RAMBUTAN WAREHOUSE

Mini Project Report

Submitted by

Shanu Sara Shibu

Reg. No.: AJC23MCA-2058

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS (MCA)

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Approved by AICTE, Accredited by NAAC, Accredited by NBA.
Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**RAMBUTAN WAREHOUSE"** is the bonafide work of **SHANU SARA SHIBU (Regno: AJC23MCA-2058)** in partial fulfillment of the requirements for the award of the Degree of Regular Master of Computer Applications under **Amal Jyothi College of Engineering Autonomous, Kanjirappally** during the year 2024-25.

Mr. Ajith G. S
Internal Guide

Mr. Binumon Joseph
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

DECLARATION

I hereby declare that the project report **"RAMBUTAN WAREHOUSE"** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Regular Master of Computer Applications (MCA) from Amal Jyothi College of Engineering Autonomous during the academic year 2024-2025.

Date: 06-11-2024	**SHANU SARA SHIBU**
KANJIRAPPALLY	**Reg: AJC23MCA-2058**

ACKNOWLEDGEMENT

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

I owe a great depth of gratitude towards our Head of the Department **Rev. Fr. Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr. Binumon Joseph** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr. Ajith G. S** for his inspiration and helping hand.

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

SHANU SARA SHIBU

ABSTRACT

The Rambutan Warehouse project aims to develop an integrated online platform that streamlines the buying and selling of rambutan fruit and related products. Designed to cater to the needs of various stakeholders in the rambutan trade industry, the platform provides a single space where farmers can sell their produce, wholesale buyers can make bulk purchases, and normal consumers can buy fresh or processed rambutan goods.

The platform addresses several key challenges faced by the rambutan trade. Firstly, it offers a solution to the limited market access that many farmers face. By providing a digital marketplace, the platform expands farmers' reach, allowing them to sell their produce to a wider audience. Secondly, it eliminates the need for intermediaries, who often take a significant portion of the profits, thereby allowing farmers to earn a fair price for their produce. Finally, the platform also addresses the perishability issue by facilitating faster and more efficient sales and deliveries.

The Rambutan Warehouse includes several key modules. These modules include a farmer management system, which allows farmers to list their products, track their inventory, and monitor sales. Wholesale buyers are provided with tools to place bulk orders, negotiate prices, and schedule deliveries. Regular buyers can browse and purchase various rambutan products, while administrators oversee the platform's operations, ensuring smooth transactions and handling any issues that arise.

Overall, the Rambutan Warehouse project aims to revolutionize the rambutan trade by providing a seamless, user-friendly platform that benefits all stakeholders.

CONTENT

List of Abbreviations

- Django: Python Framework
- HTML: Hypertext Markup Language
- CSS: Cascading Style Sheets
- JS: JavaScript
- URL: Uniform Resource Locator
- SMTP: Simple Mail Transfer Protocol
- API: Application Program Interface

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Rambutan Warehouse project aims to develop an integrated online platform that streamlines the buying and selling of rambutan fruit and related products. Designed to cater to the needs of various stakeholders in the rambutan trade industry, the platform provides a single space where farmers can sell their produce, wholesale buyers can make bulk purchases, and normal consumers can buy fresh or processed rambutan goods.

The platform addresses several key challenges faced by the rambutan trade. Firstly, it offers a solution to the limited market access that many farmers face. By providing a digital marketplace, the platform expands farmers' reach, allowing them to sell their produce to a wider audience. Secondly, it eliminates the need for intermediaries, who often take a significant portion of the profits, thereby allowing farmers to earn a fair price for their produce. Finally, the platform also addresses the perishability issue by facilitating faster and more efficient sales and deliveries.

The Rambutan Warehouse includes several key modules. These modules include a farmer management system, which allows farmers to list their products, track their inventory, and monitor sales. Wholesale buyers are provided with tools to place bulk orders, negotiate prices, and schedule deliveries. Regular buyers can browse and purchase various rambutan products, while administrators oversee the platform’s operations, ensuring smooth transactions and handling any issues that arise.

Overall, the Rambutan Warehouse project aims to revolutionize the rambutan trade by providing a seamless, user-friendly platform that benefits all stakeholders.

The platform is developed with the following modules:

- **Farmers**, who can manage their product listings, inventories, and track their sales;
- **Wholesale Buyers**, who can purchase in bulk, negotiate prices, and schedule deliveries;
- **Administrator**, who oversee operations, resolve issues, and ensure smooth transactions;
- **Normal Buyers**, who can browse and buy fresh and processed rambutan products directly.

Modules

• Farmers:

- List product.
- Manage inventory.

- Track sales.

• Wholesale Fresh Rambutan Buyers:

- Place bulk orders.
- Negotiate prices.

• Normal Buyers:

- Access a variety of rambutan products.
- Purchase fresh fruit and processed goods directly.

• Admin:

- Oversee platform operations.
- Ensure smooth transactions.
- Resolve issues.

1.2 PROJECT SPECIFICATION

The Rambutan Warehouse system is built as a web-based platform that caters to multiple user roles, each with specific functionalities. The core features of the platform revolve around efficient transaction handling, real-time inventory management, and secure payment processing.

From a frontend development perspective, the system uses HTML and CSS to ensure a user-friendly interface that is both responsive and easy to navigate. The layout is designed to allow users to browse products, place orders, and manage their accounts effortlessly. In addition, the system supports multiple user roles, each of which is granted access to different sections of the platform. Farmers can upload their produce, manage inventory, and track sales; wholesale buyers can negotiate prices, place orders, and arrange deliveries; and regular consumers can browse and purchase products.

On the backend, the system is powered by Django, a Python-based web framework that offers robustness and scalability. The use of Django ensures that the system can handle large volumes of data and transactions efficiently. SQLite is used as the database, providing lightweight yet powerful data management that allows for real-time updates on product listings, orders, and inventory levels. In addition to the real-time inventory tracking, the platform also supports secure payment processing, ensuring that all transactions are safe and reliable.

The platform’s architecture is designed to ensure high availability and responsiveness, with a focus on user experience and system performance. Features such as automated email notifications, real-time updates, and a secure login system are built-in to provide seamless interaction across the various user roles.

Overall, the Rambutan Warehouse platform is a highly functional, scalable, and secure system that meets the needs of all its users, from farmers and buyers to administrators.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

System study is a crucial phase in the development of any software system as it helps to identify the existing challenges and gaps that need to be addressed through the new solution. In the Rambutan Warehouse, the study focuses on understanding the rambutan trade ecosystem, its stakeholders, and the issues that impact the smooth operation of rambutan sales and distribution. This stage involved gathering data from farmers, buyers, and agricultural experts to form a complete picture of the current situation and how it affects the trade.

The rambutan trade traditionally relies heavily on local markets and intermediaries, which limit the farmer's access to larger buyers and more lucrative markets. Furthermore, because rambutan is a highly perishable fruit, any delay in finding buyers or distributing the produce can lead to significant losses for farmers. This system study identifies the pain points in this traditional model and lays the groundwork for the design of a digital solution that connects all participants in the rambutan trade in a more efficient and effective way.

The system study also explores the technical requirements, economic feasibility, and behavioral aspects of the system, ensuring that the Rambutan Warehouse platform will be viable in terms of its functionality, cost-effectiveness, and usability. By thoroughly analyzing the current system and designing a solution that addresses its limitations, the project aims to revolutionize how rambutan farmers, buyers, and consumers interact in the market.

2.2 EXISTING SYSTEM

The current system is largely dependent on physical markets or intermediaries. Farmers typically sell their produce to middlemen, who then distribute it to larger markets or buyers. While this system ensures some level of sales, it often results in farmers receiving lower prices for their produce. Additionally, the perishable nature of rambutan means that delays in finding buyers can lead to significant losses for farmers. Wholesale buyers, on the other hand, often rely on established relationships with middlemen or travel to local markets to make bulk purchases.
The absence of a centralized, digital platform to connect farmers and buyers results in inefficiencies, with farmers having limited control over pricing and access to markets.

2.2.1 NATURAL SYSTEM STUDIED

The natural system studied in the context of Rambutan Warehouse refers to the traditional processes involved in the cultivation, sale, and distribution of rambutan. In this natural system, farmers are responsible for growing and harvesting the fruit, while the sale and distribution processes involve several intermediaries. Once harvested, rambutan is typically transported to local markets, where it is sold either to consumers directly or to intermediaries, who in turn sell the produce to larger buyers such as wholesalers and retailers. This natural system has several drawbacks. The reliance on intermediaries reduces the profits that farmers can make, as these middlemen take a cut of the final sale price. Additionally, the natural system is limited by geography, as farmers primarily sell their produce in local markets. This limits their access to a wider range of buyers, particularly wholesale buyers who may be interested in purchasing large quantities of rambutan but are unable to connect with farmers directly. Furthermore, the natural system is inefficient when it comes to managing inventory. Farmers must transport their produce to markets and rely on physical sales, which can be time-consuming and lead to wastage if the fruit is not sold quickly. Given the highly perishable nature of rambutan, this often results in financial losses for farmers, who may be forced to sell their produce at lower prices to avoid spoilage. From the perspective of buyers, the natural system provides limited options for comparing prices or evaluating the quality of the produce. Buyers must often rely on intermediaries or travel to different markets to source rambutan, which can be a time-consuming process. As a result, there is a lack of transparency in the pricing and quality of the fruit, making it difficult for buyers to make informed purchasing decisions.

2.2.2 DESIGNED SYSTEM STUDIED

The Rambutan Warehouse platform is designed to modernize the rambutan trade by creating a direct link between farmers and buyers, eliminating intermediaries. It serves as an online marketplace where farmers can easily list their rambutan produce, manage inventory, and receive secure payments. Buyers, including both wholesale and regular consumers, can browse products, compare prices, and place orders directly with farmers, ensuring transparency and fair pricing.

The platform features real-time inventory management, enabling farmers to update their stock levels and track sales efficiently. Secure payment processing is integrated, allowing smooth and safe transactions. Wholesale buyers can also schedule deliveries, ensuring that the perishable fruit arrives in good condition, which is critical for maintaining freshness.

Administrator oversee the platform’s operations, managing transactions, resolving disputes, and ensuring everything runs smoothly. The system is user-friendly, making it accessible even to those with limited technical knowledge. Additionally, it is scalable, meaning it can handle a growing number of users and transactions without performance issues.

Overall, the designed system simplifies the rambutan trade by providing farmers with a broader market and buyers with easy access to fresh produce. This digital platform benefits all stakeholders by ensuring efficiency, transparency, and better control over the buying and selling process.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Limited Market Access**: Farmers often have restricted access to larger markets, relying primarily on local markets or intermediaries to sell their rambutan.
- **Presence of Intermediaries**: Middlemen reduce farmers' profits by taking a portion of the sale, leading to lower income for farmers.
- **Perishability of Produce**: Rambutan is highly perishable, and delays in finding buyers can result in significant losses for farmers.
- **Lack of Pricing Transparency**: There is often no clear, consistent pricing mechanism, leading to unfair or fluctuating prices for both farmers and buyers.
- **Inefficient Inventory Management**: Farmers manually track their stock, which is time-consuming and prone to errors, potentially leading to overselling or wastage.
- **Limited Buyer Reach**: Wholesale buyers must rely on physical markets or intermediaries, making it harder to connect directly with farmers for bulk purchases.
- **High Operational Costs**: Transportation, middlemen fees, and market commissions increase the overall cost of sales, reducing the profitability for farmers.

2.4 PROPOSED SYSTEM

The proposed system, Rambutan Warehouse, is designed to modernize the rambutan trade by providing a digital platform that connects farmers directly with buyers. This system eliminates intermediaries, allowing farmers to sell their produce at fair prices while providing buyers with access to fresh rambutan and related products. Key features of the system include real-time inventory management, which enables farmers to update their stock levels instantly, reducing the risk of overselling or wastage. Secure payment gateways are integrated, ensuring safe and efficient transactions, with farmers receiving payments directly into their accounts. Wholesale buyers can schedule deliveries, ensuring the timely arrival of fresh produce. The platform is overseen by administrators who manage disputes, monitor operations, and ensure smooth functionality. Designed to be scalable, the platform can handle a growing number of users and transactions, while remaining user-friendly for farmers and buyers alike. Overall, the proposed system simplifies and streamlines the rambutan trade, making it more efficient, transparent, and profitable for all stakeholders involved.

2.5 ADVANTAGES OF PROPOSED SYSTEM

The Rambutan Warehouse platform provides several key advantages over the traditional rambutan trade system:

- Direct Connection Between Farmers and Buyers: By eliminating intermediaries, the platform allows farmers to sell their produce directly to buyers. This results in higher profit margins for farmers and lower prices for buyers. Farmers can set their own prices and negotiate directly with buyers, ensuring that they receive fair compensation for their produce.
- Transparent Pricing: The platform provides a transparent pricing mechanism, allowing buyers to compare prices from different sellers. This fosters a competitive market environment, where buyers are assured of getting the best deals.

- Real-Time Inventory Management: Farmers can track their inventory in real-time, ensuring that they always have up-to-date information on their stock levels. This helps farmers manage their produce more effectively and avoid overselling or underselling.
- Secure Payment Processing: The platform integrates secure payment gateways, ensuring that all transactions are safe and reliable. Buyers can make payments with confidence, and farmers receive their earnings without delays.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility study is an important phase in software development process. It enables developers to have a clear picture of the product being developed in terms of outcomes of the product, operational requirements for implementing it, etc. A feasibility study is conducted to determine whether the project will, upon completion, fulfil the objectives of the organization in relation to the work, effort, and time invested in it. As a result, a new application often undergoes a feasibility assessment before approved for development.

The feasibility study for the Rambutan Warehouse project evaluates the practicality of developing, implementing, and maintaining the system. It focuses on three key areas: economic, technical, and behavioral feasibility. Each aspect is analyzed to ensure that the project can meet its goals and provide value to all stakeholders involved, from farmers and wholesale buyers to administrators and regular users. By addressing the financial, technological, and user adoption challenges, the feasibility study aims to confirm that the project is both viable and sustainable in the long term.

3.1.1 Economical Feasibility

Economic feasibility evaluates whether the project can be developed and operated within a reasonable budget while offering sufficient returns on investment. The Rambutan Warehouse system requires initial investments in software development, database infrastructure, and server hosting. However, by utilizing open-source technologies such as Django for the backend and SQLite3 for database management, the project can significantly reduce costs. Development can be achieved with a modest team of developers, given the straightforward architecture of the system, which further lowers operational expenses.

In the long term, the platform promises to provide a high return on investment for its primary stakeholders. By eliminating middlemen and enabling direct sales between farmers and buyers, the platform allows farmers to capture a larger share of the revenue. Wholesale buyers benefit from competitive prices, enhancing their business operations. Additionally, by offering value-added services such as secure payment processing, the platform can generate income through transaction fees or premium services. These streams of revenue make the Rambutan Warehouse financially viable, ensuring that the project not only covers its costs but also becomes profitable over time.

3.1.2 Technical Feasibility

The technical feasibility examines whether the technology required to develop and operate the Rambutan Warehouse system is available and suitable. The project relies on widely-used, stable technologies, which increases its feasibility. The backend of the platform is built using Django, a robust Python-based framework that allows for rapid development and seamless scalability. SQLite 3, a reliable and efficient database management system, is used to store user information, product listings, transactions, and inventory data. The frontend is developed with HTML, CSS, Java Script ensuring a responsive and user-friendly interface.

Scalability is a key consideration in the system's design, allowing it to handle increasing volumes of transactions and users as the platform grows. With adequate server resources and cloud-based infrastructure, the system can scale to accommodate growing demands without significant technical constraints. Security features such as secure authentication, data encryption, and integration with payment gateways like PayPal or Stripe ensure that the system meets industry standards for data protection and transaction safety.

Given the widespread availability of these technologies and the development team's expertise, the technical challenges of building and maintaining the Rambutan Warehouse system are minimal. The platform can be developed, deployed, and scaled with relative ease, making it technically feasible.

3.1.3 Behavioral Feasibility

Behavioral feasibility assesses whether the target users are willing to adopt and engage with the system. In the case of Rambutan Warehouse, the primary users include rambutan farmers, wholesale buyers, regular consumers, and administrators. User research conducted during the requirement-gathering phase reveals a high level of interest from these stakeholders. Farmers, in particular, expressed a need for a platform that offers direct access to buyers and a way to manage their inventory and sales more efficiently. Wholesale buyers also demonstrated enthusiasm for features such as bulk ordering, price negotiation, and delivery scheduling.

The system's user-friendly design is a key factor in its potential for adoption. Farmers and buyers are accustomed to traditional methods of trading, but the intuitive interface of the Rambutan Warehouse ensures that they can quickly adapt to the platform. Additionally, the system offers significant benefits, such as eliminating middlemen and providing transparent pricing, which further incentivizes its use. Regular consumers, who are familiar with e-commerce platforms, are likely to find the Rambutan Warehouse appealing due to its wide range of fresh and processed rambutan products.

Considering the benefits the platform offers and the positive response from its target audience, behavioral feasibility is high. Users are likely to adopt the platform willingly, driven by its simplicity, convenience, and value. Thus, the Rambutan Warehouse system is not only technically sound and financially viable but also has a strong likelihood of user engagement and satisfaction.

3.1.4 Feasibility Study Questionnaire

1. **Project Overview?**

Ans: The Rambutan Warehouse is an online platform designed to facilitate the sale and distribution of rambutan fruits and related products. It connects farmers, wholesale buyers, and regular consumers by providing a centralized digital marketplace. The platform allows farmers to list their rambutan produce, manage inventory, and track sales, while buyers can place bulk orders, negotiate prices, and schedule deliveries. It leverages modern e-commerce technologies like Django for backend development and SQLite 3 as the database engine, ensuring efficiency and scalability. The platform aims to enhance market accessibility for farmers and offer buyers a streamlined purchasing experience.

2. **To what extend the system is proposed for?**

Ans: The system is proposed as a full-scale e-commerce platform that supports real-time transactions, secure payments, and logistics management, tailored specifically for the rambutan trade.

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

Ans: The platform is designed for farmers, wholesale buyers, regular consumers, and administrators, each having different roles and functionalities in the system.

4. **Identify the users in your project?**

Ans: Users include rambutan farmers, wholesale buyers, regular consumers purchasing rambutan products, and platform administrators managing the system.

5. **Who owns the system?**

Ans: Administrator

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

Ans: The system is related to the agriculture and e-commerce industries, focusing on the sale and distribution of rambutan fruits and their processed products.

7. Details of person that you have contacted for data collection?

Ans: Data was collected from Shibu Thomas (farmer) and Tony (agriculture officer) to gain insights into the challenges and needs of the rambutan trade.

8. List the modules in your system

Ans: Key modules of the system include:

- Farmer Management Module: Allows farmers to register, list their produce, manage inventory, and monitor sales.
- Wholesale Buyer Module: Enables bulk orders, pricing negotiations, and delivery scheduling.
- Product Management Module: Manages the listing of various rambutan products, including fresh fruits and processed goods.
- User Authentication Module: Handles user registration, login, and secure authentication processes.
- Payment Processing Module: Facilitates secure and efficient transaction handling for all users.
- Administrator Module: Provides tools for system oversight, user management, and issue resolution.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	-	Intel i5
RAM	-	8 G B
Hard disk	-	S S D

3.2.2 Software Specification

Front End	-	HTML, CSS, Java Script
Back End	-	Python-Django
Database	-	SQLite 3
Client on PC	-	Windows 11
Technologies used	-	JS, HTML, J Query, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 Django

Django is a popular and powerful open-source web framework written in Python, designed to facilitate rapid development and maintainable web applications. It follows the Model-View Template (MVT) architectural pattern, which is similar to the Model-View-Controller (MVC) pattern. Django provides a structured and efficient way to build web applications, offering several key components and features.

At its core, Django includes a robust Object-Relational Mapping (ORM) system that simplifies database interactions, allowing developers to work with Python objects instead of raw SQL queries. It also includes a URL dispatcher for mapping URLs to view functions, an automatic admin interface for managing application data, and a templating engine for creating dynamic and reusable user interfaces.

Django places a strong emphasis on security, with built-in features to protect against common web vulnerabilities. It offers authentication and authorization systems, middleware support for global request and response processing, and compatibility with various databases.

The framework's scalability, extensibility, and a vibrant community of developers make it a popular choice for building web applications, from simple websites to complex, high-traffic platforms. Django's extensive documentation and ecosystem of third-party packages further enhance its appeal for web developers.

3.3.2 SQLite

SQLite is a self-contained and serverless relational database management system (RDBMS) known for its simplicity and efficiency. It stores the entire database in a single file, eliminating the need for a separate server process, which simplifies deployment. This makes SQLite an ideal choice for embedded systems, mobile applications, and small to medium-sized projects. Developers interact with the database directly through function calls, making it an embedded database well-suited for various applications, including mobile apps, desktop software, and web applications.

Despite its lightweight nature, SQLite is ACID-compliant, ensuring data integrity with support for transactions. It offers a wide range of data types, and its compatibility with multiple programming languages, including Python, C/C++, and Java, makes it accessible to a diverse developer community. SQLite is open-source, free, and known for its speed and efficiency, particularly for read-heavy workloads. While not intended for extremely high-concurrency or large-scale applications, SQLite excels in scenarios where simplicity, portability, and low resource consumption are key requirements.

