include:
  - Subscription Fees: Offering premium features to farmers for a monthly subscription.
  - Advertising: Allowing businesses to advertise their products or services on the platform.
- Cost of Development: While development costs are significant, they can be mitigated through efficient resource allocation, open-source tools, and cloud-based infrastructure.
- Profitability: The potential for profitability is high, given the large market demand and potential revenue streams.

Conclusion: The project is economically feasible with a strong market demand, diverse revenue streams, and the potential for significant profitability.

4.3 Scope Feasibility:

Scope feasibility is about ensuring that the project's objectives are achievable within the defined scope and resources. This involves a thorough evaluation of the project's technical requirements, functionality, and limitations. Here's an assessment of the scope feasibility of the "Revolutionizing Agriculture" project:

Scope Definition:

Objectives:

- To create a broker-free marketplace that connects farmers directly with consumers.
- To provide farmers with a platform to list their products, manage orders, and communicate with buyers.
- To offer consumers an easy-to-use platform to search for and purchase fresh, locally sourced produce.
- To promote transparency and fair pricing for farmers.

Core Features:

- Farmer registration and profile management.
- Product listing and management for farmers.
- Search and browse functionalities for consumers.
- Secure online payment integration.
- Order tracking and communication features.
- Rating and review system.

Chapter 05 System Design

5.1 Diagrams

5.1.1 Database Diagram

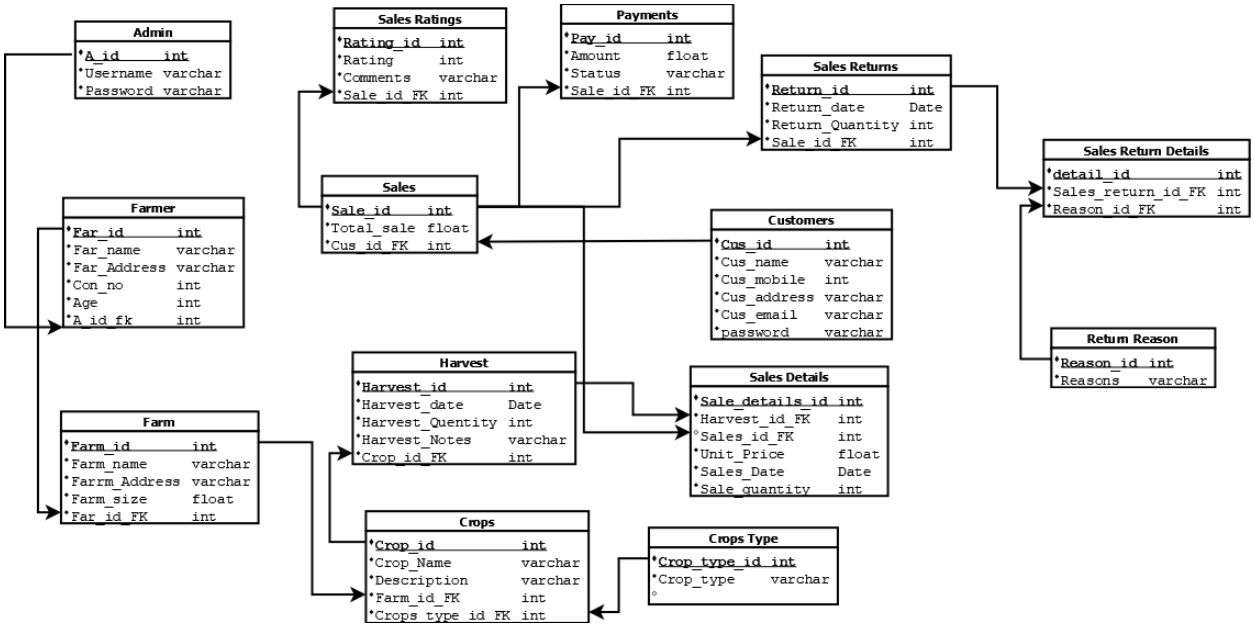


Figure 2 Database Diagram

Diagram Breakdown Structure

- **Admin:**

A_id: Primary Key.

Username: Admin's username.

Password: Admin's password.

Relationships: Admins have multiple farmers (Far_id in Farmer table as a foreign key).

- **Farmer:**

Far_id: Primary Key.

Far_name: Farmer's name.

Far_Address: Farmer's address.

Con_no: Contact number.

Age: Age of the farmer.

A_id_fk: Foreign Key referencing A_id in the Admin table.

Relationships: Each farmer is linked to an admin and manages multiple farms (Farm_id in Farm table as a foreign key).

- **Farm:**

Farm_id: Primary Key.

Farm_name: Name of the farm.

Farm_Address: Address of the farm.

Farm_size: Size of the farm.

Far_id_FK: Foreign Key referencing Far_id in the Farmer table.

Relationships: Each farm is managed by one farmer and produces multiple crops (Crop_id in Crops table as a foreign key).

- **Crops:**

Crop_id: Primary Key.

Crop_Name: Name of the crop.

Description: Description of the crop.

Farm_id_FK: Foreign Key referencing Farm_id in the Farm table.

Crops_type_id_FK: Foreign Key referencing Crops_type_id in the Crops Type table.

Relationships: Each crop belongs to a farm and a crop type, and each harvest record (Harvest_id in Harvest table) is linked to a crop.

- **Crops Type:**

Crops_type_id: Primary Key.

Crop_type: Type of crop.

Relationships: Each crop type can have multiple crops.

- **Harvest:**

Harvest_id: Primary Key.

Harvest_date: Date of harvest.

Harvest_Quantity: Quantity harvested.

Harvest_Notes: Notes about the harvest.

Crop_id_FK: Foreign Key referencing Crop_id in the Crops table.

Relationships: Each harvest record is linked to a crop and can be linked to multiple sales details records (Sales_details_id in Sales Details table).

- **Sales:**

Sale_id: Primary Key.

Total_sale: Total sale amount.

Cus_id_FK: Foreign Key referencing Cus_id in the Customers table.

Relationships: Each sale record is linked to a customer and can have multiple sales details records (Sales_details_id in Sales Details table).

- **Sales Details:**

Sales_details_id: Primary Key.

Harvest_id_FK: Foreign Key referencing Harvest_id in the Harvest table.

Sales_id_FK: Foreign Key referencing Sale_id in the Sales table.

Unit_Price: Unit price of the crop sold.

Sales_Date: Date of sale.

Sale_quantity: Quantity sold.

Relationships: Each sales detail is linked to a harvest record and a sale record.

- **Customers:**

Cus_id: Primary Key.

Cus_name: Customer's name.

Cus_mobile: Customer's mobile number.

Cus_address: Customer's address.

Cus_email: Customer's email.

password: Customer's password.

Relationships: Each customer can have multiple sales records.

- **Payments:**

Pay_id: Primary Key.

Amount: Payment amount.

Status: Payment status.

Sale_id_FK: Foreign Key referencing Sale_id in the Sales table.

Relationships: Each payment is linked to a sale record.

- **Sales Ratings:**

Rating_id: Primary Key.

Rating: Rating given by the customer.

Comments: Comments on the sale.

Sale_id_FK: Foreign Key referencing Sale_id in the Sales table.

Relationships: Each sales rating is linked to a sale record.

- **Sales Returns:**

Return_id: Primary Key.

Return_date: Date of return.

Return_Quantity: Quantity returned.

Sale_id_FK: Foreign Key referencing Sale_id in the Sales table.

Relationships: Each sales return is linked to a sale record and can have multiple sales return details records (detail_id in Sales Return Details table).

- **Sales Return Details:**

detail_id: Primary Key.

Sales_return_id_FK: Foreign Key referencing Return_id in the Sales Returns table.

Reason_id_FK: Foreign Key referencing Reason_id in the Return Reason table.

Relationships: Each sales return detail is linked to a sales return record and a return reason.

- **Return Reason:**

Reason_id: Primary Key.

Reasons: Reason for the return.

Relationships: Each return reason can be linked to multiple sales return details records.

