

Eco Hive Organic Product Store

Mini Project Report

Submitted by

Don K Joseph

Reg. No.: AJC19MCA-I023

In Partial fulfillment for the Award of the Degree of

INTEGRATED MASTER OF COMPUTER APPLICATIONS

(INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



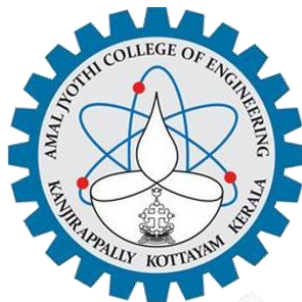
AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

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

2023-2024

DEPARTMENT OF COMPUTER APPLICATIONS
AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY



CERTIFICATE

This is to certify that the Project report, “**ECO HIVE**” is the bona fide work of **DON K JOSEPH (Regno: AJC19MCA-I023)** in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Ms. Gloriya Mathew

Internal Guide

Ms. Meera Rose Mathew

Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

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

Date: 25-10-2023

Don K joseph

KANJIRAPPALLY

Reg: AJC19MCA-I023

ACKNOWLEDGEMENT

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

I 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 **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Gloriya Mathew** for her inspiration and helping hand.

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

DON K JOSEPH

ABSTRACT

The "Eco-Friendly Organic Product Store Website" mini project aims to create an innovative and sustainable online platform. This platform connects environmentally conscious consumers with sellers offering a wide range of eco-friendly and organic products. The website's user-friendly interface empowers customers to explore, purchase, and review eco-conscious products, while also supporting sellers in showcasing their environmentally friendly offerings. Additionally, the project introduces a basic Legal Advisor role to verify and display eco-certification information, promoting transparency and trust.

The problem addressed by this project stems from the increasing demand for eco-friendly products. While consumers seek reliable sources for such products, sellers face challenges in promoting their eco-conscious offerings due to the lack of a dedicated marketplace. The project seeks to bridge this gap by providing a seamless and trustworthy platform for eco-conscious consumers and sellers. The website offers a diverse catalog of eco-friendly and organic products, encourages sustainable shopping, and introduces a Legal Advisor role to verify eco-certification information, ensuring transparency and building consumer trust.

Users in the system have different roles and responsibilities. Administrators are responsible for tasks such as logging into the system, managing users (including adding, updating, and removing them), modifying products, deleting products, and monitoring and moderating reviews and comments.

Customers, on the other hand, are tasked with registering and logging into the system, providing product reviews and ratings, adding items to their shopping cart, searching for products, viewing products with prices, ratings, and descriptions, and accessing order details.

Sellers have their own set of responsibilities, which includes registering and logging into the system, managing their seller profile, adding products with prices, descriptions, and images, applying for certification approval, and checking the status of their application. Lastly, legal advisors are responsible for verifying seller certificates.

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	2
2	SYSTEM STUDY	4
2.1	INTRODUCTION	5
2.2	EXISTING SYSTEM	5
2.3	DRAWBACKS OF EXISTING SYSTEM	7
2.4	PROPOSED SYSTEM	8
2.5	ADVANTAGES OF PROPOSED SYSTEM	9
3	REQUIREMENT ANALYSIS	10
3.1	FEASIBILITY STUDY	11
3.1.1	ECONOMICAL FEASIBILITY	11
3.1.2	TECHNICAL FEASIBILITY	11
3.1.3	BEHAVIORAL FEASIBILITY	12
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	12
3.2	SYSTEM SPECIFICATION	15
3.2.1	HARDWARE SPECIFICATION	15
3.2.2	SOFTWARE SPECIFICATION	15
3.3	SOFTWARE DESCRIPTION	15
3.3.1	DJANGO	15
3.3.2	SQLite	16
4	SYSTEM DESIGN	17
4.1	INTRODUCTION	18
4.2	UML DIAGRAM	18
4.2.1	USE CASE DIAGRAM	19
4.2.2	SEQUENCE DIAGRAM	21
4.2.3	STATE CHART DIAGRAM	23
4.2.4	ACTIVITY DIAGRAM	24
4.2.5	CLASS DIAGRAM	26
4.2.6	OBJECT DIAGRAM	28
4.2.7	COMPONENT DIAGRAM	30

4.2.8	DEPLOYMENT DIAGRAM	32
4.2.9	COLLABORATION DIAGRAM	34
4.3	USER INTERFACE DESIGN USING FIGMA	35
4.4	DATABASE DESIGN	39
5	SYSTEM TESTING	49
5.1	INTRODUCTION	50
5.2	TEST PLAN	50
5.2.1	UNIT TESTING	51
5.2.2	INTEGRATION TESTING	52
5.2.3	VALIDATION TESTING	52
5.2.4	USER ACCEPTANCE TESTING	52
5.2.5	AUTOMATION TESTING	52
5.2.6	SELENIUM TESTING	53
6	IMPLEMENTATION	63
6.1	INTRODUCTION	64
6.2	IMPLEMENTATION PROCEDURE	64
6.2.1	USER TRAINING	65
6.2.2	TRAINING ON APPLICATION SOFTWARE	65
6.2.3	SYSTEM MAINTENANCE	65
7	CONCLUSION & FUTURE SCOPE	66
7.1	CONCLUSION	67
7.2	FUTURE SCOPE	67
8	BIBLIOGRAPHY	69
9	APPENDIX	71
9.1	SAMPLE CODE	72
9.2	SCREEN SHOTS	93

List of Abbreviation

IDE - Integrated Development Environment

HTML - Hyper Text Markup Language.

CSS - Cascading Style Sheet

SQL - Structured Query Language

UML - Unified Modelling Language

JS – JavaScript

AJAX - Asynchronous JavaScript and XML Environment

XML - Extensible Markup Language

UI - User Interface

1NF - First Normal Form

2NF - Second Normal Form

3NF - Third Normal Form

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The "Eco-Friendly Organic Product Store Website" project is a forward-thinking initiative designed to create a user-friendly online platform connecting environmentally conscious consumers with sellers offering a diverse range of eco-friendly and organic products. The platform addresses a growing demand for sustainable consumption and aims to bridge the gap between conscious shoppers and reliable sellers.

The primary objectives of this project are threefold. First, it entails the development of a comprehensive website that simplifies the process of exploring, purchasing, and reviewing eco-friendly and organic products. Second, it provides a valuable space for sellers to exhibit their eco-conscious products and access a wider audience. Third, the project introduces a fundamental Legal Advisor role, responsible for verifying and displaying eco-certification information. This step enhances transparency and fosters trust among consumers regarding the genuineness of eco-friendly claims.