It is commonly used as a local data store, cache, or embedded database within larger applications, contributing to its versatility and widespread adoption in software development.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

The initial stage of developing any engineered product or system is the design phase, which involves a creative approach. A well-crafted design plays a critical role in ensuring the successful functioning of a system. Design is defined as the process of employing various techniques and principles to define a process or system in enough detail to enable its physical realization. This involves using different methods to describe a machine or system, explaining how it operates, in sufficient detail for its creation. In software development, design is a crucial step that is always present, regardless of the development approach. System design involves creating a blueprint for building a machine or product. Careful software design is essential to ensure optimal performance and accuracy. During the design phase, the focus shifts from the user to the programmers or those working with the database. The process of creating a system typically involves two key steps: Logical Design and Physical Design.

4.2 UML DIAGRAM

UML, which stands for Unified Modeling Language, is a standardized language used for specifying, visualizing, constructing, and documenting the elements of software systems. The Object Management Group (OMG) is responsible for the creation of UML, with the initial draft of the UML 1.0 specification presented to OMG in January 1997. Unlike common programming languages such as C++, Java, or COBOL, UML is not a programming language itself. Instead, it is a graphical language that serves as a tool for creating software blueprints.
UML is a versatile and general-purpose visual modeling language that facilitates the visualization, specification, construction, and documentation of software systems. While its primary use is in modeling software systems, UML's applications are not limited to this domain. It can also be employed to represent and understand processes in various contexts, including non-software scenarios like manufacturing unit processes.
It's important to note that UML is not a programming language, but it can be utilized with tools that generate code in different programming languages based on UML diagrams. UML encompasses eight core diagrams that aid in representing various aspects of a system.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram

● Activity diagram

● State chart diagram

● Deployment diagram

● Component diagram

4.2.1 USE CASE DIAGRAM

A use case is a tool for understanding a system's requirements and organizing them, especially in the context of creating or using something like a product delivery website. These tools are represented using "use case" diagrams within the Unified Modeling Language, a standardized way of creating models for real-world things and systems.

A use case diagram consists of these main elements:

● The boundary, which delineates the system and distinguishes it from its surroundings.

● Actors, representing individuals or entities playing specific roles within the system.

● The interactions between different people or elements in specific scenarios or problems.

● The primary purpose of use case diagrams is to document a system's functional specifications. To create an effective use case diagram, certain guidelines must be followed;

● Providing clear and meaningful names for use cases and actors.

● Ensuring that the relationships and dependencies are well-defined.

● Including only necessary relationships for the diagram's clarity.

● Utilizing explanatory notes when needed to clarify essential details.

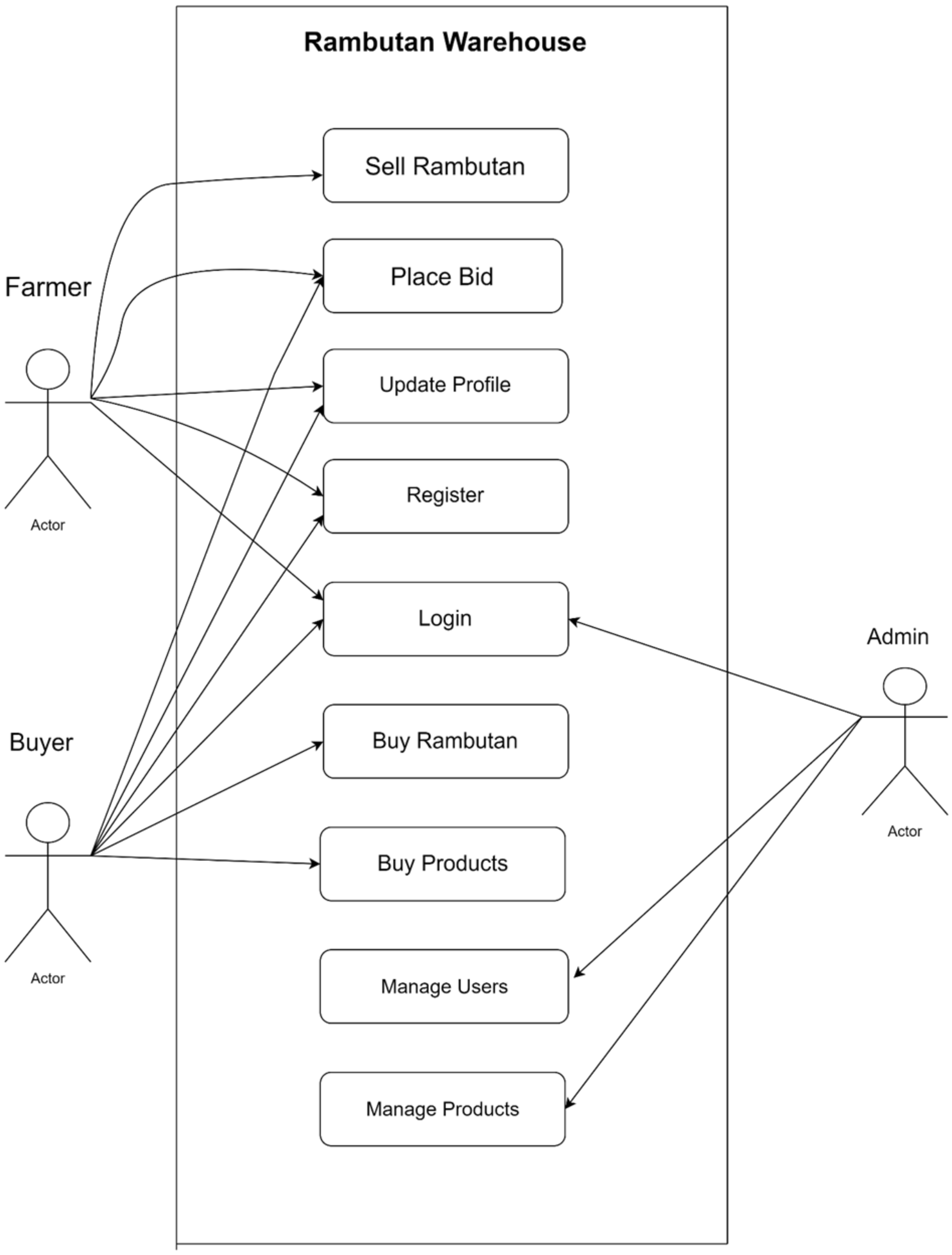


fig 4.2. 1- Usecase diagram

4.2.2 SEQUENCE DIAGRAM

A sequence diagram illustrates the specific order in which objects interact with each other, showcasing the sequential flow of events. This type of diagram is also known as event diagrams or event scenarios. Sequence diagrams serve the purpose of elucidating how various components of a system collaborate and the precise sequence in which these actions occur. These diagrams find frequent application among business professionals and software developers, aiding in the understanding and depiction of requirements for both new and existing systems.

Sequence Diagram Notations:

- Actors - Within a UML diagram, actors represent individuals who utilize the system and its components. Actors are not depicted within the UML diagram as they exist outside the system being modeled. They serve as role-players in a story, encompassing people and external entities. In a UML diagram, actors are represented by simple stick figures. It's possible to depict multiple individuals in a diagram that portrays the sequential progression of events.
- Lifelines - In a sequence diagram, each element is presented as a lifeline, with lifeline components positioned at the top of the diagram.
- Messages - Communication between objects is achieved through the exchange of messages, with messages being arranged sequentially on the lifeline. Arrows are employed to represent messages, forming the core structure of a sequence diagram.
- Guards - Within the UML, guards are employed to denote various conditions. They are used to restrict messages in the event that specific conditions are met, providing software developers with insights into the rules governing a system or process.

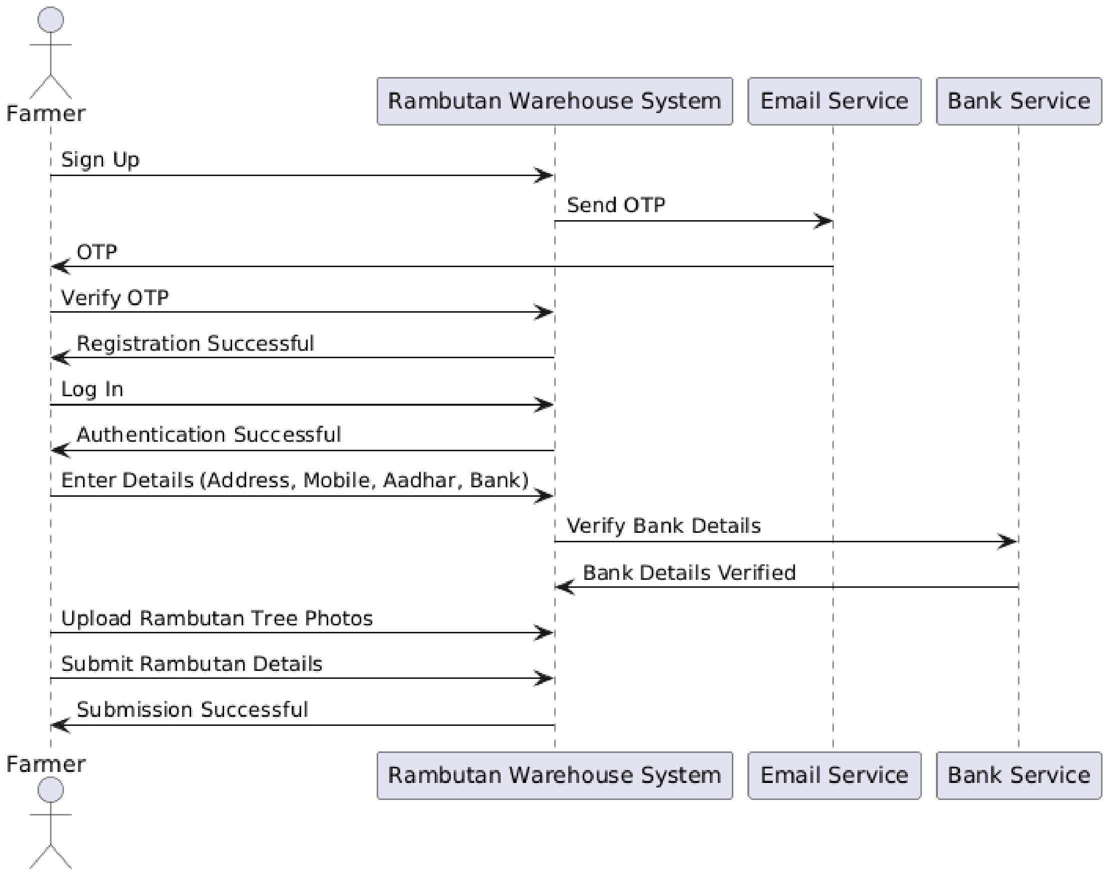


fig 4.2.2-Sequence Diagram

4.2.3 State Chart Diagram

A state machine diagram, also known as a state chart, visually represents the various states an object undergoes within a system and the sequence in which these states are traversed. It serves as a record of the system's behavior, illustrating the collaborative functioning of a group of entities, whether it's a team, a collection of students, a large assembly, or an entire organization. State machine diagrams are a valuable method for depicting the interactions of diverse components within a system, outlining how objects evolve in response to events and elucidating the diverse conditions that each entity or component can inhabit.

Notations within a state machine diagram encompass:

● Initial state: Symbolized by a black circle, this signifies the commencement of a process.

● Final state: Representing the conclusion of a process, this is denoted by a filled circle within another circle.

● Decision box: Shaped like a diamond, it aids in decision-making by considering the evaluation of a guard.

● Transition: Whenever a change in authority or state occurs due to an event, it is termed a transition. Transitions are depicted as arrows with labels indicating the triggering event.

● State box: It portrays the condition or state of an element within a group at a specific moment. These are typically represented by rectangles with rounded corners.

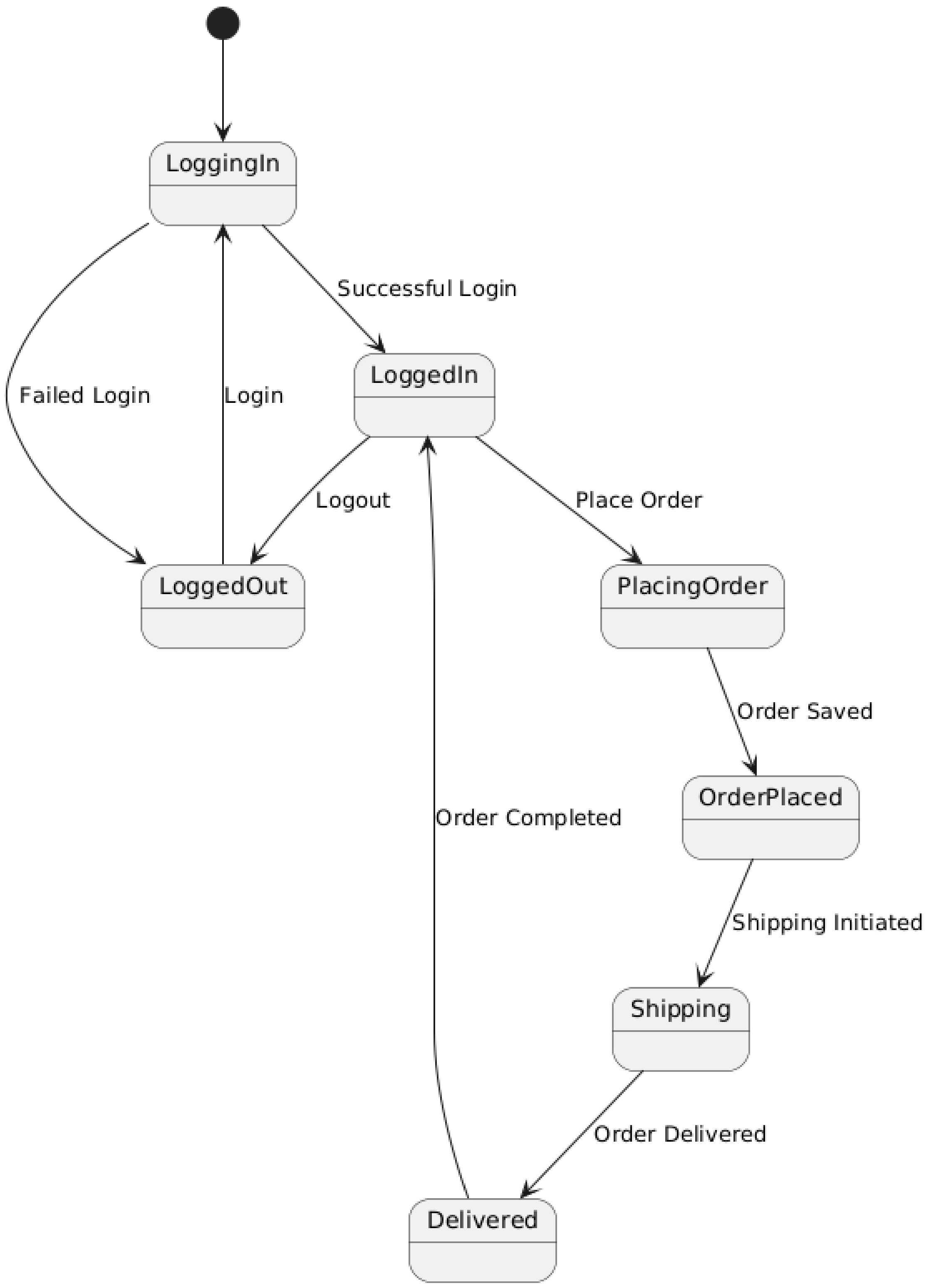


fig 4.2.3-State Chart Diagram

4.2.4Activity Diagram

An activity diagram is a visual representation of how events unfold simultaneously or sequentially. It aids in understanding the flow of activities, emphasizing the progression from one task to the next. Activity diagrams focus on the order in which tasks occur and can depict various types of flows, including sequential, parallel, and alternative paths. To facilitate these flows, activity diagrams incorporate elements such as forks and join nodes, aligning with the concept of illustrating the functioning of a system in a specific manner.

Key Components of an Activity Diagram include:
a) Activities: Activities group behaviors into one or more actions, forming a network of interconnected steps. Lines between these actions outline the step-by-step sequence of events. Actions encompass tasks, controlling elements, and resources utilized in the process.
b) Activity Partition/Swim Lane: Swim lanes are used to categorize similar tasks into rows or columns, enhancing modularity in the activity diagram. They can be arranged vertically or horizontally, though they are not mandatory for every activity diagram.

c) Forks: Fork nodes enable the simultaneous execution of different segments of a task. They represent a point where one input transforms into multiple outputs, resembling the diverse factors influencing a decision.

d) Join Nodes: Join nodes are distinct from fork nodes and employ a Logical AND operation to ensure that all incoming data flows converge to a single point, promoting synchronization.

Notations in an Activity Diagram include:
- Initial State: Depicting the beginning or first step in the process.
- Final State: Signifying the completion of all actions with no further progress.
- Decision Box: Ensuring that activities follow a specific path.
- Action Box: Representing the specific tasks or actions to be performed in the process.

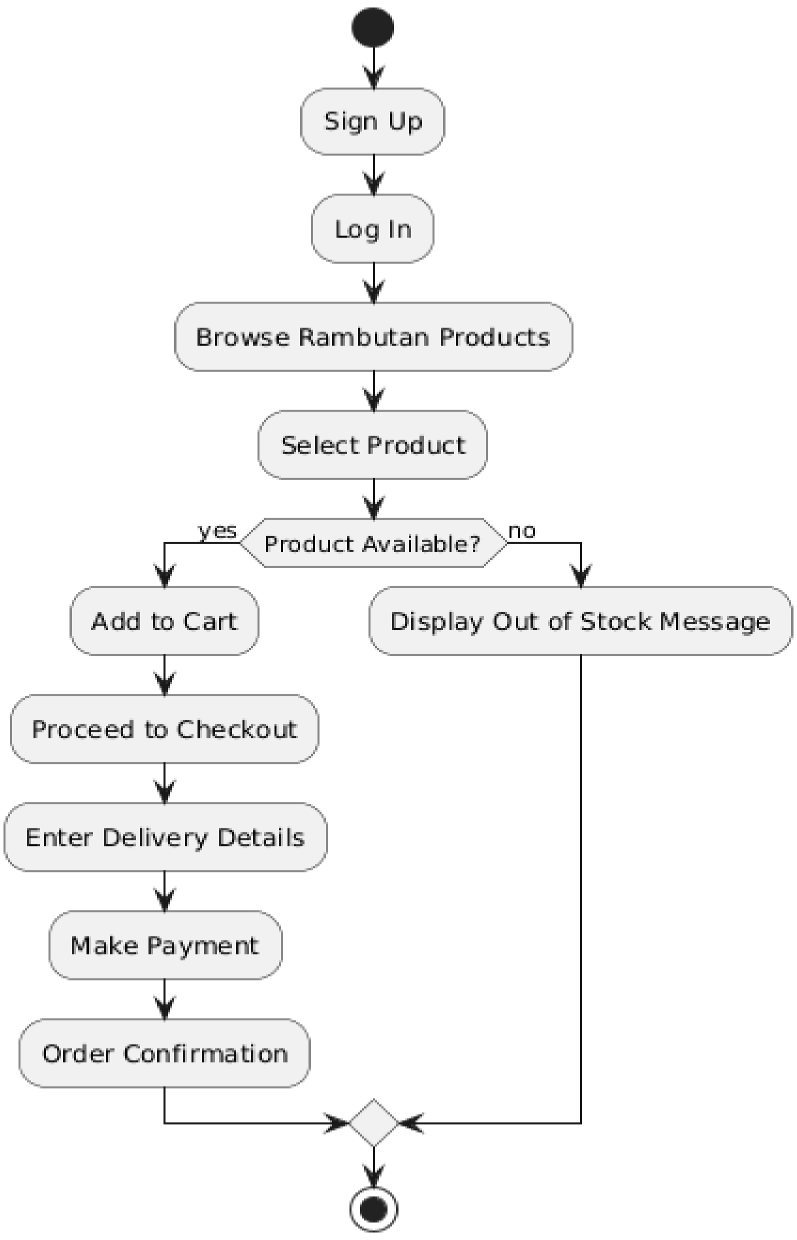


fig 4.2.4-Activity Diagram

4.2.5 Class Diagram

A class diagram serves as a static blueprint for an application, illustrating its components and their relationships when the system is in a dormant state. It provides insights into the system's structure, showcasing the various elements it comprises and how they interact. In essence, a class diagram acts as a visual guide for software development, aiding in the creation of functional applications.

Key Aspects of a Class Diagram:

● System Overview: A class diagram offers a high-level representation of a software system, presenting its constituent parts, associations, and collaborative dynamics. It serves as an organizational framework for elements such as names, attributes, and methods, simplifying the software development process.

● Structural Visualization: The class diagram is a structural diagram that combines classes, associations, and constraints to define the system's architecture.

Components of a Class Diagram:

The class diagram comprises three primary sections:

● Upper Section: This top segment features the class name, representing a group of objects that share common attributes, behaviors, and roles. Guidelines for displaying groups of objects include capitalizing the initial letter of the class name, positioning it in the center, using bold lettering, and employing slanted writing style for abstract class titles.

● Middle Section: In this part, the class's attributes are detailed, including their visibility indicators, denoted as public (+), private (-), protected (#), or package (~).

● Lower Section: The lower section elaborates on the class's methods or operations, presented in a list format with each method on a separate line. It outlines how the class interacts with data.

In UML, relationships within a class diagram fall into three categories:

▪ Dependency: Signifying the influence of one element's changes on another.

▪ Generalization: Representing a hierarchical relationship where one class acts as a parent, and another serves as its child.

▪ Association: Indicating connections between elements.

▪ Multiplicity: Defining constraints on the number of instances allowed to possess specific characteristics, with one being the default value when not specified.

▪ Aggregation: An aggregation is a group that is a part of a relationship called association.

▪ Composition: "Composition" is like a smaller part of "aggregation.". This describes how a parent and child need each other, so if one is taken away, the other won't work anymore.