5.1.2 Use Case Diagram

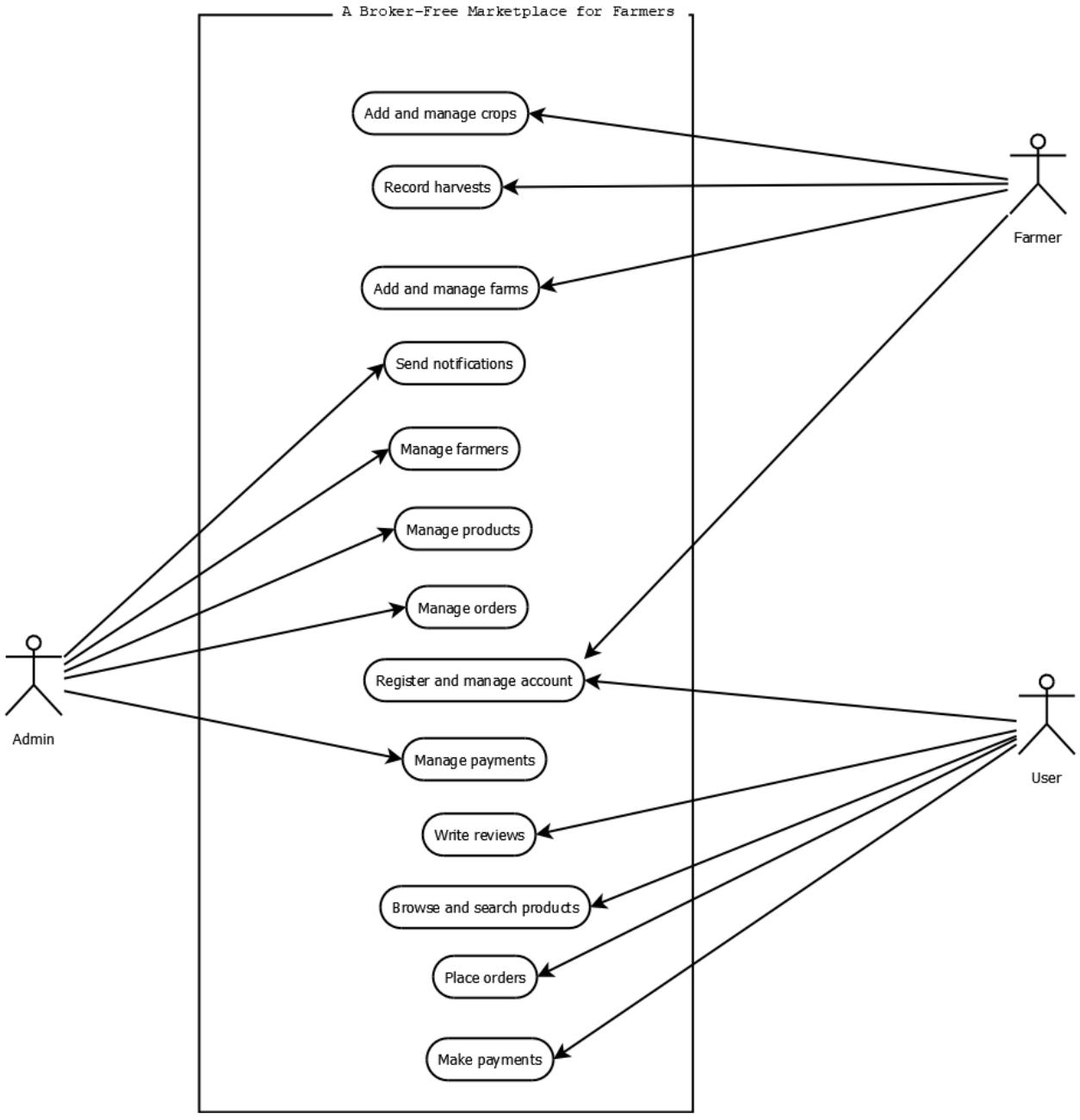


Figure 3 Use Case Diagram

5.2 User Interface Design

5.2.1 Web Applications

1. Admin Login

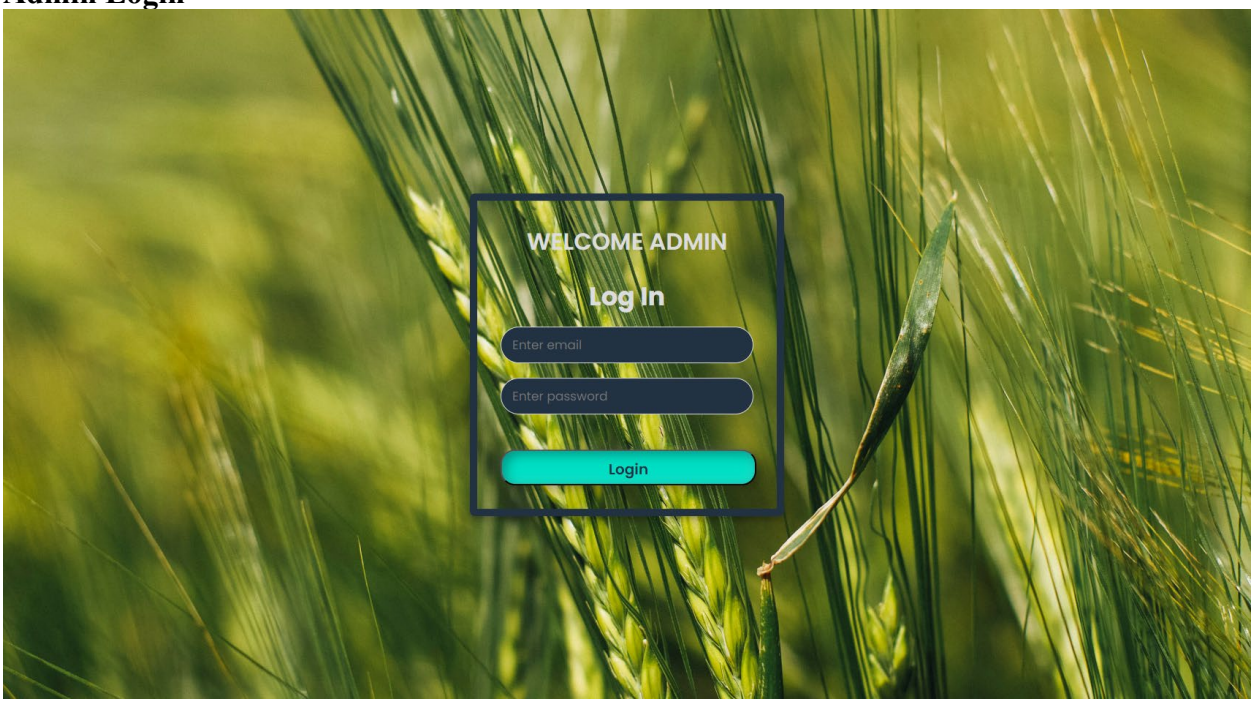


Figure 4 Admin Login

2. Admin Dashboard

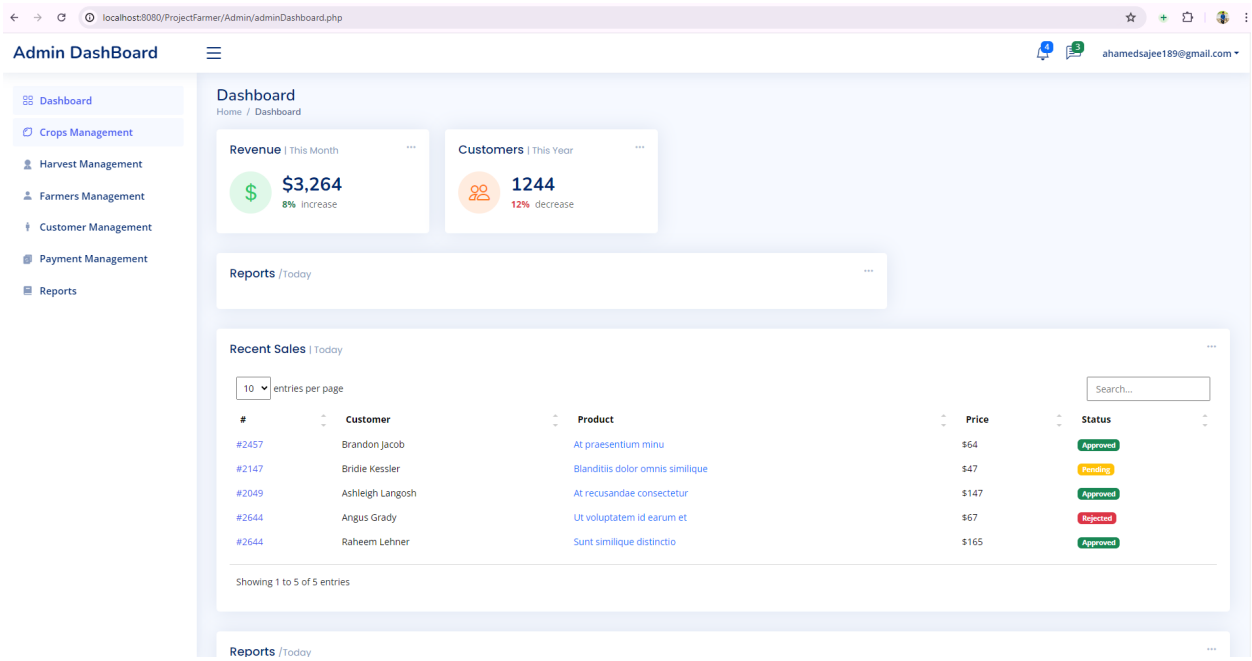


Figure 5 Admin Dashboard

3. Add Crops Form

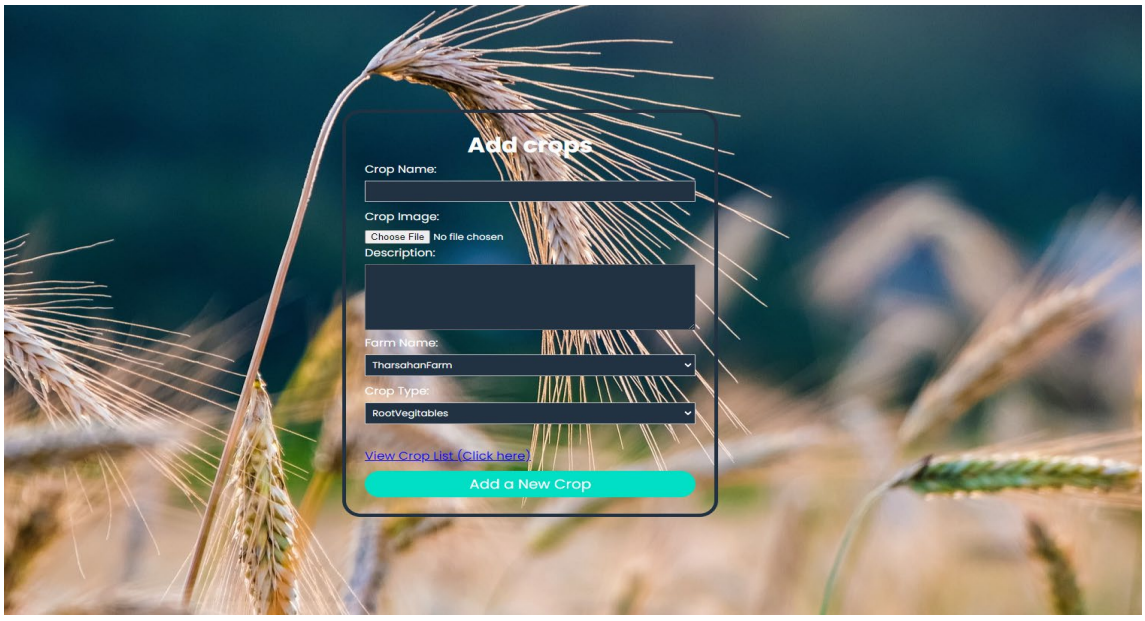


Figure 6Add Crops

4. View Crops

Figure 7 View Crops

5. Delete Crops

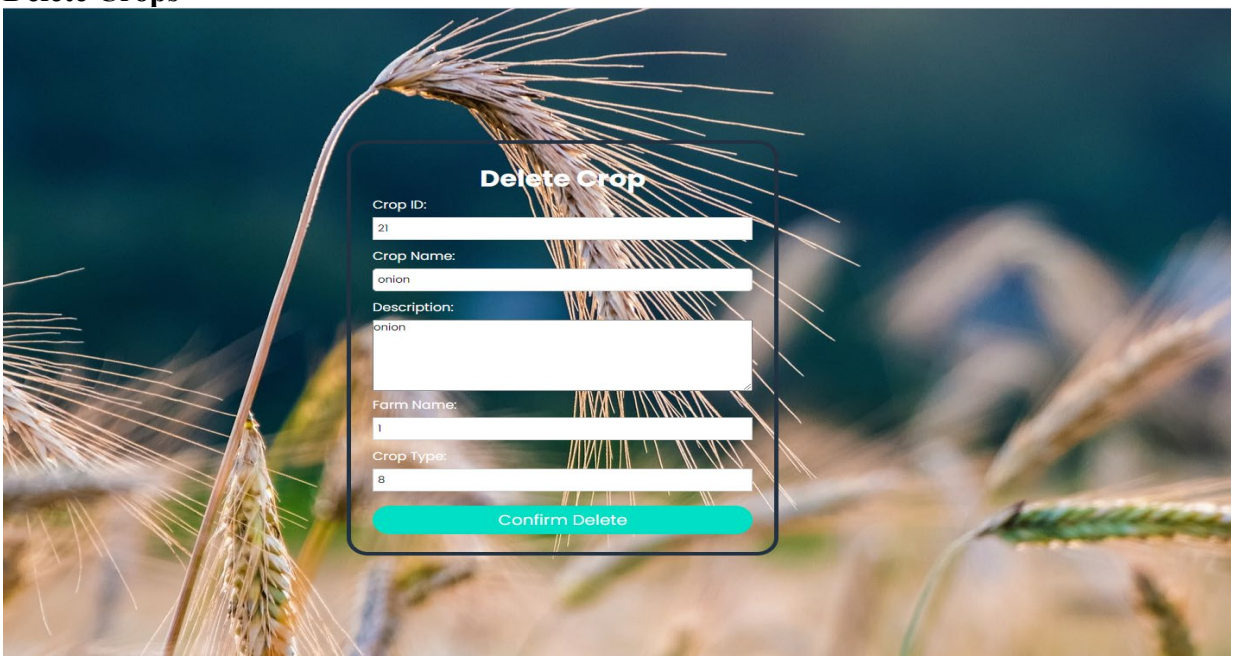


Figure 8 Delete Crops

6. **Add Harvest**

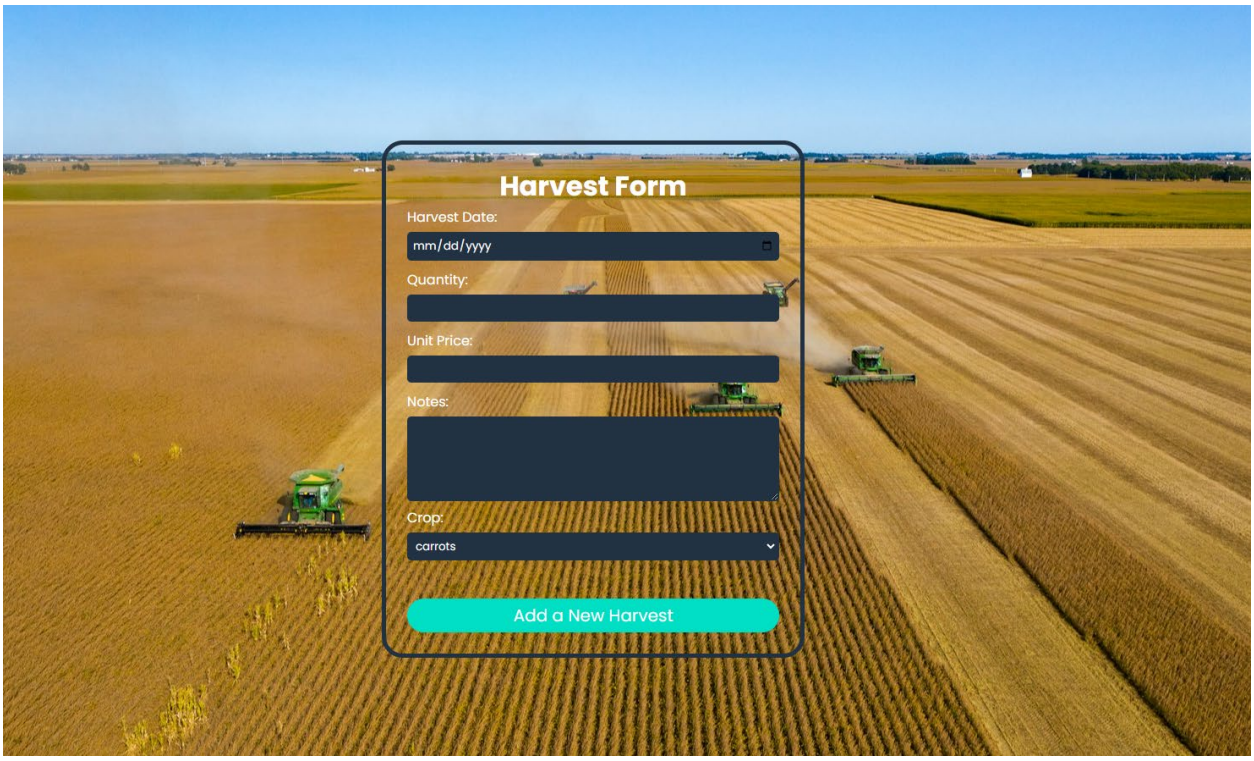


Figure 9 Add a Harvest

7. View harvest Details

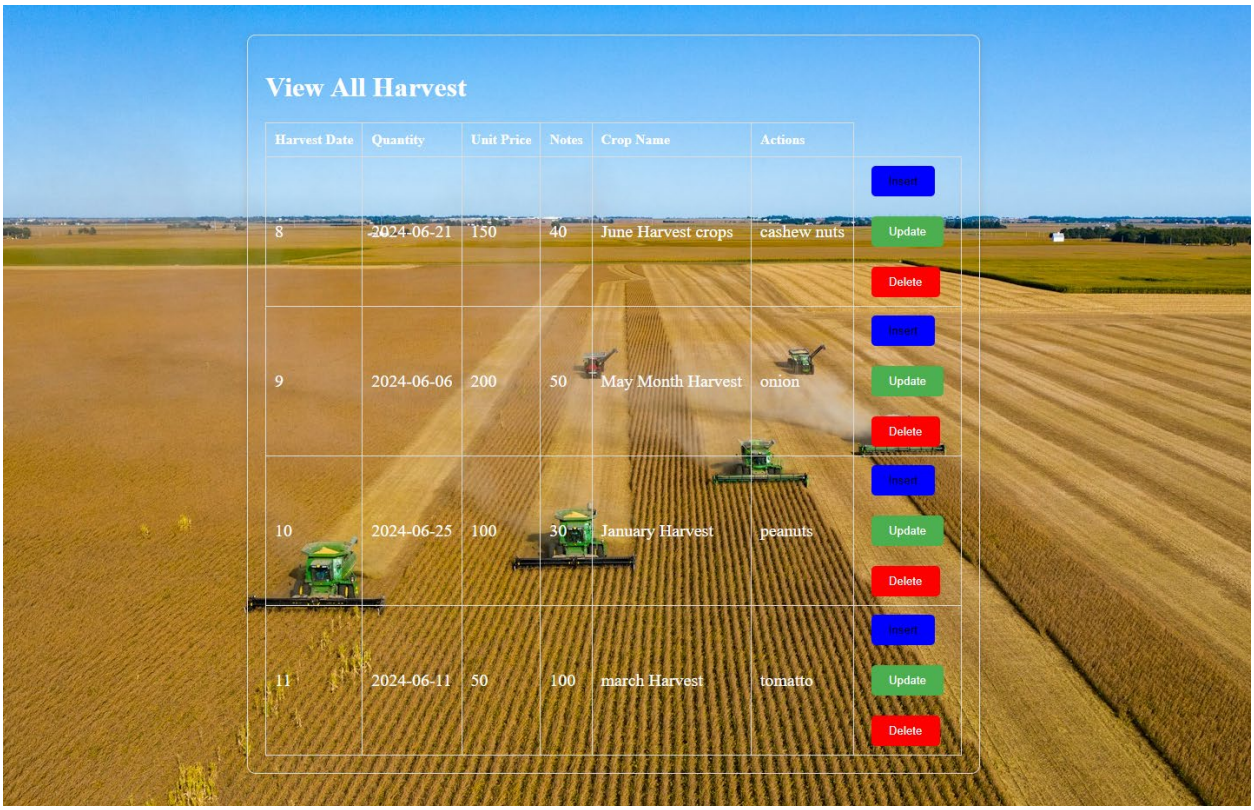


Figure 10 View Harvest Details

8. Update Harvest Details

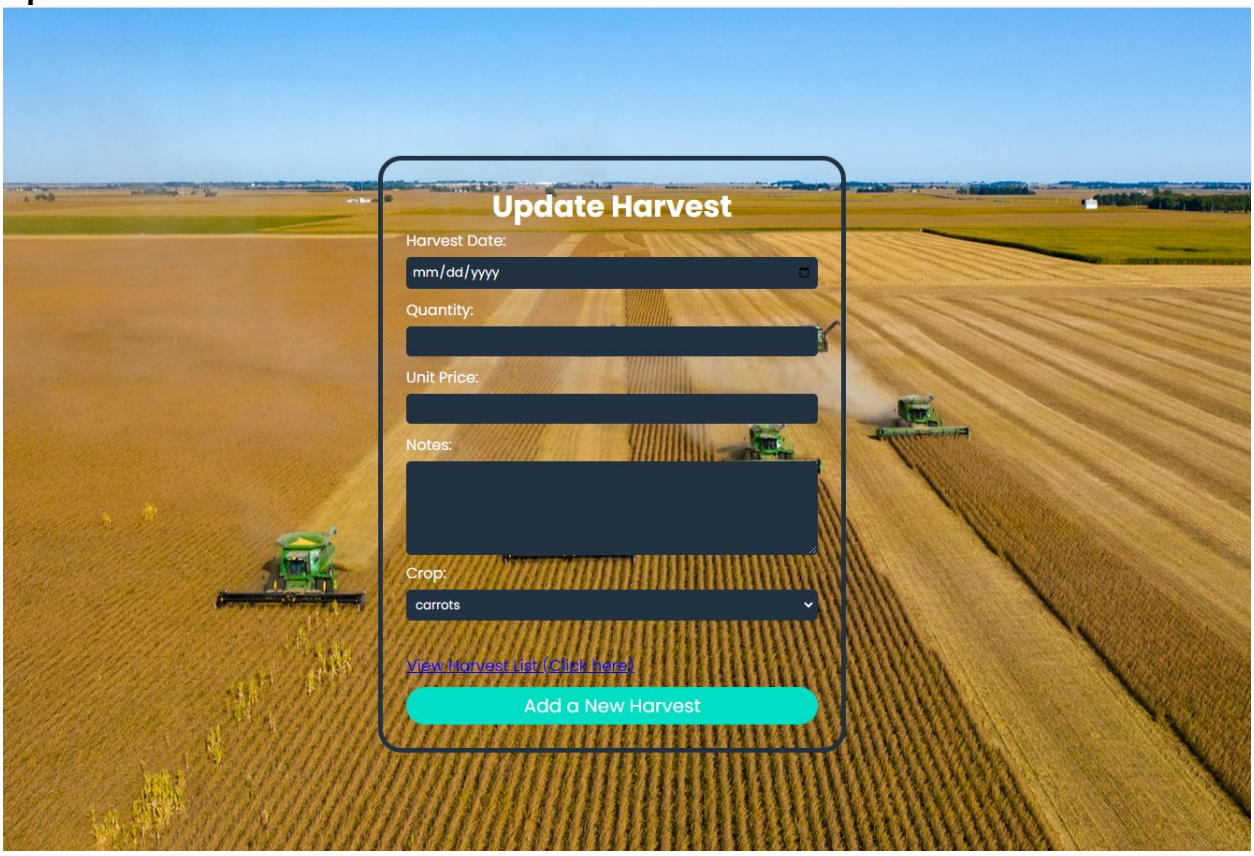


Figure 11 Update Harvest

9. **Add Farm**

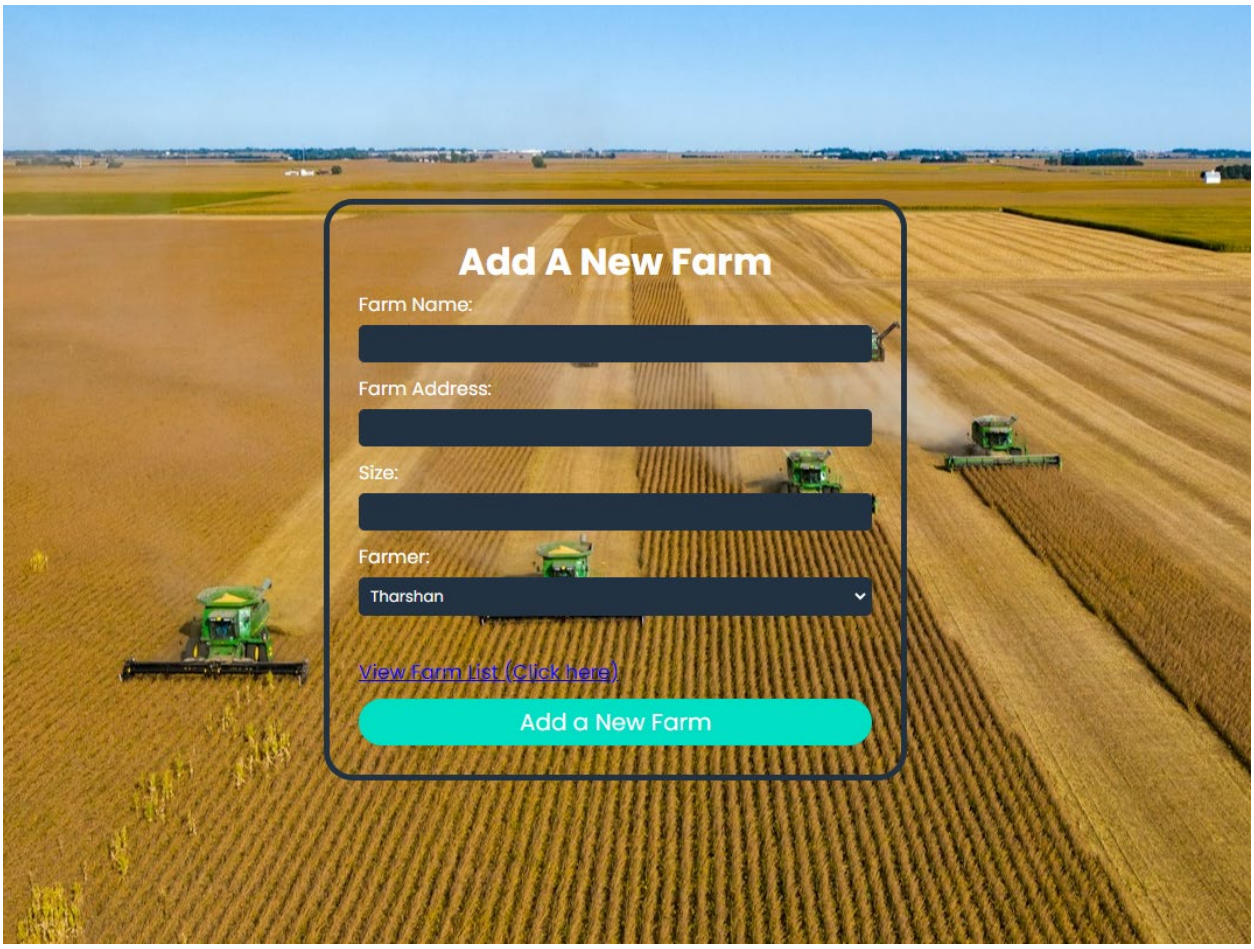


Figure 12 Add A New Farm

10. update Farm

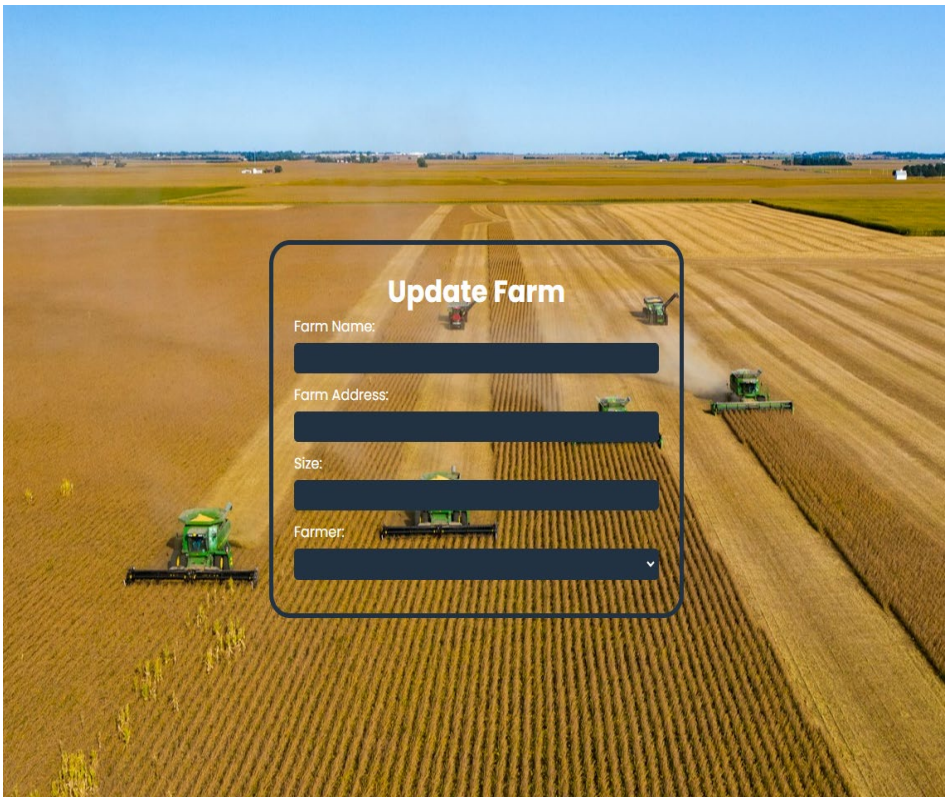


Figure 13 Update Farm

5.2.2 Mobile Application

1. **Starting Screen with logo**

Figure 14 Start app screen

2. **Customer Login Screen**

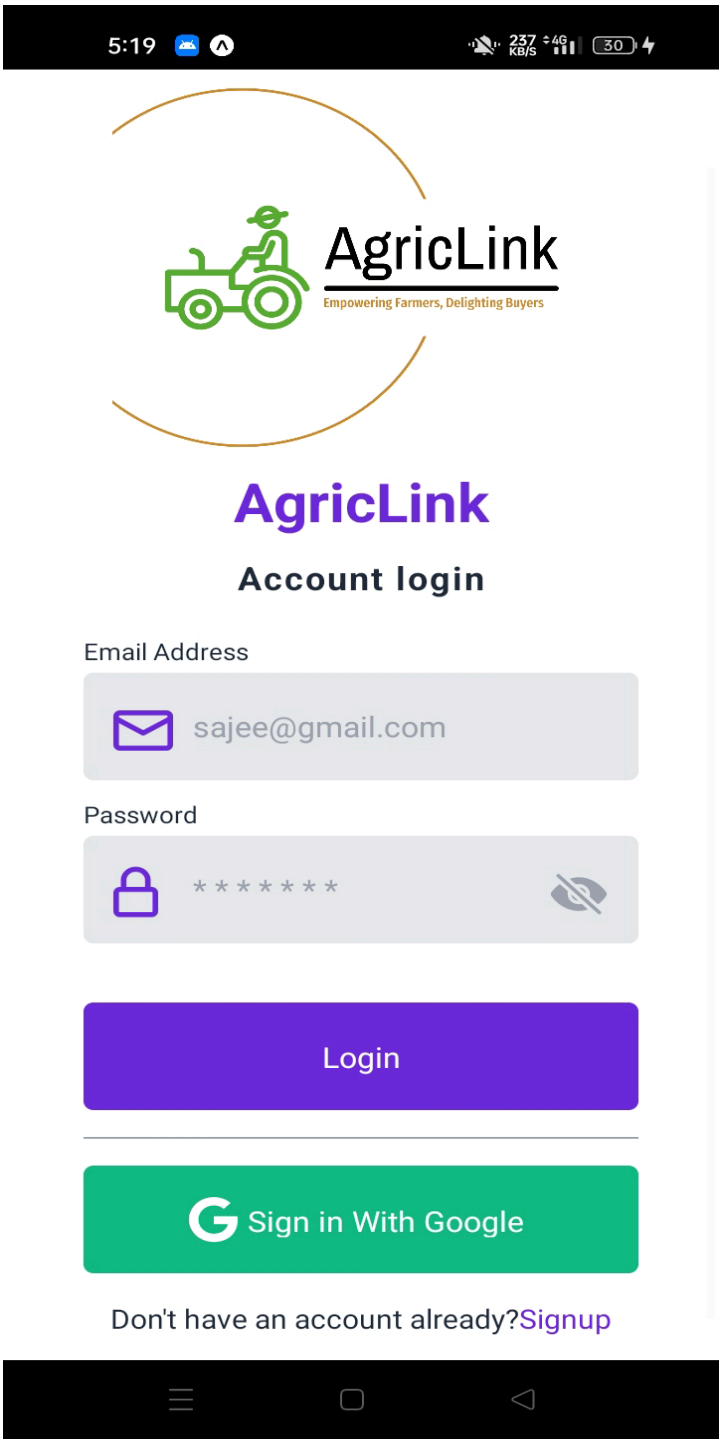


Figure 15 Customer Login Screen

3. **Customer Registration Screen**

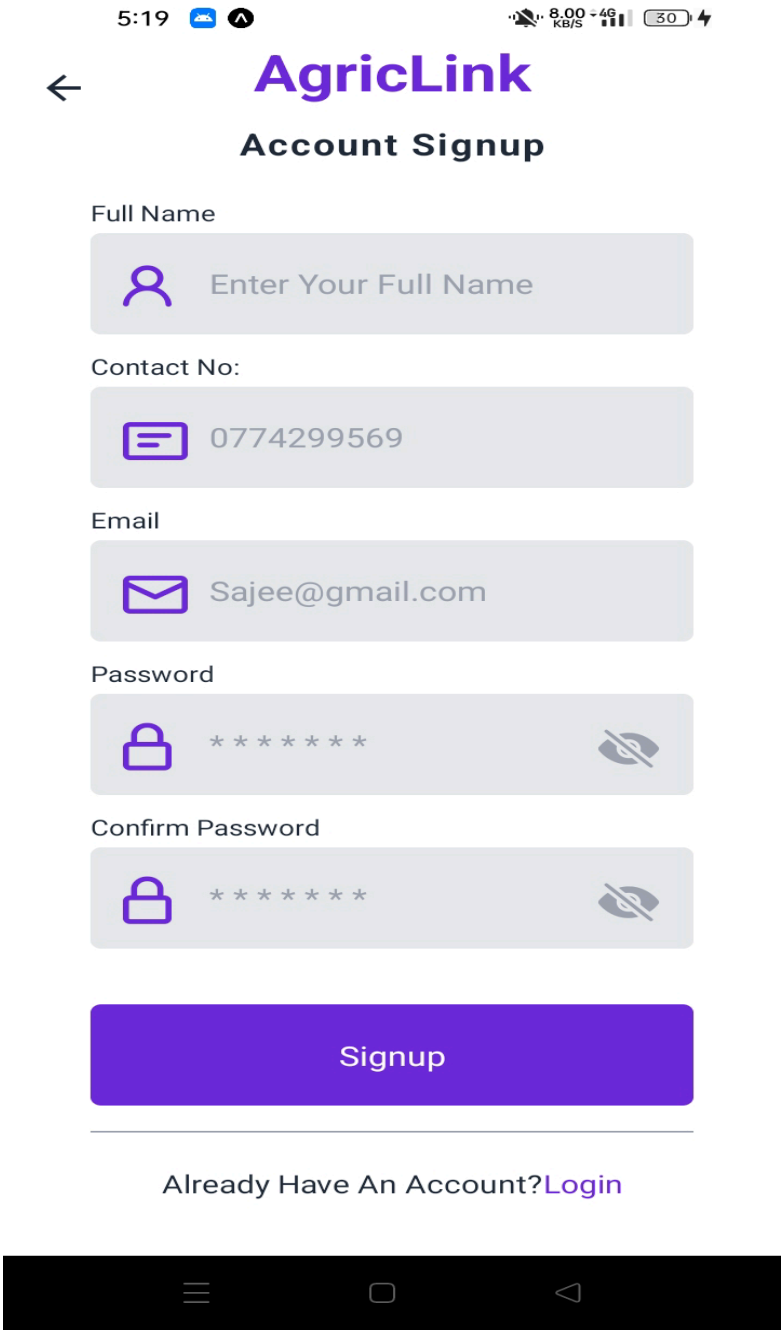


Figure 16 Customer Registration Screen

4. **Welcome Screen**

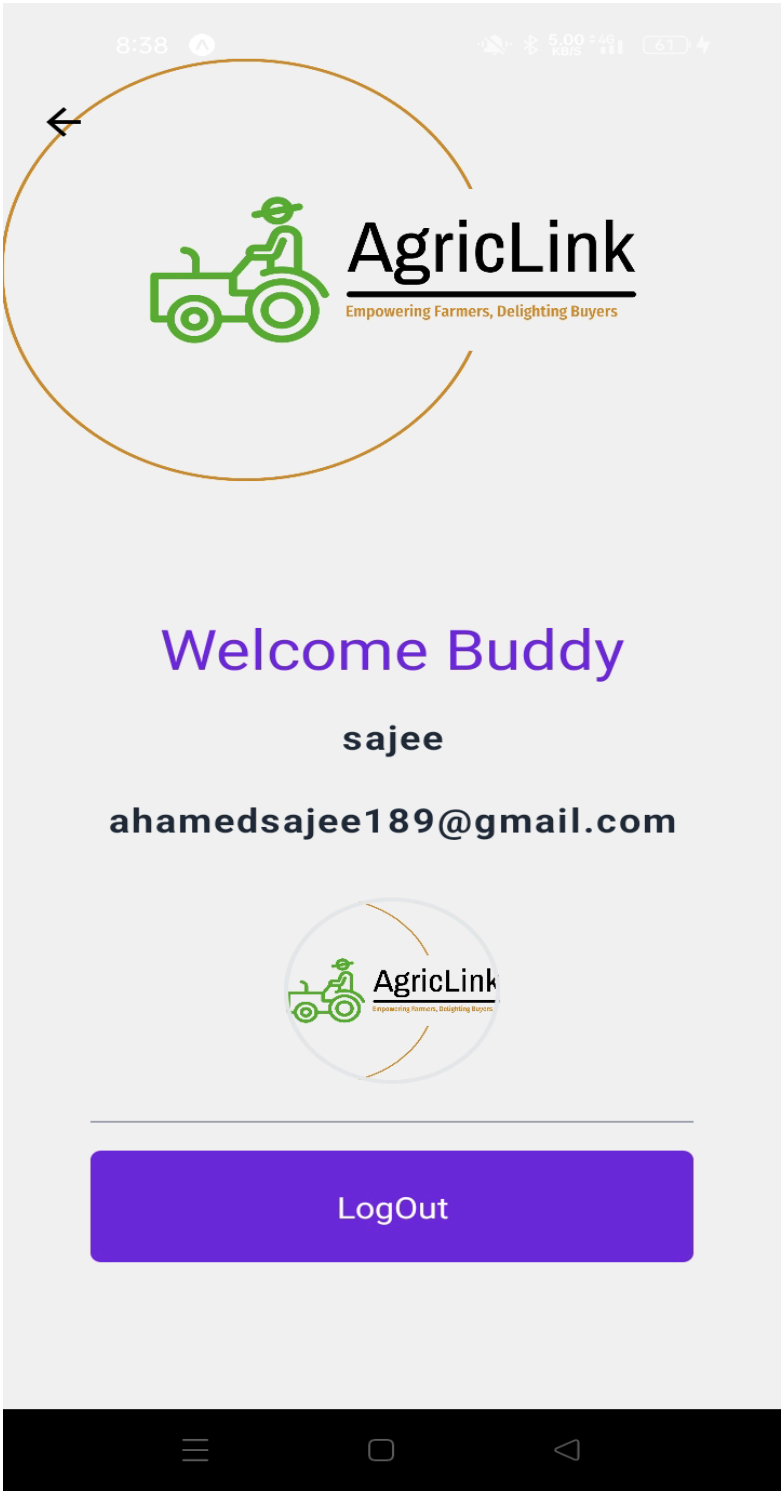


Figure 17 Welcome Screen

5. **Product List Screen**

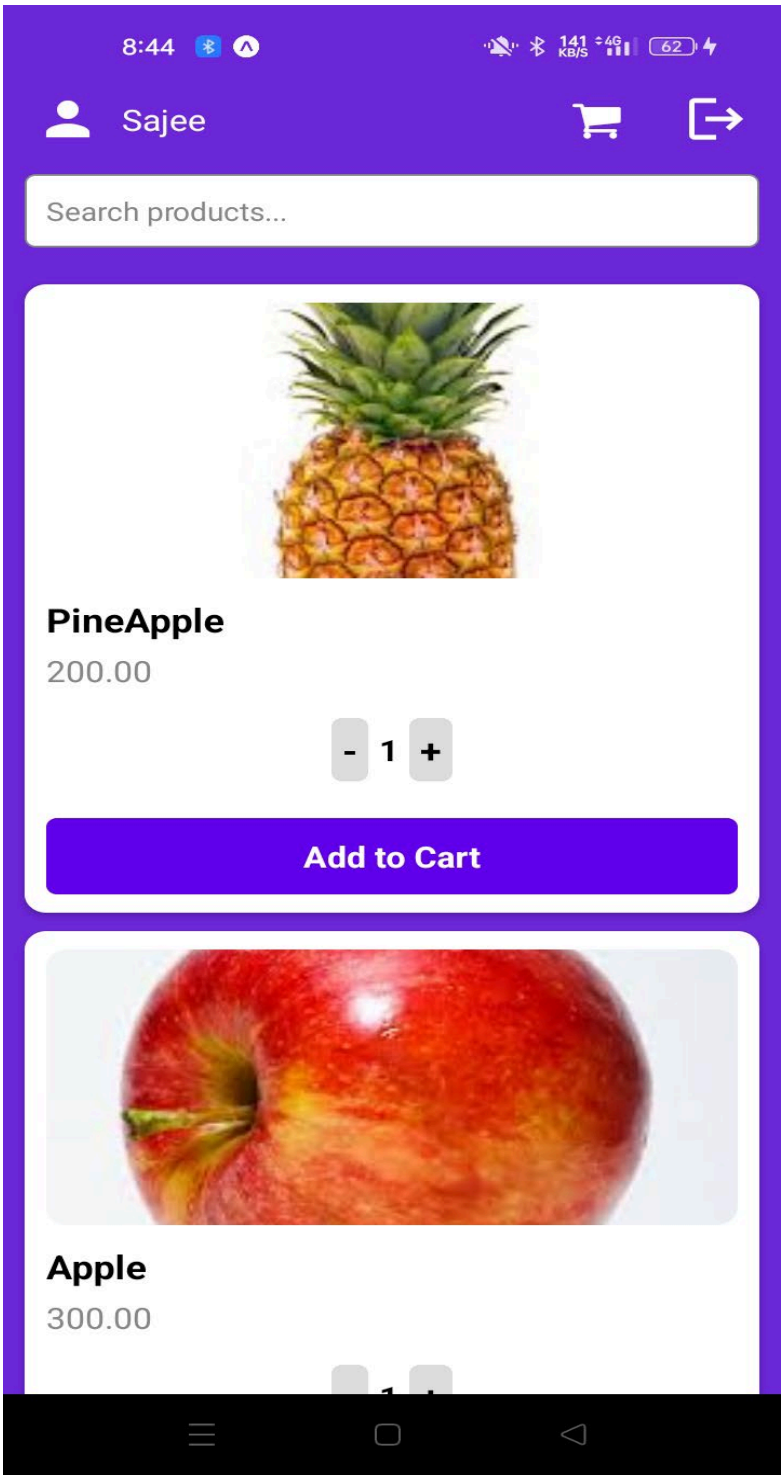


Figure 18 Product List Screen

6. **Add to Cart Screen**

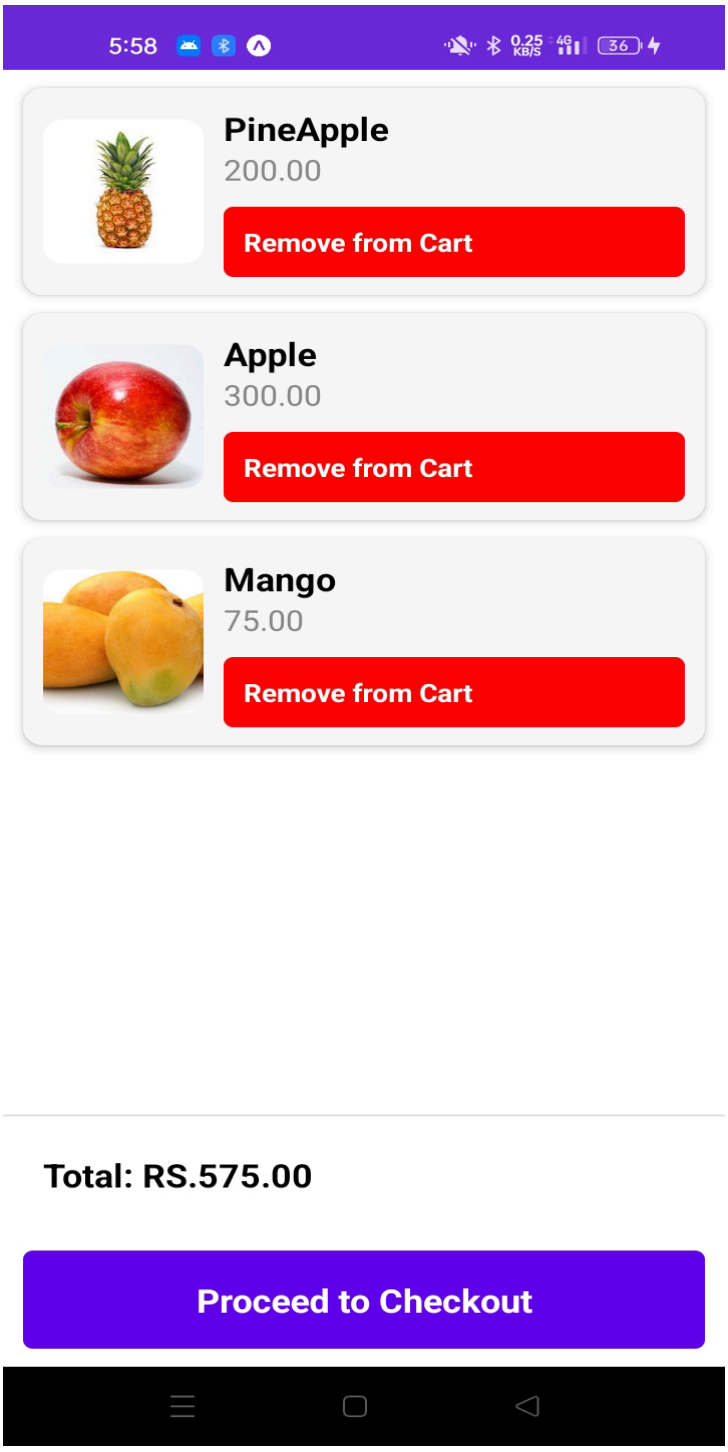


Figure 19 Add to Cart Screen

7. **Checkout Screen**

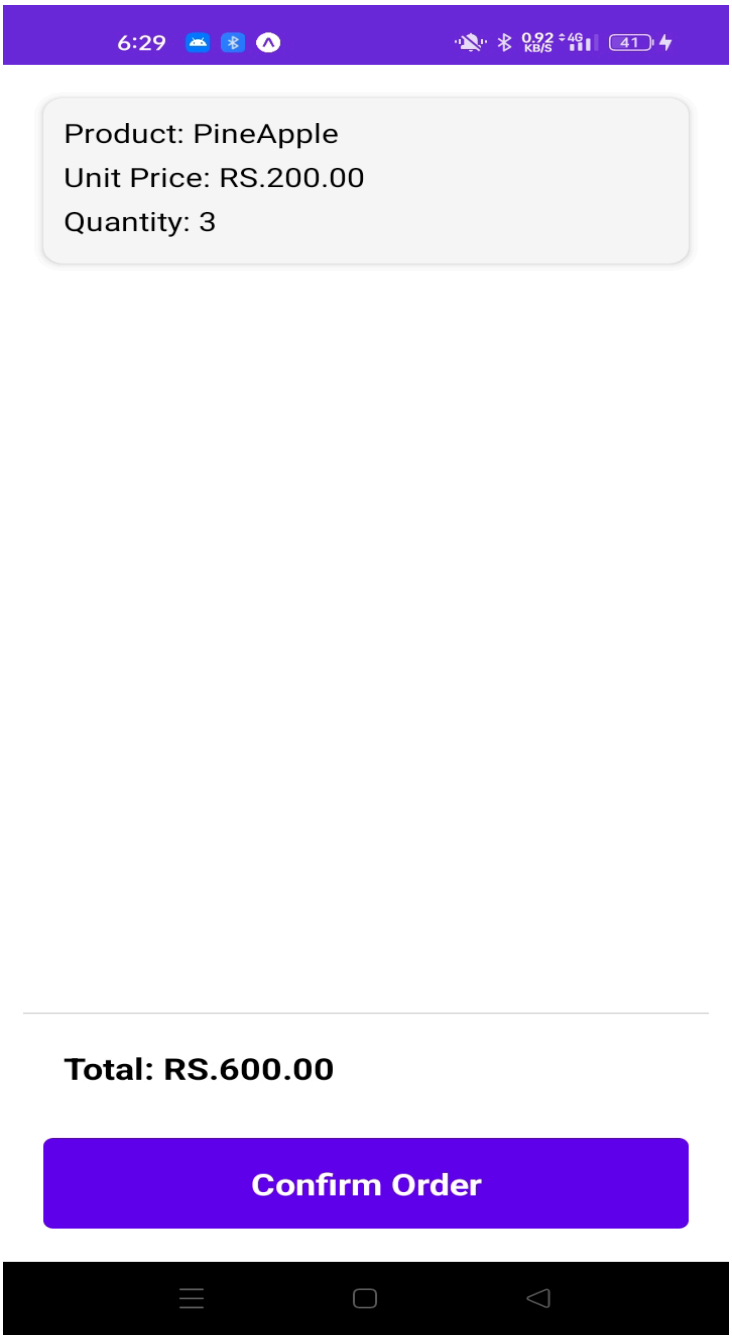


Figure 20 Checkout. Screen

8. **Payment screen**

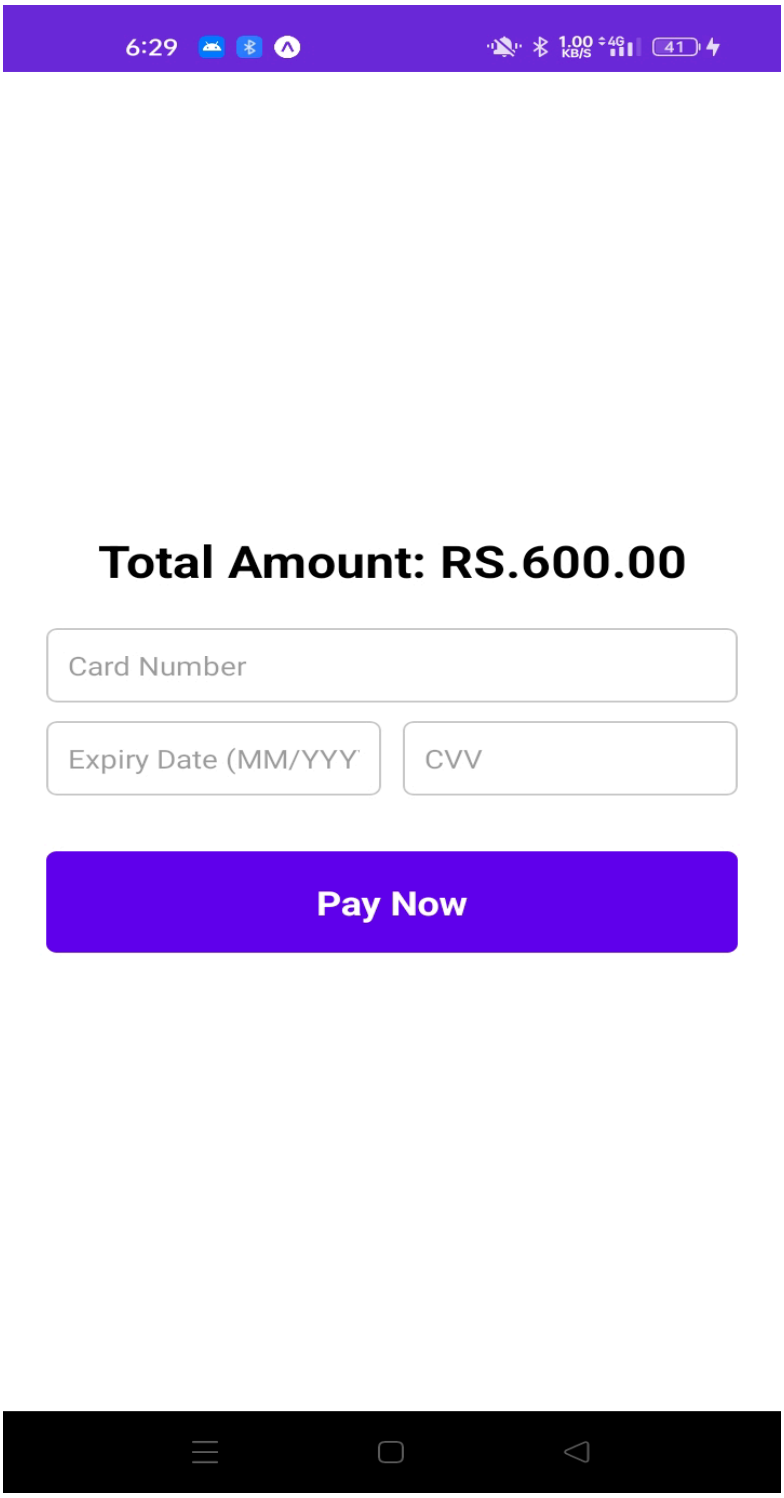


Figure 21 Payment Screen

9. Order Complete Screen

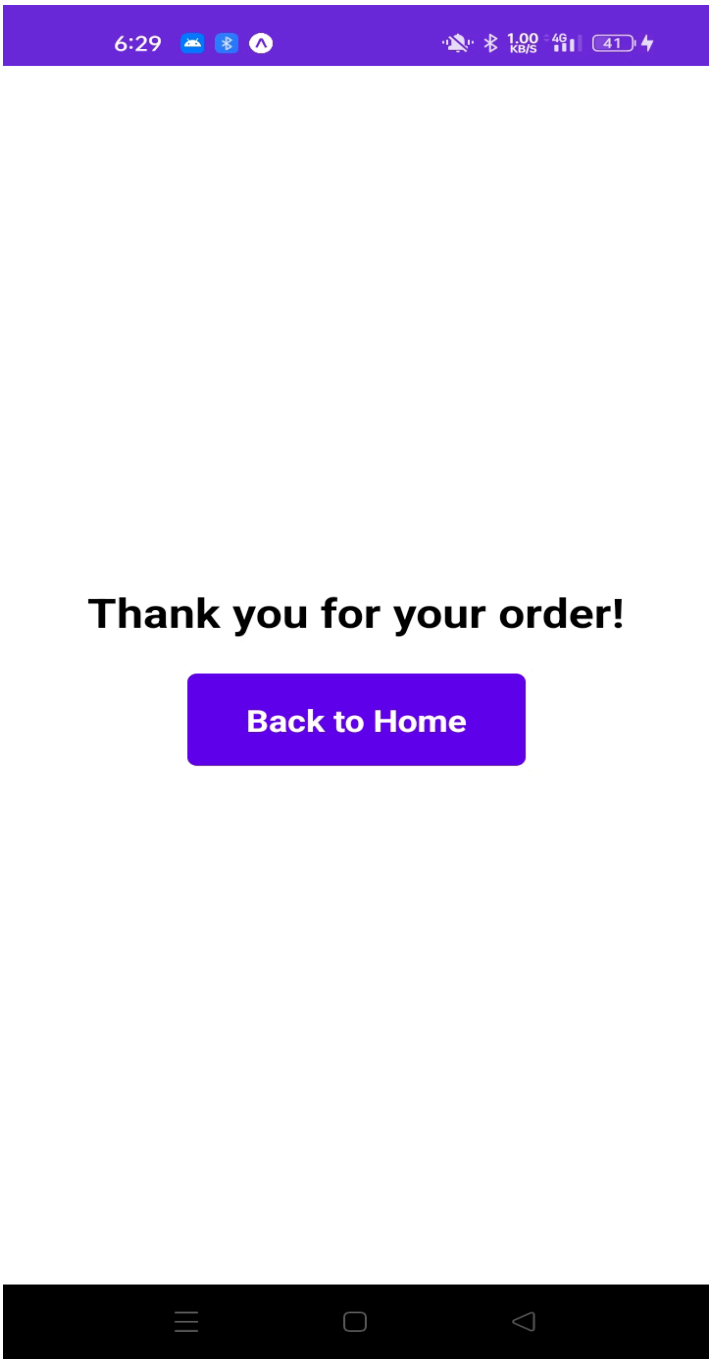


Figure 22 Order Complete Screen

5.3 Resource Requirements

5.3.1 Hardware Requirements

A. Development Environment

- **Developer Workstation:**
  - **Processor:** Intel Core i5 or equivalent (multi-core for performance)
  - **RAM:** 8GB (minimum), 16GB recommended for smoother development
  - **Storage:** SSD drive for faster loading times
  - **Display:** A high-resolution monitor for detailed code viewing and user interface design
  - **Mobile Devices (for testing):**
    - Android Phone (various models)
    - iPhone (various models)

B. Development Environment:

- Server Infrastructure (Consider clod hosting and Dedicated Server)
  - **Processor:** Powerful processor with multiple cores for handling high traffic and data processing
  - **RAM:** At least 16GB (more for increased traffic), potentially scalable to meet demand
  - **Storage:** High-capacity storage (SSD or NVMe for speed) for database and application files
  - **Network Bandwidth:** Sufficient bandwidth for smooth data transfer and user interactions

5.3.2 Software Requirements

A. Development:

- **Code Editors/IDEs:**
  - Visual Studio Code (popular, cross-platform)
- **Web Server for Local Development:**
  - XAMPP (cross-platform, includes Apache, MySQL, PHP)

- **Database Management Tool:**
  - phpMyAdmin (web-based, integrated with XAMPP)
- **Version Control System:**