The project's significance lies in its response to the lack of a dedicated marketplace for eco-friendly products, a challenge that hampers both consumers seeking dependable sources and sellers eager to promote their sustainable goods. By amalgamating a user-centric interface, seller support features, and certification verification, the "Eco-Friendly Organic Product Store Website" aims to contribute to sustainable shopping practices, environmental preservation, and the promotion of authentic eco-friendly products. It aspires to be more than just a marketplace; it's a solution that champions the cause of eco-conscious consumption and environmental stewardship.

The "Eco-Friendly Organic Product Store Website" project promises to make a meaningful impact on the landscape of sustainable e-commerce.

1.2 PROJECT SPECIFICATION

- **Administrators**
 - **User Management:** Administrators can create, edit, and remove user accounts.
 - **Product Management:** Admins have the authority to modify and delete product listings.
 - **Moderation:** Administrators can monitor and moderate user reviews and comments.

- **Customers**

- **Registration and Login:** Customers can create accounts and log in securely.
- **Product Browsing and Shopping:** Customers can view, search, and filter products by various criteria such as price, ratings.
- **Product Reviews and Ratings:** Customers can read and write reviews, rate products, and view the ratings of products.
- **Cart Management:** Customers can add products to their shopping carts and proceed to checkout.
- **Searching for Products:** Customers can search for products using keywords.
- **Order Details:** Customers can view order history and details of their purchases.

- **Sellers**

- **Registration and Login:** Sellers can create accounts and log in.
- **Seller Profile Management:** Sellers can manage their profiles, providing information about their businesses and eco-friendly initiatives.
- **Adding Products:** Sellers can add products for sale, including details like price, descriptions, and images.
- **Certification Approval:** Sellers can apply for eco-certification approval for their products.
- **Checking Certification Status:** Sellers can verify the status of their eco-certification applications.

- **Legal Advisors**

- **Certification Verification:** Legal Advisors are responsible for verifying the authenticity of eco-certifications provided by sellers.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The "Eco-Friendly Organic Product Store Website" represents a pioneering online initiative at the intersection of sustainability and e-commerce. This system study delves into the core objectives, scope, and requirements of the project, with the aim of creating an environmentally conscious and user-friendly platform.

In an era where environmental awareness and the pursuit of eco-friendly products have gained paramount importance, the need for a dedicated marketplace that unites eco-conscious consumers with trustworthy sellers is evident. The project's primary goal is to address this need by crafting a robust online platform that seamlessly connects consumers and sellers in pursuit of eco-friendly and organic goods.

This platform intends to empower both sellers and consumers. Sellers, who are eager to share their eco-conscious products with a broader audience, will find a platform for reaching a niche market. Meanwhile, consumers will benefit from an intuitive interface that simplifies the exploration, purchase, and review of eco-friendly items.

Additionally, the introduction of a Legal Advisor role bolsters the project's commitment to authenticity and transparency, enabling the verification of eco-certifications and enhancing consumer trust.

2.2 EXISTING SYSTEM

In the existing system, there is a lack of a dedicated online platform specifically designed to cater to the needs of environmentally conscious consumers and sellers of eco-friendly and organic products. Currently, consumers seeking such products must navigate various general-purpose e-commerce websites, making it challenging to find and verify authentic eco-friendly offerings. Sellers face limitations in promoting their products to a niche market of environmentally conscious individuals.

The absence of a centralized platform for eco-friendly products poses challenges in terms of product discovery, eco-certification verification, and trust. This existing gap in the market highlights the need for our "Eco-Friendly Organic Product Store Website" project, which aims to bridge this divide by offering a dedicated, user-friendly, and trustworthy platform for eco-conscious consumers and sellers.

2.2.1 NATURAL SYSTEM STUDIED

In the conventional approach, environmentally conscious customers seeking eco-friendly and organic products face numerous limitations and inconveniences. Typically, they must resort to traditional methods to obtain such products, which often involve direct contact with sellers or relying on various means to make purchases. However, this system has significant drawbacks. Customers often encounter the inconvenience of traveling long distances to physical stores or sellers offering eco-friendly products. Upon arrival, there's no guarantee that the specific eco-friendly products they seek will be available, resulting in wasted time and effort. This process can be time-consuming and doesn't assure customers of finding the exact eco-conscious products they desire. Moreover, the lack of detailed product information, limited eco-certification verification, inefficient customer-seller interaction, and minimal support for sustainability ratings in the existing system present challenges for informed and eco-conscious purchasing. The limitations of this natural system highlight the pressing need for an innovative and sustainable online platform like the "Eco-Friendly Organic Product Store Website," which aims to rectify these shortcomings by introducing a user-friendly interface, robust eco-certification verification, and comprehensive product information to empower customers and promote eco-conscious consumption.

2.2.2 DESIGNED SYSTEM STUDIED

In contrast to the limitations of the conventional system, the "Eco-Friendly Organic Product Store Website" presents a transformative solution. This innovative system is designed to provide environmentally conscious consumers with a seamless and sustainable online platform that addresses the drawbacks of traditional methods. The new system enhances convenience significantly, eliminating the need for customers to travel long distances to physical stores. It offers a comprehensive range of eco-friendly and organic products, ensuring that customers can conveniently find the exact items they desire from the comfort of their homes. Furthermore, the platform offers an abundance of comprehensive product information, including eco-certifications, sustainability ratings, and detailed product descriptions. This empowers customers to make informed purchasing decisions and trust the authenticity of eco-friendly claims. Unlike the conventional system, the designed system implements a systematic and transparent method for verifying eco-certifications, and it

facilitates efficient and direct interaction between customers and sellers, streamlining communication. The platform also prioritizes sustainability ratings, providing comprehensive information about the eco-friendliness of products, enabling customers to assess and compare the environmental impact of different products with ease. The "Eco-Friendly Organic Product Store Website" represents a significant departure from the limitations of the conventional system, offering an innovative and sustainable solution that empowers environmentally conscious consumers while promoting eco-conscious consumption.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Limited Eco-Friendly Product Access:** The existing system relies on customers physically traveling to brick-and-mortar stores or interacting with sellers through various means, which can be inconvenient and time-consuming. This approach may result in customers not being able to access a comprehensive range of eco-friendly and organic products. The lack of online access restricts their choices and may not guarantee the availability of specific eco-conscious products they seek.
- **Insufficient Product Information:** The conventional approach lacks comprehensive product information, making it challenging for customers to make informed purchasing decisions. Customers often receive limited data regarding eco-certifications, sustainability ratings, and detailed product descriptions, which are critical for assessing the authenticity of eco-friendly claims. Without this information, customers may face difficulties in understanding the environmental impact and benefits of the products they intend to purchase.
- **Absence of Eco-Certification Verification:** Unlike the innovative approach proposed in your project, the existing system lacks a systematic method for verifying eco-certifications. Customers cannot easily verify the accuracy of sellers' eco-friendly claims, which can lead to trust issues and potential misunderstandings. This gap in verification may undermine the credibility of eco-friendly claims in the market.
- **Inefficient Customer-Seller Interaction:** The traditional system often necessitates direct and potentially inefficient interactions between customers and sellers. These interactions may involve back-and-forth communication through various means, resulting in communication gaps and delays. Inefficient customer-seller interactions can be a barrier to smooth and convenient purchasing processes.
- **Limited Support for Sustainability Ratings:** The conventional system does not

prioritize sustainability ratings or offer detailed information about the eco-friendliness of products. This limitation makes it challenging for customers to assess and compare the environmental impact of different products, hindering their ability to make environmentally conscious choices.