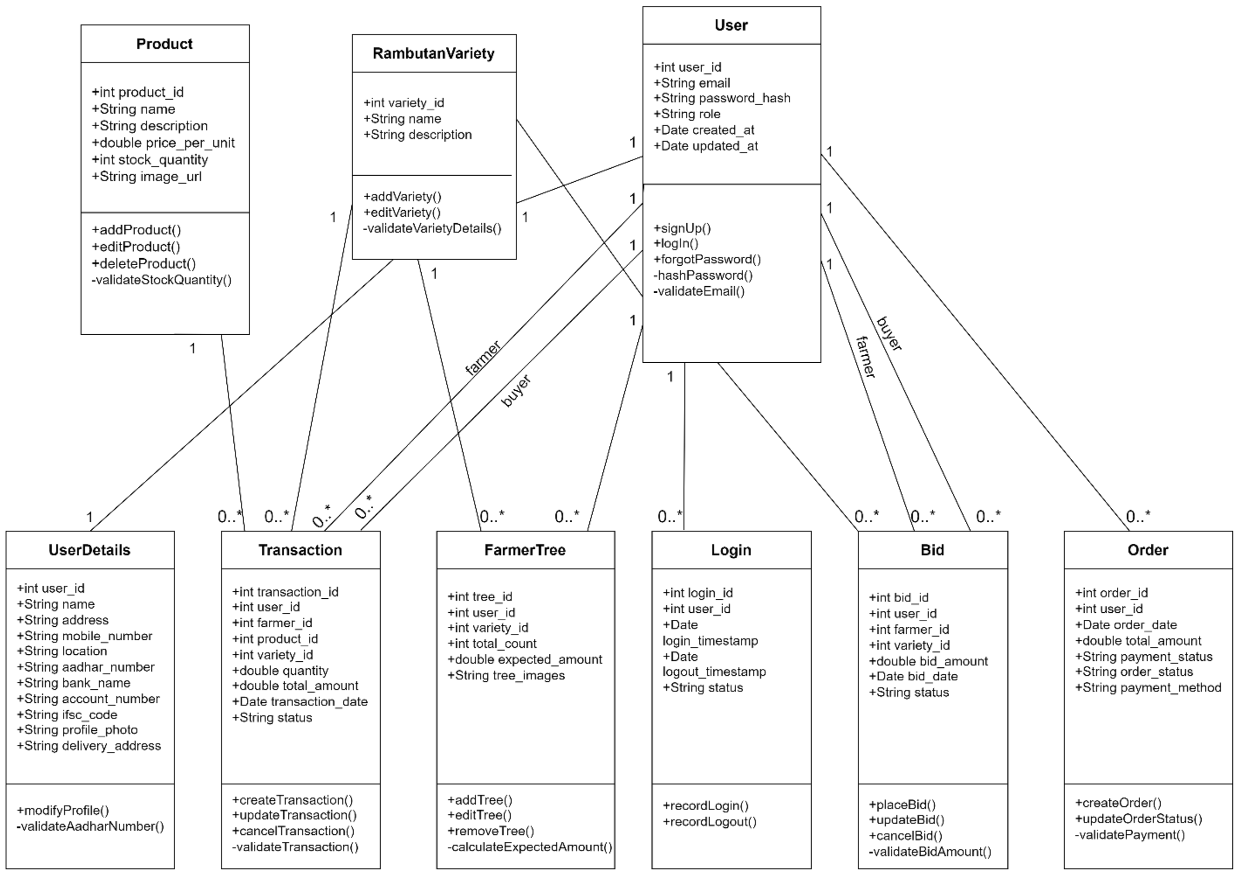


fig 4.2.5-Class Diagram

4.2.6 Object Diagram

Object diagrams are derived from class diagrams and rely on them to provide a visual representation. They offer an illustration of a collection of objects related to a particular class. Object diagrams provide a snapshot of objects in an object-oriented system at a specific point in time.

Object diagrams and class diagrams share similarities, but they also have distinctions. Class diagrams are more generalized and do not portray specific objects. This abstraction in class diagrams simplifies the comprehension of a system's functionality and structure.

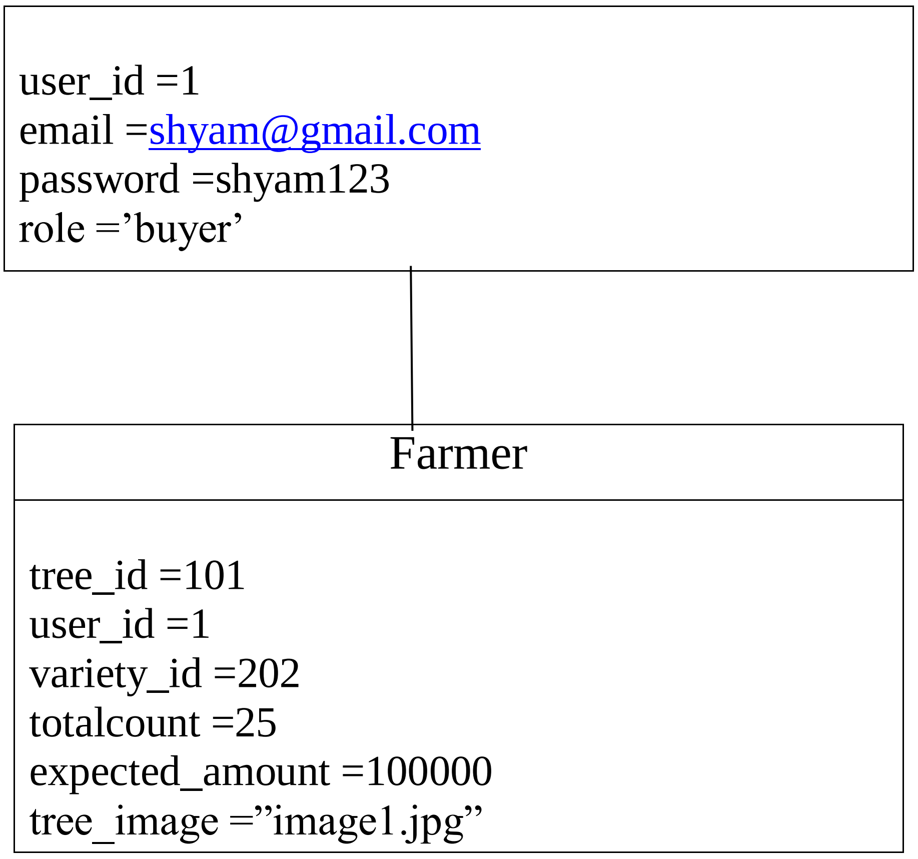


fig 4.2.6-Object Diagram

4.2.7 Component Diagram

A component diagram serves the purpose of breaking down a complex system that utilizes objects into more manageable segments. It offers a visual representation of the system, showcasing its internal components such as programs, documents, and tools within the nodes. This diagram elucidates the connections and organization of elements within a system, resulting in the creation of a usable system.

In the context of a component diagram, a component refers to a system part that can be modified and operates independently. It retains the secrecy of its internal operations and requires a specific method to execute a task, resembling a concealed box that functions only when operated correctly. Notation for a Component Diagram includes:

- A component
- A node

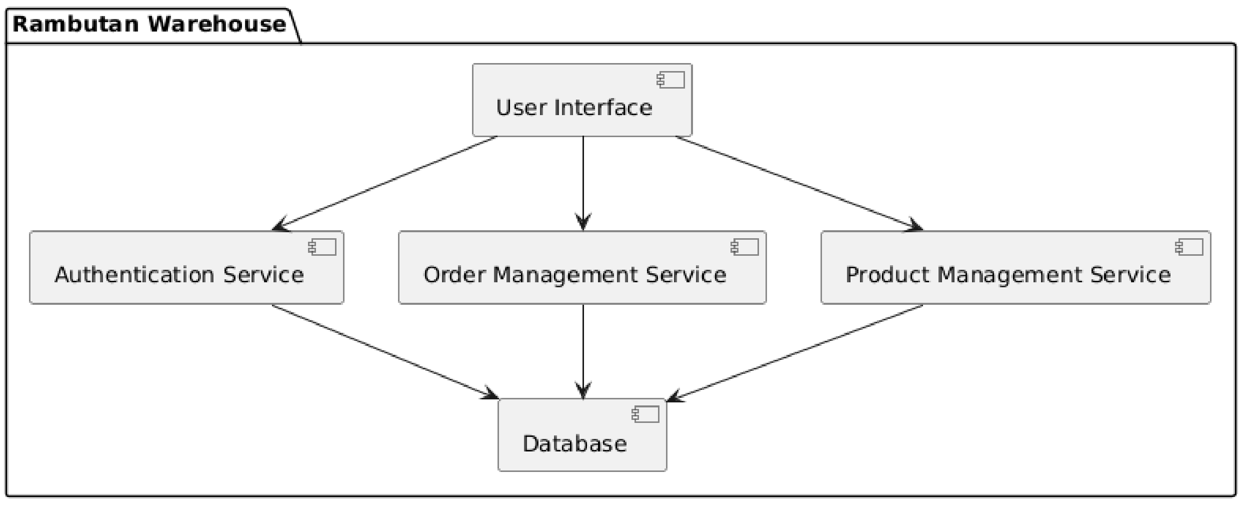


fig 4.2.7-Component Diagram

4.2.8 Deployment Diagram

A deployment diagram provides a visual representation of how software is positioned on physical computers or servers. It depicts the static view of the system, emphasizing the arrangement of nodes and their connections.

This type of diagram delves into the process of placing programs on computers, elucidating how software is constructed to align with the physical computer system.

Notations in a Deployment Diagram include:

- A component
- An artifact
- An interface
- A node

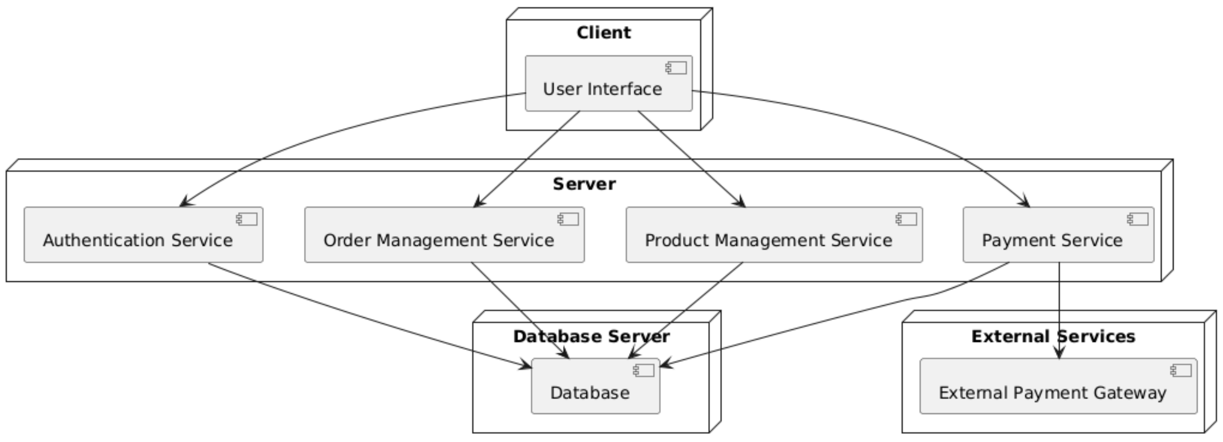


fig 4.2.8-Deployment Diagram

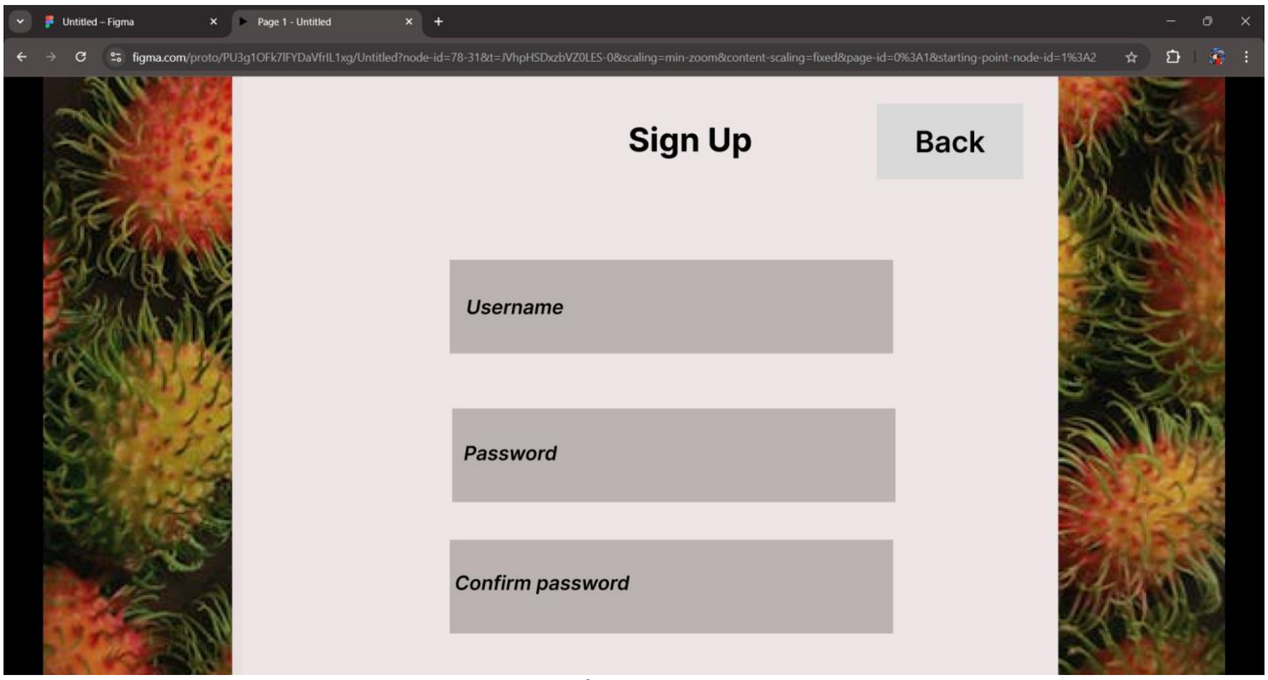
4.2.9 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is a type of behavior diagram in Unified Modeling Language (UML) that shows the interactions between objects and their messages in a system. It is used to visualize the interactions and communication patterns between objects in a system.

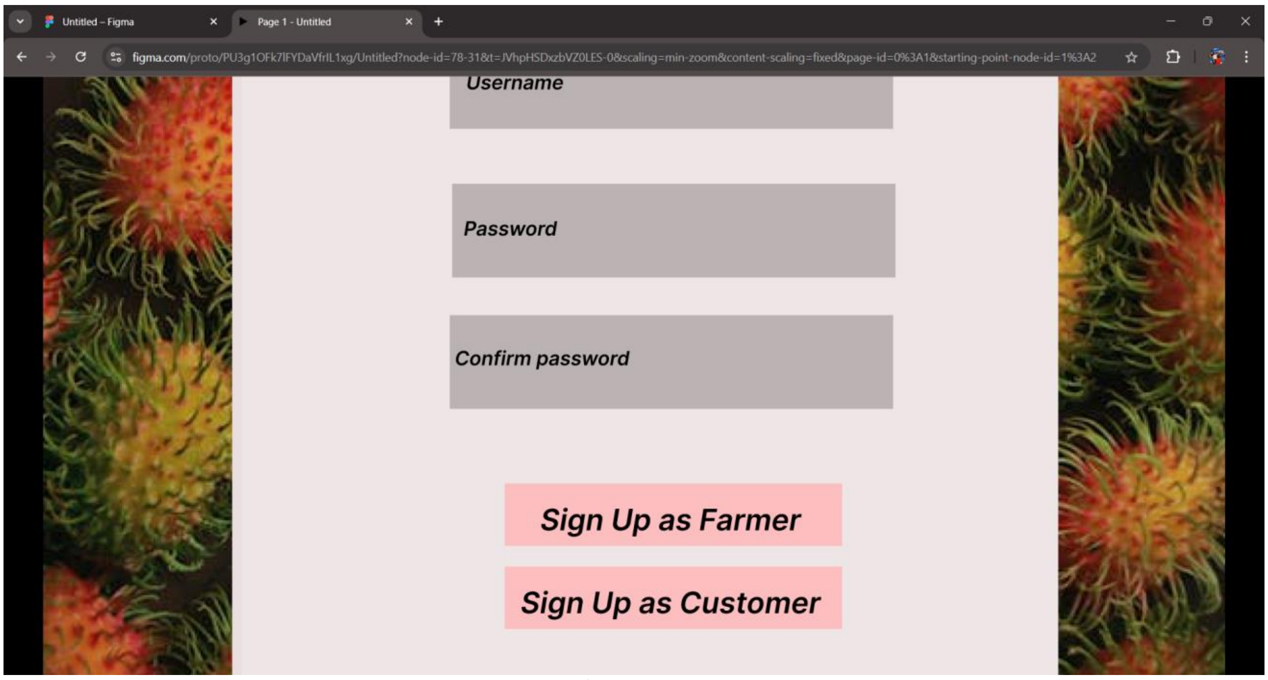
In a collaboration diagram, the objects that participate in the interaction are represented by rectangles, and the messages exchanged between them are represented by arrows. The sequence of messages are shown from top to bottom, with the initial message at the top and the final message at the bottom. Collaboration diagrams can be used to model any system that involves multiple objects and interactions between them, such as a software system, a business process, or a physical system.
They are useful for identifying the objects that participate in the interaction, and for understanding the sequence of messages exchanged between them. Collaboration diagrams can be used during the design phase of software development to help in designing the interactions between objects and to identify potential problems or conflicts in the interaction. They are also useful for documentation and communication purposes, as well as for testing and validation of the system's interaction. Collaboration diagrams can be used in conjunction with other UML diagrams, such as use case diagrams and sequence diagrams, to provide a complete view of the system's behavior.

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Registration page

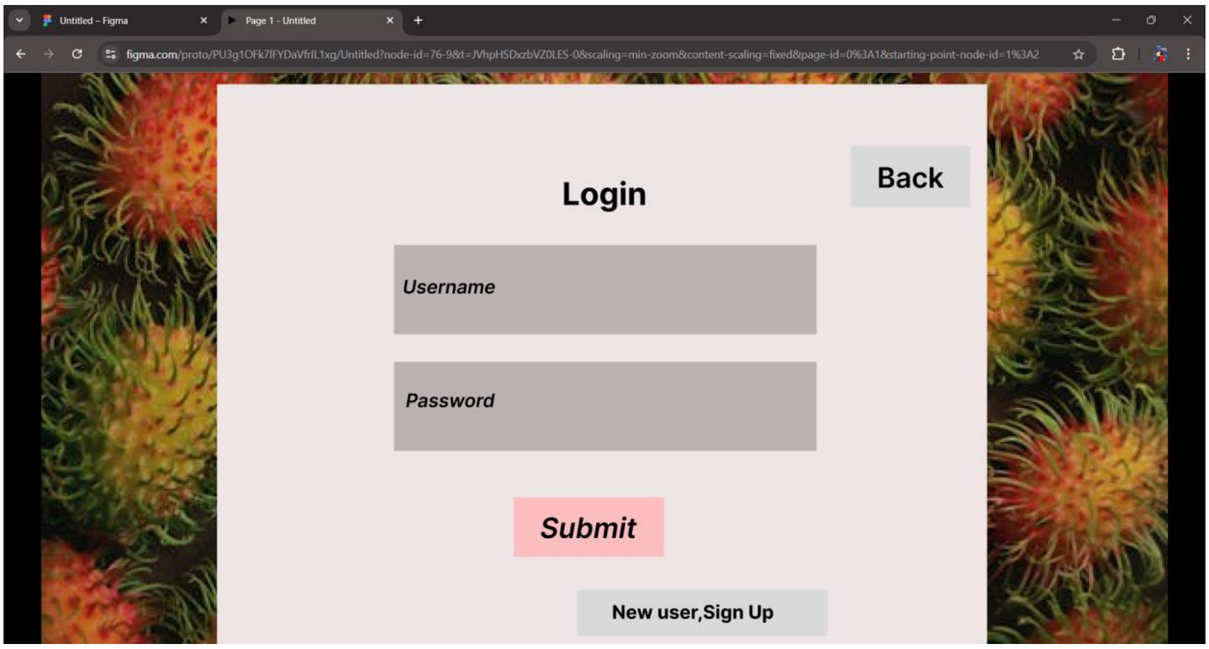

Untitled – Figma
Page 1 - Untitled
figma.com/proto/PU3g1OFk7lFYDaVfrlL1xg/Untitled?node-id=78-31&t=JVhpHSDxzbVZ0LES-0&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A2
Sign Up
Back
Username
Password
Confirm password

form 4.3.1-signup


Untitled – Figma
Page 1 - Untitled
figma.com/proto/PU3g1OFk7lFYDaVfrlL1xg/Untitled?node-id=78-31&t=JVhpHSDxzbVZ0LES-0&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A2
Username
Password
Confirm password
Sign Up as Farmer
Sign Up as Customer