  - Git (recommended for code management and collaboration)
- **Mobile App Development:**
  - React Native CLI and Node.js (for cross-platform development)
- **Web Browsers (for testing):**
  - Google Chrome
- **Debugging Tools:**
  - PHP debugging tools (e.g., Xdebug)
  - JavaScript debugging tools (e.g., Chrome DevTools)

B. Deployment

- Web Server:
  - Apache (popular, open-source)
- Database Management System:
  - MySQL (popular, open-source)
- Operating System (for servers):
  - Cloud-based operating systems (e.g., AWS Linux, Azure Linux)
- Security Tools:
  - Web Application Firewall (WAF)

C. Additional Software:

- Payment Gateway:
  - Stripe

5.4 Evaluation of Solutions

This section evaluates how well the proposed solution - a broker-free online marketplace for farmers - addresses the problem of limited market access and unfair pricing faced by farmers.

1. **Problem: Limited Market Access for Farmers**

   Solution Evaluation:

- **Direct Connection to Consumers:** The platform removes intermediaries, enabling farmers to directly connect with a wider consumer base. This expands their potential market and reduces dependence on brokers.
- **Geographic Reach:** The online platform allows farmers to reach consumers beyond their local area, potentially expanding their market to new regions.
- **Accessibility and Convenience:** Both farmers and consumers can access the platform easily through their computers or mobile devices, increasing accessibility and convenience.

Strength: This solution effectively addresses the problem of limited market access by creating a platform that connects farmers with a larger and more diverse consumer base.

2. **Problem: Unfair Pricing and Exploitation by intermediaries**

   Solution Evaluation:

   - **Transparent Pricing:** The platform allows farmers to set their own prices, giving them control over their earnings and eliminating the need to negotiate with intermediaries.
   - **Direct Payment:** Farmers receive payments directly from consumers, removing the need for intermediaries and ensuring a larger share of the profit.
   - **Consumer Choice and Competition:** Competition among farmers on the platform incentivizes them to offer fair prices and high-quality produce to attract consumers.

   Strength: This solution mitigates unfair pricing by empowering farmers to control their pricing and receive direct payments from consumers.

3. **Additional Benefits:**
    - **Reduced Food Waste:** The platform can help reduce food waste by connecting farmers with consumers who need their produce, potentially increasing demand and reducing spoilage.
    - **Increased Transparency:** The online platform fosters transparency by providing consumers with information about the origin and quality of their food.
    - **Empowerment of Farmers:** The platform empowers farmers by providing them with greater control over their business and a more equitable share of the value chain.
    - **Sustainable Practices:** The platform can promote sustainable farming practices by allowing farmers to showcase their environmentally friendly methods and connect with consumers who value sustainability.