2.4 PROPOSED SYSTEM

The "Eco-Friendly Organic Product Store Website" introduces a groundbreaking solution to the challenges inherent in the existing system. This proposed system revolutionizes the way environmentally conscious consumers discover and access eco-friendly and organic products. It presents a user-friendly online platform that mitigates the inconveniences of traditional shopping methods.

Expanded Accessibility: In contrast to the existing system's limitations, the proposed system eliminates the need for physical travel and offers a vast, readily accessible catalog of eco-friendly and organic products. Customers can conveniently explore, purchase, and review these products from the comfort of their homes.

Comprehensive Product Information: The platform provides extensive product details, including eco-certifications, sustainability ratings, and in-depth product descriptions. This wealth of information empowers customers to make well-informed decisions and trust the authenticity of eco-friendly claims.

Eco-Certification Verification: Unlike the absence of systematic verification in the existing system, the proposed system introduces a rigorous process for eco-certification verification, ensuring that customers can rely on the accuracy of sellers' eco-friendly claims.

Efficient Customer-Seller Interaction: The proposed system streamlines communication between customers and sellers, creating a direct and efficient channel for inquiries and assistance. This enhances the overall purchasing experience and reduces communication gaps and delays.

Strong Emphasis on Sustainability Ratings: Sustainability ratings and comprehensive information about product eco-friendliness are at the core of the proposed system. This enables customers to assess, compare, and choose products based on their environmental impact.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Convenience and Accessibility:** The proposed system eliminates the need for customers to travel to physical stores, offering a convenient, 24/7 accessible platform. This enhances the shopping experience, allowing customers to explore and purchase eco-friendly products from anywhere.
- **Comprehensive Product Information:** Unlike the existing system, the proposed system provides in-depth information about eco-friendly products, including eco-certifications, sustainability ratings, and detailed product descriptions. This empowers customers to make well-informed choices.
- **Verified Eco-Certifications:** The system introduces a rigorous verification process, assuring customers that eco-friendly claims made by sellers are accurate. This builds trust and confidence in the authenticity of eco-conscious products.
- **Efficient Customer-Seller Interaction:** The proposed system streamlines communication between customers and sellers, allowing for efficient inquiries and assistance. This direct interaction reduces delays, enhances customer support, and fosters a sense of community.
- **Sustainability Ratings:** With a strong emphasis on sustainability ratings, customers can easily assess and compare the environmental impact of products, making eco-conscious choices more accessible.
- **Support for Sellers:** Seller's benefit from an expansive platform to showcase their eco-friendly products and gain access to a broader audience, enhancing their reach and business growth.
- **Legal Advisor Role:** The introduction of Legal Advisors ensures the reliability of eco-certifications, promoting transparency and trust in the authenticity of eco-friendly claims.
- **Promotion of Eco-Conscious Consumption:** The system actively promotes sustainable shopping practices and contributes to environmental conservation by connecting environmentally conscious consumers with eco-friendly products.
- **User-Centric Interface:** The proposed system offers a user-friendly interface that simplifies product browsing, selection, and purchase, enhancing the overall customer experience.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A systematic assessment of a project or venture's viability and likelihood of success is called a feasibility study. In order to determine whether the project is viable and worthwhile, it evaluates a variety of elements, including technical, economic, legal, and operational considerations. Decision-makers can decide whether to move on with the project or make necessary adjustments by weighing the costs, benefits, potential hazards, and market demand. The study aids in the early identification of potential barriers and difficulties, enabling stakeholders to reduce risks and maximize resource allocation. In the end, a thorough feasibility analysis gives project planning a strong foundation and lessens the possibility of costly errors and failures during execution.

3.1.1 Economical Feasibility

Conducting an economic feasibility analysis is crucial to determining the time and financial commitment required to evaluate the worth of a new project. It comprises a thorough investigation of each matter that might have an impact on how the effort turns out. The suggested approach, Eco-Friendly Organic Product Store Website's economic feasibility assessment shows that it is a workable business venture. The anticipated expenditures for building the website, integrating eco-certifications, and covering operational costs line up with prospective revenue sources including product sales and seller commissions. An optimistic financial outlook is reflected in the predicted return on investment (ROI) and appropriate payback duration. The platform's linkage with escalating consumer demand for environmentally friendly goods and potential intangible advantages further strengthen its feasibility from an economic standpoint. With these things taken into account, the project is financially viable and shows promise for meeting the association's financial goals. Continuous success and financial sustainability will be ensured by regular financial assessments.

3.1.2 Technical Feasibility

The technical feasibility study for the "Eco-Friendly Organic Product Store Website" demonstrates a high probability of successful implementation. The development teams possess the required expertise and resources to build the website using modern web technologies. Integration of eco-certification verification systems is feasible, ensuring the

accuracy of organic product listings. The platform's scalability is assured through efficient coding practices and scalable infrastructure. Technical assessments indicate that the project aligns with industry standards and can handle increasing user traffic and product listings. Overall, the technical feasibility study confirms that the project is technically achievable and in line with the association's objectives.

3.1.3 Behavioral Feasibility

User Acceptance: Customer acceptance is a critical aspect of this feasibility study. The platform is designed to enhance the shopping experience, provide valuable product information, and streamline the purchasing process. The success of the project depends on users readily adopting the system for their eco-friendly shopping needs.

Seller Engagement: Sellers' willingness to join the platform, manage their profiles, and list eco-friendly products is crucial. Behavioral feasibility will assess their readiness to embrace this new digital marketplace and engage proactively.

Administrative Adaptation: Administrators play a pivotal role in managing the platform. Their acceptance of the system and its functionalities is fundamental to the project's smooth operation. They should be willing to use the system to manage users, products, and reviews effectively.

Legal Advisor Participation: The role of legal advisors in verifying eco-certifications is central to the platform's credibility. Behavioral feasibility evaluates the willingness of qualified professionals to take on this role and ensure eco-friendly claims are genuine.

User Training and Support: The project should consider the need for user training and support to ensure seamless adoption. Behavioral feasibility also assesses the effectiveness of support mechanisms in helping users navigate the platform.