form 4.3.2-signup

Form Name: Login page



Login

Back

Username

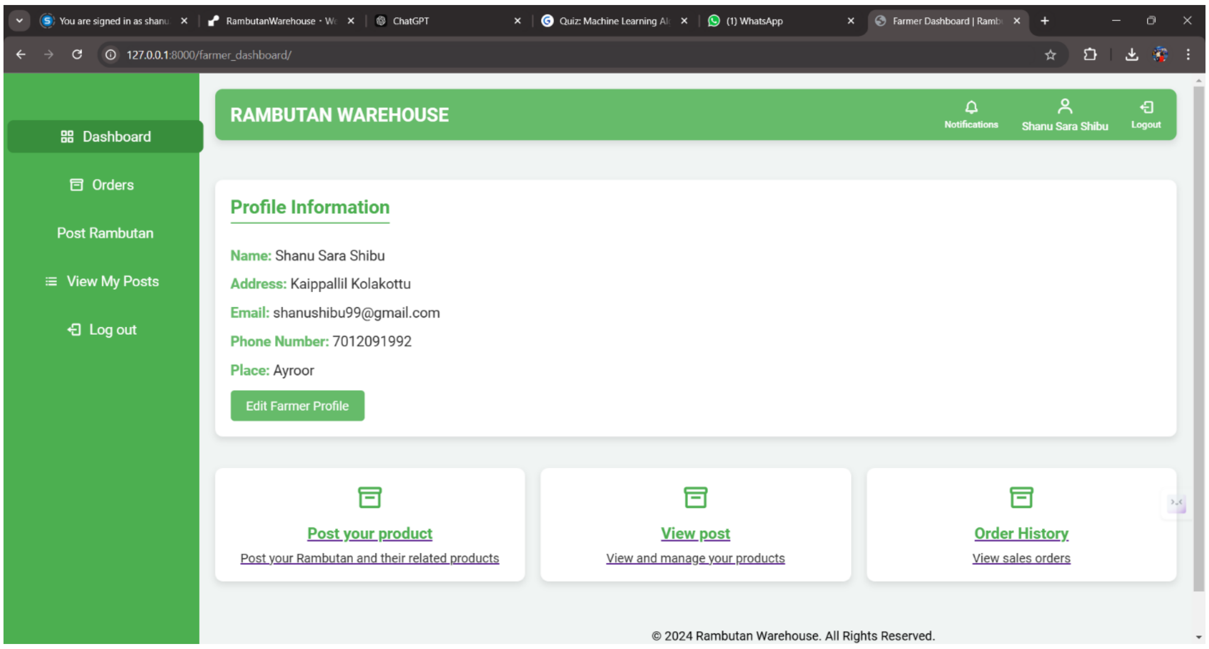
Password

Submit

New user,Sign Up

form 4.3.3-login

Form Name: farmer_dashboard.html



RAMBUTAN WAREHOUSE

Dashboard

Orders

Post Rambutan

View My Posts

Log out

Notifications Shanu Sara Shibu Logout

Profile Information

Name: Shanu Sara Shibu

Address: Kaippallil Kolakottu

Email: shanushibu99@gmail.com

Phone Number: 7012091992

Place: Ayroor

Edit Farmer Profile

Post your product
Post your Rambutan and their related products

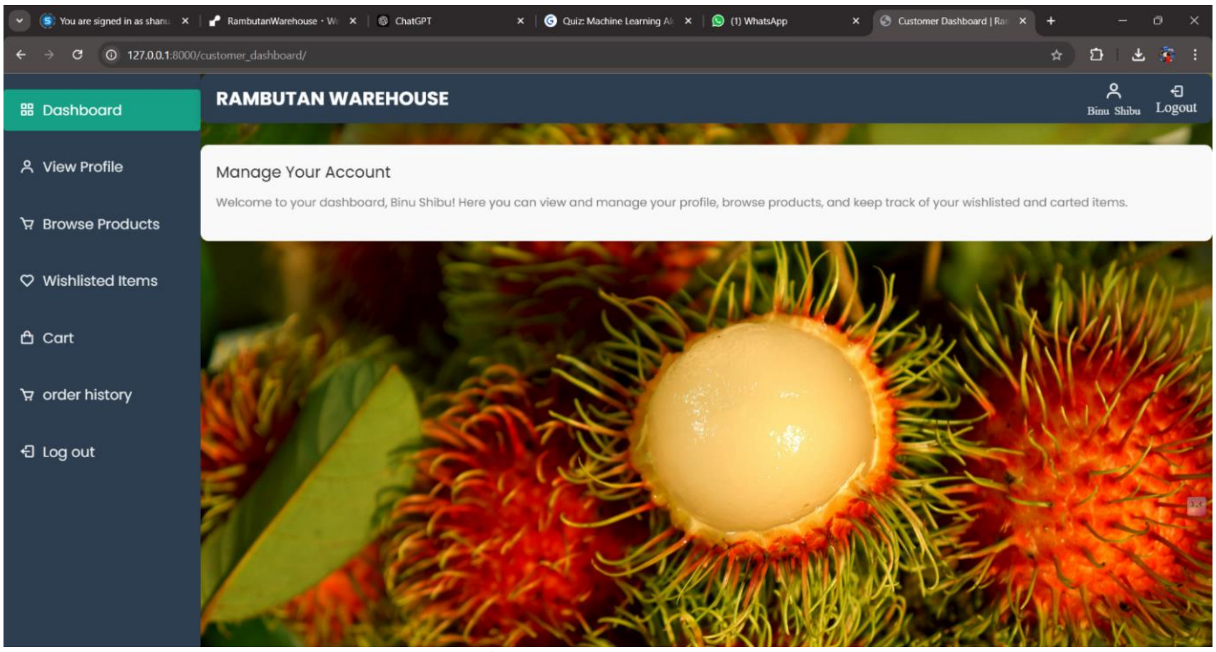
View post
View and manage your products

Order History
View sales orders

© 2024 Rambutan Warehouse. All Rights Reserved.

form 4.3.4-farmer dashboard

Form Name: customer_dashboard.html



Manage Your Account

Welcome to your dashboard, Binu Shibu! Here you can view and manage your profile, browse products, and keep track of your wishlisted and carted items.

form 4.3.5-customer dashboard

4.4 DATABASE DESIGN

A database is an organized collection of structured data that is stored and managed on a computer system. It is designed to efficiently store, retrieve, and manage large amounts of data, making it easy to access and manipulate information. Databases can range from small, single-user systems to large, complex systems that are used by millions of users simultaneously.

Database design is the process of creating a structured representation of data that allows for efficient storage, retrieval, and manipulation of information. It involves identifying the data requirements of an organization, modeling these requirements using a database schema, and implementing the schema using a database management system (DBMS).

The primary goal of database design is to ensure that data is organized in a way that supports the business processes and applications that will be using it. This involves understanding the relationships between different types of data, defining tables and columns to store this data, and establishing rules for how the data can be accessed and manipulated.

Good database design also takes into account factors such as scalability, performance, security, and data integrity. It requires a thorough understanding of the requirements of the organization and the various stakeholders who will be interacting with the data.

Effective database design can result in significant benefits for an organization, such as improved data quality, increased efficiency, better decision-making, and reduced costs.

4.4.1 Relational Database Management System (RDBMS)

A database is a system designed to store and facilitate the retrieval and utilization of information. The integrity and security of the data within a database are of paramount importance. Creating a database involves two essential steps. Initially, it is crucial to understand the user's requirements and establish a database structure that aligns with their needs. The first step entails devising an organizational plan for the information, which is not reliant on any particular computer program. Subsequently, this plan is employed to craft a design tailored to the specific computer program that will be employed to construct the database system. This phase is centered on determining how data will be stored within the chosen computer program.

Some popular RDBMS software include MySQL, Oracle Database, Microsoft SQL Server, and PostgreSQL. These systems are commonly used in various industries, including finance, healthcare, education, and e-commerce, to store and manage large amounts of data efficiently and securely.

Key aspects of database development include:

● Data Integrity: Ensuring the accuracy and consistency of data stored within the database.

● Data Independence: The ability to modify the data storage structure without affecting the application's access to the data.

4.4.2 Normalization

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multistep process that sets the data into tabular form and removes the duplicated data from the relational tables. Normalization is a process of decomposing the relations into relations with fewer attributes. Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.

Data modification anomalies can be categorized into three types:

• Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.

• Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

• Updation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

1) First Normal Form: A relation is in 1NF if it contains an atomic value.

2) Second Normal Form: A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.

3) Third Normal Form: A relation will be in 3NF if it is in 2NF and no transition dependency exists.

4) Boyce Codd Normal Form: A stronger definition of 3NF is known as Boyce Codd's normal form.

5) Fourth Normal Form: A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.

6)Fifth Normal Form: A relation will be in 5NF if it is in 4NF and does not contain any join dependency, joining should be lossless.

4.4.3 Sanitization

Django incorporates various in-built mechanisms for data sanitization. To begin with, it provides a diverse range of field types that come with automatic validation and sanitization capabilities. For instance, the CharField performs automatic validation and cleaning of text input, while the EmailField ensures that email addresses adhere to the correct format. These field types serve as protective measures to prevent the storage of malicious or invalid data in the database. Moreover, Django strongly advocates for the utilization of form validation to sanitize user input. Django's forms are equipped with predefined validation methods and validators that can be applied to form fields. These validators execute a range of checks and cleansing operations, such as confirming that numerical values fall within specified ranges or verifying that uploaded files adhere to specific formats. These combined features in Django promote data integrity and security, making it a reliable choice for web application development.

4.4.4 Indexing

The index keeps track of a certain piece of information or group of information, arranged in order of its value. Arranging the index entries helps find things quickly and easily that match exactly or are within a certain range. Indexes make it easy to find information in a database without having to search through every record every time the database is used. An index is like a roadmap for finding information in a database. It helps you look up data quickly and also makes it easy to find records that are in a certain order. It can be based on one or more columns in the table.

4.5 TABLE DESIGN

1.warehouse_registeruser

Primary key: **id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Registered user id
2.	username	EmailField	unique=True	User's email for login
3.	name	CharField (50)	Not Null	Name of the user
4.	password	CharField (50)	Not Null	Hashed password
5.	role	ENUM ('farmer', 'customer')	Not Null	User role type
6.	contact	CharField (250)	Not Null	User's contact information
7.	address	CharField (250)	Not Null	User's address
8.	created_at	TIMESTAMP	Not Null	Account creation timestamp

Table 4.5.1

2. warehouse_farmerdetails

Primary key: **id**
Foreign key: **user_id** references table **warehouse_registeruser**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Registered farmer's user id
2.	user_id	INT	Foreign key	Foreign Key to Users table
3.	address	TextField	Not Null	Farmer's address
4.	mobile_number	CharField (10)	Not Null	Farmer's mobile number
5.	location	CharField (255)	Not Null	Farmer's location
6.	aadhar_number	CharField (12)	Not Null	Farmer's Aadhar number
7.	bank_name	CharField (255)	Not Null	Farmer's bank name
8.	account_number	CharField (18)	Not Null	Farmer's bank account number
9.	ifsc_code	CharField (11)	Not Null	Farmer's bank IFSC code

Table 4.5.2

3.warehouse_customerdetails

Primary key: **id**
Foreign key: **user_id** references table **warehouse_registeruser**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Registered cutomer's user id
2.	user_id	INT	Foreign key	Foreign Key to Users table
3.	address	TextField	Not Null	Customer's address
4.	mobile_number	CharField (10)	Not Null	Customer's mobile number
5.	location	CharField (255)	Not Null	Customer's location

Table 4.5.3

4.warehouse_rambutanpost

Primary key: **id**
Foreign key: **id** references table **warehouse_farmerdetails**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Registered farmer's post id
2.	id	INT	Foreign key	Link to the farmer details table
3.	name	CharField (255)	Not Null	Name of the rambutan post
4.	variety	CharField (100)	Not Null	Variety of rambutan
5.	quantity	INT	Not Null	Total quantity available
6.	quantity_left	INT	Not Null	Quantity left after sales
7.	price_per_kg	DecimalField (10, 2)	Not Null	Price per kg of rambutan
8.	image	ImageField	Not Null	Image of rambutan
9.	description	TextField	Not Null	Description of the rambutan post
10.	created_at	DateTimeField	Not Null	Post creation date
11.	is_available	BooleanField	Not Null	Availability status of the rambutan

Table 4.5.4

5.warehouse_wishlist

Primary key: **id**
Foreign key: **id** references table **warehouse_registeruser**
Foreign key: **id** references table **warehouse_rambutanpost**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Wishlist id
2.	id	INT	Foreign key	Link to the register user table
3.	id	INT	Foreign key	Link to the rambutan post
4.	added_at	DateTimeField	auto_now_add =True	Timestamp when the post was added to the wishlist

Table 4.5.5

6.warehouse_cart

Primary key: **id**
Foreign key: **id** references table **warehouse_registeruser**
Foreign key: **id** references table **warehouse_rambutanpost**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Cart id
2.	id	INT	Foreign key	Link to the register user table
3.	id	INT	Foreign key	Link to the rambutan post
4.	quantity	INT	Not Null	Quantity added to the cart
5.	price	DecimalField (10, 2)	Not Null	Price of the rambutan post
6.	total_price	DecimalField (10, 2)	Not Null	Total price calculated based on quantity
7.	added_at	DateTimeField	auto_now_add =True	Timestamp when the item was added to the cart

Table 4.5.6

7.warehouse_billingdetail

Primary key: **id**
Foreign key: **id** references table **warehouse_registeruser**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Billing details id
2.	id	INT	Foreign key	Link to the register user table
3.	first_name	CharField (100)	Not Null	First name of the user
4.	last_name	CharField (100)	Not Null	Last name of the user
5.	country	CharField (100)	Not Null	Country of the user
6.	street_address	CharField (255)	Not Null	Street address of the user
7.	city	CharField (100)	Not Null	City of the user
8.	postcode	CharField (20)	Not Null	Postcode of the user
9.	phone	CharField (20)	Not Null	Phone number of the user

10.	email	EmailField	Not Null	Email of the user
11.	created_at	DateTimeField	auto_now_add=True	Timestamp of billing detail creation

Table 4.5.7

8.warehouse_order

Primary key: **id**
Foreign key: **id** references table **warehouse_registeruser**
Foreign key: **id** references table **warehouse_billingdetail**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Order id
2.	id	INT	Foreign key	Link to the register user table
3.	id	INT	Foreign key	Link to the billing details
4.	total_amount	DecimalField (10, 2)	Not Null	Total amount for the order
5.	payment_method	CharField (50)	Not Null	Payment method used
6.	created_at	DateTimeField	auto_now_add =True	Timestamp of order creation

Table 4.5.8

9.warehouse_ordernotification

Primary key: **id**
Foreign key: **id** references table **warehouse_registeruser**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	id	INT	Primary key	Notification id
2.	id	INT	Foreign key	Link to the register user table
3.	order_number	CharField (50)	Not Null	Order number related to the notification
4.	item_name	CharField (255)	Not Null	Name of the item
5.	quantity	INT	Not Null	Quantity of the item ordered
6.	price	DecimalField (10, 2)	Not Null	Price of the item
7.	created_at	DateTimeField	auto_now_add=True	Timestamp of notification creation

Table 4.5.9

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

Software testing is an essential procedure employed to verify if a computer program functions as intended. It is conducted to ensure that the software performs its designated tasks accurately and complies with the prescribed requirements and standards. Validation is the process of inspecting and assessing software to confirm its adherence to the specified criteria. Software testing is a method for assessing the performance of a program, often in conjunction with techniques like code inspection and program walkthroughs. Validation ensures that the software aligns with the user's expectations and requirements.

Several principles and objectives guide the process of software testing, including:

- Testing is the practice of executing a program with the primary aim of identifying errors.
- An effective test case is one that has a high likelihood of uncovering previously undiscovered errors.
- A successful test is one that exposes previously undiscovered errors.

When a test case operates effectively and accomplishes its objectives, it can detect flaws within the software. This demonstrates that the computer program is functioning as intended and is performing well. The process of evaluating a computer program encompasses three primary aspects:

- Correctness assessment
- Evaluation of implementation efficiency
- Examination of computational complexity

5.2 TEST PLAN

A test plan serves as a comprehensive set of instructions for conducting various types of tests. It can be likened to a map that outlines the steps to follow when evaluating a computer program. Software developers create instructions for both using and organizing the necessary information for the program's proper functionality. They ensure that each component of the program performs its intended functions. To ensure the thorough testing of the software, a group known as the ITG (Information Technology Group) is responsible for verifying its functionality, instead of relying solely on the software's creators for testing.

The objectives of testing should be clearly defined and measurable. A well-structured test plan should encompass details about the frequency of failures, the associated repair costs, the occurrence rate of issues, and the time required for complete testing. The levels of testing typically include:

- Unit testing
- Integration testing
- Data validation testing
- Output testing

5.2.1 Unit Testing

Unit testing checks the smallest part of a software design - the software component or module. Testing important control paths within a module using the design guide to find errors. This means how difficult the tests are for each small part of a program and what parts of the program haven't been tested yet. Unit testing is a type of testing that looks at how the code works inside and can be done at the same time for different parts of the program.

Before starting any other test, we need to check if the data flows correctly between different parts of the computer program. If the information doesn't move in and out correctly, all other checks are pointless. When designing something, it's important to think about what could go wrong and make a plan for how to deal with those problems. This can mean redirecting the process or stopping it completely.

The Sell-Soft System was tested by looking at each part by itself and trying different tests on it. Some mistakes in the design of the modules were discovered and then fixed. After writing the instructions for different parts, each part is checked and tried out separately. We got rid of extra code and made sure everything works the way it should.

5.2.2 Integration Testing

Integration testing is a critical process in software development that involves constructing a program while simultaneously identifying errors in the interaction between different program components. The primary objective is to utilize tested individual parts and assemble them into a program according to the initial plan. This comprehensive testing approach assesses the entire program to ensure its proper and correct functionality.

As issues are identified and resolved during integration testing, it's not uncommon for new problems to surface, leading to an ongoing cycle of testing and refinement. After each individual component of the system is thoroughly examined, these components are integrated to ensure they function harmoniously. Additionally, efforts are made to standardize all programs to ensure uniformity rather than having disparate versions.

5.2.3 Validation Testing or System Testing

The final phase of testing involves a comprehensive examination of the entire system to ensure the correct interaction of various components, including different types of instructions and building blocks. This testing approach is referred to as Black Box testing or System testing.

Black Box testing is a method employed to determine if the software functions as intended. It assists software engineers in identifying all program issues by employing diverse input types. Black Box testing encompasses the assessment of errors in functions, interfaces, data access, performance, as well as initialization and termination processes. It is a vital technique to verify that the software meets its intended requirements and performs its functions correctly.

5.2.4 Output Testing or User Acceptance Testing

System testing is conducted to assess user satisfaction and alignment with the company's requirements. During the development or update of a computer program, it's essential to maintain a connection with the end-users. This connection is established through the following elements:

- Input Screen Designs.
- Output Screen Designs.

To perform system testing, various types of data are utilized. The preparation of test data plays a crucial role in this phase. Once the test data is gathered, it is used to evaluate the system under investigation. When issues are identified during this testing, they are addressed by following established procedures. Records of these corrections are maintained for future reference and improvement. This process ensures that the system functions effectively and meets the needs of both users and the organization.

5.2.5 Automation Testing

Automated testing is a method employed to verify that software functions correctly and complies with established standards before it is put into official use. This type of testing relies on written instructions that are executed by testing tools. Specifically, UI automation testing involves the use of specialized tools to automate the testing process. Instead of relying on manual interactions where individuals click through the application to ensure its proper functioning, scripts are created to automate these tests for various scenarios. Automating testing is particularly valuable when it is necessary to conduct the same test across multiple computers simultaneously, streamlining the testing process and ensuring consistency.

5.2.6 Selenium Testing

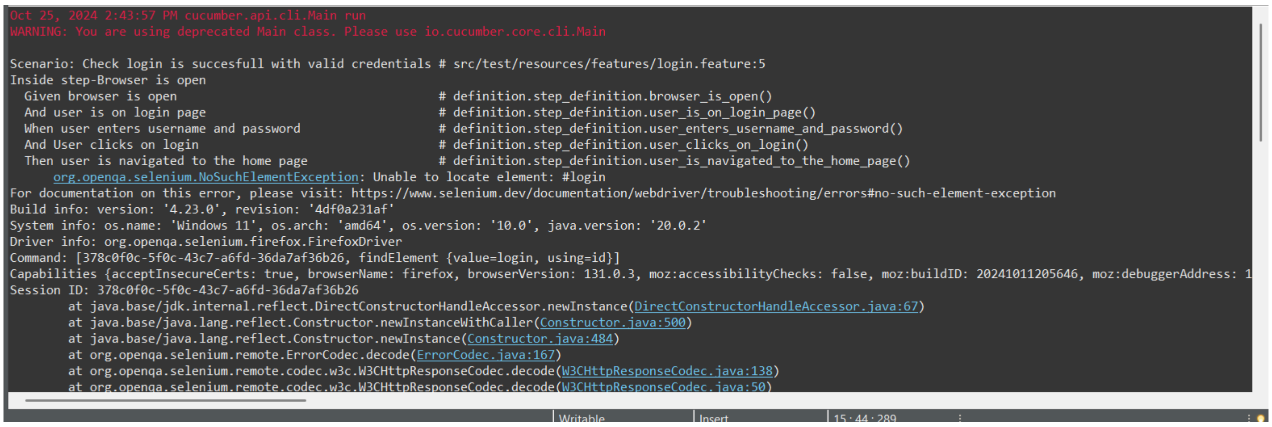
Selenium is a valuable and free tool designed for automating website testing. It plays a crucial role for web developers as it simplifies the testing process. Selenium automation testing refers to the practice of using Selenium for this purpose. Selenium isn't just a single tool; it's a collection of tools, each serving distinct functions in the realm of automation testing. Manual testing is a necessary aspect of application development, but it can be monotonous and repetitive. To alleviate these challenges, Jason Huggins, an employee at Thoughtworks, devised a method for automating testing procedures, replacing manual tasks. He initially created a tool named the JavaScriptTestRunner to facilitate automated website testing, and in 2004, it was rebranded as Selenium.