Chapter 06 Implementation

6.1 Customer side Mobile Application Implimentation

6.1.1 Create a main.js file to communicate MySQL Database

```
// main.js

const express = require("express");
const mysql = require("mysql");
const app = express();
const cors = require("cors");
app.use(express.json());
app.use(cors());

const con = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "",
    database: "farmers",
});

con.connect((err) => {
    if (err) {
        console.log("Error connecting to database:", err);
    } else {
        console.log("Connected to database");
    }
});

app.post("/Register", (req, res) => {
    const { Cus_Name, Cus_Mobile, Cus_email, Password } = req.body;
    console.log("Received data:", req.body); // Log the received data

    if (!Cus_Name || !Cus_Mobile || !Cus_email || !Password) {
        return res.status(400).json({ message: "All fields are required" });
    }

    con.query(
        "INSERT INTO customers (Cus_Name, Cus_Mobile, Cus_email, Password) VALUES (?, ?, ?, ?)",
        [Cus_Name, Cus_Mobile, Cus_email, Password],
        (err, result) => {
            if (err) {
                console.log("Error inserting data:", err);
                res.status(500).json({ message: "Error inserting data", error: err });
            } else {
                res.json({ message: "User registered successfully", status: "SUCCESS", data: result });
            }
        }
    );
});
```

Figure 23 Main.Js Code

```
app.post("/login", (req, res) => {
    const { Cus_email, Password } = req.body;

    con.query(
        "SELECT * FROM customers WHERE Cus_email = ?",
        [Cus_email],
        (err, result) => {
            if (err) {
                console.log("Error retrieving data:", err);
                res.status(500).send("Error retrieving data");
            } else if (result.length === 0) {
                res.status(401).send("User not found");
            } else {
                const user = result[0];
                if (Password === user.Password) {
                    res.send({ status: "SUCCESS", data: user });
                } else {
                    res.status(401).send("Invalid credentials");
                }
            }
        }
    );
});

app.get("/crops", (req, res) =>{
    con.query("SELECT * From crops", (err,result) =>{
        if(err){
            console.log("Error fetching crops Details:",err);
            res.status(500).json({error:"Error fetching Crops"});
        }
        else{
            res.json(result);
        }
    })
})

app.listen(3000, (err) => {
    if (err) {
        console.log("Error starting server:", err);
    } else {
        console.log("Server running on port 3000");
    }
});

```

Figure 24 Main.js Code

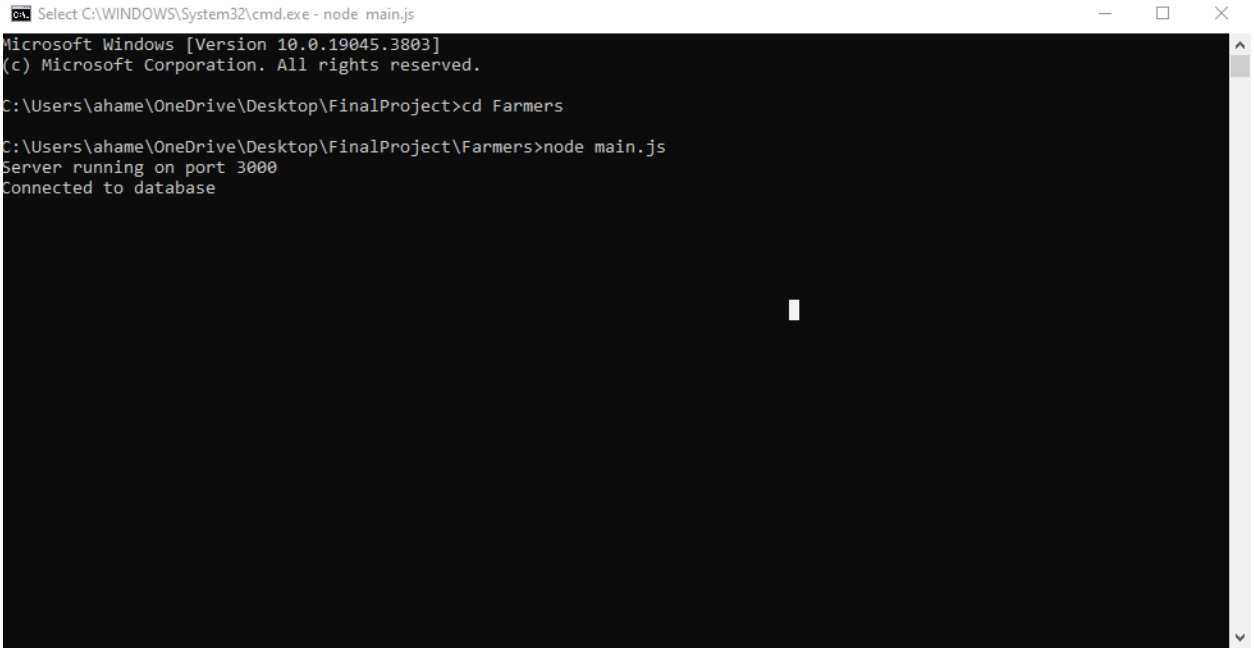


Figure 25 Check the server successfully running

6.1.1 User Login Screen

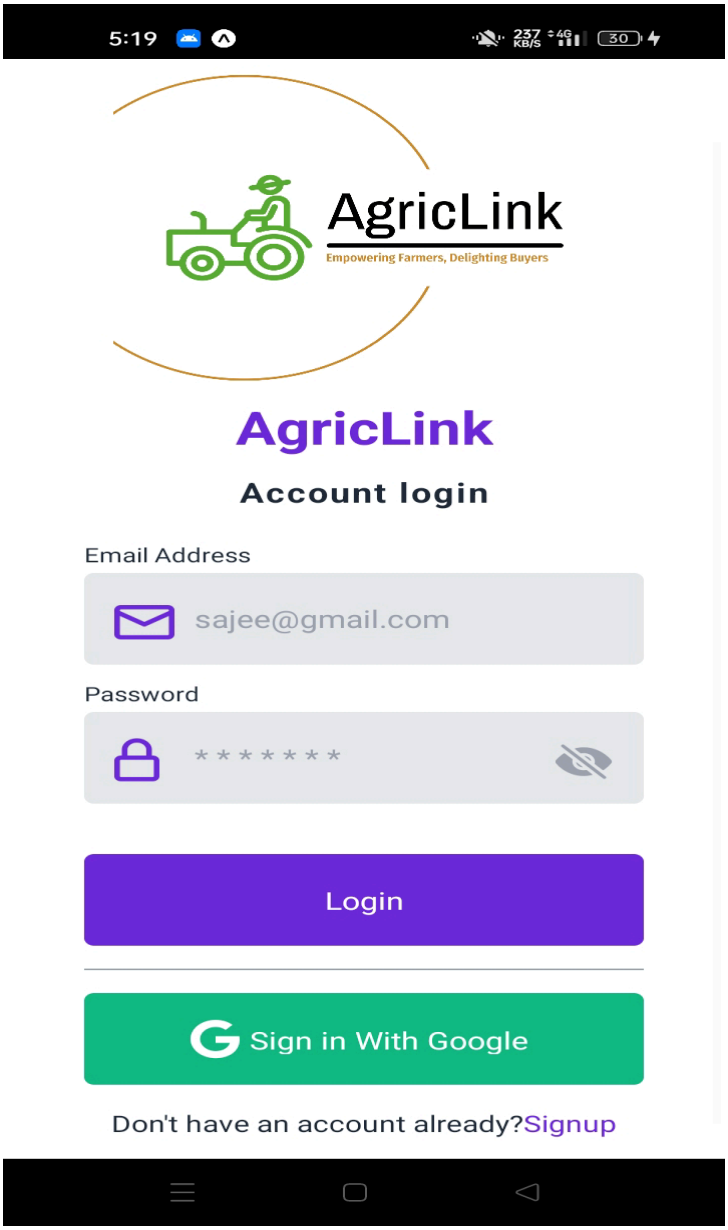


Figure 26 Login Screen

```
const Login = ({ navigation }) => {
    const [hidePassword, setHidePassword] = useState(true);
    const [message, setMessage] = useState(null);
    const [messageType, setMessageType] = useState(null);
    const [googleSubmitting, setGoogleSubmitting] = useState(false);

    const handleLogin = (credentials, setSubmitting) => {
        setMessage(null);
        const url = 'http://192.168.84.198:3000/login';

        axios
            .post(url, credentials)
            .then((response) => {
                const result = response.data;
                const { message, status, data } = result;

                if (status !== 'SUCCESS') {
                    handleMessage(message, status);
                } else {
                    navigation.navigate('Welcome', { ...data });
                }
                setSubmitting(false);
            })
            .catch((error) => {
                console.error('Login error:', error); // Log the detailed error
                console.log('Error details:', error.toJSON()); // Log error details
                setSubmitting(false);
                if (error.response) {
                    // Server responded with a status other than 200 range
                    handleMessage(error.response.data, 'FAILED');
                } else if (error.request) {
                    // Request was made but no response received
                    handleMessage('No response from server. Check your network.', 'FAILED');
                } else {
                    // Something else happened while setting up the request
                    handleMessage('An error occurred. Try again.', 'FAILED');
                }
            });
    };

    const handleMessage = (message, type = 'FAILED') => {
        setMessage(message);
        setMessageType(type);
    };
```

Figure 27 Login Handling Code

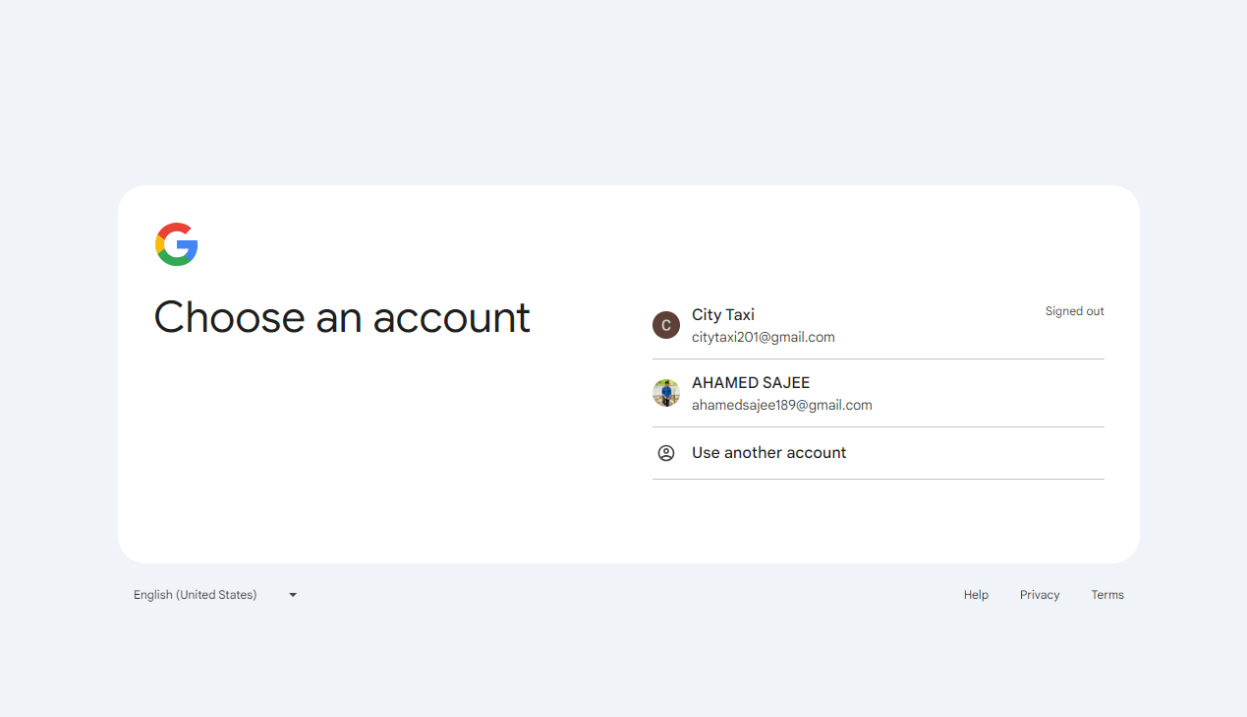


Figure 28 Google Sign in

```
const { brand, darkLight, primary } = Colors;

// Replace with your actual OAuth client IDs obtained from Google Developer Console
const config = {
    iosClientId: '872507830674-dv0rnhk9q4hvm5f10af480smp5b8sd5p.apps.googleusercontent.com',    "rnhk": Unknown word.
    androidClientId: '872507830674-6meh3qu9bhq8bn0r58uqt4a44mgj5ogn.apps.googleusercontent.com',    "googleusercontent": Unknown word.
    scopes: ['profile', 'email']
};
```

Figure 29 google Authentication Keys

```
const handleGoogleSignin = async () => {    "Signin": Unknown word.

    try {
        const redirectUrl = AuthSession.makeRedirectUri({ useProxy: true });
        const authUrl = `https://accounts.google.com/o/oauth2/auth?response_type=id_token&client_id=${
            Platform.OS === 'ios' ? config.iosClientId : config.androidClientId
        }&redirect_uri=${encodeURIComponent(redirectUrl)}&scope=${encodeURIComponent(config.scopes.join(' '))}`;

        const result = await AuthSession.startAsync({ authUrl });

        if (result.type === 'success') {
            const { id_token } = result.params;

            // Example: Use id_token for verification or server-side validation
            // Typically, you would send this id_token to your server for validation

            handleMessage('Google Sign-In Successful', 'SUCCESS');
            setTimeout(() => navigation.navigate('Welcome'), 1000);
        } else if (result.type === 'error') {
            handleMessage('Google Sign-In Error', 'FAILED');
        } else {
            handleMessage('Google Sign-In Cancelled', 'FAILED');
        }
    } catch (error) {
        console.error('Google Sign-In Error:', error);
        handleMessage('Google Sign-In failed. Try again.', 'FAILED');
    }

    setGoogleSubmitting(false);
};
```

Figure 30 Google Sign in Handling Code

```
const MyTextInput = ({ label, icon, isPassword, hidePassword, setHidePassword, ...props }) => {
    return (
        <View>
            <LeftIcon>
                <Octicons name={icon} size={30} color={brand} />
            </LeftIcon>
            <StyledInputLabel>{label}</StyledInputLabel>
            <StyledTextInput {...props} />
            {isPassword && (
                <RightIcon onPress={() => setHidePassword(!hidePassword)}>
                    <Ionicons name={hidePassword ? 'eye-off' : 'eye'} size={30} color={darkLight} />    "Ionicons": Unknown word.
                </RightIcon>
            )}
        </View>
    );
};

export default Login;
```

Figure 31 Hidden Password Code

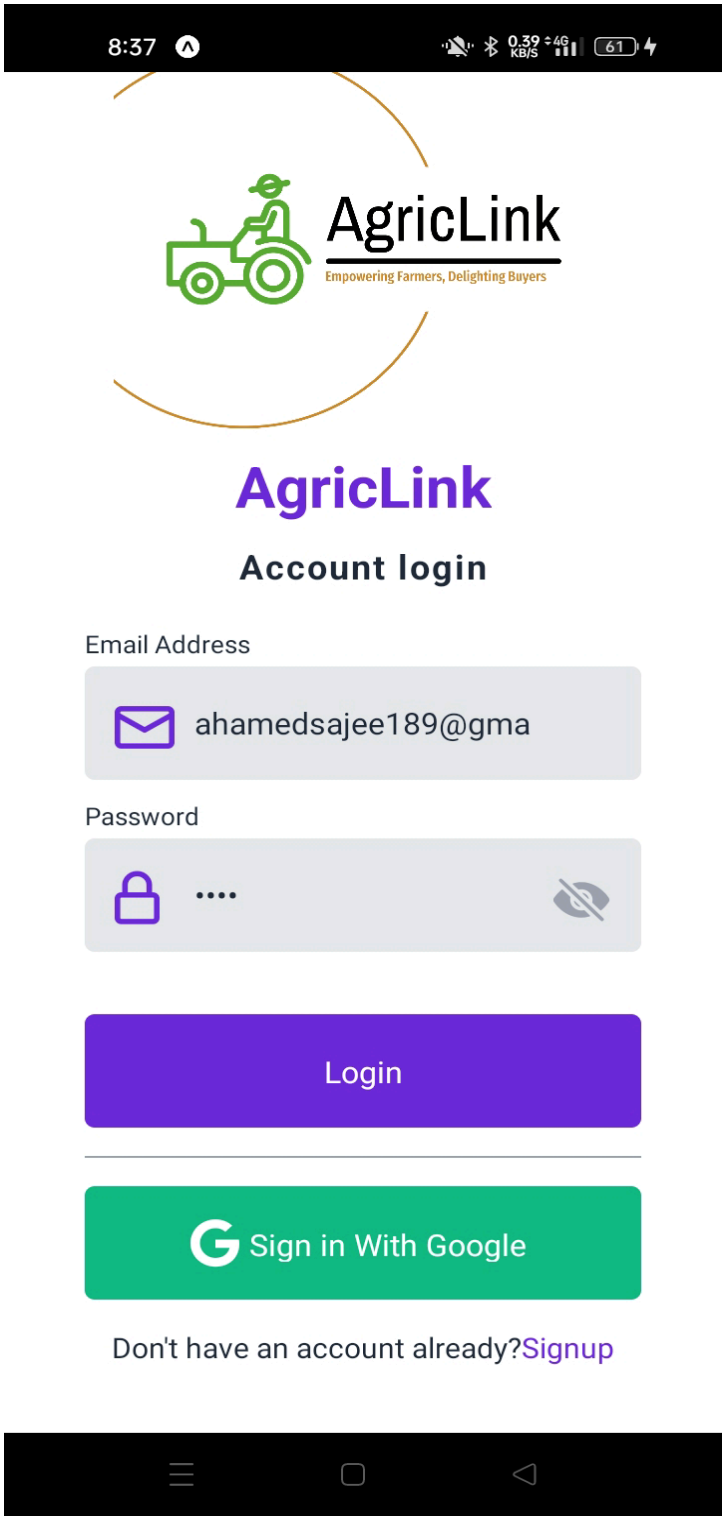


Figure 32 Enter Email Address and Password

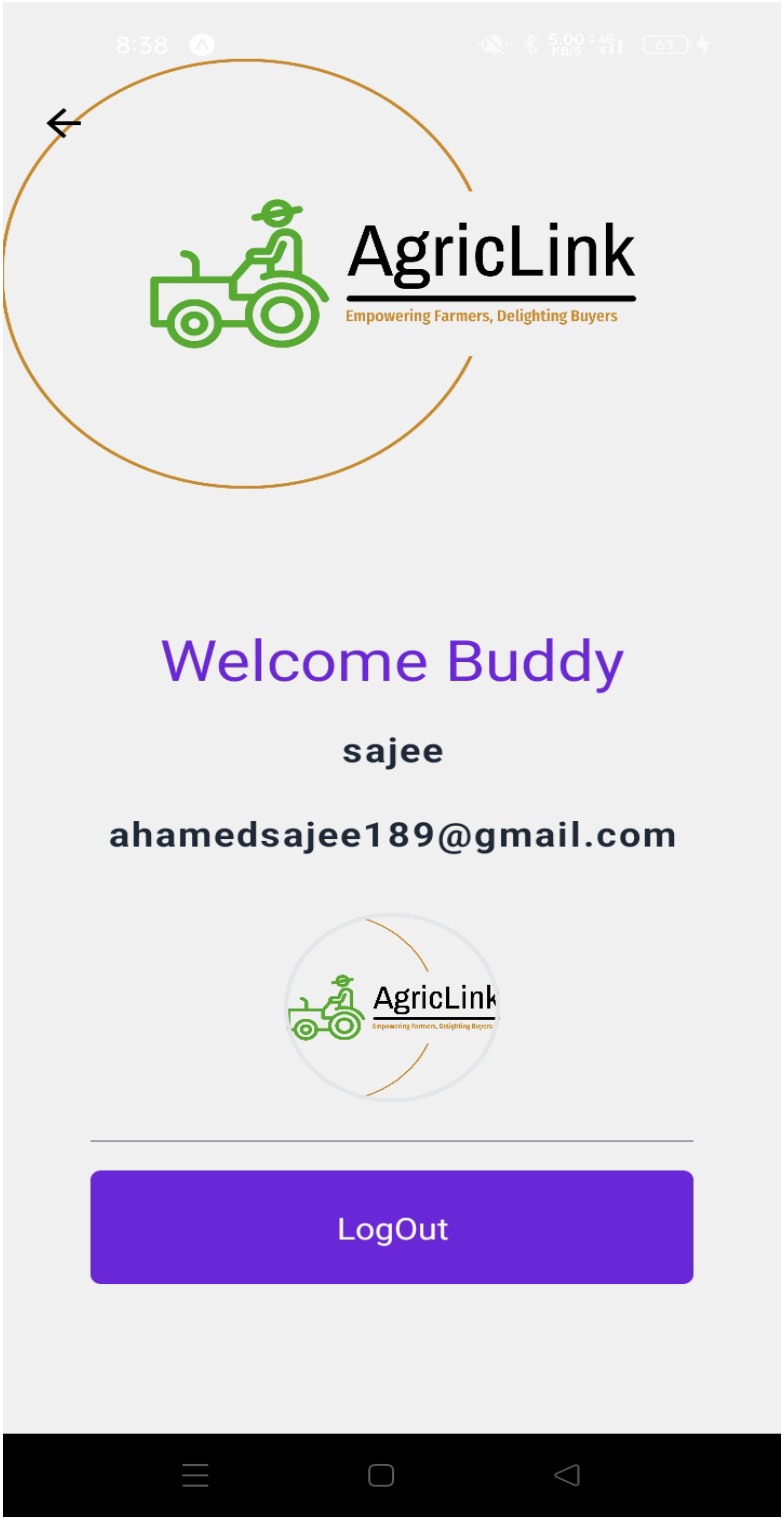


Figure 33 Login Success full The page Redirect to Welcome page

Login Success You reached a welcome page after in 3 second the page automatically redirect to Order page.