3.1.4 Feasibility Study Questionnaire

1. Project Overview

The project's primary objective is to develop an online storefront that provides clients with a wide selection of organic and environmentally friendly goods. The website will serve customers that care about the environment and are looking for real, sustainable products. The platform will make it easier for buyers and sellers to communicate, giving consumers access to a wide range of environmentally friendly goods. While the Legal Advisor function oversees the eco-certification verification procedure to ensure product authenticity, sellers

will have the chance to display their items. The platform will be run by administrators, who will be in charge of user accounts, product listings, and any potential conflicts. The initiative seeks to foster environmentally beneficial behaviors, stimulate the use of organic products, and support environmental sustainability.

2. To What Extent the System Is Proposed For?

The "Eco-Friendly Organic Product Store Website" system under consideration intends to establish an online marketplace where users can look for and purchase eco-friendly and organic goods. Sellers are able to display their goods, and legal counsel guarantees the legitimacy of those goods. Platform management and problem-solving are done by administrators. The concept promotes sustainable lifestyle choices and links buyers interested in the environment with businesses selling genuinely eco-friendly goods.

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

Potential Customers, General Visitors, Online Shoppers

4. List the Modules included in your System?

Admin, Guest Users, Customers, Sellers, Legal Advisor

5. Identify the users in your project?

Guest Users, Customers

6. Who owns the system?

Administrator

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

e-commerce industry

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

Antony Mathew (Organica, Kottayam)

9. Questionnaire to collect details about the project?

1. How do you buy products for the shop?

Buying from the sellers.

2. How do you decide on the pricing of your appliances?

The pricing model is based on a combination of factors, including the manufacturer's suggested retail price (MSRP), our purchasing costs.

3. How the payment is collected from the user?

By cash or credit card facility

4. Do you use any inventory management software or systems to keep track inventory?

I manually manage my inventory and keep track of sales using pen and paper.

5. How do you source and verify the authenticity of the organic products you sell?

I personally visit farms and suppliers to ensure organic certifications and verify the sourcing practices of the products I sell.

6. Would you find value in a system that allows you to showcase eco-certification details for your products to build customer trust?

Yes, showcasing eco-certification details would build trust among customers and increase sales.

7. What are the main challenges you face in running your organic shop?

The main challenges I face include reaching a broader customer base, managing inventory efficiently, and verifying the authenticity of organic products.

8. What marketing strategies do you think would be effective in attracting customers to an online platform for organic products?

Online advertisements, social media promotions, and partnerships with eco-friendly organizations would be effective marketing strategies.

9. How important is customer reviews and ratings for your products in building customer confidence?

Customer reviews and ratings are critical in building customer confidence and driving sales.

10. How do you handle product deliveries?

In addition to in-house delivery, also partner with reliable courier services for certain deliveries.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - Intel core i5

RAM - 8 GB

Hard disk - 500 GB

3.2.2 Software Specification

Front End - HTML5, Bootstrap, CSS

Back End - Python

Database - SQLite

Client on PC - Windows 7 and above.

Technologies used - DJANGO, HTML5, JS, AJAX, J Query, CSS, Bootstrap

3.3 SOFTWARE DESCRIPTION

3.3.1 Django Framework:

The Django Framework stands as a popular and robust web framework specifically designed for Python developers. Its reputation is built on its attributes of simplicity, clean code, and rapid development capabilities. Django follows the Model-Template-Views (MTV) architectural pattern, which bears resemblance to the Model-View-Controller (MVC) pattern seen in other frameworks. Notably, it incorporates an Object-Relational Mapping (ORM) system, which simplifies database interactions by representing database tables as Python objects. This abstraction effectively eliminates the need for writing raw SQL queries, significantly simplifying database operations.

Django's offerings include a built-in administrative interface that streamlines content management. Its URL routing system empowers developers to establish clean and user-friendly URLs for web applications. Moreover, Django comprehensively supports form

handling, data validation, and user authentication, reducing the intricacies of common web development tasks. Emphasizing security, Django comes equipped with built-in defenses against common web vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).

The framework's modular architecture encourages extensibility, enabling developers to seamlessly integrate third-party packages or create custom components. Thanks to a thriving community and a wide array of reusable packages, Django proves to be a versatile choice suitable for web development projects of varying sizes and complexities.

3.3.2 SQLite:

SQLite stands out as a lightweight, self-contained, and serverless relational database management system. Unlike conventional databases, SQLite doesn't necessitate a separate server; instead, it's embedded directly within the application. One of its remarkable attributes is the self-contained nature of SQLite databases, where the entire database resides within a single file. This characteristic simplifies database management, backups, and transfers. Despite its lightweight architecture, SQLite adheres to the principles of ACID compliance, ensuring data integrity through transaction support, which makes it apt for multi-user environments.

SQLite boasts cross-platform compatibility, working seamlessly on various operating systems, making it a versatile choice for applications targeting diverse platforms. Its small code footprint and minimal resource usage render it suitable for resource-constrained environments, particularly mobile devices. SQLite is renowned for its speed and efficiency, especially when handling read-heavy workloads. It finds common application in mobile applications, desktop software, and embedded systems, where deploying a full-fledged database server might be excessive, and portability remains a critical concern.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

The development of any system or product begins with the design phase, a crucial step in ensuring the efficiency and effectiveness of the final outcome. In the context of our project, the "Eco-Friendly Organic Product Store Website," a well-executed design is fundamental. It's a creative process that involves the application of various methodologies and concepts to define the system comprehensively, facilitating its actual implementation.

Regardless of the chosen development model, the design phase holds paramount importance in software engineering. It aims to provide the architectural details essential for constructing a robust system, forming the technical foundation of the entire software engineering process. Our project has undergone a meticulous design phase, optimizing aspects of effectiveness, performance, and accuracy. This design phase transforms a user-oriented document into a blueprint tailored for programmers and database specialists, ensuring the project's successful realization.

4.2 UML DIAGRAM

A standardized dialect known as Unified Modeling Language (UML) serves as a vital tool for conceptualizing, defining, designing, and describing software systems. The Object Management Group (OMG) was responsible for the development of UML, and the initial UML 1.0 draft was introduced in January 1997. It's important to note that UML is distinct from programming languages like Java, C++, and COBOL. UML is a generic visual modeling language employed for computer program systems and a pictorial language used for program designs. Although UML is widely employed in representing software systems, its utility extends beyond software and encompasses various applications, including manufacturing processes.

UML encompasses a range of diagrams, each tailored to specific purposes:

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- State chart diagram

- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A use case diagram is a visual representation that illustrates how users and other external entities interact with the internal components of a system. Its primary function is to identify, outline, and organize a system's functional requirements from the perspective of its users. Typically constructed using the Unified Modeling Language (UML), which serves as a standard language for modeling physical entities and systems.