Test Case 1-Login

Code

```
package definition;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class step_definition {
 WebDriver driver = null;
@Given("browser is open")
public void browser_is_open() {
System.out.println("Inside step-Browser is open");
System.setProperty("webdriver.gecko.marionette","E:\\Program
Files\\Java\\eclipse1\\987\\src\\test\\resources\\drivers\\geckodriver.exe");
driver = new FirefoxDriver();
driver.manage().window().maximize();
}
@And("user is on login page")
public void user_is_on_login_page() throws Exception {
driver.navigate().to("http://127.0.0.1:8000/login/");
Thread.sleep(2000);
}
 @When("user enters username and password")
public void user_enters_username_and_password() throws Throwable{
driver.findElement(By.id("email")).sendKeys("shanushibu99@gmail.com");
 driver.findElement(By.id("password")).sendKeys("Shanu@123");
}
 @When("User clicks on login")
 public void user_clicks_on_login() {
 driver.findElement(By.id("login")).click(); }
@Then("user is navigated to the home page")
public void user_is_navigated_to_the_home_page() throws Exception {
driver.findElement(By.id("login")).isDisplayed();
Boolean isLogoutDisplayed = driver.findElement(By.id("logout")).isDisplayed();
 if (isLogoutDisplayed) {
System.out.println("Login successful and user is on the home page");
} else {
System.out.println("Login failed or not navigated to the home page");
}
Thread.sleep(2000);
driver.close();
driver.quit();
}
}
```

Screenshot:



```
Oct 25, 2024 2:43:57 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Check login is succesfull with valid credentials # src/test/resources/features/login.feature:5
Inside step-Browser is open
  Given browser is open                                 # definition.step_definition.browser_is_open()
  And user is on login page                             # definition.step_definition.user_is_on_login_page()
  When user enters username and password                # definition.step_definition.user_enters_username_and_password()
  And User clicks on login                              # definition.step_definition.user_clicks_on_login()
  Then user is navigated to the home page               # definition.step_definition.user_is_navigated_to_the_home_page()
      org.openqa.selenium.NoSuchElementException: Unable to locate element: #login
For documentation on this error, please visit: https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#no-such-element-exception
Build info: version: '4.23.0', revision: '4df0a231af'
System info: os.name: 'Windows 11', os.arch: 'amd64', os.version: '10.0', java.version: '20.0.2'
Driver info: org.openqa.selenium.firefox.FirefoxDriver
Command: [378c0f0c-5f0c-43c7-a6fd-36da7af36b26, findElement {value=login, using=id}]
Capabilities {acceptInsecureCerts: true, browserName: firefox, browserVersion: 131.0.3, moz:accessibilityChecks: false, moz:buildID: 20241011205646, moz:debuggerAddress: 1
Session ID: 378c0f0c-5f0c-43c7-a6fd-36da7af36b26
	at java.base/jdk.internal.reflect.DirectConstructorHandleAccessor.newInstance(DirectConstructorHandleAccessor.java:67)
	at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:500)
	at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:484)
	at org.openqa.selenium.remote.ErrorCodec.decode(ErrorCodec.java:167)
	at org.openqa.selenium.remote.codec.w3c.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.java:138)
	at org.openqa.selenium.remote.codec.w3c.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.java:50)
```

Writable | Insert | 15 : 44 : 289

Test Report

Test Case 1

Project Name: Rambutan Warehouse					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Shanu**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 23/10/2024**		
Module Name: Login Module			**Test Executed By: Mr. Ajith G. S**		
Test Title: User Login			**Test Execution Date: 23/10/2024**		
Description: Verify user login with valid email/password					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	Pass
2	Enter valid email	shanushibu99@gmail.com	User should be able to login	User logs in	Pass
3	Enter valid password	Shanu@123			
4	Click on login button				
5	Verify navigation to dashboard page		Dashboard page should be displayed	Dashboard page should be displayed	Pass
Post-Condition: User has valid email and password credentials					

Test Case 2: Post Rambutan

Code

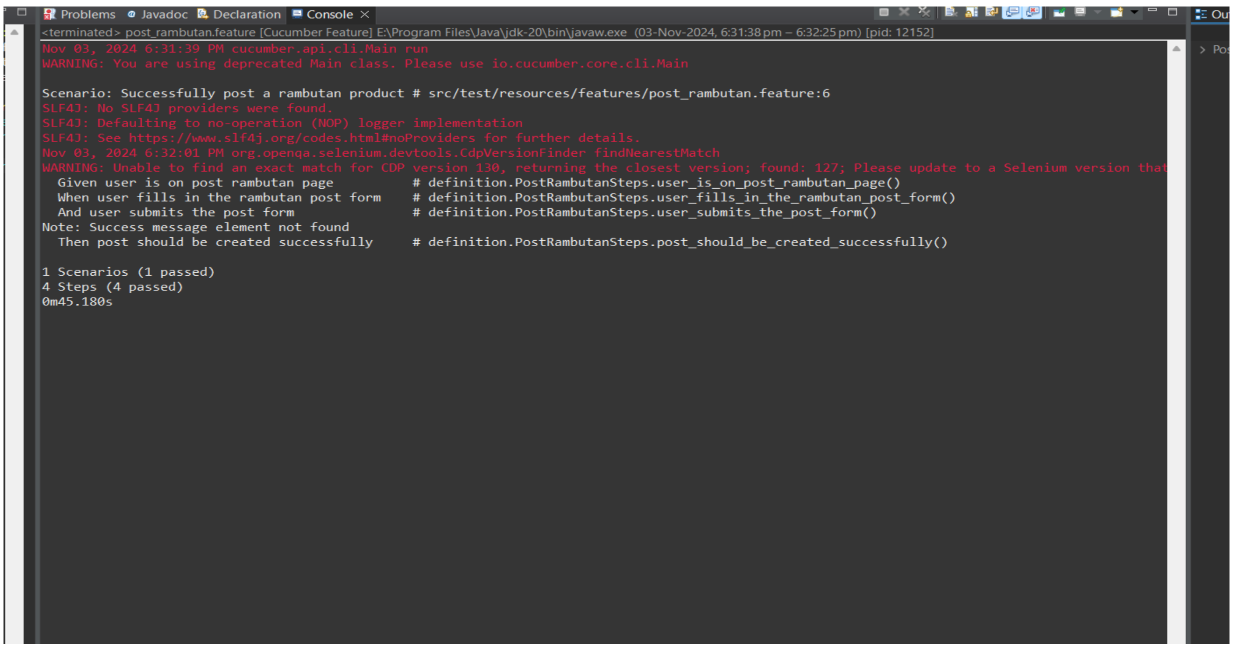
```
package definition;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.support.ui.WebDriverWait;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import org.junit.Assert;
import io.github.bonigarcia.wdm.WebDriverManager;
import java.time.Duration;

public class PostRambutanSteps {
private WebDriver driver;
private WebDriverWait wait;
private static final String BASE_URL = "http://127.0.0.1:8000";
private static final String LOGIN_EMAIL = "shanushibu99@gmail.com";
private static final String LOGIN_PASSWORD = "Shanu@123";

@Before
public void setup() {
try {
// Setup ChromeDriver
WebDriverManager.chromedriver().setup();

// Configure Chrome options
ChromeOptions options = new ChromeOptions();
options.addArguments("--remote-allow-origins=*");
options.addArguments("--start-maximized");
```

```
// Initialize ChromeDriver
driver = new ChromeDriver(options);
wait = new WebDriverWait(driver, Duration.ofSeconds(10));
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

} catch (Exception e) {
System.err.println("Error during setup: " + e.getMessage());
throw new RuntimeException("Failed to setup WebDriver: " + e.getMessage());
}
}

@Given("user is on post rambutan page")
public void user_is_on_post_rambutan_page() {
try {
// Navigate to login page
driver.get(BASE_URL + "/login/");
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("email")));

// Login process
WebElement emailInput = driver.findElement(By.id("email"));
WebElement passwordInput = driver.findElement(By.id("password"));
WebElement loginButton = driver.findElement(By.id("login"));

emailInput.sendKeys(LOGIN_EMAIL);
passwordInput.sendKeys(LOGIN_PASSWORD);
loginButton.click();

// Wait for dashboard and navigate to post rambutan
wait.until(ExpectedConditions.urlContains("farmer_dashboard"));
WebElement postRambutanLink = wait.until(ExpectedConditions.elementToBeClickable(
By.xpath("//a[contains(@href, 'post_rambutan')]")));
postRambutanLink.click();

// Verify navigation
```

```
wait.until(ExpectedConditions.urlContains("post_rambutan"));

} catch (Exception e) {
System.err.println("Error during navigation: " + e.getMessage());
throw new RuntimeException("Failed to navigate: " + e.getMessage());
}
}

@When("user fills in the rambutan post form")
public void user_fills_in_the_rambutan_post_form() {
try {
// Wait for form elements to be present
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("product")));

// Select product
Select productSelect = new Select(driver.findElement(By.id("product")));
productSelect.selectByValue("E35");

// Select category
Select categorySelect = new Select(driver.findElement(By.id("category")));
categorySelect.selectByValue("fresh_fruit");

// Select quantity type
Select quantityTypeSelect = new Select(driver.findElement(By.name("quantity_type")));
quantityTypeSelect.selectByValue("kg");

// Enter quantity
WebElement quantityInput = driver.findElement(By.name("quantity"));
quantityInput.clear();
quantityInput.sendKeys("10");

// Enter price
WebElement priceInput = driver.findElement(By.name("price"));
priceInput.clear();
priceInput.sendKeys("280");

// Upload image
WebElement fileInput = driver.findElement(By.name("image"));
String imagePath = System.getProperty("user.dir") +
"/src/test/resources/testdata/rambutan.jpg";
fileInput.sendKeys(imagePath);

// Enter description
WebElement descriptionInput = driver.findElement(By.name("description"));
descriptionInput.clear();
descriptionInput.sendKeys("Fresh & organic rambutan");

Thread.sleep(1000); // Small wait to ensure all fields are filled

} catch (Exception e) {
System.err.println("Error filling form: " + e.getMessage());
throw new RuntimeException("Failed to fill form: " + e.getMessage());
}
}

@And("user submits the post form")
public void user_submits_the_post_form() {
try {
WebElement submitButton = wait.until(ExpectedConditions.elementToBeClickable(
By.className("submit-button")));
submitButton.click();
Thread.sleep(2000); // Wait for form submission

} catch (Exception e) {
System.err.println("Error submitting form: " + e.getMessage());
throw new RuntimeException("Failed to submit form: " + e.getMessage());
}
}
```

```
@Then("post should be created successfully")
public void post_should_be_created_successfully() {
try {
// Wait for redirect to view posts page
wait.until(ExpectedConditions.urlContains("view_posts"));

// Optional: Verify success message if it exists
try {
WebElement successMessage = wait.until(ExpectedConditions.presenceOfElementLocated(
By.className("success-message")));
Assert.assertTrue("Success message should be displayed",
successMessage.isDisplayed());
} catch (TimeoutException e) {
System.out.println("Note: Success message element not found");
}

} catch (Exception e) {
System.err.println("Error verifying post creation: " + e.getMessage());
throw new RuntimeException("Failed to verify post creation: " + e.getMessage());
}
}

@After
public void tearDown() {
if (driver != null) {
try {
driver.quit();
} catch (Exception e) {
System.err.println("Error during teardown: " + e.getMessage());
}
}
}
}
```

Screenshot



```
<terminated> post_rambutan.feature [Cucumber Feature] E:\Program Files\Java\jdk-20\bin\javaw.exe  (03-Nov-2024, 6:31:38 pm – 6:32:25 pm) [pid: 12152]
Nov 03, 2024 6:31:39 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Successfully post a rambutan product # src/test/resources/features/post_rambutan.feature:6
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Nov 03, 2024 6:32:01 PM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 130, returning the closest version; found: 127; Please update to a Selenium version that
  Given user is on post rambutan page          # definition.PostRambutanSteps.user_is_on_post_rambutan_page()
  When user fills in the rambutan post form    # definition.PostRambutanSteps.user_fills_in_the_rambutan_post_form()
  And user submits the post form               # definition.PostRambutanSteps.user_submits_the_post_form()
Note: Success message element not found
  Then post should be created successfully     # definition.PostRambutanSteps.post_should_be_created_successfully()

1 Scenarios (1 passed)
4 Steps (4 passed)
0m45.180s
```

Test report

Test Case 2

Project Name: Rambutan Warehouse					
Post Rambutan Test Case					
Test Case ID: Test_2			**Test Designed By: Shanu**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 26/10/2024**		
Module Name: Post Rambutan Module			**Test Executed By: Mr. Ajith G. S**		
Test Title: Post Rambutan			**Test Execution Date: 26/10/2024**		
Description: Verify that a farmer can successfully post a rambutan product for buyers to view and purchase.					
Pre-Condition: User must be logged in as a farmer and should be on the post rambutan page.					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	Navigate to post rambutan		Post rambutan page should be displayed	Post rambutan page is displayed	Pass

	page				
2	Fill in rambutan post form details	Product: E35 Price: Rs 280 Quantity: 10 kg Image Description	Form fields should accept valid input	Form fields accepted input	Pass
3	Submit the post form		Post should be created and displayed to buyers	Post created and visible to buyers	Pass
4	Verify post appears in view post		Rambutan post should appear in view post	Rambutan post appears in listings	Pass
Post-Condition: The rambutan product post is created successfully and is visible to buyers in the product listings. Session details and post information are logged in the database.					

Test Case 3: Farmer details

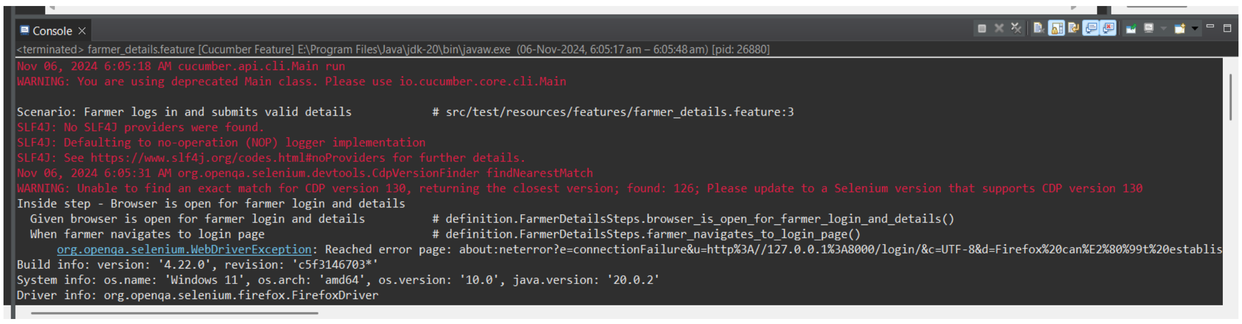
Code

```
package definition;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.And;
import java.time.Duration;

public class FarmerDetailsSteps {
WebDriver driver = null;
WebDriverWait wait;

@Given("browser is open for farmer login and details")
public void browser_is_open_for_farmer_login_and_details() {
```

```
System.out.println("Inside step - Browser is open for farmer login and details");
System.setProperty("webdriver.gecko.marionette","E:\\Program
Files\\Java\\eclipse1\\987\\src\\test\\resources\\drivers\\geckodriver.exe");
driver = new FirefoxDriver();
driver.manage().window().maximize();
wait = new WebDriverWait(driver, Duration.ofSeconds(10));
}

@When("farmer navigates to login page")
public void farmer_navigates_to_login_page() throws Exception {
driver.navigate().to("http://127.0.0.1:8000/login/");
Thread.sleep(2000);
}

@And("farmer enters valid login credentials")
public void farmer_enters_valid_login_credentials() {
driver.findElement(By.id("email")).sendKeys("shanusara2001@gmail.com");
driver.findElement(By.id("password")).sendKeys("Shibu@123");
}

@And("farmer clicks on login button")
public void farmer_clicks_on_login_button() {
driver.findElement(By.id("login")).click();
}

@Then("farmer should be redirected to dashboard")
public void farmer_should_be_redirected_to_dashboard() {
wait.until(ExpectedConditions.urlContains("/farmer_dashboard"));
System.out.println("Farmer logged in successfully and is on the dashboard");
}

@When("farmer clicks on Post Rambutan in the sidebar")
public void farmer_clicks_on_post_rambutan_in_sidebar() {
WebElement postRambutanLink =
wait.until(ExpectedConditions.elementToBeClickable(By.linkText("Post
Rambutan")));
postRambutanLink.click();
}

@Then("farmer should see the farmer details form")
public void farmer_should_see_farmer_details_form() {
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("address")));
System.out.println("Farmer details form is displayed");
}

@When("farmer enters valid farmer details")
public void farmer_enters_valid_farmer_details() {
driver.findElement(By.id("address")).sendKeys("123 Farm Road, Rural Area");
driver.findElement(By.id("mobile_number")).sendKeys("9876543210");
driver.findElement(By.id("aadhar_number")).sendKeys("123456789012");
driver.findElement(By.id("account_number")).sendKeys("1234567890");
driver.findElement(By.id("ifsc_code")).sendKeys("ABCD0123456");
driver.findElement(By.id("bank_name")).sendKeys("Farmer's Bank");
driver.findElement(By.id("location")).sendKeys("Green Valley");
}

@And("farmer submits the details form")
public void farmer_submits_the_details_form() {
driver.findElement(By.cssSelector("button[type='submit']")).click();
}

@Then("farmer should see a details submission success message")
public void farmer_should_see_a_details_submission_success_message() throws
Exception {
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".success-
message")));
Boolean isSuccessMessageDisplayed = driver.findElement(By.cssSelector(".success-
message")).isDisplayed();
```

```
if (isSuccessMessageDisplayed) {
System.out.println("Farmer details submitted successfully");
} else {
System.out.println("Farmer details submission failed or success message not
displayed");
}
Thread.sleep(2000);
driver.close();
driver.quit();
}
}
```

Screenshot



```
<terminated> farmer_details.feature [Cucumber Feature] E:\Program Files\Java\jdk-20\bin\javaw.exe  (06-Nov-2024, 6:05:17 am – 6:05:48 am) [pid: 26880]
Nov 06, 2024 6:05:18 AM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Farmer logs in and submits valid details           # src/test/resources/features/farmer_details.feature:3
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Nov 06, 2024 6:05:31 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 130, returning the closest version; found: 126; Please update to a Selenium version that supports CDP version 130
Inside step - Browser is open for farmer login and details
  Given browser is open for farmer login and details         # definition.FarmerDetailsSteps.browser_is_open_for_farmer_login_and_details()
  When farmer navigates to login page                        # definition.FarmerDetailsSteps.farmer_navigates_to_login_page()
      org.openqa.selenium.WebDriverException: Reached error page: about:neterror?e=connectionFailure&u=http%3A//127.0.0.1%3A8000/login/&c=UTF-8&d=Firefox%20can%E2%80%99t%20establis
Build info: version: '4.22.0', revision: 'c5f3146703*'
System info: os.name: 'Windows 11', os.arch: 'amd64', os.version: '10.0', java.version: '20.0.2'
Driver info: org.openqa.selenium.firefox.FirefoxDriver
```

Test report

Test Case 3

Project Name: Rambutan Warehouse					
Farmer Details Test Case					
Test Case ID: Test_3			**Test Designed By: Shanu**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 26/10/2024**		
Module Name: Farmer Module			**Test Executed By: Shanu**		
Test Title: Farmer Details			**Test Execution Date: 26/10/2024**		
Description: Verify that the farmer can log in and submit their details for posting rambutan products.					
Pre-Condition: Farmer has a valid login account and is prepared to enter personal and rambutan product details.					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**

1	Open browser for farmer login		Browser opens successfully	Browser opened	Pass
2	Navigate to login page		Login page should be displayed	Login page is displayed	Pass
3	Enter valid login credentials	Username: shanusara2001@gmail.com Password: Shibu@123	Credentials are accepted	Credentials accepted	Pass
4	Click on login button		Farmer should be redirected to the dashboard	Redirected to the dashboard	Pass
5	Click on "Post Rambutan" in sidebar		Farmer details form should be displayed	Farmer details form displayed	Pass
6	Enter valid farmer details	Enter valid farmer details	Form should accept valid farmer details	Details accepted	Pass
7	Submit the form		Should be redirected to the post rambutan	Redirected to the post rambutan	Pass
Post-Condition: Farmer is logged in and has successfully submitted their details, which are stored in the database. Farmer can now proceed to post rambutan products for sale.					