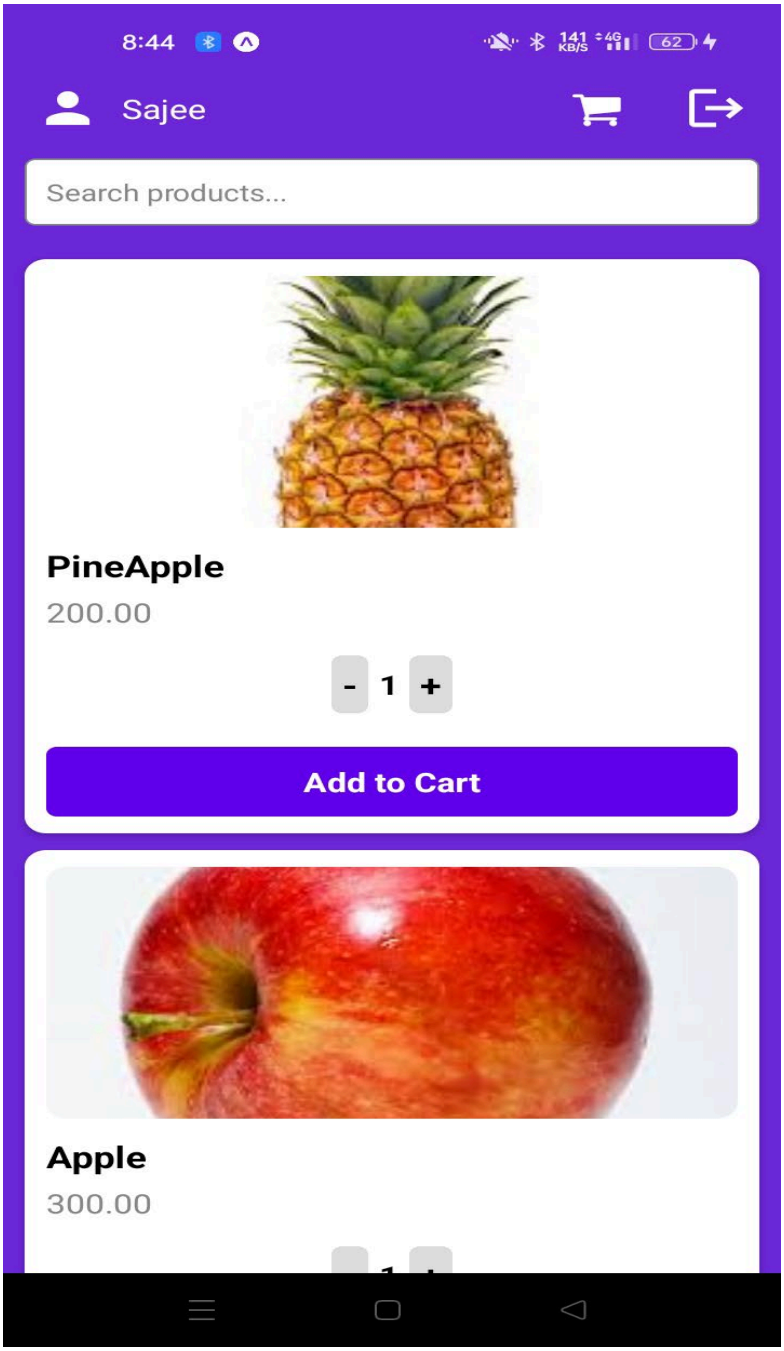


Figure 34 Order Page

6.1.2 User Registration Screen

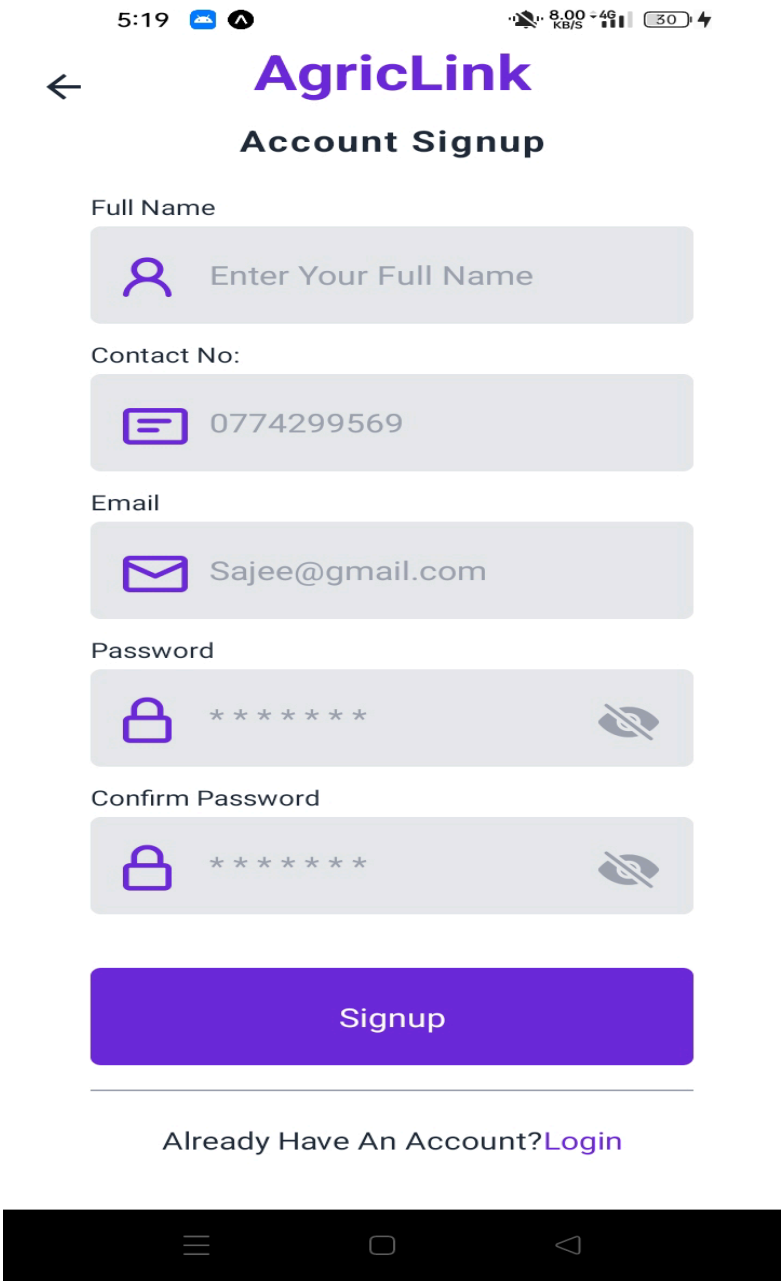


Figure 35 User Registration Screen

```
const Signup = ({ navigation }) => {
    const [hidePassword, setHidePassword] = useState(true);
    const [message, setMessage] = useState('');
    const [messageType, setMessageType] = useState('');

    const handleSignup = (credentials, setSubmitting) => {
        handleMessage(null);
        const url = 'http://192.168.84.198:3000/Register';

        axios.post(url, credentials)
            .then((response) => {
                const result = response.data;
                const { message, status, data } = result;

                if (status !== 'SUCCESS') {
                    handleMessage(message, 'FAILED');
                } else {
                    navigation.navigate('Login', { ...data });
                }
                setSubmitting(false);
            })
            .catch(error => {
                console.error("Signup error:", error); // Log the detailed error
                setSubmitting(false);
                if (error.response) {
                    handleMessage(error.response.data.message || 'An error occurred', 'FAILED');
                } else if (error.request) {
                    handleMessage("No response from server. Check your network.", 'FAILED');
                } else {
                    handleMessage("An error occurred. Try again.", 'FAILED');
                }
            });
    };

    const handleMessage = (message, type = 'FAILED') => {
        setMessage(message);
        setMessageType(type);
    };
```

Figure 36 User Signup Handling Code

```
const handleMessage = (message, type = 'FAILED') => {
    setMessage(message);
    setMessageType(type);
};

return (
    <KeyboardAvoidingWrapper>
        <StyledContainer>
            <StatusBar style="dark" />
            <InnerContainer>
                <PageTitle>AgricLink</PageTitle>
                <SubTitle>Account Signup</SubTitle>

                <Formik    "Formik": Unknown word.
                    initialValues={{ Cus_Name: '', Cus_Mobile: '', Cus_email: '', password: '', confirmPassword: '' }}
                    onSubmit={(values, { setSubmitting }) => {
                        if (values.Cus_Name === '' || values.Cus_Mobile === '' || values.Cus_email === '' || values.password === '' || values.confirmPassword === '') {
                            handleMessage('Please fill in all fields');
                            setSubmitting(false);
                        } else if (values.password !== values.confirmPassword) {
                            handleMessage('Passwords do not match');
                            setSubmitting(false);
                        } else {
                            handleSignup({
                                Cus_Name: values.Cus_Name,
                                Cus_Mobile: values.Cus_Mobile,
                                Cus_email: values.Cus_email,
                                Password: values.password
                            }, setSubmitting);
                        }
                    }}
                >
                    {({ handleChange, handleBlur, handleSubmit, values, isSubmitting }) => (
                        <StyledFormArea>
                            <MyTextInput
                                label="Full Name"
                                icon="person"
                                placeholder="Enter Your Full Name"
                                placeholderTextColor={darkLight}
                                onChangeText={handleChange('Cus_Name')}
                                onBlur={handleBlur('Cus_Name')}
                                value={values.Cus_Name}
                            />
```

Figure 37 User Signup Set Submitting Code

```
const MyTextInput = ({ label, icon, isPassword, hidePassword, setHidePassword, ...props }) => {
    return (
        <View>
            <LeftIcon>
                <Octicons name={icon} size={30} color={brand} />
            </LeftIcon>
            <StyledInputLabel>{label}</StyledInputLabel>
            <StyledTextInput {...props} />
            {isPassword && (
                <RightIcon onPress={() => setHidePassword(!hidePassword)}>
                    <Ionicons name={hidePassword ? 'eye-off' : 'eye'} size={30} color={darkLight} />    "Ionicons": Unknown word.
                </RightIcon>
            )}
        </View>
    );
};

export default Signup;
```

Figure 38 Hide and show Password box Values code

6.1.3 Order Page Screen

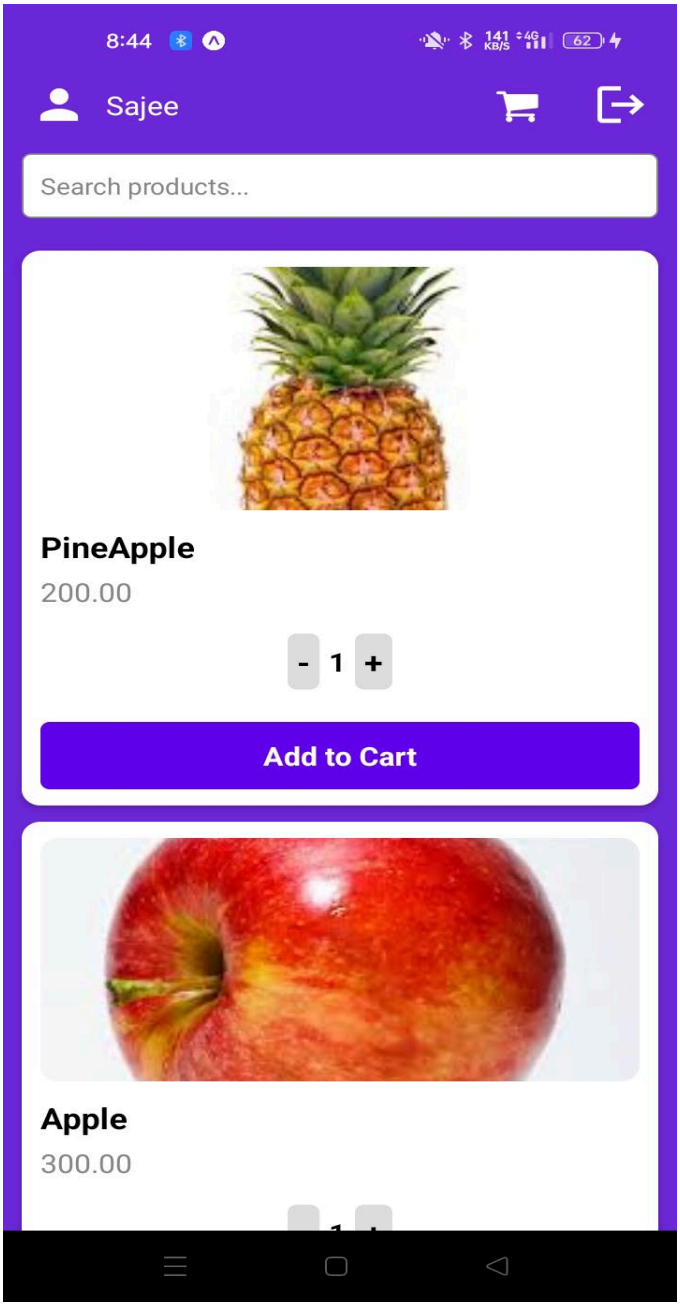


Figure 39 Order Screen

```
const renderProduct = ({ item }) => (
  <View style={styles.card}>
    <Image source={{ uri: item.image }} style={styles.image} />
    <Text style={styles.name}>{item.name}</Text>
    <Text style={styles.price}>{item.price}</Text>
    <View style={styles.quantityContainer}>
      <TouchableOpacity onPress={() => handleQuantityChange(item.id, -1)} style={styles.quantityButton}>
        <Text style={styles.quantityButtonText}>-</Text>
      </TouchableOpacity>
      <Text style={styles.quantity}>{quantities[item.id] || 1}</Text>
      <TouchableOpacity onPress={() => handleQuantityChange(item.id, 1)} style={styles.quantityButton}>
        <Text style={styles.quantityButtonText}>+</Text>
      </TouchableOpacity>
    </View>
    <TouchableOpacity style={styles.button} onPress={() => addToCart(item)}>
      <Text style={styles.buttonText}>Add to Cart</Text>
    </TouchableOpacity>
  </View>
);

return (
  <View style={styles.container}>
    <StatusBar barStyle="light-content" backgroundColor="#6D28D9" />
    <View style={styles.navbar}>
      <View style={styles.profileSection}>
        {customer.profileImage ? (
          <Image source={{ uri: customer.profileImage }} style={styles.profileImage} />
        ) : (
          <Icon name="person" type="material" color="#fff" size={30} />
        )}
        <Text style={styles.profileName}>{customer.name}</Text>
      </View>
      <TouchableOpacity style={styles.cartIcon} onPress={() => navigation.navigate('Cart')}>
        <Icon name="shopping-cart" type="font-awesome" color="#fff" />
        {cart.length > 0 && (
          <View style={styles.cartBadge}>
            <Text style={styles.cartBadgeText}>{cart.length}</Text>
          </View>
        )}
      </TouchableOpacity>
      <TouchableOpacity onPress={handleLogout}>
        <Icon name="logout" type="material" color="#fff" size={30} />
      </TouchableOpacity>
    </View>
```

Figure 40 Order Screen Code

6.1.4 Add to Cart Screen

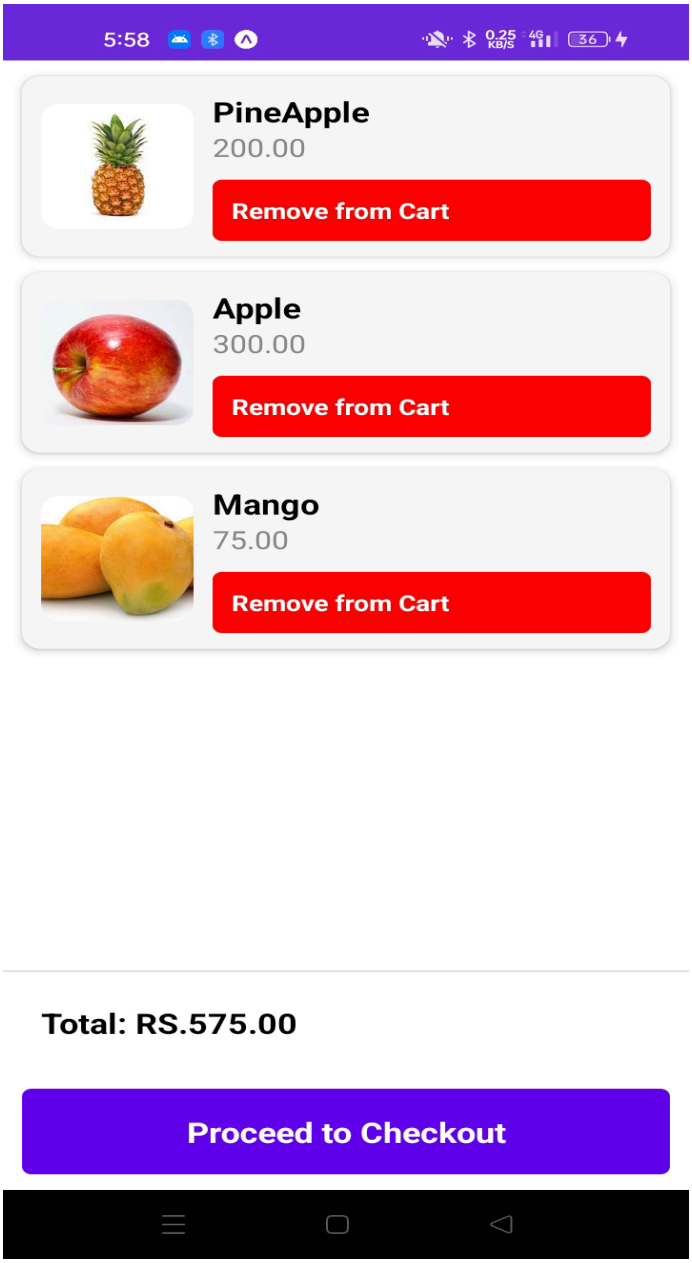


Figure 41 Add to Cart Screen

```
const Cart = () => {
  const { cart, setCart } = useCart(); // Use the cart context
  const [updatedCart, setUpdatedCart] = useState(cart);
  const navigation = useNavigation();

  const getTotal = () => {
    return updatedCart.reduce((total, item) => total + parseFloat(item.price.replace('$', '')), 0).toFixed(2);
  };

  const handleCheckout = () => {
    navigation.navigate('Checkout', { updatedCart }); // Pass updatedCart to Checkout
  };

  const removeFromCart = (item) => {
    const newCart = updatedCart.filter(cartItem => cartItem.id !== item.id);
    setUpdatedCart(newCart);
    setCart(newCart);
    if (newCart.length === 0) {
      navigation.navigate('Order');
    }
  };

  const renderCartItem = ({ item }) => (
    <View style={styles.cartItem}>
      <Image source={{ uri: item.image }} style={styles.image} />
      <View style={styles.info}>
        <Text style={styles.name}>{item.name}</Text>
        <Text style={styles.price}>{item.price}</Text>
        <TouchableOpacity onPress={() => removeFromCart(item)} style={styles.removeButton}>
          <Text style={styles.removeButtonText}>Remove from Cart</Text>
        </TouchableOpacity>
      </View>
    </View>
  );

  return (
    <View style={styles.container}>
      <StatusBar barStyle="light-content" backgroundColor="#6D28D9" />
      <FlatList
        data={updatedCart}
        renderItem={renderCartItem}
        keyExtractor={(item) => item.id.toString()} // Ensure each key is unique and is a string
        contentContainerStyle={styles.cartList}
      />
      <View style={styles.totalContainer}>
        <Text style={styles.totalText}>Total: ${getTotal()}</Text>
      </View>
      <TouchableOpacity
        style={styles.checkoutButton}
        onPress={handleCheckout}
      >
        <Text style={styles.checkoutButtonText}>Proceed to Checkout</Text>
      </TouchableOpacity>
    </View>
  );
};
```

Figure 42Add to Cart Implementation codes

6.1.5 Checkout Screen

Figure 43 Checkout Screen

```
const Checkout = ({ route }) => {
  const { updatedCart } = route.params; // Ensure updatedCart is received correctly
  const navigation = useNavigation();

  const getTotal = () => {
    return updatedCart.reduce((total, item) => total + (parseFloat(item.price.replace('RS.', '')) * item.quantity), 0).toFixed(2);
  };

  const renderSaleDetail = ({ item }) => (
    <View style={styles.saleDetail}>
      <Text style={styles.text}>Product: {item.name}</Text>
      <Text style={styles.text}>Unit Price: RS.{item.price}</Text>
      <Text style={styles.text}>Quantity: {item.quantity}</Text>
    </View>
  );

  const handleConfirm = () => {
    const totalAmount = getTotal();
    // Here you would send saleDetails and totalSale to your backend to create the order
    console.log('Order confirmed:', updatedCart, totalAmount);
    navigation.navigate('Payment', { totalAmount }); // Navigate to the payment page with total amount
  };

  return (
    <View style={styles.container}>
      <FlatList
        data={updatedCart}
        renderItem={renderSaleDetail}
        keyExtractor={(item) => item.id.toString()}
        contentContainerStyle={styles.list}
      />
      <View style={styles.totalContainer}>
        <Text style={styles.totalText}>Total: RS.{getTotal()}</Text>
      </View>
      <TouchableOpacity style={styles.confirmButton} onPress={handleConfirm}>
        <Text style={styles.confirmButtonText}>Confirm Order</Text>
      </TouchableOpacity>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    padding: 10,
  },
  list: {
    padding: 10,
  },
  saleDetail: {
    padding: 10,
    backgroundColor: '#f8f8f8',
    borderRadius: 10.
```

Figure 44Checkout Screen Implementation

6.1.6 Payment Screen

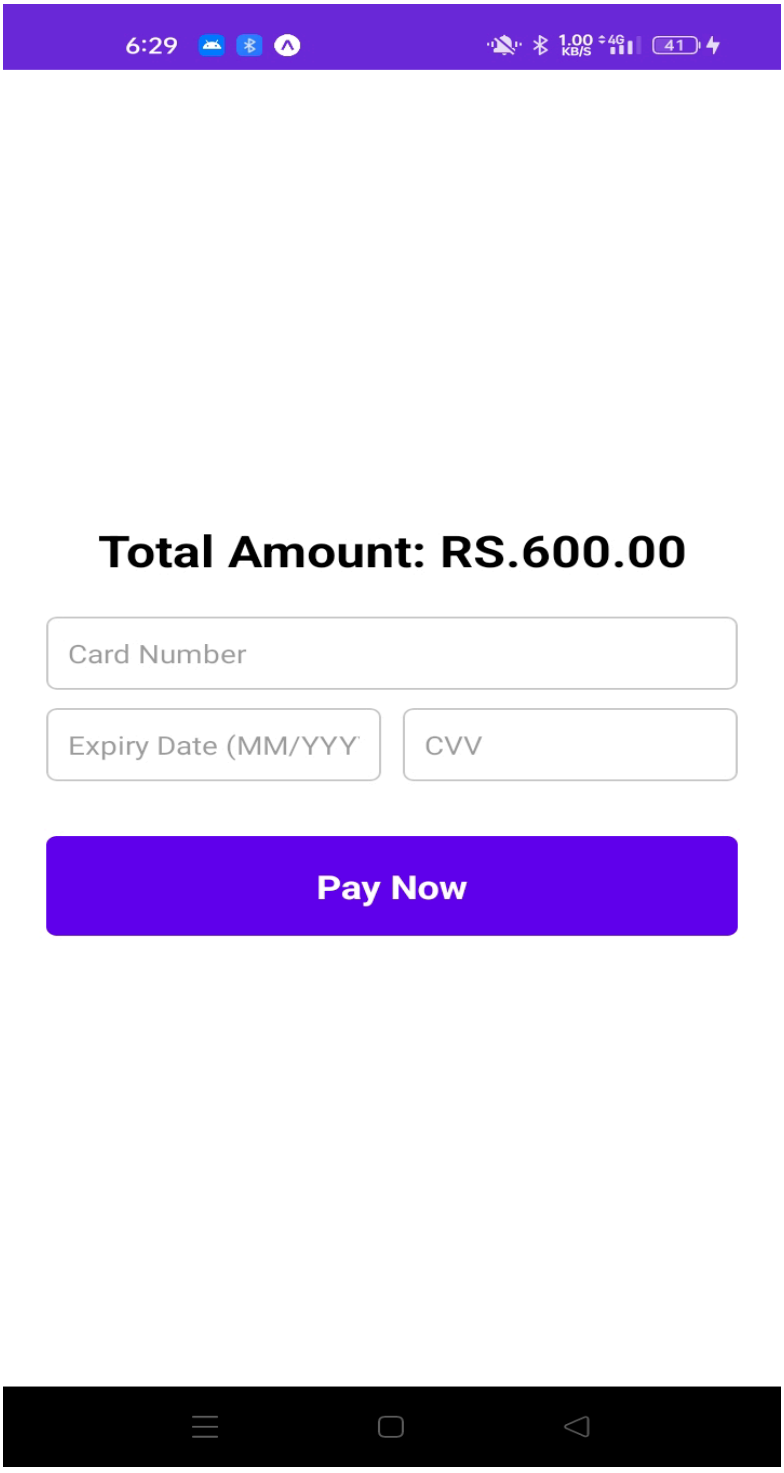


Figure 45 Payment screen With using strip Api

```
const Payment = ({ route }) => {
  const { totalAmount } = route.params; // Receive total amount from the route parameters
  const navigation = useNavigation();
  const { confirmPayment } = useStripe();

  const [cardDetails, setCardDetails] = useState(null);

  const handlePayment = async () => {
    if (!cardDetails?.complete) {
      console.log('Please enter complete card details');
      return;
    }

    try {
      const paymentIntentClientSecret = await fetchPaymentIntentClientSecret(totalAmount);

      const { error } = await confirmPayment(paymentIntentClientSecret, {
        type: 'Card',
        billingDetails: {
          // Add any additional billing details you need here
        },
      });

      if (error) {
        console.log('Payment failed:', error);
      } else {
        console.log('Payment successful for amount:', totalAmount);
        navigation.navigate('Ordercomplete'); // Navigate to the order completion page after successful payment    "Ordercomplete": Unknown word.
      }
    } catch (e) {
      console.log('Payment failed:', e);
    }
  };

  const fetchPaymentIntentClientSecret = async (amount) => {
    // Replace this with your backend endpoint that creates a PaymentIntent
    const response = await fetch('sk_test_51PWhjaFCV2h1o1RK0Evkfigp0jCorzdvLcBmrGU8oNUYkW15a3iOlQ9Iyezja4TP42MNO82fBMug2RBGusO7PCJk00JWkjc7AI', {    "Whja": Unknown word.
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ amount }),
    });
    const { clientSecret } = await response.json();
    return clientSecret;
  };

  return (
    <View style={styles.container}>
      <Text style={styles.amountText}>Total Amount: RS.{totalAmount}</Text>
      <CardField
        postalCodeEnabled={true}
        placeholder={{
          number: '4242 4242 4242 4242',
        }}
```