Use cases have a broad range of applications, including defining essential requirements, validating hardware designs, testing and debugging software, creating online help references, or fulfilling customer support roles. Examples of use cases in the context of product sales include customer support, product purchase, catalog updates, and payment processing.

A use case diagram is composed of system boundaries, actors, use cases, and their interconnections. The system boundary delineates the system's limits concerning its environment. Actors are characterized based on their roles and represent individuals or systems that interact with the system. Use cases detail the specific actions or behaviors that actors perform within or in close proximity to the system. The diagram also visually displays the relationships between actors and use cases, along with the use cases themselves.

Use case diagrams are visual tools that capture a system's functional requirements. When creating a use case diagram, it is important to adhere to these guidelines for an efficient and effective representation:

- Select descriptive names for use cases that accurately convey their functionalities.
- Assign appropriate names to actors to clarify their roles in the system.
- Ensure that relationships and dependencies are clearly depicted in the diagram.
- Avoid overcomplicating the diagram with unnecessary relationships, focusing on identifying the essential requirements.
- Use notes when necessary to provide additional context or explanations.

By following these guidelines, a clear and concise use case diagram can be created, accurately depicting the system's functional requirements.

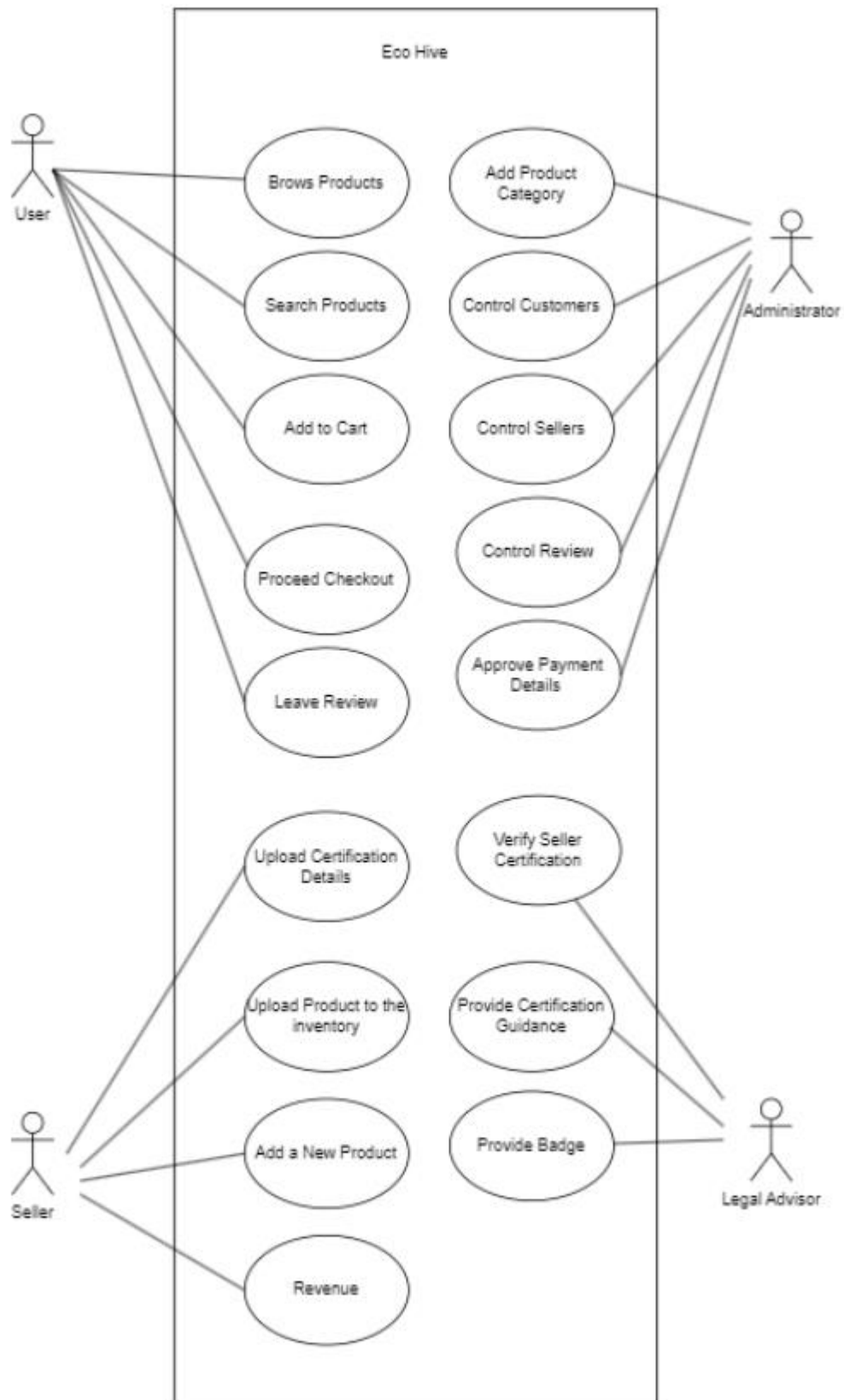


Fig 1: Use case diagram for Eco Hive

4.2.2 SEQUENCE DIAGRAM

A sequence diagram, categorized as an interaction diagram, is a graphical representation of how various system components interact with one another over a series of messages or actions. These diagrams are also referred to as event scenarios or event scenario diagrams. Sequence diagrams are commonly used in software engineering to comprehend the requirements of both new and existing systems, aiding in the visualization of object control relationships and the identification of systemic issues.

Notations in Sequence Diagrams:

- i. **Actors:** In UML, actors symbolize roles that interact with the system and its objects. Typically, actors exist outside the system being depicted in the UML diagram. They can play various roles, including those of external entities or human users, and are represented as stick figures in UML diagrams. Depending on the scenario being modeled, a sequence diagram can involve multiple actors.
- ii. **Lifelines:** A lifeline in a sequence diagram is a vertical dashed line representing the lifespan of an object involved in the interaction. Each lifeline denotes an individual participant in the sequence of events and is labeled with the participant's name. Lifelines visually portray the timeline of events for each participant, drawn as vertical lines from the activation point to the deactivation point.
- iii. **Messages:** Messages are fundamental in sequence diagrams, portraying interactions and communication between system objects or components. Messages come in various types, including synchronous and asynchronous messages, create and delete messages, self-messages, reply messages, found messages, and lost messages. Guards are used to model conditions and constraints on message flow.
- iv. **Guards:** Guards in UML serve to model conditions and are utilized to regulate the flow of messages when specific conditions are met. They are crucial in informing software developers about constraints and limitations within a system or a particular process.