Test Case 4: Customer update profile

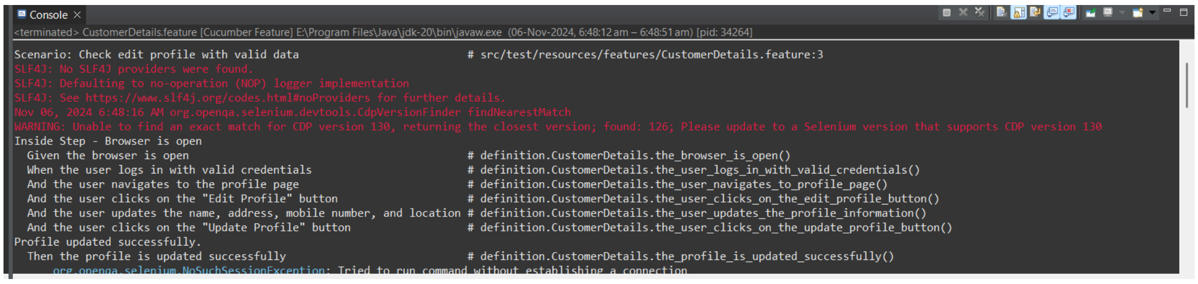
Code

```
package definition;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import java.time.Duration;

public class CustomerDetails {
private WebDriver driver = null;
private WebDriverWait wait;
@Given("the browser is open")
public void the_browser_is_open() {
System.out.println("Inside Step - Browser is open");
System.setProperty("webdriver.gecko.marionette",
"E:\\Program
Files\\Java\\eclipse1\\987\\src\\test\\resources\\drivers\\geckodriver.exe");
driver = new FirefoxDriver();
driver.manage().window().maximize();
wait = new WebDriverWait(driver, Duration.ofSeconds(10));
}
@When("the user logs in with valid credentials")
public void the_user_logs_in_with_valid_credentials() {
try {
driver.navigate().to("http://127.0.0.1:8000/login/");
Thread.sleep(2000);
// Wait for login form elements to be present
wait.until(ExpectedConditions.presenceOfElementLocated(By.name("email")))
.sendKeys("shanusarashibu01@gmail.com");
wait.until(ExpectedConditions.presenceOfElementLocated(By.name("password")))
.sendKeys("Sharon@123");

wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("button[type=
'submit']")))
.click();
Thread.sleep(2000); // Wait for login to complete
} catch (InterruptedException e) {
System.out.println("Login process was interrupted: " + e.getMessage());
}
}
```

```
@And("the user navigates to the profile page")
public void the_user_navigates_to_profile_page() {
try {
driver.navigate().to("http://127.0.0.1:8000/profile_view/");
Thread.sleep(2000);
} catch (InterruptedException e) {
System.out.println("Navigation was interrupted: " + e.getMessage());
}
}
@When("the user clicks on the \"Edit Profile\" button")
public void the_user_clicks_on_the_edit_profile_button() {
try {
wait.until(ExpectedConditions.elementToBeClickable(By.linkText("Edit Profile")))
.click();
Thread.sleep(1000);
} catch (InterruptedException e) {
System.out.println("Click operation was interrupted: " + e.getMessage());
}
}
@And("the user updates the name, address, mobile number, and location")
public void the_user_updates_the_profile_information() {
try {
// Update Name
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("name"))).clear();
driver.findElement(By.id("name")).sendKeys("Sharon T Shibu");
Thread.sleep(500);
// Update Address
driver.findElement(By.id("address")).clear();
driver.findElement(By.id("address")).sendKeys("Cholakathu (H)");
Thread.sleep(500);
// Update Contact
driver.findElement(By.id("contact")).clear();
driver.findElement(By.id("contact")).sendKeys("9876543210");
Thread.sleep(500);
```

```
// Update Place
driver.findElement(By.id("place")).clear();
driver.findElement(By.id("place")).sendKeys("Vennikulam");
Thread.sleep(500);
} catch (InterruptedException e) {
System.out.println("Update operation was interrupted: " + e.getMessage());
}
}
@And("the user clicks on the \"Update Profile\" button")
public void the_user_clicks_on_the_update_profile_button() {
try {
wait.until(ExpectedConditions.elementToBeClickable(By.className("submit-
button")))
.click();
Thread.sleep(1000);
} catch (InterruptedException e) {
System.out.println("Submit operation was interrupted: " + e.getMessage());
}
}
@Then("the profile is updated successfully")
public void the_profile_is_updated_successfully() {
try {
// Wait for success message or updated profile view
boolean isUpdateSuccessful = false;
try {
// First try to find a success message
wait.until(ExpectedConditions.presenceOfElementLocated(
By.className("profile-update-success")));
isUpdateSuccessful = true;
} catch (Exception e) {
// If no success message, check if we're back on profile view page
String currentUrl = driver.getCurrentUrl();
if (currentUrl.contains("profile_view")) {
isUpdateSuccessful = true;
}
}
if (isUpdateSuccessful) {
System.out.println("Profile updated successfully.");
} else {
System.out.println("Profile update failed.");
}
Thread.sleep(2000);
} catch (InterruptedException e) {
System.out.println("Verification was interrupted: " + e.getMessage());
} finally {
if (driver != null) {
driver.close();
driver.quit();
}
}
}
}
```

Screenshot



```
<terminated> CustomerDetails.feature [Cucumber Feature] E:\Program Files\Java\jdk-20\bin\javaw.exe  (06-Nov-2024, 6:48:12 am – 6:48:51 am) [pid: 34264]
Scenario: Check edit profile with valid data                            # src/test/resources/features/CustomerDetails.feature:3
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Nov 06, 2024 6:48:16 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 130, returning the closest version; found: 126; Please update to a Selenium version that supports CDP version 130
Inside Step - Browser is open
  Given the browser is open                                             # definition.CustomerDetails.the_browser_is_open()
  When the user logs in with valid credentials                          # definition.CustomerDetails.the_user_logs_in_with_valid_credentials()
  And the user navigates to the profile page                            # definition.CustomerDetails.the_user_navigates_to_profile_page()
  And the user clicks on the "Edit Profile" button                      # definition.CustomerDetails.the_user_clicks_on_the_edit_profile_button()
  And the user updates the name, address, mobile number, and location # definition.CustomerDetails.the_user_updates_the_profile_information()
  And the user clicks on the "Update Profile" button                    # definition.CustomerDetails.the_user_clicks_on_the_update_profile_button()
Profile updated successfully.
  Then the profile is updated successfully                              # definition.CustomerDetails.the_profile_is_updated_successfully()
      org.openqa.selenium.NoSuchSessionException: Tried to run command without establishing a connection
```

Test report

Test Case 4	
Project Name: Rambutan Warehouse	
Update Profile Test Case	
Test Case ID: Test_4	**Test Designed By: Shanu**
Test Priority (Low/Medium/High): High	**Test Designed Date: 01/11/2024**
Module Name: Profile Module	**Test Executed By: Shanu**

Test Title: Edit Profile			**Test Execution Date: 01/11/2024**		
Description: Verify that the user can successfully update their profile information with valid data.					
Pre-Condition: User has an existing account, valid login credentials, and is on the profile page.					
Step	**Test Step**	**Test Data**	**Expecte dResult**	**Actual Result**	**Status (Pass/ Fail)**
1	Open browser		Browser opens successfully	Browser opened	Pass
2	Log in with valid credential	Username: shanusarashibu01@gmail.com Password: Sharon@123	User should be logged in successfully	Logged in successfully	Pass
3	Navigate to the profile page		Profile page should be displayed	Profile page displayed	Pass
4	Click on "Edit Profile" button		Edit Profile form should be displayed	Edit Profile form is displayed	Pass
5	Update profile information	Name: Sharon T Shibu Address: Cholakathu (H) Mobile: 9876543210 Place: Vennikulam	Fields should accept and display the updated information	Information updated successfully	Pass
6	Click on "Update Profile" button		Profile should be updated successfully	Profile updated successfully	Pass
7	Verify profile update success		Profile view is displayed	Profile update confirmed	Pass
Post-Condition: User's profile information is updated and saved in the database. Updated details should reflect on the profile page.					

CHAPTER 6
IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the process of translating software designs and specifications into a functional software system. It involves the actual coding, testing, and integration of software components to create a working software product. The implementation phase is a critical step in the software development life cycle (SDLC) because it involves turning the abstract ideas and plans into tangible software products that can be used by end-users. The implementation process involves several steps, including:

- Coding: Writing code in the programming language specified in the design phase.
- Testing: Ensuring that the code functions correctly and meets the design specifications.
- Testing can include unit testing, integration testing, and system testing.
- Debugging: Identifying and fixing any errors or defects in the code.
- Integration: Combining the individual software components into a cohesive system that functions as intended.
- Deployment: Installing the software on the target hardware and making it available to endusers.

During the implementation phase, it is important to follow coding and testing standards to ensure the software is reliable, maintainable, and scalable. It is also important to document the code and maintain version control to facilitate future maintenance and updates. Collaboration between developers and other stakeholders, such as testers, project managers, and end-users, can help to ensure that the software meets the needs and expectations of all stakeholders.

6.2 IMPLEMENTATION PROCEDURES

The implementation phase of the software development life cycle (SDLC) involves a series of procedures that must be followed to ensure that the software product is developed according to the design specifications and meets the quality and performance standards. Below are some of the typical procedures involved in the implementation phase:

- Coding: This involves writing the actual code for the software using the programming language specified in the design phase. Developers must follow coding standards and guidelines to ensure that the code is maintainable and scalable.
- Testing: Testing is a critical part of the implementation phase, and it involves testing the software at different levels, including unit testing, integration testing, and system testing. Testers

must develop test cases and test scripts based on the design specifications to ensure that the software functions correctly.

• Debugging: During testing, issues and bugs may be identified in the software code. Developers must identify and fix these errors or defects to ensure that the software functions as intended.

• Integration: After individual software components have been tested and debugged, they must be integrated into a cohesive system. The integration process must be carefully managed to ensure that the software functions as intended and that there are no conflicts or errors.

• Deployment: Once the software has been integrated, it must be deployed to the target hardware and made available to end-users. Deployment can be a complex process that involves several steps, including installation, configuration, and testing.

• Documentation: It is essential to document the software code and the implementation procedures to ensure that the software can be easily maintained and updated in the future. Documentation can include user manuals, technical documentation, and code comments.

• Version control: Version control is critical during the implementation phase to ensure that changes to the software code are tracked and managed properly. Version control tools such as Git or SVN can be used to manage code changes and track different versions of the software product.

By following these procedures, software developers can ensure that the software product is developed according to the design specifications and meets the quality and performance standards required by the end-users.

6.2.1 User Training

User training is a critical component of any software development project. It involves providing end-users with the necessary knowledge and skills to effectively use the software product. The goal of user training is to ensure that end-users can use the software product efficiently and effectively to meet their business needs.

Below are some common steps involved in user training:

• Identify training needs: Before training can begin, it is essential to identify the training needs of the end-users. This can be done by conducting a needs analysis, which involves assessing the current skills and knowledge of the end-users and identifying any gaps that need to be addressed.

• Develop training materials: Once the training needs have been identified, training materials can be developed. This can include user manuals, online tutorials, and training videos. The training materials should be designed to be easy to understand and follow and should cover all the necessary features and functionalities of the software product.

• Conduct training sessions: Training sessions can be conducted in-person or online, depending on the needs of the end-users. The training sessions should be interactive and engaging, and should provide opportunities for end-users to ask questions and practice using the software product.

• Assess learning: After the training sessions, it is important to assess whether the end-users have learned the necessary skills and knowledge to effectively use the software product. This can be done through quizzes, assessments, or practical exercises.

• Provide ongoing support: Even after the initial training, it is important to provide ongoing support to end-users. This can include a helpdesk or support team that can provide assistance when needed, as well as updates to the training materials to reflect any changes or updates to the software product.
Effective user training can help to ensure that end-users are able to use the software product efficiently and effectively, which can lead to increased productivity and improved business outcomes.

6.2.2 Training on the Application Software

Training on application software is an essential component of software implementation. It is the process of providing end-users with the knowledge and skills necessary to operate and use the application software effectively. The goal of training is to ensure that end-users are able to utilize all the features and functionalities of the software to perform their tasks and achieve their goals. Below are some best practices for providing effective training on application software:

• Identify the target audience: Before developing training materials, it is essential to identify the target audience and their specific needs. Different groups of end-users may require different levels of training or different types of training materials.

• Develop training materials: Training materials can include user manuals, online tutorials, videos, and live training sessions. The training materials should be designed to be easy to understand and follow, and should cover all the necessary features and functionalities of the software product.•

•Use interactive training methods: Interactive training methods such as demonstrations, simulations, and hands-on exercises can help end-users to better understand and retain the training materials.

• Provide ongoing support: After the initial training, it is important to provide ongoing support to end-users. This can include a helpdesk or support team that can provide assistance when needed, as well as updates to the training materials to reflect any changes or updates to the software product.

• Evaluate training effectiveness: It is important to evaluate the effectiveness of the training by assessing the end-users' knowledge and skills before and after the training. This can help identify areas that may require additional training or support.

• Provide refresher training: As the software product evolves, it is important to provide refresher training to end-users to ensure that they are up-to-date with the latest features and functionalities.

Effective training on application software can help to ensure that end-users are able to use the software product effectively, which can lead to increased productivity and improved business outcomes.

6.2.3 System Maintenance

System maintenance is the process of keeping a software system up-to-date and functioning properly after it has been deployed. It involves a variety of activities that are designed to ensure that the system remains stable, secure, and efficient over time. Some of the key activities involved in system maintenance include:

• Updates and patches: One of the most important aspects of system maintenance is keeping the system up-to-date with the latest software updates and security patches. These updates and patches are released periodically by software vendors to fix bugs, add new features, and address security vulnerabilities. Applying these updates in a timely manner is essential to ensure that the system remains secure and reliable.

• Performance monitoring: Regular performance monitoring is another key aspect of system maintenance. This involves tracking system performance metrics such as CPU usage, memory utilization, and network traffic to identify potential issues and optimize system performance.

• Backup and recovery: Backing up system data and ensuring that it can be recovered in the event of a system failure or data loss is an important part of system maintenance. This involves creating regular backups of system data and testing the recovery process to ensure that it works as expected.

• Security management: Maintaining system security is critical to protecting sensitive data and preventing unauthorized access to the system. This involves implementing security controls such as firewalls, antivirus software, and access controls, and regularly monitoring the system for potential security breaches.

• Hardware maintenance: Ensuring that hardware components such as servers, storage devices, and network devices are functioning properly is another key aspect of system maintenance. This involves performing regular maintenance tasks such as cleaning, repairs, and upgrades to ensure that the hardware components remain in good working condition.

- Effective system maintenance is critical to ensuring that a software system remains reliable, secure, and efficient over time. By implementing a proactive system maintenance strategy, organizations can minimize downtime, prevent system failures, and optimize system performance.

6.2.4 Hosting

Hosting refers to the process of storing and serving website files on a remote server, which can be accessed by visitors over the internet. When you create a website, you need to have it hosted on a server so that it can be available for others to view. There are various types of hosting options available such as shared hosting, dedicated hosting, VPS hosting, cloud hosting, etc. The choice of hosting depends on the size of the website, its traffic volume, and the level of control and flexibility required by the owner.

Render

Render is a cloud hosting service that provides a streamlined platform for deploying web applications, APIs, static sites, and databases. It simplifies the process of hosting applications by automating tasks such as server setup, scaling, and load balancing, which allows developers to focus on building their applications rather than managing infrastructure. Render supports various languages and frameworks, including Django, and offers seamless integration with Git for continuous deployment.

Procedure for hosting a website on Render:

Step 1: Login to your render account

Step 2: Create a new project

Step 3: Create a requirements.txt file which will include all your dependencies to be installed.

Step 4: Upload the content which is to be hosted into GitHub

Step 5: Change the setting of your project to include the host also and then deploy.

Hosted Website: Render

Hosted Link: https://rambutanwarehouse.onrender.com

Hosted QR Code:

RAMBUTAN WAREHOUSE

Screenshot

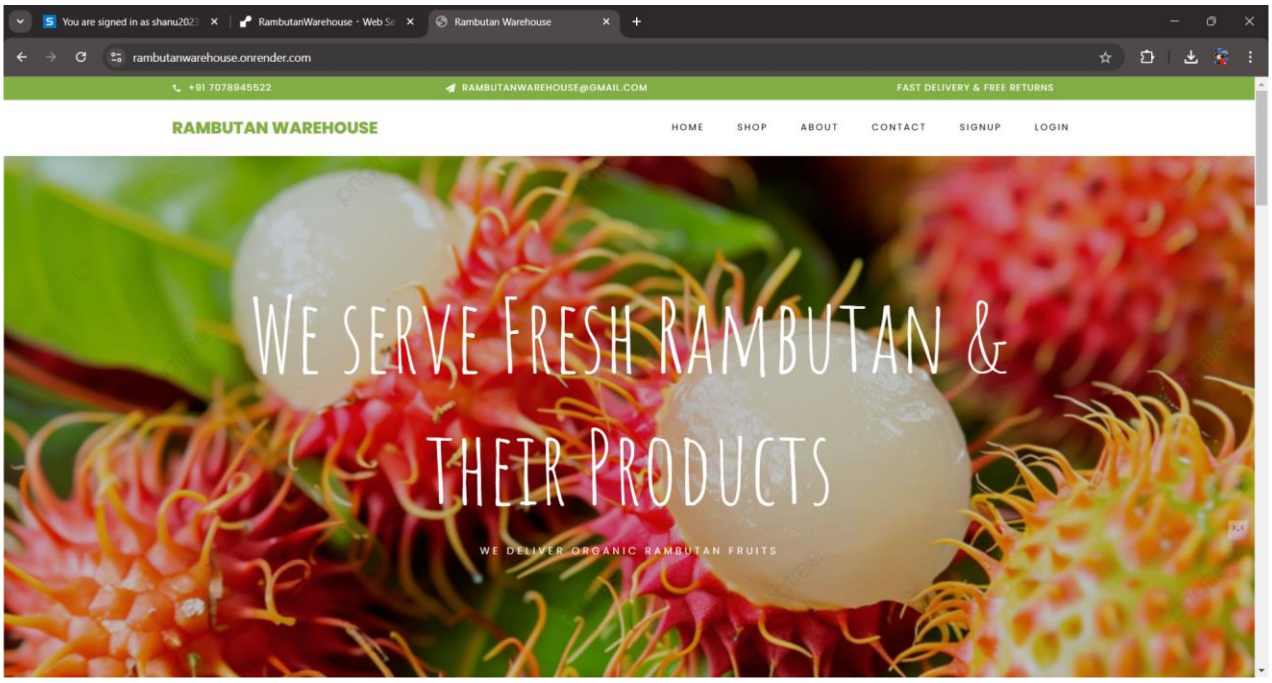

You are signed in as shanu2023
RambutanWarehouse · Web Se
Rambutan Warehouse
rambutanwarehouse.onrender.com
+91 7078945522
RAMBUTANWAREHOUSE@GMAIL.COM
FAST DELIVERY & FREE RETURNS
RAMBUTAN WAREHOUSE
HOME
SHOP
ABOUT
CONTACT
SIGNUP
LOGIN
WE SERVE FRESH RAMBUTAN & THEIR PRODUCTS
WE DELIVER ORGANIC RAMBUTAN FRUITS

Figure 6.2.4-Hosting

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Rambutan Warehouse project has developed a comprehensive online platform to support the rambutan supply chain, fostering a streamlined experience for all participants. By facilitating direct connections between farmers, wholesale buyers, administrators, and regular buyers, the platform contributes significantly to the rambutan distribution process. Farmers benefit from inventory management and sales tracking tools, while buyers can easily access a variety of fresh and processed rambutan products. Administrators play a key role in ensuring platform reliability, managing operations, and resolving any issues that arise. This Django-based system with a responsive HTML/CSS frontend achieves its goal of creating a user-friendly environment that not only encourages rambutan sales but also promotes the fruit's wider distribution. The project demonstrates a promising approach to bridging the gap between agricultural producers and consumers, using technology to create a seamless, accessible marketplace.

7.2 FUTURE SCOPE

Looking forward, the Rambutan Warehouse platform has several avenues for growth and enhancement. One promising direction is the integration of machine learning algorithms to aid farmers and buyers in making data-driven decisions. For instance, predictive models could assist in demand forecasting, price optimization, and inventory management, helping farmers achieve better pricing and efficient stock turnover. Developing a mobile application is another important step, as it would increase accessibility for users, particularly in rural areas, thereby broadening the platform's reach and user base.