Figure 46 Payment Screen implementation

6.1.7 Order Complete Screen

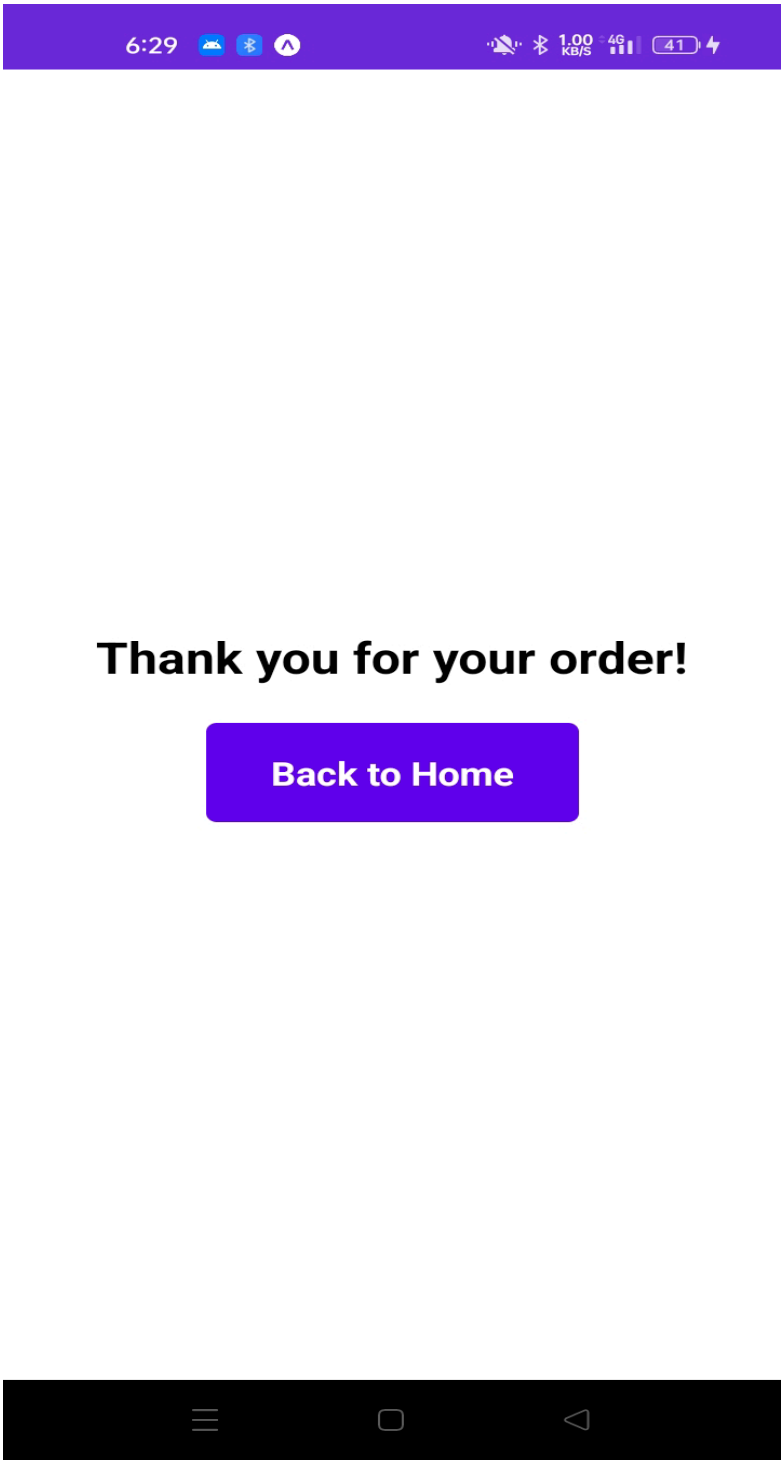


Figure 47 Order Complete Screen

6.2 Farmer and Customer Side Implementation

6.2.1 Database Connection Code

```php
<?php
// Database connection settings
define('DB_HOST', 'localhost');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', '');
define('DB_NAME', 'farmers');

// Create a MySQL connection
$conn = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD, DB_NAME);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>    The closing ?> tag should be omitted from files containing only PHP.
```

Figure 48 Database Connection Code

6.2.2 Admin Login

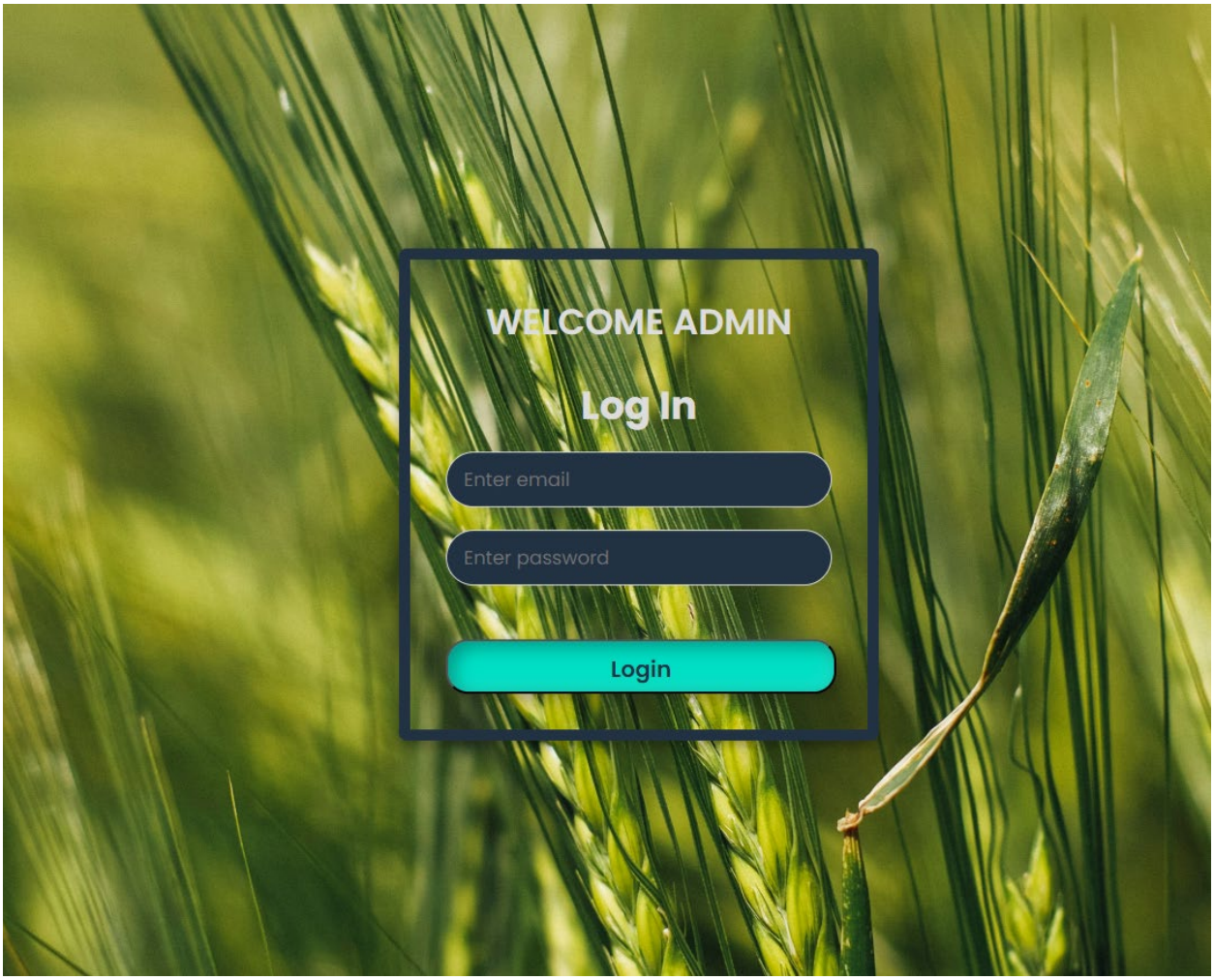


Figure 49Admin Login

```php
<?php
session_start();
include 'config.php'; // Include the database configuration file

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Get email and password from POST request
    $email = $_POST['email'];
    $password = $_POST['password'];

    // Validate credentials
    if (!empty($email) && !empty($password)) {

        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }

        // Prepare a SQL statement to prevent SQL injection
        $sql = "SELECT * FROM Admin WHERE email = ?";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param('s', $email);
        $stmt->execute();
        $result = $stmt->get_result();

        // Check if an admin with the given email exists
        if ($result->num_rows === 1) {
            $admin = $result->fetch_assoc();

            // Verify the password
            if ($password === $admin['password']) { // Plain text password comparison
                // Start a session and store admin information
                $_SESSION['admin_id'] = $admin['A_id'];
                $_SESSION['admin_email'] = $admin['email'];
                header('Location: adminDashboard.php'); // Redirect to admin dashboard
                exit();
            } else {
                $error = "Invalid email or password.";
            }
        } else {
            $error = "Invalid email or password.";
        }

        // Close the statement and connection
        $stmt->close();
        $conn->close();
    } else {
        $error = "Please fill in all fields.";
    }
}
?>
```

Figure 50 Admin Login Codes

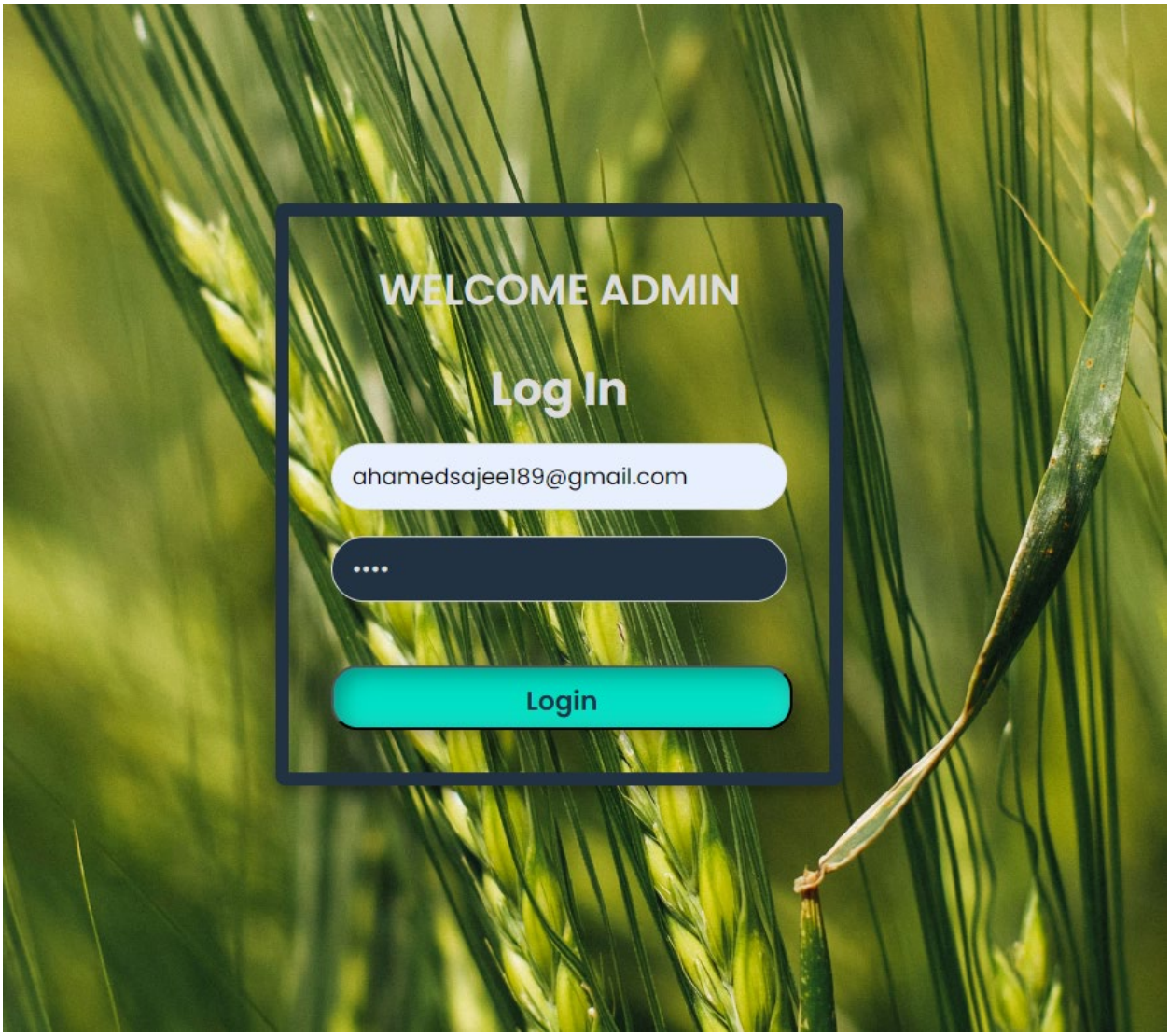


Figure 51 Enter Admin Email and Password

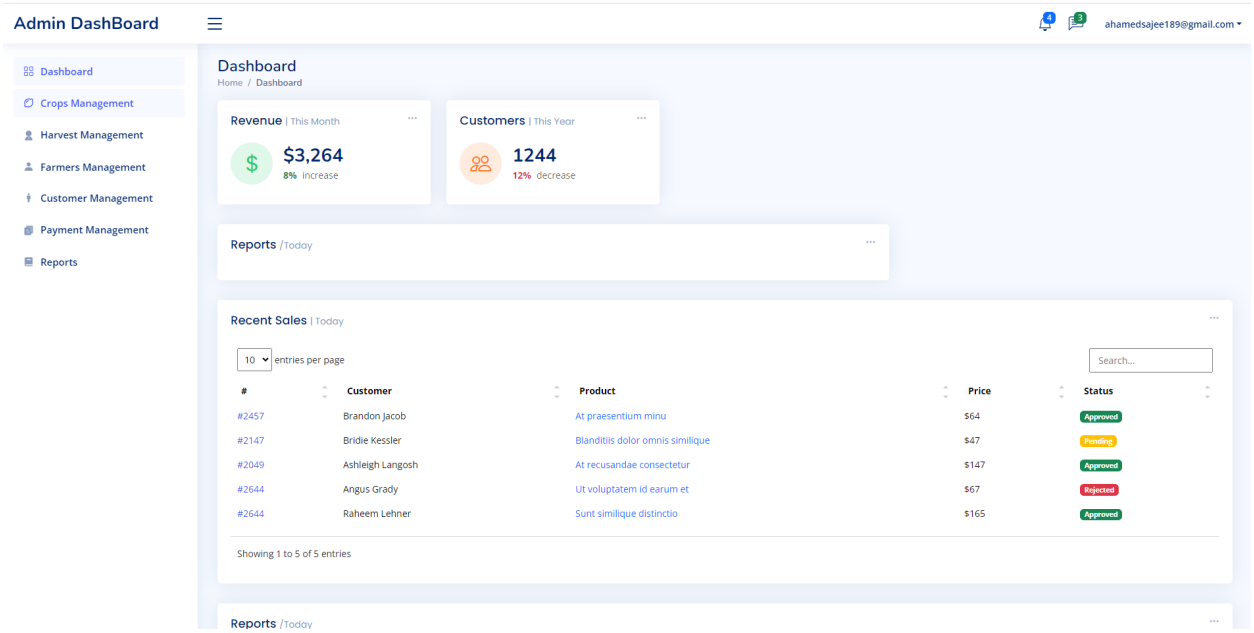


Figure 52 Login Success Page Redirect to Admin Dashboard Page

```
<li class="nav-item dropdown pe-3">
  <a class="nav-link nav-profile d-flex align-items-center pe-0" href="#" data-bs-toggle="dropdown">
    <span class="d-none d-md-block dropdown-toggle ps-2"><?php echo htmlspecialchars($_SESSION['admin_email']); ?></span>
  </a>
  <ul class="dropdown-menu dropdown-menu-end dropdown-menu-arrow profile">
    <li class="dropdown-header">
      <h6><?php echo htmlspecialchars($_SESSION['admin_email']); ?></h6>
      <span>Admin</span>
    </li>
    <li>
      <hr class="dropdown-divider">
    </li>
    <li>
```

Figure 53 Use Session to get Current Login Admin email

6.2.3 Add crops Page

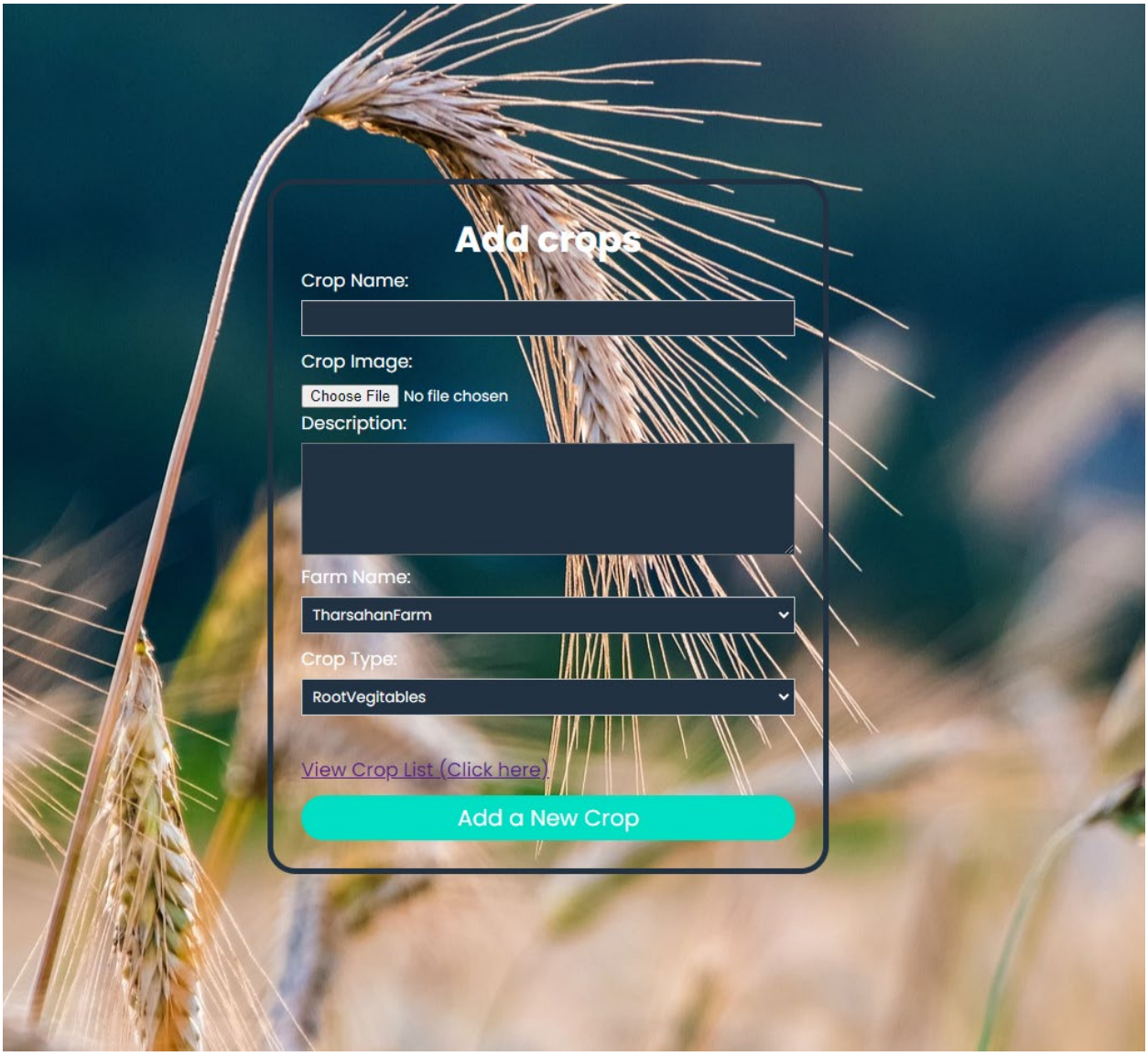


Figure 54 Add corps Page

```php
<?php
include 'config.php';

// Initialize variables for messages
$errorMsg = '';
$successMsg = '';

// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $crop_name = $conn->real_escape_string($_POST['CropName']);
    $description = $conn->real_escape_string($_POST['Description']);
    $farm_id = $conn->real_escape_string($_POST['FarmName']);
    $crops_type_id = $conn->real_escape_string($_POST['CropType']);

    // File upload handling
    $targetDir = "uploads/"; // Directory where images will be stored
    $targetFile = $targetDir . basename($_FILES["image"]["name"]);
    $uploadOk = 1;
    $imageFileType = strtolower(pathinfo($targetFile, PATHINFO_EXTENSION));

    // Check if image file is a actual image or fake image
    if (isset($_FILES["image"]) && $_FILES["image"]["error"] == 0) {
        $check = getimagesize($_FILES["image"]["tmp_name"]);
        if ($check !== false) {
            $uploadOk = 1;
        } else {
            $errorMsg = "File is not an image.";
            $uploadOk = 0;
        }
    } else {
        $errorMsg = "No file uploaded.";
        $uploadOk = 0;
    }

    // Check file size
    if ($_FILES["image"]["size"] > 500000) {
        $errorMsg = "Sorry, your file is too large.";
        $uploadOk = 0;
    }

    // Allow certain file formats
    $allowedFormats = ["jpg", "jpeg", "png", "gif"];
    if (!in_array($imageFileType, $allowedFormats)) {
        $errorMsg = "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
        $uploadOk = 0;
    }
```

Figure 55 Add crops page implementation

```php
    // Try to upload file if $uploadOk is set to 1
    if ($uploadOk == 1) {
        if (move_uploaded_file($_FILES["image"]["tmp_name"], $targetFile)) {
            $successMsg = "The file " . htmlspecialchars(basename($_FILES["image"]["name"])) . " has been uploaded successfully.";
            $image_url = $targetFile; // Store this URL in the database

            // SQL query to insert data into the crops table if file upload was successful
            $sql = "INSERT INTO crops (Crop_Name, image, Description, Farm_id_fk, Crops_type_id_fk) VALUES ('$crop_name', '$image_url', '$description', $farm_id, $crops_type_id)";

            if ($conn->query($sql) === TRUE) {
                $successMsg .= " New crop added successfully.";
            } else {
                $errorMsg = "Error: " . $sql . "<br>" . $conn->error;
            }
        } else {
            $errorMsg = "Sorry, there was an error uploading your file.";
        }
    }
}

// Function to reload page after 3 seconds
0 references
function reloadPage() {
    echo '<script>
            setTimeout(function() {
                window.location.reload();
            }, 3000);
          </script>';
}
?>
```