Applications of Sequence Diagrams:

- Modeling and visualizing complex functions, operations, or procedures.
- Detailing UML use case diagrams.
- Understanding the intricate functionality of current or future systems.
- Visualizing the flow of messages and tasks between system objects or components.

- Overall, sequence diagrams are valuable for representing the interaction flow between objects in a system, aiding both business stakeholders and software engineers in comprehending and communicating system requirements and behaviors.

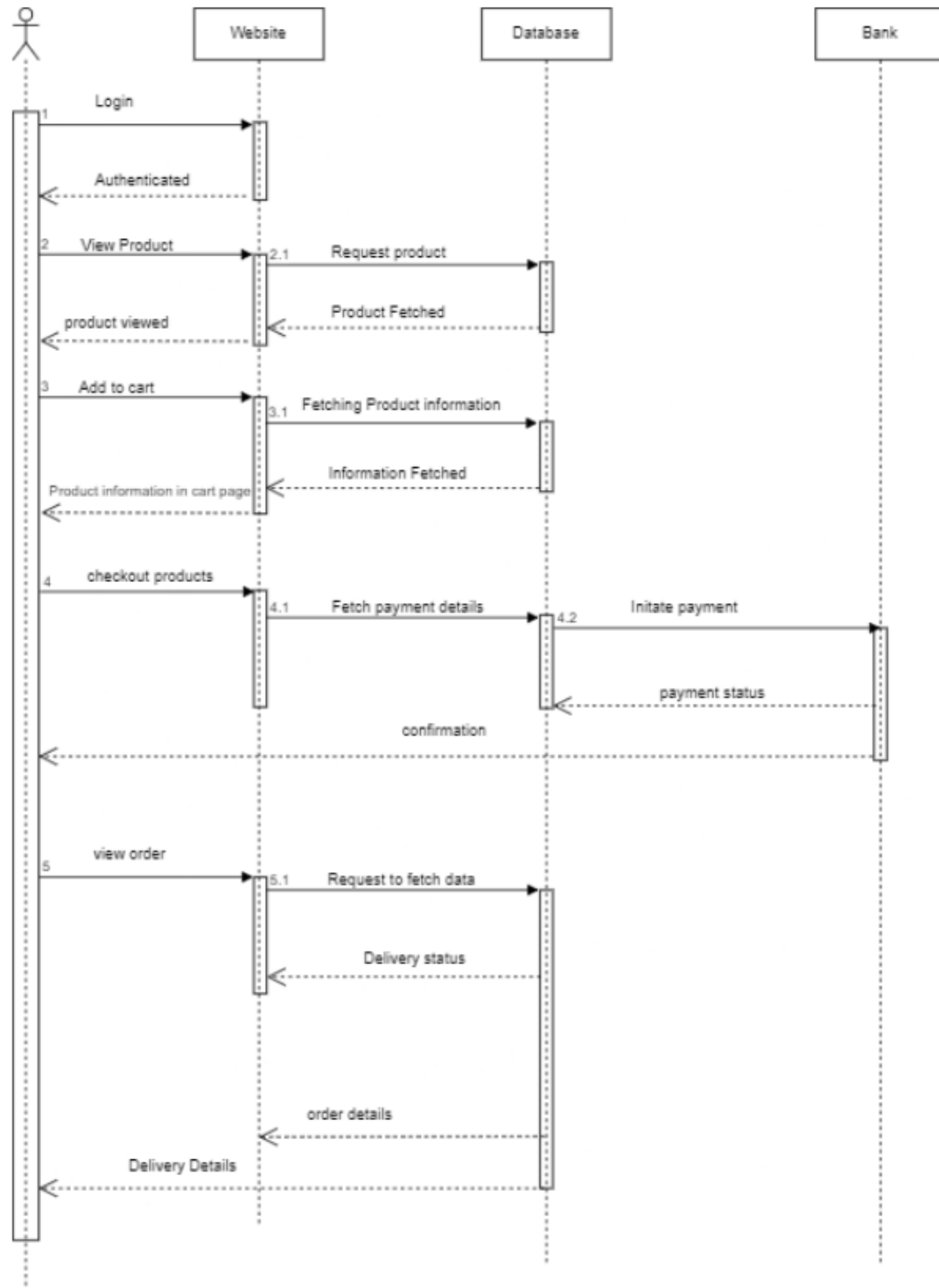


Fig 2: Sequence diagram for Eco Hive

4.2.3 State Chart Diagram

A state diagram, often created using the Unified Modeling Language (UML), is a visual representation that illustrates the different states an object can assume and the transitions between these states. It is alternatively known as a state machine diagram or a state chart diagram.

Understanding State Chart Diagrams:

A State Chart Diagram, a type of behavioral diagram in UML, provides insight into the behavior of a system or an object over time. It comprises various key elements:

- **Initial State:** This state marks the system or object's starting point and is depicted by a solid black circle.
- **State:** These elements describe the system or object's current condition at a specific moment and are symbolized by rectangles with rounded corners.
- **Transition:** Represented by arrows, transitions illustrate the movement of the system or object from one state to another.
- **Event and Action:** An event acts as a trigger that initiates a transition, while an action denotes the behavior or consequence of that transition.
- **Signal:** Signals, triggered by events, are messages sent to a state, prompting a transition.
- **Final State:** The State Chart Diagram concludes with a Final State element, recognizable by a solid black circle with a dot inside. It signifies the completion of the system or object's behavior.