In addition, expanding the payment options to support international currencies and integrating more advanced payment gateways would simplify transactions and enhance security, making the platform attractive to global buyers. Real-time analytics could empower farmers and administrators with actionable insights into sales trends and seasonal demand, enabling more strategic decision-making. Incorporating blockchain technology for transaction transparency would further instill trust among users by providing a tamper-proof ledger of all sales and interactions. Finally, the platform could grow its product offerings to include other tropical fruits and rambutan-derived goods, giving farmers more opportunities to diversify their income. By pursuing these advancements, Rambutan Warehouse could solidify its role as a leading solution in the digital transformation of agricultural commerce.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- "Django for Beginners" by William S. Vincent
- "Django Essentials" by Samuel Dauzon
- "Django for Beginners: Build Websites with Python and Django" by Mark G. Carter
- "Beginning Django: Web Application Development and Deployment with Python" by Daniel Rubio

WEBSITES:

- Django Official Documentation: https://docs.djangoproject.com/
- SQLite Documentation: https://www.sqlite.org/docs.html
- "Django Best Practices: Projects vs. Apps" by LearnDjango.com: https://learndjango.com/tutorials/django-best-practices-projects-vs-apps
- "How to Create a Django Project?" by GeeksforGeeks: https://www.geeksforgeeks.org/how-to-create-a-django-project/

CHAPTER 9
APPENDIX

9.1 Sample Code

farmer_dashboard.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="UTF-8">
  <title>Farmer Dashboard | Rambutan Warehouse</title>
  <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10"></script>
  <link href='https://unpkg.com/boxicons@2.0.7/css/boxicons.min.css' rel='stylesheet'>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Google Fonts -->
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&family=Fredoka+O
ne&display=swap" rel="stylesheet">
   <!-- Boxicons for Icons -->
   <link href="https://unpkg.com/boxicons@2.0.7/css/boxicons.min.css" rel="stylesheet">
   <!-- SweetAlert2 CSS -->
   <link href="https://cdn.jsdelivr.net/npm/sweetalert2@11/dist/sweetalert2.min.css"
rel="stylesheet">
   <!-- SweetAlert2 JS -->
   <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    :root {
      --primary-color: #4CAF50;
      --secondary-color: #66BB6A;
      --accent-color: #388e3c;
      --light-bg: #f0f4f3;
      --dark-bg: #333;
      --text-color: #333;
      --white: #ffffff;
      --font-family: 'Roboto', sans-serif;
      --font-family-logo: 'Fredoka One', cursive;
    }
    body {
      font-family: var(--font-family);
      background-color: var(--light-bg);
      display: flex;
      min-height: 100vh;
    }
    .sidebar {
      background-color: var(--primary-color);
      width: 250px;
      position: fixed;
      top: 0;
      left: 0;
      height: 100%;
      padding-top: 40px;
      transition: width 0.3s ease;
      z-index: 1000;
    }
    .sidebar .logo-details {
      text-align: center;
      margin-bottom: 40px;
    }
    .sidebar .logo_name {
      font-family: var(--font-family-logo);
      font-size: 28px;
      color: var(--white);
    }
    .sidebar ul {
      list-style: none;
    }
    .sidebar ul li {
      margin: 20px 0;
      text-align: center;
    }
    .sidebar ul li a {
      text-decoration: none;
      color: var(--white);
      font-size: 18px;
      display: flex;
      align-items: center;
      justify-content: center;
      padding: 10px 20px;
      border-radius: 8px;
      transition: background-color 0.3s, transform 0.3s;
    }
    .sidebar ul li a:hover,
    .sidebar ul li a.active {
      background-color: var(--accent-color);
      transform: translateX(5px);
    }
    .sidebar ul li a i {
      margin-right: 10px;
      font-size: 20px;
    }
    .home-section {
      margin-left: 250px;
      padding: 20px;
      width: calc(100% - 250px);
      transition: margin-left 0.3s ease, width 0.3s ease;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
```

```
}
/* Footer Styling */
footer {
  width: calc(100% - 250px);
  margin-left: 250px;
  margin-top: 40px;
  padding: 20px;
  color: var(--black);
  text-align: center;
  border-radius: 10px;
}
footer a {
  color: var(--black);
  text-decoration: none;
  margin: 0 10px;
  transition: color 0.3s;
}
footer a:hover {
  color: var(--accent-color);
  text-decoration: underline;
}
@media (max-width: 768px) {
  .sidebar {
    width: 80px;
  }
  .home-section {
    margin-left: 80px;
    width: calc(100% - 80px);
  }
  .sidebar .logo_name {
    display: none;
  }
  .sidebar ul li a .links_name {
    display: none;
  }
  .sidebar ul li a i {
    margin-right: 0;
  }
  .top-navbar .logo img {
    height: 35px;
  }
  .home-section nav.top-navbar .logo img {
    height: 35px;
  }
}
.top-navbar {
  background-color: var(--secondary-color);
  padding: 10px 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  margin-bottom: 20px;
}
.logo-text {
  font-size: 24px;
  color: white;
  font-weight: bold;
  text-transform: uppercase;
}
.top-navbar .logo a {
  display: flex;
  align-items: center;
  text-decoration: none;
}

.top-navbar .logo img {
```

```
  height: 50px;
  margin-right: 10px;
}
.top-navbar ul.nav-icons {
  list-style: none;
  display: flex;
  gap: 30px;
  align-items: center;
}
.top-navbar ul.nav-icons li a {
  text-decoration: none;
  color: var(--white);
  font-size: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
  transition: color 0.3s;
  position: relative;
}
.top-navbar ul.nav-icons li a:hover {
  color: var(--accent-color);
}
.top-navbar ul.nav-icons li a .icon-name {
  font-size: 12px;
  margin-top: 5px;
  font-weight: 500;
}
.top-navbar ul.nav-icons li.user-profile a i {
  font-size: 24px;
  color: var(--white);
}
.top-navbar ul.nav-icons li.user-profile a .icon-name {
  font-size: 14px;
  color: var(--white);
```

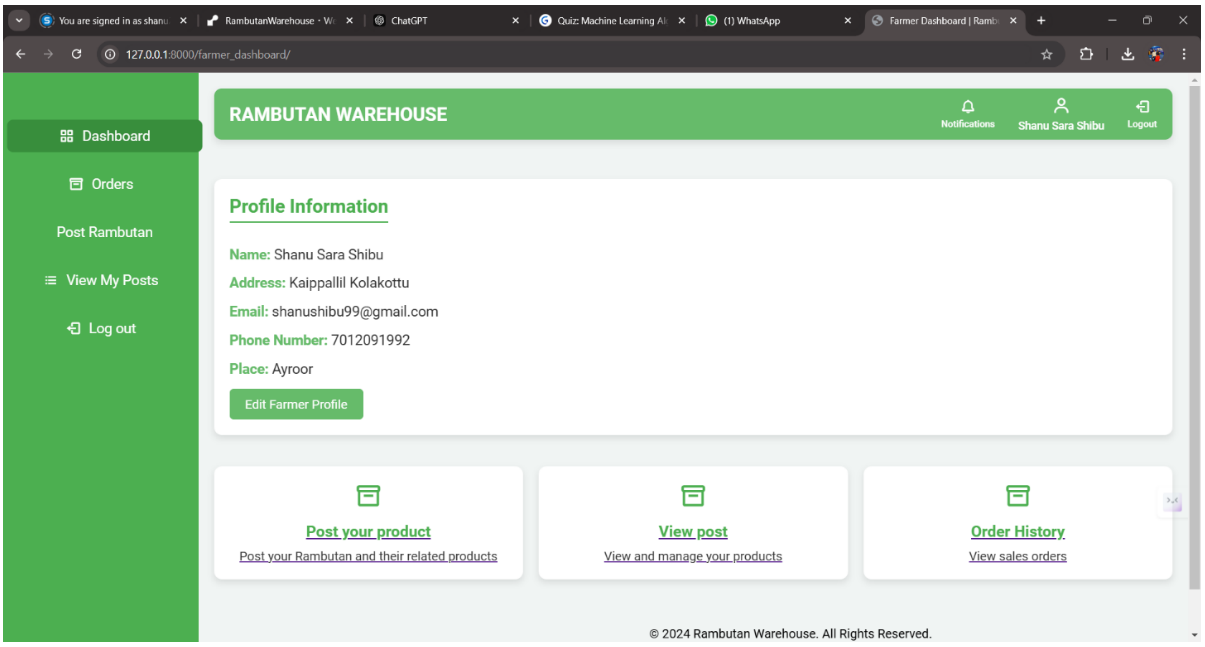
```
}
.main-nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 20px;
  background-color: var(--white);
  color: var(--secondary-color);
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  margin-bottom: 20px;
}
.main-nav .dashboard {
  font-size: 24px;
  font-weight: 700;
}
.main-nav .profile-details {
  display: flex;
  align-items: center;
  cursor: pointer;
  position: relative;
}
.main-nav .profile-details img {
  display: none;
}

.main-nav .profile-details span {
  font-size: 16px;
  font-weight: 500;
  margin-right: 5px;
}

.main-nav .dropdown-menu {
  display: none;
  position: absolute;
  top: 60px;
  right: 0;
  background-color: var(--white);
  color: var(--text-color);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  border-radius: 8px;
  width: 180px;
  z-index: 1001;
  padding: 10px 0;
  transition: opacity 0.3s ease;
}
.main-nav .dropdown-menu.active {
  display: block;
}
.main-nav .dropdown-menu li {
  list-style: none;
  padding: 10px 20px;
}
.main-nav .dropdown-menu li a {
  text-decoration: none;
  color: var(--text-color);
  display: block;
  transition: background-color 0.3s;
}
.main-nav .dropdown-menu li a:hover {
  background-color: var(--light-bg);
  border-radius: 5px;
}
.profile-view {
  margin-top: 30px;
  background-color: var(--white);
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.profile-view h2 {
  font-size: 24px;
  color: var(--primary-color);
  margin-bottom: 20px;
  border-bottom: 2px solid var(--primary-color);
  display: inline-block;
  padding-bottom: 5px;
}
.profile-view .info {
  display: flex;
  flex-direction: column;
  gap: 15px;
  margin-top: 10px;
}
.profile-view .info div {
  font-size: 18px;
  color: var(--text-color);
}
.profile-view .info div strong {
  color: var(--secondary-color);
}
.quick-actions {
    display: flex;
    justify-content: space-between;
    margin-top: 40px;
    gap: 20px;
}
.quick-actions .card {
    background-color: var(--white);
    width: 100%;
    border-radius: 10px;
    padding: 20px;
    text-align: center;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    transition: box-shadow 0.3s ease;
}
.quick-actions .card:hover {
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);
}
.quick-actions .card h3 {
    color: var(--primary-color);
    font-size: 20px;
    margin-bottom: 10px;
}

.quick-actions .card p {
    font-size: 16px;
    color: var(--text-color);
}
.quick-actions .card i {
    font-size: 36px;
    color: var(--primary-color);
    margin-bottom: 15px;
}
.edit {
  margin-top: 15px; /* Add some spacing above the button */
}
.edit-button {
  display: inline-block;
  background-color: var(--secondary-color); /* Use your secondary color */
  color: var(--white);
  padding: 10px 20px; /* Add padding for a better clickable area */
  text-decoration: none;
  border-radius: 5px; /* Slightly rounded corners */
  font-size: 16px; /* Font size adjustment */
  transition: background-color 0.3s, transform 0.2s; /* Transition effects */
```

```
    }
    .edit-button:hover {
      background-color: var(--accent-color); /* Change background on hover */
      transform: translateY(-2px); /* Lift the button slightly on hover */
    }
  </style>
</head>
<body>
  <div class="sidebar">
    <ul class="nav-links">
      <li>
        <a href="{% url 'farmer_dashboard' %}" class="active">
          <i class='bx bx-grid-alt'></i>
          <span class="links_name">Dashboard</span>
        </a>
      </li>
      <li>
        <a href="{% url 'farmer_orders' %}">
          <i class='bx bx-box'></i>
          <span class="links_name">Orders</span>
        </a>
      </li>
      {% comment %} <li>
        <a href="#">
          <i class='bx bx-cart'></i>
          <span class="links_name">New Sales</span>
        </a>
      </li> {% endcomment %}
      <li>
        <a href="{% url 'post_rambutan' %}">
          <i class='bx bx-leaf'></i>
          <span class="links_name">Post Rambutan</span>
        </a>
      </li>
```

```
          <li>
            <a href="{% url 'view_posts' %}">
              <i class='bx bx-list-ul'></i>
              <span class="links_name">View My Posts</span>
            </a>
          </li>
          <li class="log_out">
            <a href="{% url 'logout' %}">
              <i class='bx bx-log-out'></i>
              <span class="links_name">Log out</span>
            </a>
          </li>
        </ul>
      </div>
      <section class="home-section">
        <!-- Top Navigation Bar -->
        <nav class="top-navbar">
          <div class="logo">
            <a href="#">
              <span class="logo-text">Rambutan Warehouse</span>
            </a>
          </div>
          <ul class="nav-icons">
            <li>
              <a href="{% url 'order_notifications' %}">
                <i class="bx bx-bell"></i>
                <span class="icon-name">Notifications</span>
              </a>
            </li>