Figure 56 Add Crops page Implementation

6.2.4 View Crops Page

Figure 57 View Crops Details

```php
<?php
include 'config.php';

// Fetch crop data
$sql = "SELECT crops.Crop_id, crops.Crop_Name,crops.image,crops.Description, farm.F_name AS Farm_Name, crops_type.C_type AS Crop_Type FROM crops JOIN farm ON crops.Farm_id_fk = farm.F_id JOIN crops_type ON crops.Crops_type_id_fk = crops_type.C_type_id;";
$result = $conn->query($sql);

$crops = [];
if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        $crops[] = $row;
    }
}

$conn->close();
?>


<!-- view_all_crops.html -->
<!DOCTYPE html>
<html>
<head>
  <title>View All Crops</title>
  <link rel="stylesheet" href="viewcrops.css">    "viewcrops": Unknown word.
</head>
<body style="background-image:url('crops.jpg');">
  <div class="container" style="width:max-content">
    <h1>View All Crops</h1>
    <table>
      <thead>
        <tr>
          <th>Crop ID</th>
          <th>Crop Name</th>
          <th>Crop Description</th>
          <th>Farm Name</th>
          <th>Crop Type</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        <?php foreach ($crops as $crop) {?>
        <tr style="font-size:20px">
          <td><?php echo $crop['Crop_id'];?></td>
          <td><?php echo $crop['Crop_Name'];?></td>
          <td><?php echo $crop['Description'];?></td>
          <td><?php echo $crop['Farm_Name'];?></td>
          <td><?php echo $crop['Crop_Type'];?></td>
          <td>
            <button class="btn btn-insert" onclick="insertCrop()" style="background-color:blue">Insert</button><br>
            <button class="btn btn-update" onclick="updateCrop(<?php echo $crop['Crop_id']; ?>)">Update</button><br>
            <button class="btn btn-delete" onclick="deleteCrop(<?php echo $crop['Crop_id']; ?>)">Delete</button>
          </td>
        </tr>
        <?php }?>
      </tbody>
    </table>
  </div>
```

Figure 58 View Crops Details Code

6.2.5 Update Crop Details

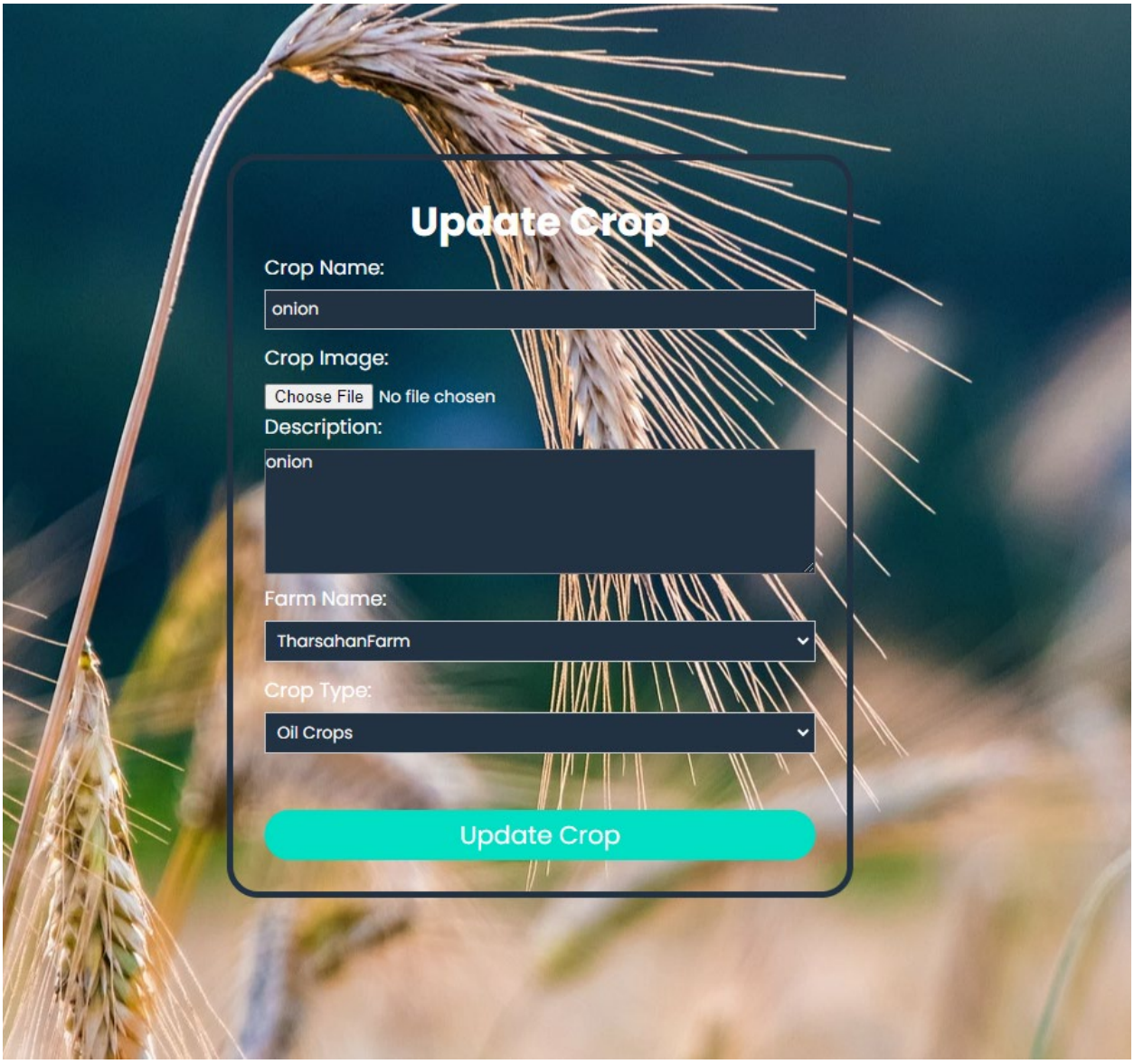


Figure 59 Update Crops

```php
<?php
include 'config.php';

// Initialize variables
$errorMsg = '';
$successMsg = '';

// Check if Crop_id is provided in the URL
if (isset($_GET['Crop_id'])) {
    $crop_id = $conn->real_escape_string($_GET['Crop_id']);

    // Fetch crop data for the selected Crop_id
    $sql = "SELECT * FROM crops WHERE Crop_id = $crop_id";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $crop = $result->fetch_assoc();
    } else {
        $errorMsg = "Crop not found.";
    }
} else {
    $errorMsg = "Crop ID not specified.";
}

// Process form submission
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $crop_id = $_POST['Crop_id'];
    $crop_name = $conn->real_escape_string($_POST['CropName']);
    $description = $conn->real_escape_string($_POST['Description']);
    $farm_id = $conn->real_escape_string($_POST['FarmName']);
    $crop_type_id = $conn->real_escape_string($_POST['CropType']);

    // File upload handling
    $image_url = ''; // Initialize image URL

    if ($_FILES['image']['error'] === UPLOAD_ERR_OK) {
        $targetDir = "uploads/";
        $targetFile = $targetDir . basename($_FILES["image"]["name"]);
        $imageFileType = strtolower(pathinfo($targetFile, PATHINFO_EXTENSION));

        // Check file size
        if ($_FILES["image"]["size"] > 500000) {
            $errorMsg = "Sorry, your file is too large.";
        } elseif (!in_array($imageFileType, ['jpg', 'jpeg', 'png', 'gif'])) {
            $errorMsg = "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
        } elseif (move_uploaded_file($_FILES["image"]["tmp_name"], $targetFile)) {
            $image_url = $targetFile;
        } else {
            $errorMsg = "Sorry, there was an error uploading your file.";
        }
    } elseif ($_FILES['image']['error'] !== UPLOAD_ERR_NO_FILE) {
        $errorMsg = "Error uploading image: " . $_FILES['image']['error'];
    }
```

Figure 60 Update Crops Code

```php
    // Update query if no file upload error
    if (empty($errorMsg)) {
        $sql = "UPDATE crops SET Crop_Name = '$crop_name', Description = '$description', Farm_id_fk = '$farm_id', Crops_type_id_fk = '$crop_type_id'";

        if (!empty($image_url)) {
            $sql .= ", image = '$image_url'";
        }

        $sql .= " WHERE Crop_id = $crop_id";

        if ($conn->query($sql) === TRUE) {
            $successMsg = "Crop updated successfully.";
        } else {
            $errorMsg = "Error updating crop: " . $conn->error;
        }
    }

    // Function to reload page after 3 seconds
    echo '<script>
            setTimeout(function() {
                window.location.href = "Viewcrops.php";    "Viewcrops": Unknown word.
            }, 3000);
          </script>';
}
?>

<!DOCTYPE html>
<html>
<head>
  <title>Update Crop</title>
  <link rel="stylesheet" href="crops.css">
  <style>
    .message-box {
      display: none;
      position: fixed;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
      padding: 20px;
      background-color: #fff;
      color: #000;
      border: 2px solid;
```

Figure 61 Update crop Codes

6.2.6 Add Harvest Page

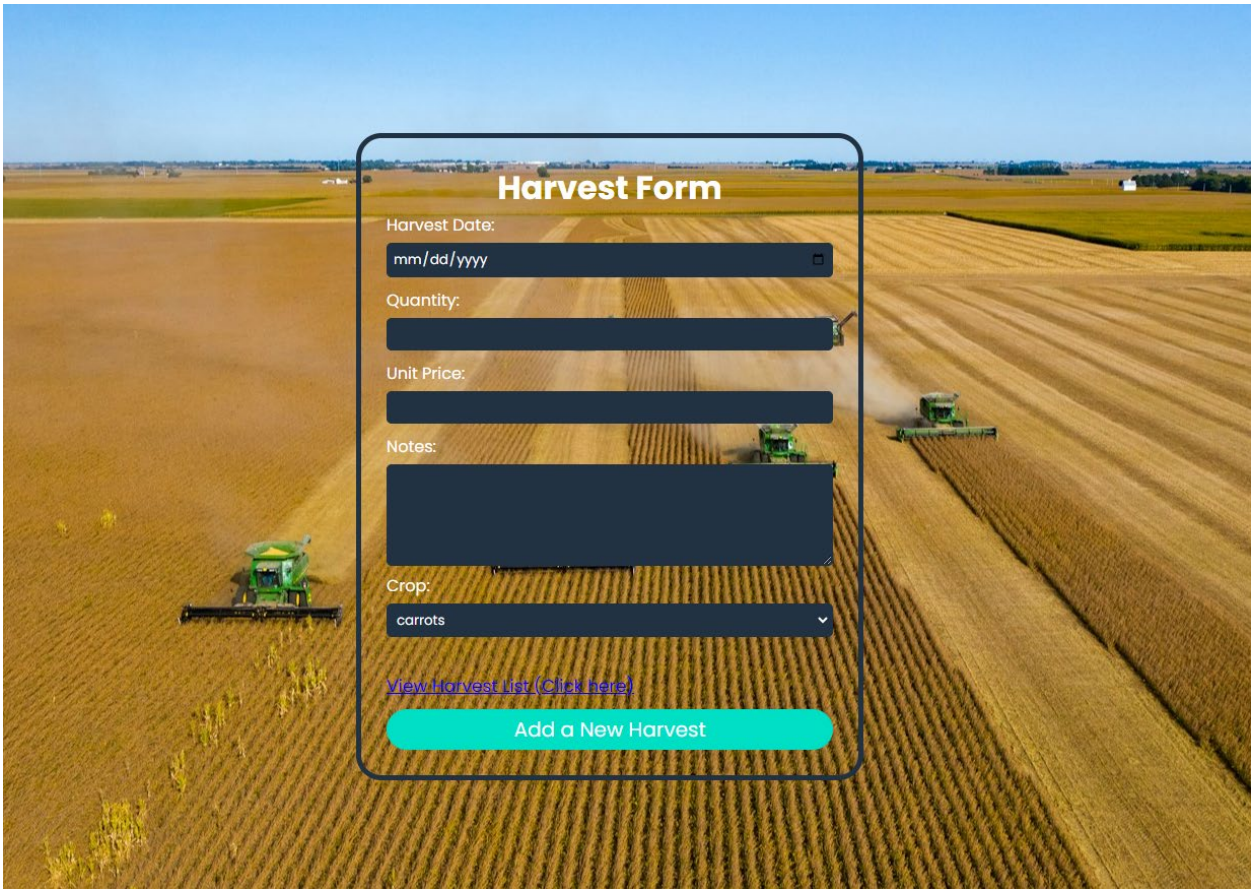


Figure 62 Add Harvest Page

```php
<?php
session_start();
include 'config.php';

// Initialize the message variable
$message = '';

// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $Harvest_date = $conn->real_escape_string($_POST['harvest_date']);
    $Harvest_Quantity = $conn->real_escape_string($_POST['quantity']);
    $Unit_price = $conn->real_escape_string($_POST['Price']);
    $Notes = $conn->real_escape_string($_POST['notes']);
    $crop_id = $conn->real_escape_string($_POST['crop_id']);

    // SQL query to insert data into the harvest table
    $sql = "INSERT INTO harvest (Harvest_date, Harvest_Quantity, Unit_Price, Harvest_Notes, Crop_id_fk) VALUES ('$Harvest_date', '$Harvest_Quantity', '$Unit_price', '$Notes', '$crop_id')";

    if ($conn->query($sql) === TRUE) {
        $_SESSION['message'] = "New Harvest Added successfully";
    } else {
        $_SESSION['message'] = "Error: " . $sql . "<br>" . $conn->error;
    }

    // Redirect to the same page to prevent resubmission
    header("Location: " . $_SERVER['PHP_SELF']);
    exit();
}
?>

<!DOCTYPE html>
<html>
<head>
  <title>Harvest Form</title>
  <link rel="stylesheet" href="harvest.css">
  <style>
        .message {
            text-align: center;
            background-color: green;
            color: white;
            padding: 10px;
            margin-top: 20px;
        }
        .container {
            position: relative;
        }
    </style>
</head>
<body style="background-image:url('harvest.jpg');">
  <div class="container">
    <?php
    if (isset($_SESSION['message'])) {
        echo "<div id='message' class='message'>" . $_SESSION['message'] . "</div>";
        unset($_SESSION['message']);
    }
```

Figure 63 Add Harvest Page Codes

6.2.7 View Harvest Details page

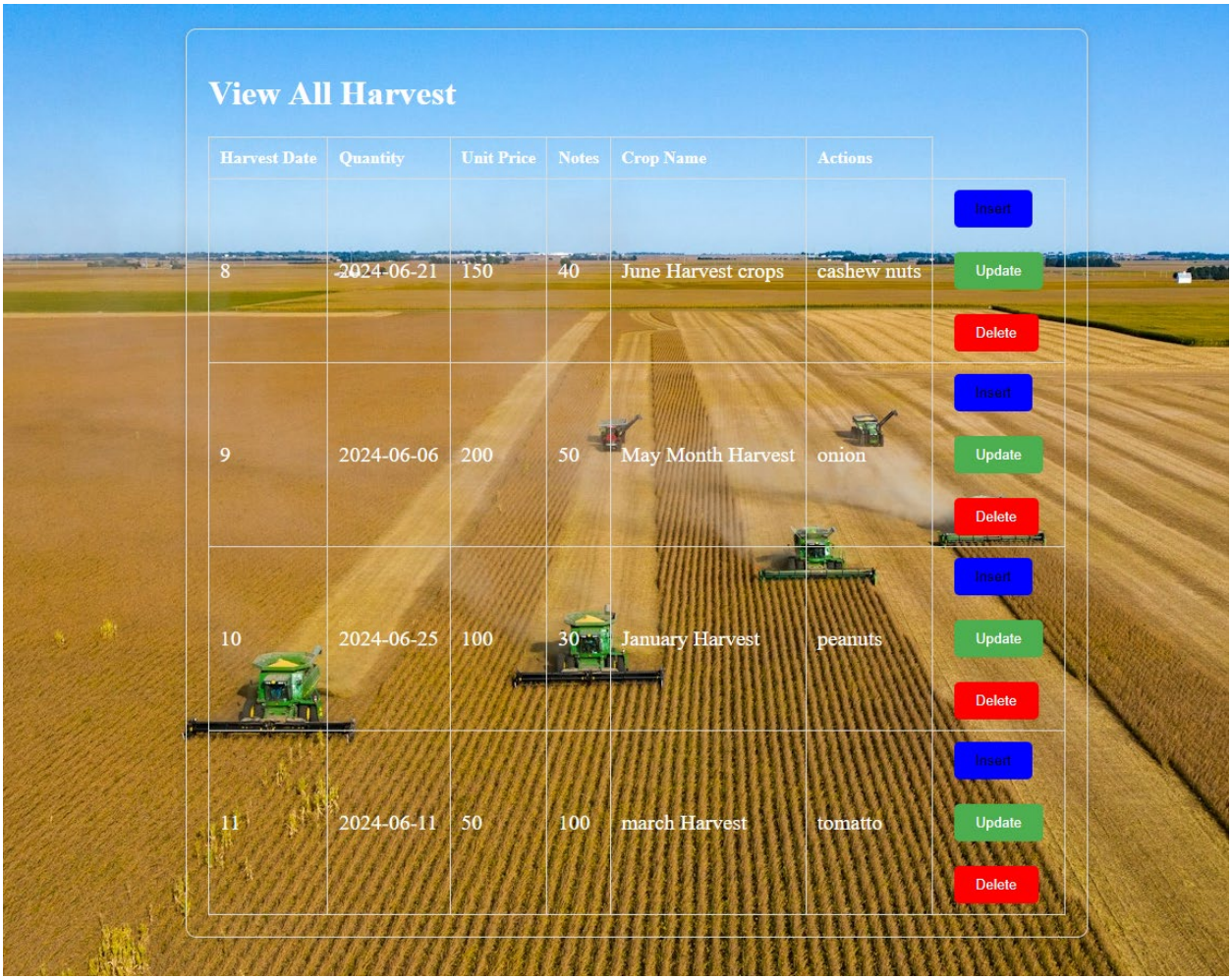


Figure 64 View Harvest Details Code

```php
<?php
include 'config.php';

// Fetch crop data
$sql = "SELECT harvest.Harvest_id, harvest.Harvest_date,harvest.Harvest_Quantity,harvest.Unit_Price,harvest.Harvest_Notes,crops.Crop_Name FROM harvest JOIN crops ON harvest.Crop_id_fk = crops.Crop_id;";
$result = $conn->query($sql);

$crops = [];
if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        $crops[] = $row;
    }
}

$conn->close();
?>


<!-- view_all_crops.html -->
<!DOCTYPE html>
<html>
<head>
  <title>View All Harvest</title>
  <link rel="stylesheet" href="Viewharvest.css">    "Viewharvest": Unknown word.
</head>
<body style="background-image:url('harvest.jpg');">
  <div class="container" style="width:max-content">
    <h1>View All Harvest</h1>
    <table>
      <thead>
        <tr>
          <th>Harvest Date</th>
          <th>Quantity</th>
          <th>Unit Price</th>
          <th>Notes</th>
          <th>Crop Name</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        <?php foreach ($crops as $crop) {?>
        <tr style="font-size:20px">
          <td><?php echo $crop['Harvest_id'];?></td>
          <td><?php echo $crop['Harvest_date'];?></td>
          <td><?php echo $crop['Harvest_Quantity'];?></td>
          <td><?php echo $crop['Unit_Price'];?></td>
          <td><?php echo $crop['Harvest_Notes'];?></td>
          <td><?php echo $crop['Crop_Name'];?></td>
          <td>
            <button class="btn btn-insert" onclick="insertHarvest()" style="background-color:blue">Insert</button><br>
            <button class="btn btn-update" onclick="updateHarvest(<?php echo $crop['Harvest_id']; ?>)">Update</button><br>
            <button class="btn btn-delete" onclick="deleteHarvest(<?php echo $crop['Harvest_id']; ?>)">Delete</button>
          </td>
        </tr>
        <?php }?>
      </tbody>
    </table>
  </div>
```

Figure 65 View Harvest Codes

6.2.8 Update Harvest Page

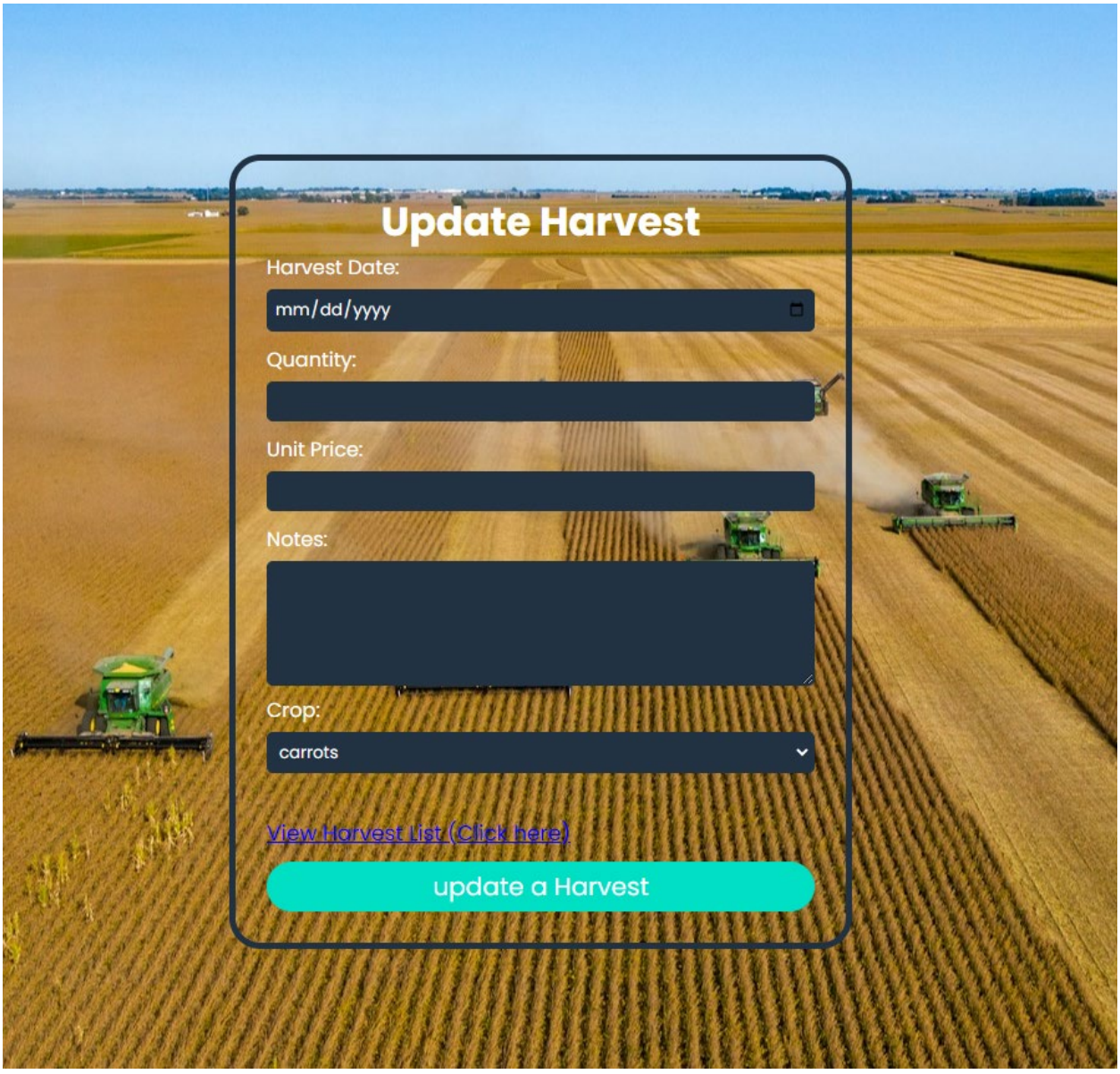


Figure 66 Update Harvest Page

```php
<?php
session_start();
include 'config.php';

// Initialize the message variable
$message = '';

// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $Harvest_id = $conn->real_escape_string($_POST['harvest_id']);
    $Harvest_date = $conn->real_escape_string($_POST['harvest_date']);
    $Harvest_Quantity = $conn->real_escape_string($_POST['quantity']);
    $Unit_price = $conn->real_escape_string($_POST['Price']);
    $Notes = $conn->real_escape_string($_POST['notes']);
    $crop_id = $conn->real_escape_string($_POST['crop_id']);

    // SQL query to update the harvest record
    $sql = "UPDATE harvest SET Harvest_date='$Harvest_date', Harvest_Quantity='$Harvest_Quantity', Unit_Price='$Unit_price', Harvest_Notes='$Notes', Crop_id_fk='$crop_id' WHERE Harvest_id='$Harvest_id'";

    if ($conn->query($sql) === TRUE) {
        $_SESSION['message'] = "Harvest record updated successfully";
    } else {
        $_SESSION['message'] = "Error: " . $sql . "<br>" . $conn->error;
    }

    // Redirect to the same page to prevent resubmission
    header("Location: " . $_SERVER['PHP_SELF'] . "?id=" . $Harvest_id);
    exit();
}

// Fetch the current harvest details if an id is provided
if (isset($_GET['id'])) {
    $Harvest_id = $conn->real_escape_string($_GET['id']);
    $sql = "SELECT * FROM harvest WHERE Harvest_id='$Harvest_id'";
    $result = $conn->query($sql);

    if ($result->num_rows == 1) {
        $harvest = $result->fetch_assoc();
    } else {
        $_SESSION['message'] = "Harvest record not found";
        header("Location: viewHarvest.php");
        exit();
    }
} else {
    $_SESSION['message'] = "No harvest record specified";
    header("Location: viewHarvest.php");
    exit();
}
?>

<!DOCTYPE html>
<html>
<head>
    <title>Update Harvest</title>
    <link rel="stylesheet" href="harvest.css">
    <style>
        .message {
            text-align: center;
            background-color: green;
            color: white;
            padding: 10px;
```

Figure 67 Update Harvest Page Codes

Chapter 07 – Testing and Verification

7.1 Scope of Testing

it mentions that user testing will be conducted with farmers to evaluate the system's usability and effectiveness. It's likely that the testing will focus on:

- Usability: How easy the platform is to use and navigate for farmers.
- Functionality: Whether the platform meets the identified functional requirements and performs as intended.
- Accessibility: Whether the platform is accessible to users with disabilities.
- Security: The effectiveness of the security measures implemented.
- Performance: The platform's ability to handle a high volume of users and transactions without impacting performance.