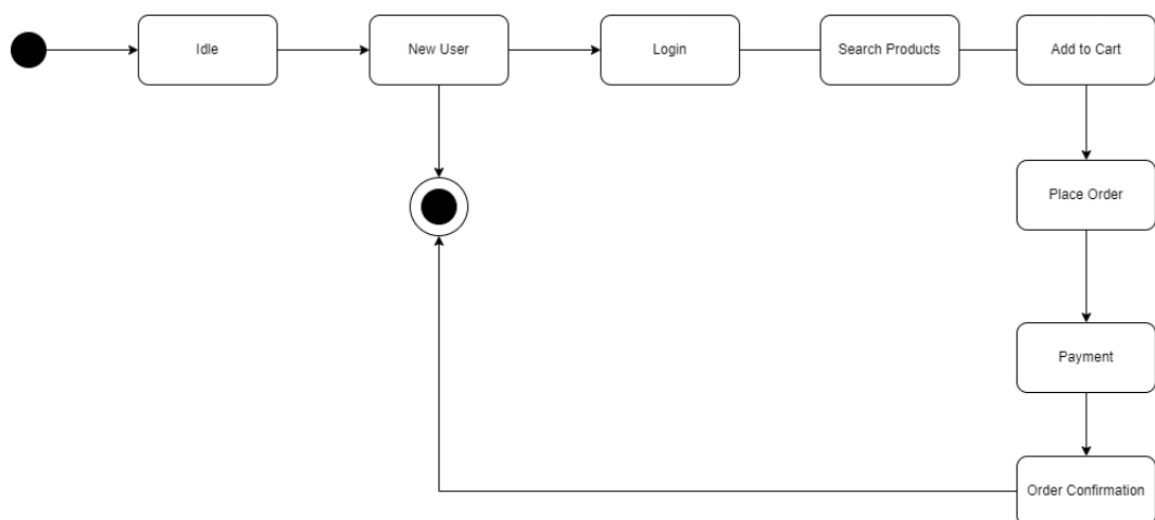


Fig 3: State Chart Diagram for Eco Hive

4.2.4 Activity Diagram

An activity diagram serves as a visual representation of a workflow, illustrating the sequential progression from one activity to another. Each activity, which is a system operation, leads to another in the control flow. These flows can take various forms, such as parallel, concurrent, or branched, and activity diagrams employ functions like branching and joining to manage these different types of flow control. Activity diagrams fall under the category of behavior diagrams and depict a system's behavior. They reveal the control flow from the starting point to the endpoint, highlighting the diverse decision paths encountered during activity execution.

Key Elements in an Activity Diagram:

- **Initial Node:** This serves as the outset of the activity diagram and is represented by a black circle.
- **Activity:** Each task or action performed by the system or entity is depicted as a rectangle with rounded corners.
- **Control Flow:** These arrows symbolize the sequence of activities or actions carried out by the system or entity.
- **Decision Node:** Recognized by a diamond shape, it signifies a point in the activity flow where decisions or branches occur.
- **Merge Node:** This element consolidates multiple branches of the activity flow into a unified flow and is depicted as a diamond shape with a plus sign inside.
- **Fork Node:** It's used to divide the activity flow into several parallel flows and is represented by a solid black circle with multiple arrows.
- **Join Node:** This component combines multiple parallel flows back into a single flow and is illustrated as a solid black circle with multiple arrows converging towards it.
- **Final Node:** The conclusion of the activity diagram, denoted by a black circle with a dot inside.
- **Object Flow:** Depicts the movement of objects or data between activities, signified by a dashed arrow.

Activity diagrams prove invaluable for simplifying intricate processes, identifying potential challenges, and effectively conveying process flows to stakeholders and project team members.

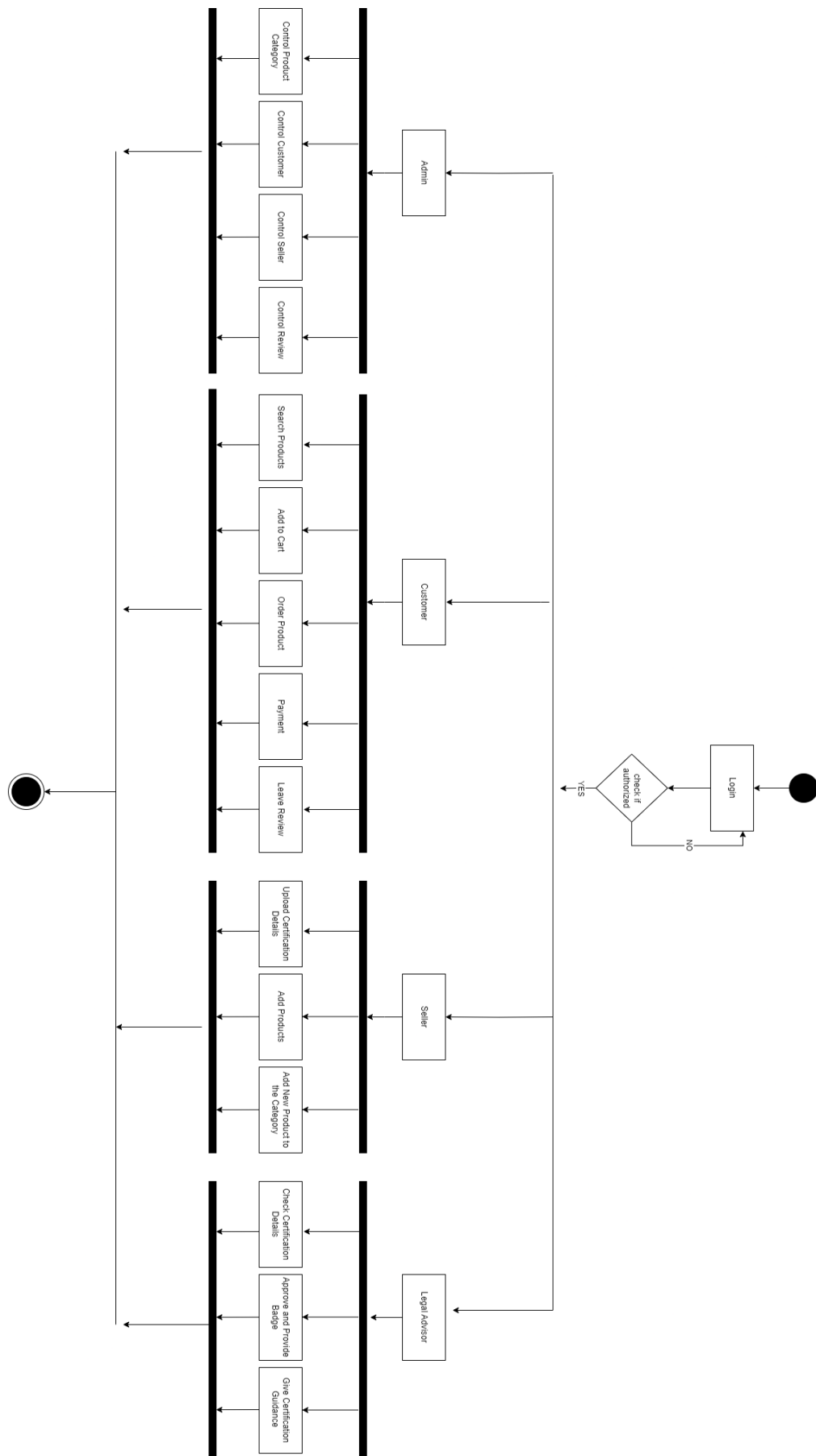


Fig 4: Activity Diagram for Eco Hive

4.2.5 Class Diagram

Class diagrams are foundational elements in object-oriented modeling and serve as the primary tool for conceptually visualizing the structure of an application. Additionally, class diagrams are valuable for detailed modeling that can be translated into actual programming code. They also find application in data modeling.