            <li class="user-profile">
              <a href="#">
                <i class='bx bx-user'></i>
                <span class="icon-name">{{ farmer.name }}</span>
              </a>
            </li>
            <li>
              <a href="{% url 'logout' %}">
                <i class="bx bx-log-out"></i>
                <span class="icon-name">Logout</span>
              </a>
            </li>
          </ul>
        </nav>
        <div class="home-content">
          <!-- Profile View -->
          <div class="profile-view">
            <h2>Profile Information</h2>
            <div class="info">
              <div><strong>Name:</strong> {{ farmer.name }}</div>
              <div><strong>Address:</strong> {{ farmer.address }}</div>
              <div><strong>Email:</strong> {{ user.username }}</div>
              <div><strong>Phone Number:</strong> {{ farmer.contact }}</div>
              <div><strong>Place:</strong> {{ farmer.place }}</div>
            </div>
            <div class="edit">
              <a href="{% url 'edit_farmer_profile' %}" class="edit-button">Edit Farmer Profile</a>
            </div>
        </div>
      <!-- Quick Actions Section -->
      <div class="quick-actions">
      <div class="card">
            <a href="{% url 'post_rambutan' %}">
                <i class="bx bx-box"></i>
                <h3>Post your product</h3>
                <p>Post your Rambutan and their related products</p>
            </a>
        </div>
        <div class="card">
            <a href="{% url 'view_posts' %}">
                  <i class="bx bx-box"></i>
                  <h3>View post</h3>
                  <p>View and manage your products</p>
            </a>
        </div>
        <div class="card">
          <a href="{% url 'farmer_orders' %}">
              <i class="bx bx-box"></i>
              <h3>Order History</h3>
              <p>View sales orders</p>
          </a>
      </div>
          <a href="#">
              <i class="bx bx-cart"></i>
              <h3>New Sale</h3>
              <p>Register a new sale of goods</p>
          </a>
      </div> {% endcomment %}
      </div>
      <!-- Footer -->
      <footer>
      <p>&copy; 2024 Rambutan Warehouse. All Rights Reserved.</p>
      <a href="#">Privacy Policy</a> |
      <a href="#">Terms & Conditions</a> |
      <a href="#">Contact Us</a>
      </footer>
      </section>
       <!-- JavaScript Files -->
       <script>
      // Confirm Logout with SweetAlert2
      document.querySelectorAll('.log_out a, .top-navbar a[href="{% url 'logout'
      %}"]').forEach(function (element) {
        element.addEventListener('click', function (event) {
          event.preventDefault();
          Swal.fire({
            title: 'Are you sure you want to logout?',
            icon: 'warning',
            showCancelButton: true,
            confirmButtonText: 'Yes, logout',
            cancelButtonText: 'Cancel'
          }).then((result) => {
            if (result.isConfirmed) {
              window.location.href = this.href;
            }
          });
        });
      });
    </script>
    </body>
    </html>
```

9.2 Screen Shots

9.2.1Farmer dashboard



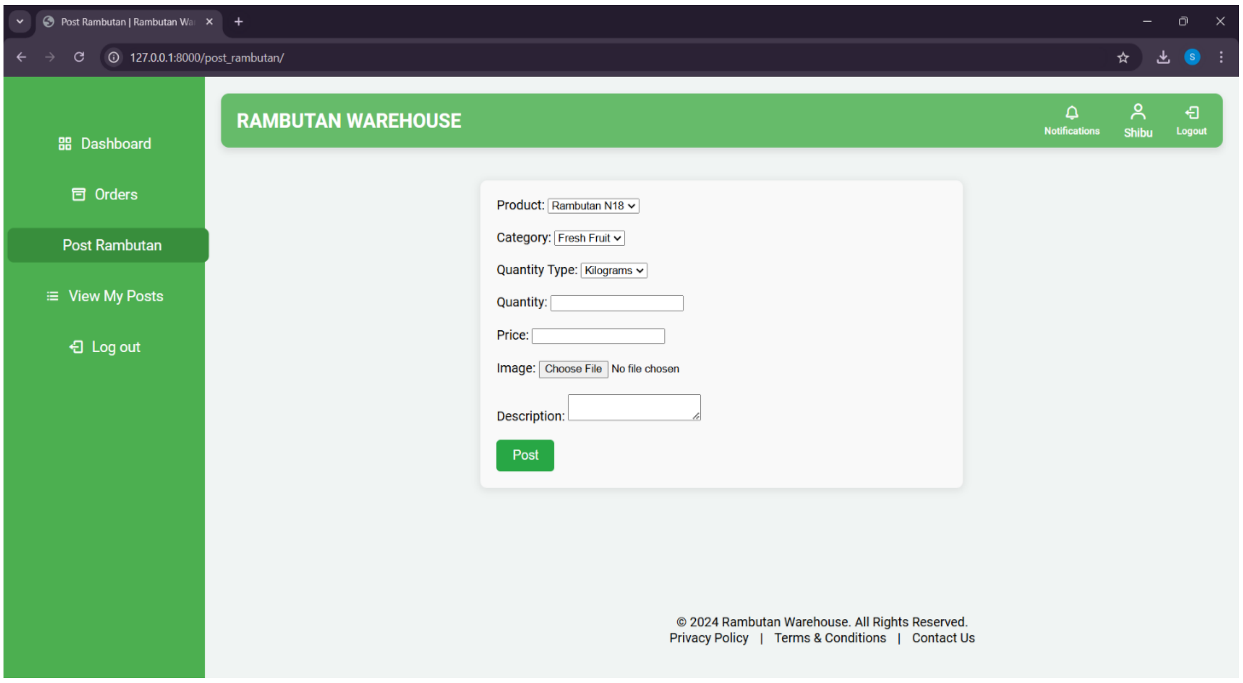
Profile Information

Name: Shanu Sara Shibu
Address: Kaippallil Kolakottu
Email: shanushibu99@gmail.com
Phone Number: 7012091992
Place: Ayroor

© 2024 Rambutan Warehouse. All Rights Reserved.

Figure 9.2.1-farmer dashboard

9.2.2 Post Rambutan

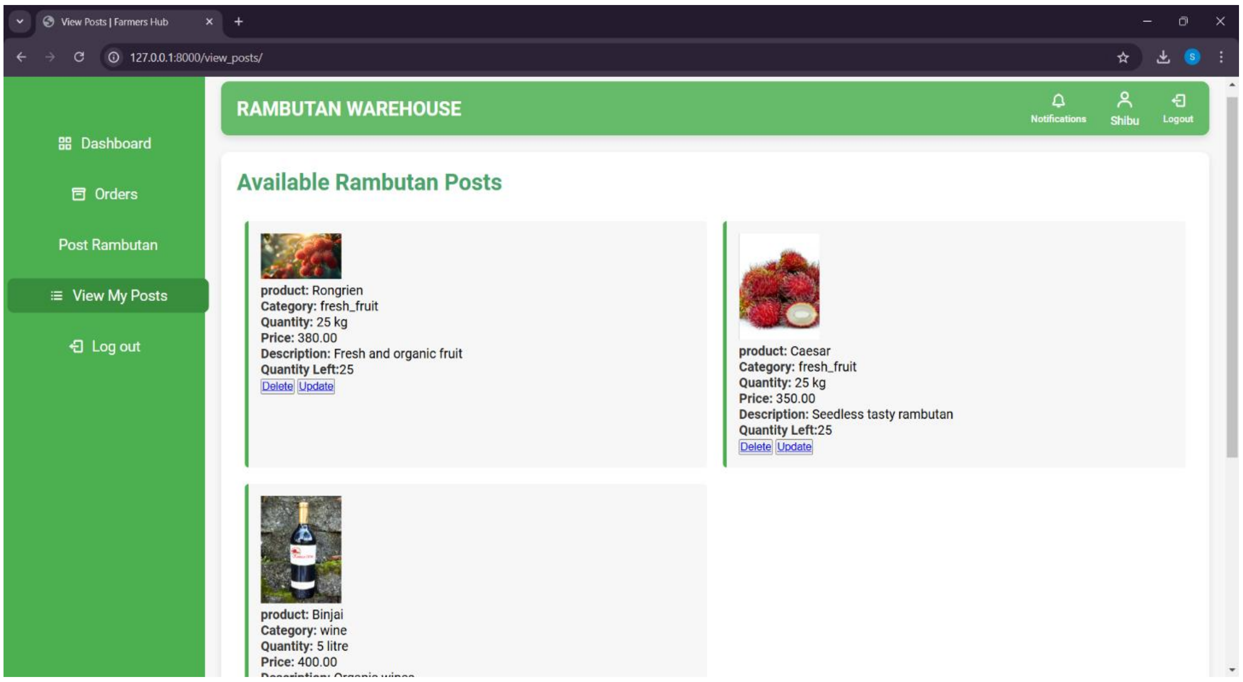


Product: Rambutan N18
Category: Fresh Fruit
Quantity Type: Kilograms
Quantity:
Price:
Image: No file chosen
Description:

© 2024 Rambutan Warehouse. All Rights Reserved.
Privacy Policy | Terms & Conditions | Contact Us

Figure 9.2.2 Post Rambutan

9.2.3 View Post



Available Rambutan Posts

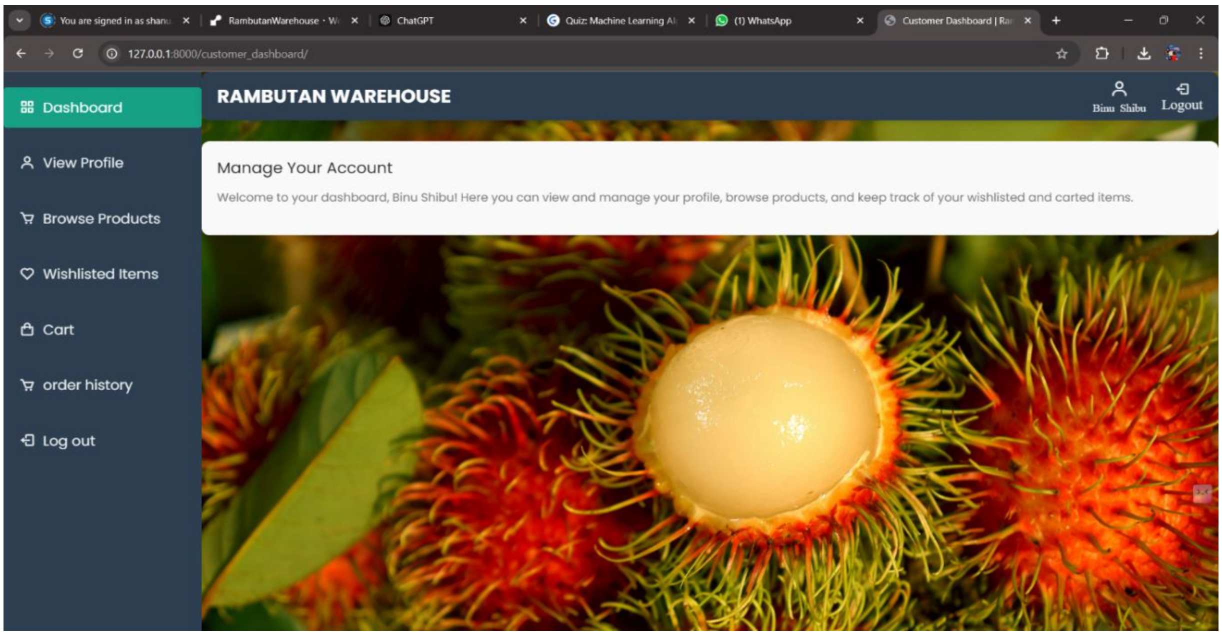
product: Rongrien
Category: fresh_fruit
Quantity: 25 kg
Price: 380.00
Description: Fresh and organic fruit
Quantity Left:25

product: Caesar
Category: fresh_fruit
Quantity: 25 kg
Price: 350.00
Description: Seedless tasty rambutan
Quantity Left:25

product: Binjai
Category: wine
Quantity: 5 litre
Price: 400.00

Figure 9.2.3 View Post

9.2.4 Customer Dashboard

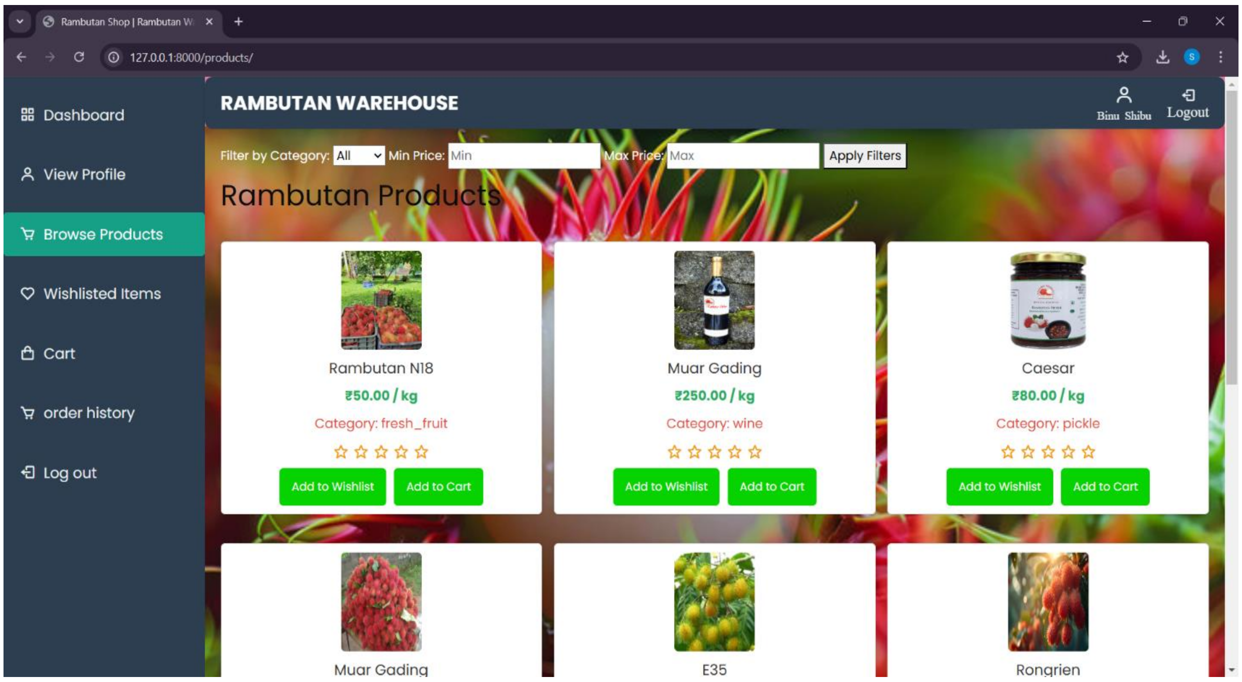


Manage Your Account

Welcome to your dashboard, Binu Shibu! Here you can view and manage your profile, browse products, and keep track of your wishlisted and carted items.

Figure 9.2.4 Customer Dashboard

9.2.5 Browse Products



Rambutan Products

Rambutan N18 ₹50.00 / kg Category: fresh_fruit

Muar Gading ₹250.00 / kg Category: wine

Caesar ₹80.00 / kg Category: pickle

Muar Gading

E35

Rongrien

Figure 9.2.5 Browse Products

9.2.6 Wishlist

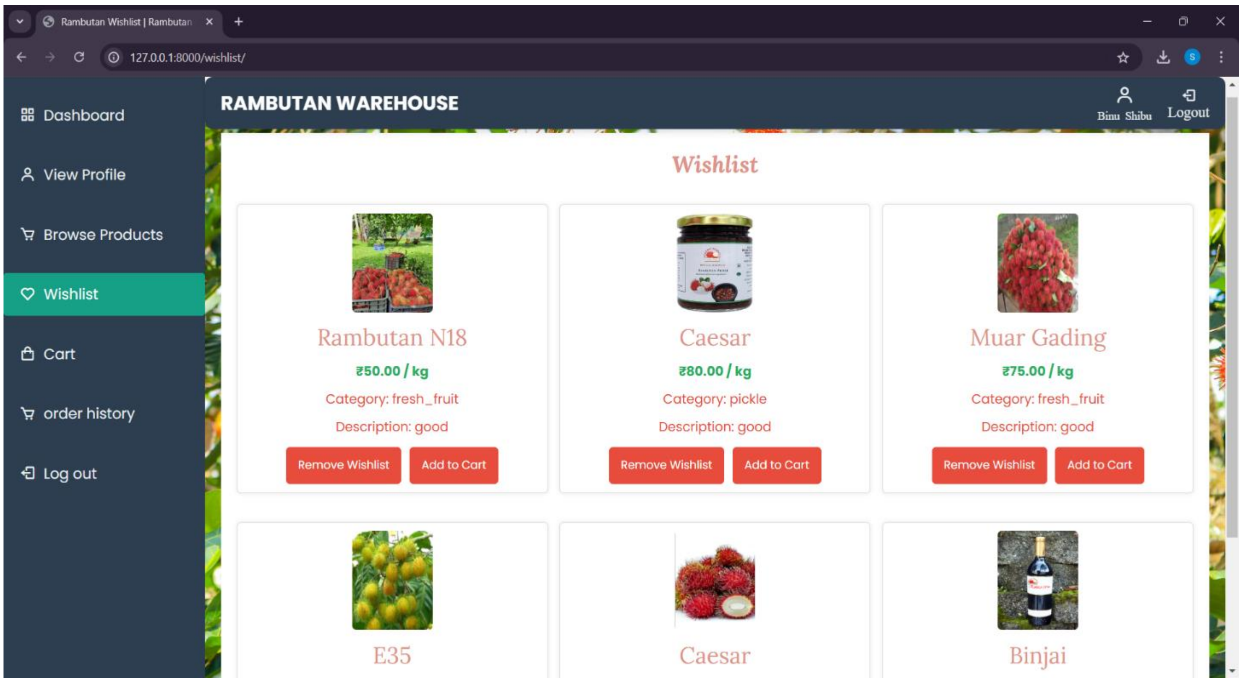


Figure 9.2.6 Wishlist

9.2.7 Cart

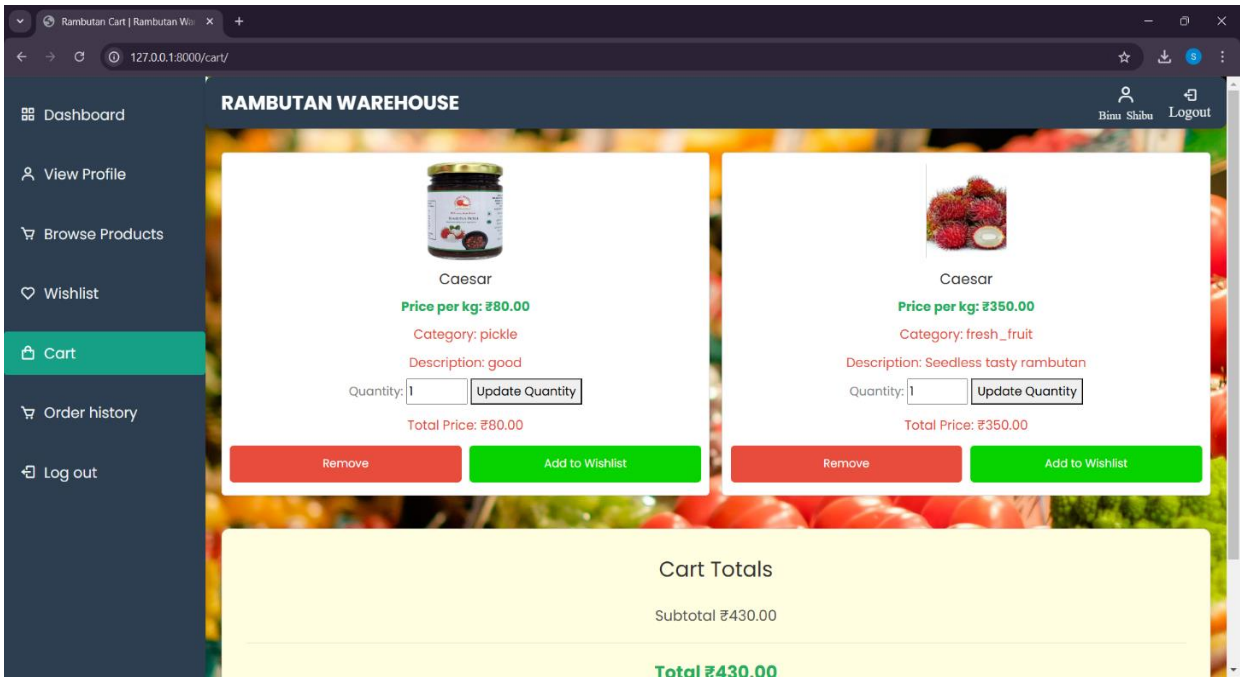


Figure 9.2.7 Cart

9.2.8 Payment

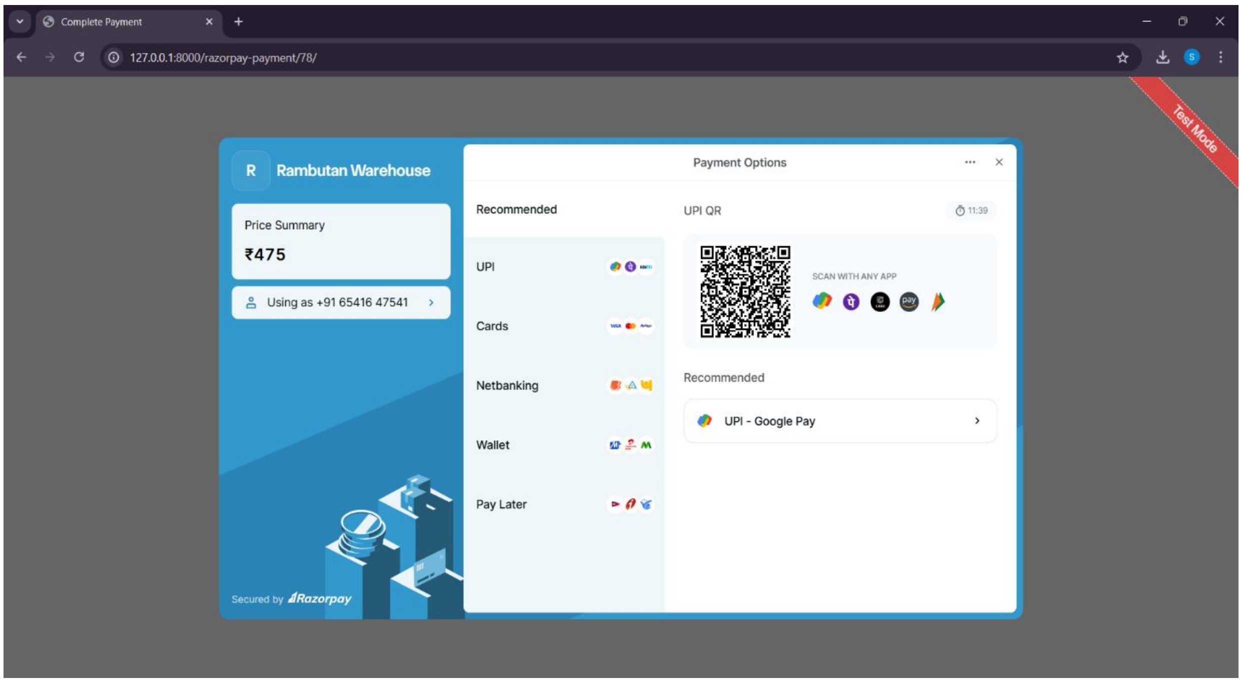


Figure 9.2.8 Payment

9.2.8.1 Payment

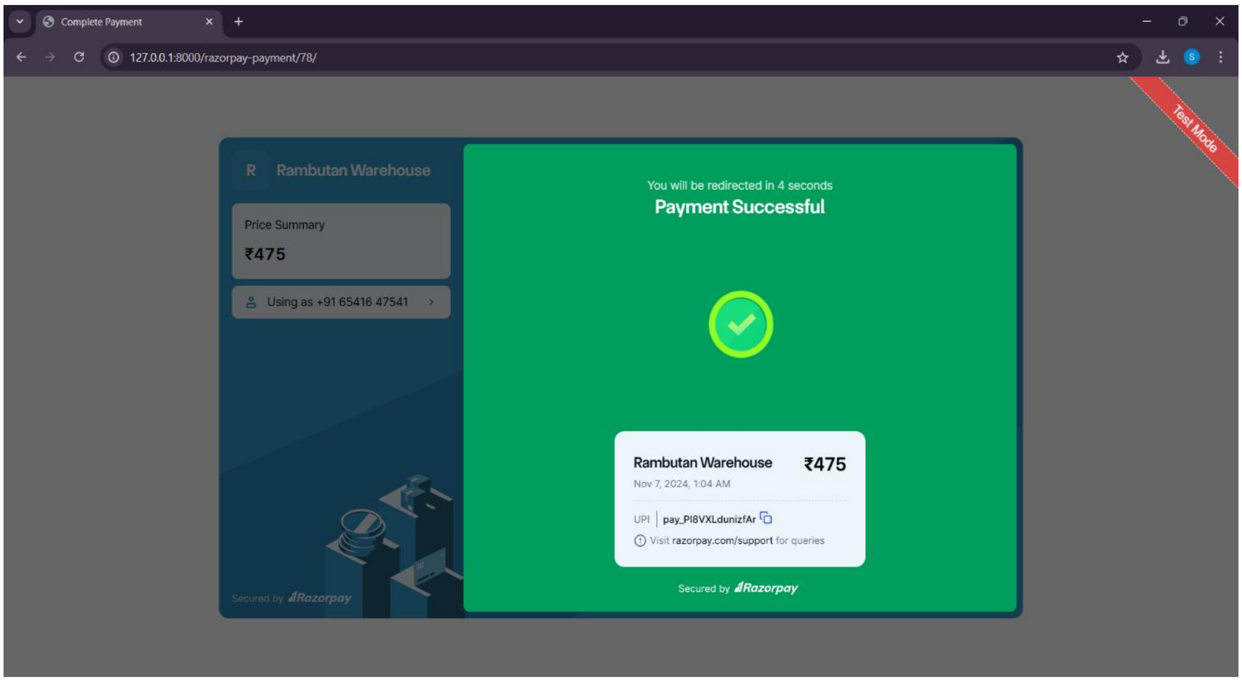
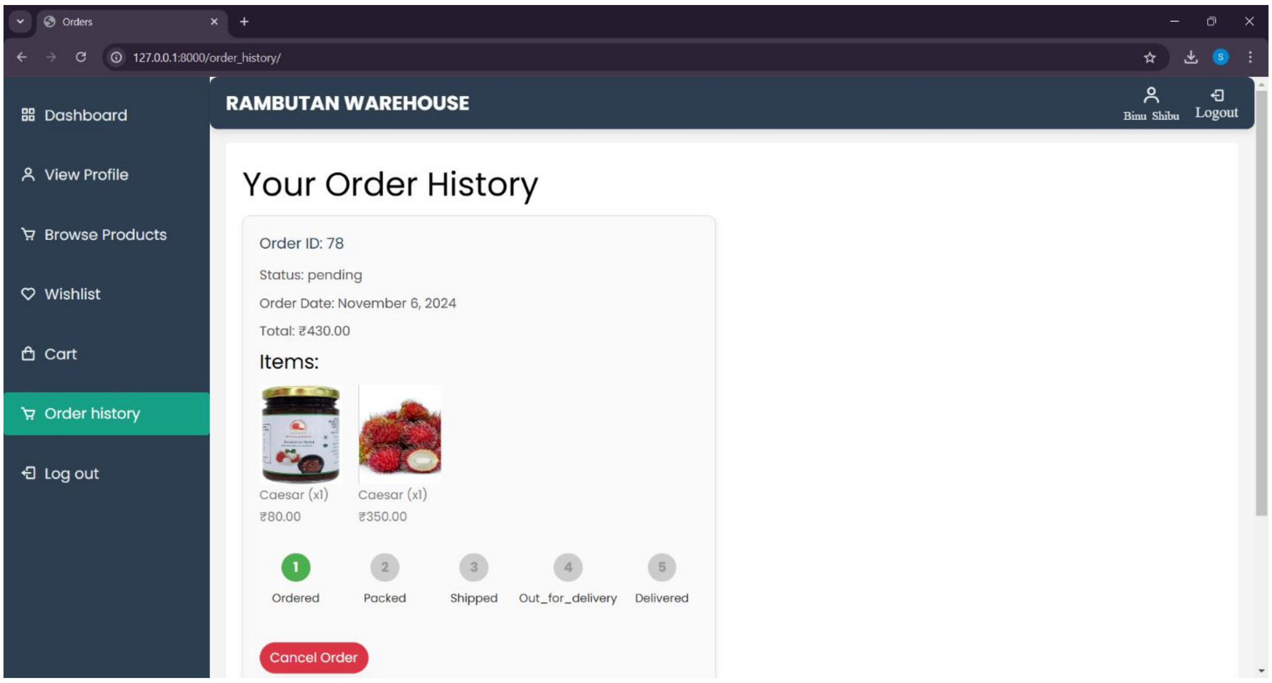


Figure 9.2.8.1 Payment

9.2.9 Order History

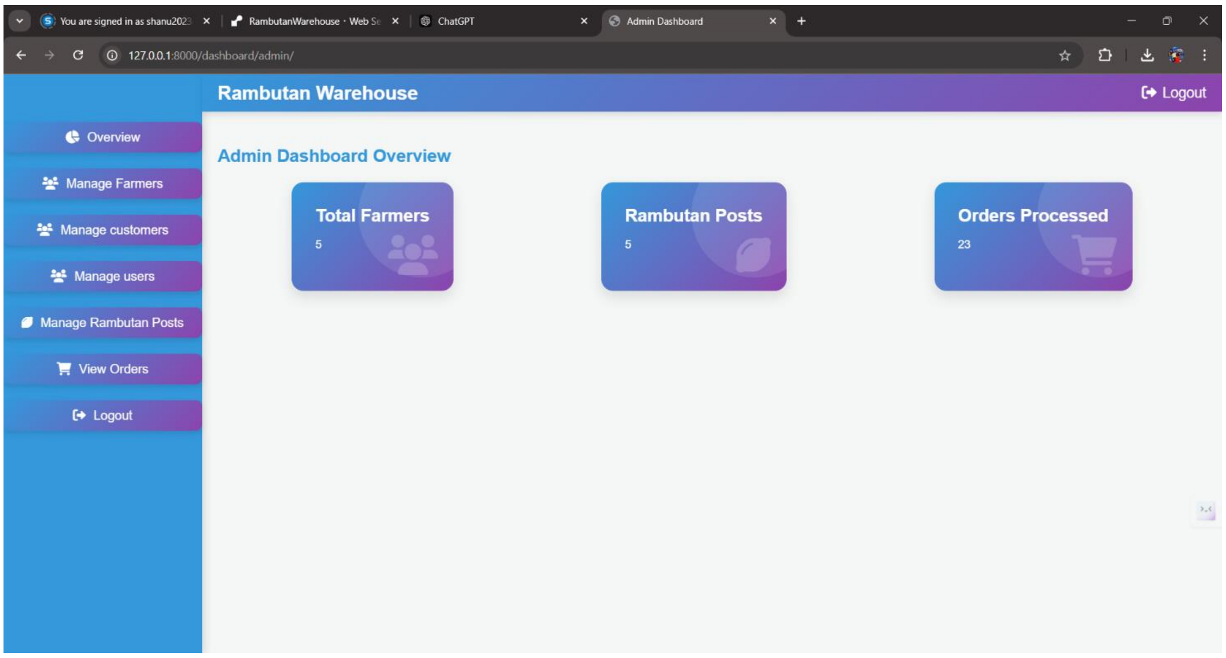


Your Order History

Order ID: 78
Status: pending
Order Date: November 6, 2024
Total: ₹430.00
Items:
Caesar (x1) ₹80.00
Caesar (x1) ₹350.00

Figure 9.2.9 Order History

9.2.10 Admin Dashboard



Admin Dashboard Overview

Total Farmers 5
Rambutan Posts 5
Orders Processed 23

Figure 9.2.10 Admin Dashboard