7.2 Test Cases

Test Case 01: Admin Login

Description	Preconditions	Steps	Expected Result	Result
Verify admin can login with valid credentials	Admin account exists	1. Navigate to the Admin page 2. Enter Valid Email and Password 3. Click on the "Login" button	Admin is Logged in and redirected to the admin Dashboard	Success

Table 5 Testcase 01

Test Case 02 : Invalid Admin Login

Description	Preconditions	Steps	Expected Result	Result
Verify admin cannot log in with invalid credentials.	Admin account exists.	1. Navigate to the Admin page 2. Enter Invalid Email and Password 3. Click on the "Login" button	Error message is displayed indicating invalid credentials	Success

Table 6 Testcase 02

Test Case 03 : Add a new Farmer

Description	Preconditions	Steps	Expected Result	Result
Verify that a new farmer can be added.	Admin is logged in.	1. Navigate to the add farmer page. 2. Enter farmer details (name, address, contact number, age, etc.). 3. Click on the "Save" button.	New farmer is added and listed in the farmer list.	Success

Table 7 Test Case 03

Test Case 04: Update Farmer Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update farmer details.	Admin is logged in, and a farmer record exists.	1. Navigate to the farmer list page. 2. Select a farmer and click on "Edit." 3. Update farmer details 4. Click on the "Save" button.	Farmer details are updated successfully.	Success

Table 8 Testcase 04

Test Case 05: Delete Farmer

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a farmer.	Admin is logged in, and a farmer record exists.	1. Navigate to the farmer list page. 2. Select a farmer and click on "Delete." 3. Confirm the deletion.	Farmer is deleted from the system.	Success

Table 9 Testcase 05

Test Case 06: Add New Farm

Description	Preconditions	Steps	Expected Result	Result
Verify that a new farm can be added.	Admin is logged in	1. Navigate to the add farm page. 2. Enter farm details (name, address, size, etc.). 3. Select a farmer. 4. Click on the "Save" button.	New farm is added and listed in the farm list.	Success

Table 10 Testcase06

Test Case 07: Update Farm Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update farm details.	Admin is logged in, and a farm record exists.	1. Navigate to the farm list page. 2. Select a farm and click on "Edit." 3. Update farm details. 4. Click on the "Save" button.	Farm details are updated successfully.	Success

Table 11 Testcase 07

Test Case 08: Delete Farm

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a farm.	Admin is logged in, and a farm record exists.	1. Navigate to the farm list page.. 2. Select a farm and click on "Delete." 3. Confirm the deletion.	Farm is deleted from the system.	Success

Table 12 Testcase 08

Test Case 09: Record New Harvest

Description	Preconditions	Steps	Expected Result	Result
Verify that a new harvest can be recorded.	Admin is logged in, farm and crop records exist.	1. Navigate to the record harvest page. 2. Enter harvest details (date, quantity, notes). 3. Select a crop. 4. Click on the "Save" button.	New harvest is recorded and listed in the harvest list.	Success

Table 13 Testcase 09

Test Case 10 : Update Harvest Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update harvest details.	Admin is logged in, and a harvest record exists.	1. Navigate to the harvest list page. 2. Select a harvest and click on "Edit." 3. Update harvest details. 4. Click on the "Save" button.	Harvest details are updated successfully.	Success

Table 14 Test Case 10

Test Case 11 : Delete Harvest

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a harvest record.	Admin is logged in, and a harvest record exists.	1. Navigate to the harvest list page. 2. Select a harvest and click on "Delete." 3. Confirm the deletion.	Harvest record is deleted from the system.	Success

Table 15 Testcase 11

Test Case 12 : Record new Sale

Description	Preconditions	Steps	Expected Result	Result
Verify that a new sale can be recorded.	Admin is logged in, customer, harvest, and crop records exist.	1. Navigate to the record sale page. 2. Enter sale details (total sale amount). 3. Select a customer and harvest. 4. Click on the "Save" button.	New sale is recorded and listed in the sales list.	Success

Table 16 Testcase 12

Test Case 13: Update Sale Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update sale details.	Admin is logged in, and a sale record exists.	1. Navigate to the sales list page. 2. Select a sale and click on "Edit." 3. Update sale details. 4. Click on the "Save" button.	Sale details are updated successfully.	Success

Table 17 Testcase 13

Test Case 14: Delete Sale

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a sale record.	Admin is logged in, and a sale record exists.	1. Navigate to the sales list page. 2. Select a sale and click on "Delete." 3. Confirm the deletion.	Sale record is deleted from the system.	Success

Table 18 Test case14

Test Case 15 : Record New Payment

Description	Preconditions	Steps	Expected Result	Result
Verify that a new payment can be recorded.	• Admin is logged in, and sale record exists.	1. Navigate to the record payment page.. 2. Enter payment details (amount, status). 3. Select a sale. 4. Click on the "Save" button.	New payment is recorded and listed in the payments list.	Success

Table 19 Testcase 15

Test Case 16: Update Payment Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update payment details	• Admin is logged in, and a payment record exists.	1. Navigate to the payments list page. 2. Select a payment and click on "Edit." 3. Update payment details. 4. Click on the "Save" button.	Payment details are updated successfully.	Success

Table 20 Testcase 16

Test Case 17 Delete Payment

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a payment record.	• Admin is logged in, and a payment record exists.	1. Navigate to the payments list page. 2. Select a payment and click on "Delete." 3. Confirm the deletion.	Payment record is deleted from the system.	Success

Table 21 Test case 17

Test Case 18: Add New Customer

Description	Preconditions	Steps	Expected Result	Result
Verify that a new customer can be added.	• Admin is logged in.	1. Navigate to the add customer page. 2. Enter customer details (name, mobile, address, email, password). 3. Click on the "Save" button.	New customer is added and listed in the customer list.	Success

Table 22 Test case 18

Test Case 19: Update Customer Details

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can update customer details.	•Admin is logged in, and a customer record exists.	1. Navigate to the add customer list page. 2. Select customer and click on “Update” 3. Update Customer Details 4. Click on the” Save” Button	Customer details are updated successfully.	Success

Table 23 TEst case 19

Test Case 20: Delete Customer

Description	Preconditions	Steps	Expected Result	Result
Verify that an admin can delete a customer record.	•Admin is logged in, and a customer record exists.	1. Navigate to the customer list page. 2. Select customer and click on "Delete" 3. Confirm Deletion	Customer record is Deleted from the system	Success

Table 24 Test case 20

Chapter 8 Evaluation and Conclusion

8.1 Evaluation

The "Revolutionizing Agriculture: A Broker-Free Marketplace for Farmers" project has proven to be a successful endeavor in its initial stages. The development of the web application, focusing on admin login, farmer registration, farmer login, crop addition, and harvest addition pages, has demonstrated the feasibility of the proposed solution. This includes a secure and user-friendly interface for farmers and administrators, facilitating the process of product listing and management. Furthermore, the chosen technologies, including PHP and MySQL, have proven efficient and cost-effective for building the platform's backend functionality. However, the evaluation also reveals the need for further development, particularly regarding the mobile application. While the wireframes have been created, the actual development and integration of the mobile app remain a priority for future work.

8.2 Conclusion

The "Revolutionizing Agriculture" project has the potential to significantly impact the agricultural sector by empowering farmers and creating a more transparent and efficient marketplace. By directly connecting farmers with consumers, this project eliminates intermediaries, reduces costs, and increases farmers' incomes. The project's success hinges on its ability to overcome challenges such as digital literacy gaps among farmers and ensuring internet connectivity in rural areas. Continued development, particularly the implementation of the mobile app and its integration with the web platform, will be critical to achieving widespread adoption and maximizing the project's positive impact on the agricultural industry.

Chapter 09 User Guide

9.1 User Guide for mobile application

Getting Started

1. **Download the app:** Search for "Revolutionizing Agriculture" in your app store (Google Play or Apple App Store) and download the application.
2. **Register/Login:**
    - If you are a new user, tap "Register" and follow the on-screen instructions to create your account.
    - Existing users can tap "Login" and enter your registered email address and password. Or click google sign in button to sign in via google.

Main Screen:

- **Welcome Screen:** after login successful the welcome screen displays your name and email address after 3 sec the screen redirect to Orders Screen
- **Order Screen:** if you are select the food or vegetables select the quantity and click the add to cart button that selected product successfully added to add to card after you finish shopping click cart button on the screen left corner click it the cart page is open
- **Cart Screen**: your cart items stored in the cart screen click the remove cart button you can remove the cart does not change you click the checkout button to screen redirect to payment page.
- **Payment screen:** if your total amount show in a top of the payment screen and you can fill your card details in fields and pay now button payment successfully completed. don't worry for payment issues we use secure payment gateway (stripe).
- **Order complete screen:** if payment completed the screen thank you for your order. if payment receive from farmer the farmer contact you shortly and get your address the product has been arrived quickly. If any farmer related issues you can call anytime. our contact details in about screen.

LONDON METROPOLITAN UNIVERSITY

Chapter 10: References and Bibliography

10.1 References

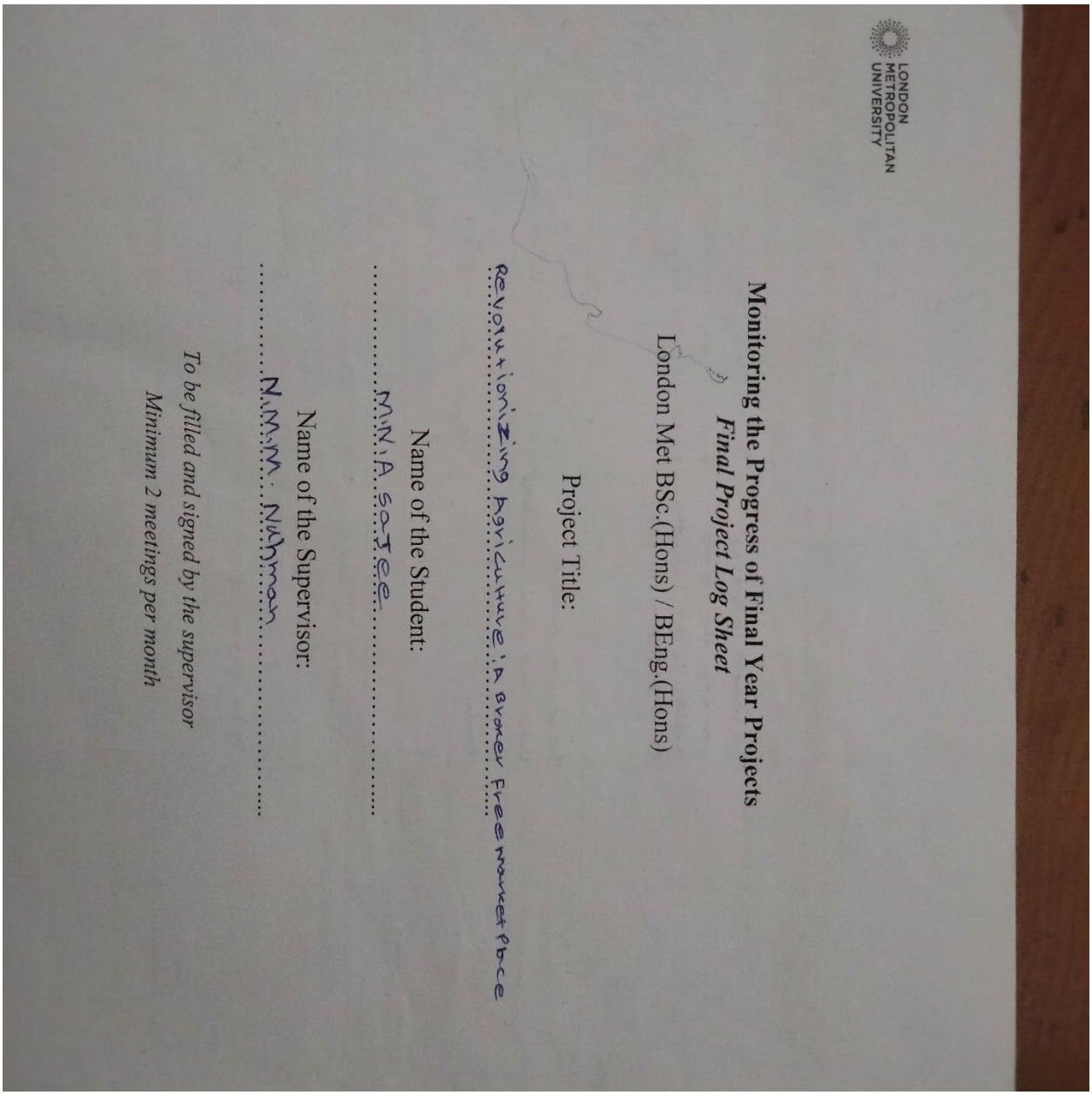
- FAO. (2021). The State of Food and Agriculture 2021: Pathways to Digitalization. Food and Agriculture Organization of the United Nations. https://www.fao.org/documents/card/en/c/cb4623en/
- IFPRI. (2020). The Role of Digital Technologies in Transforming Food Systems. International Food Policy Research Institute. https://www.ifpri.org/blog/role-digital-technologies-transforming-food-systems
- UNDP. (2019). Digital Technologies for Agriculture and Rural Development. United Nations Development Programme. https://www.undp.org/publications/digital-technologies-agriculture-and-rural-development
- World Bank. (2020). Digital Agriculture: A New Paradigm for Agricultural Transformation. World Bank. https://www.worldbank.org/en/topic/digitaldevelopment/brief/digital-agriculture-a-new-paradigm-for-agricultural-transformation
- World Economic Forum. (2018). The Future of Food: How Technology is Transforming the Way We Eat. World Economic Forum. https://www.weforum.org/agenda/2018/01/the-future-of-food-how-technology-is-transforming-the-way-we-eat/

10.2 Bibliography

11. FAO. (2021). The State of Food and Agriculture 2021: Pathways to Digitalization. Food and Agriculture Organization of the United Nations. https://www.fao.org/documents/card/en/c/cb4623en/
12. IFPRI. (2020). The Role of Digital Technologies in Transforming Food Systems. International Food Policy Research Institute. https://www.ifpri.org/blog/role-digital-technologies-transforming-food-systems
13. UNDP. (2019). Digital Technologies for Agriculture and Rural Development. United Nations Development Programme. https://www.undp.org/publications/digital-technologies-agriculture-and-rural-development
14. World Bank. (2020). Digital Agriculture: A New Paradigm for Agricultural Transformation. World Bank. https://www.worldbank.org/en/topic/digitaldevelopment/brief/digital-agriculture-a-new-paradigm-for-agricultural-transformation
15. World Economic Forum. (2018). The Future of Food: How Technology is Transforming the Way We Eat. World Economic Forum. https://www.weforum.org/agenda/2018/01/the-future-of-food-how-technology-is-transforming-the-way-we-eat/

Chapter 11 Appendices

11.1 Progress approval form and Project Commencement meeting Sheet



LONDON METROPOLITAN UNIVERSITY

Monitoring the Progress of Final Year Projects

Final Project Log Sheet

London Met BSc.(Hons) / BEng.(Hons)

Project Title:

Revolutionizing horticulture: A broker free market place

Name of the Student:

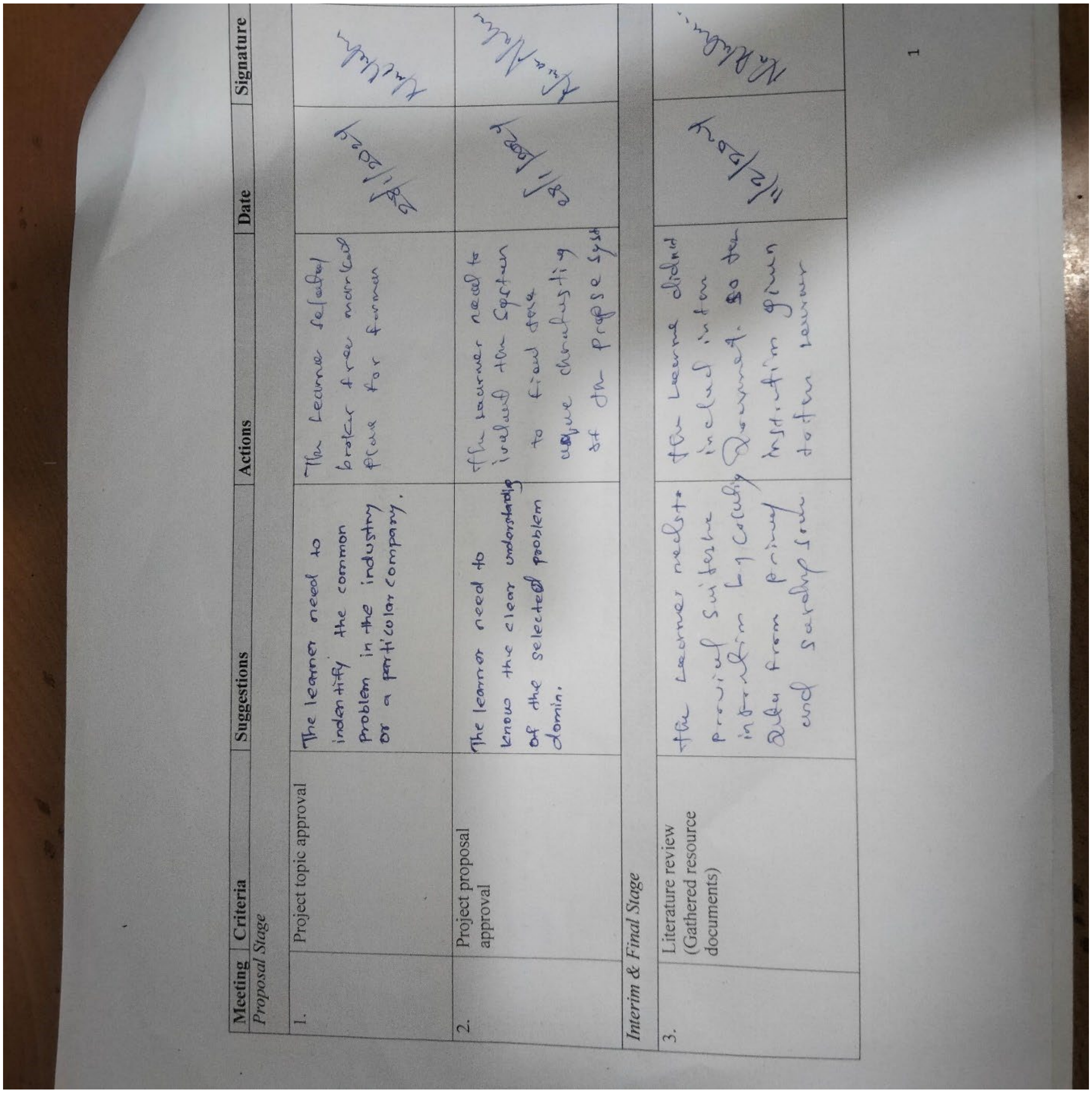
MINA SAJEE

Name of the Supervisor:

NAMAN NAHMAN

To be filled and signed by the supervisor

Minimum 2 meetings per month



Meeting	Criteria	Suggestions	Actions	Date	Signature
Proposal Stage					
1.	Project topic approval	The learner need to indentify the common problem in the industry or a particolar company.	The Learner selected broker free market place for farmer	28/1/2024	[illegible]
2.	Project proposal approval	The learner need to know the clear understanding of the selected problem domin.	The learner need to evaluate the system to find the unique characteristic of the propose syst	28/1/2024	[illegible]
Interim & Final Stage					
3.	Literature review (Gathered resource documents)	The learner needs to provide suitable information by collecting data from primary and secondary source	The learner didnot include in the document. So the instruction given to the learner	11/2/2024	[illegible]

1

Literature review (Report)	the Learner needs to evaluate the existing system	the Learner didn't provide suitable information about the existing system	11/2/2024	[illegible]
Approach (Users, Input, Output, Technologies Used)	As per the given functional and non functional Requirement the Learner needs to [illegible] select tables	the Leader didn't include in the document. but the Learner build the system in PHP	11/2/2024	[illegible]
Project Design	to fulfill the functional Requirement the Learner needs to provide suitable Design.	the Learner Database Diagram in Acceptable but the Learner need to improve the use case Diagram	11/2/2024	[illegible]
Implementation (Algorithms, Flow charts, System, etc.)				

2