Key Aspects of a Class Diagram:

- **Class:** Representing a blueprint or template for creating objects, classes are portrayed as rectangles containing the class name, attributes, and methods.
- **Interface:** Interfaces, which define a contract between a class and the external world through abstract methods, are depicted as circles with the interface name inside.
- **Object:** Objects, representing instances of a class with both state and behavior, are shown as rectangles with the object name inside.
- **Association:** Signifying a relationship between two classes, associations are visualized as lines with optional directionality, multiplicity, and role names.
- **Aggregation:** Illustrating a part-whole relationship, where a whole is composed of parts, aggregations are represented by diamond shapes on the aggregator side.
- **Composition:** Depicting a stronger form of aggregation where parts cannot exist without the whole, compositions are symbolized by filled diamond shapes on the aggregator side.
- **Inheritance:** Representing a relationship between a superclass and its subclasses, this signifies an "is-a" relationship and is shown as a line with an open arrowhead pointing from the subclass to the superclass.
- **Dependency:** Indicating a relationship where changes in one class may affect another, dependencies are visualized as dashed lines with arrowheads pointing from the dependent class to the independent class.
- **Multiplicity:** Showing the number of class instances associated with another class, multiplicity is displayed as a range of values near the association or aggregation line.

Class diagrams are indispensable in designing and modeling object-oriented software systems. They offer a visual representation of the system's structure, functionality, and the relationships between its objects. They streamline software development, maintenance, and enhance communication among team members.

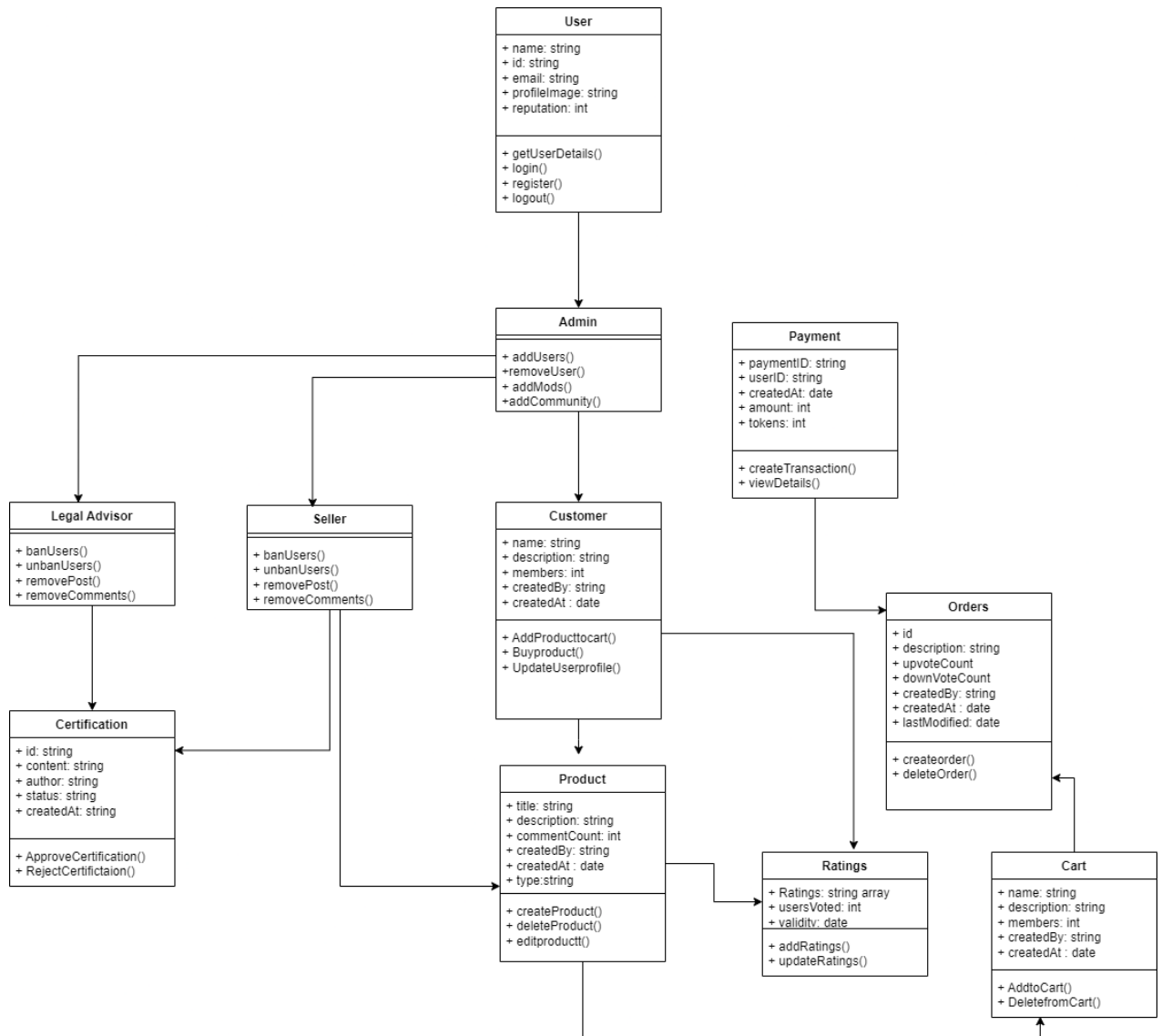


Fig 5: Class Diagram for Eco Hive

4.2.6 Object Diagram

Class diagrams and object diagrams are closely intertwined in the realm of object-oriented modeling. Object diagrams can be considered as real-world snapshots derived from class diagrams, capturing a system's state at a particular moment. Both types of diagrams employ identical concepts and notations to delineate a system's structure. While class diagrams focus on modeling the system's structure, encompassing classes, attributes, and methods, object diagrams illustrate a collection of objects and their interconnections at a precise instant.

- Object diagrams, a subset of structural UML diagrams, display class instances and their associations. The primary elements within an object diagram encompass:
- Object: Representing a concrete instance of a class that embodies a specific entity in the system, denoted as a rectangle housing the object's name.
- Class: Serving as a blueprint or prototype for generating objects, classes are portrayed as rectangles with compartments for the class name, attributes, and methods.
- Link: Indicating a relationship between two objects, signifying a connection or association via a line that can bear optional labels.
- Attribute: Conveying an object's properties or characteristics that describe its state, presented as a name-value pair within the object's rectangle.
- Value: Identifying a specific instance or setting of an attribute, appearing as a value within the attribute's name-value pair.
- Operation: Representing an action or behavior that an object can execute, typically manifesting as a method name within the class rectangle.
- Multiplicity: Signifying the number of class instances associated with another class.

Object diagrams serve as effective tools for visualizing object relationships and their attributes within a system. They are invaluable for comprehending a system's behavior at a specific point in time and for unearthing potential issues or inefficiencies within the system.

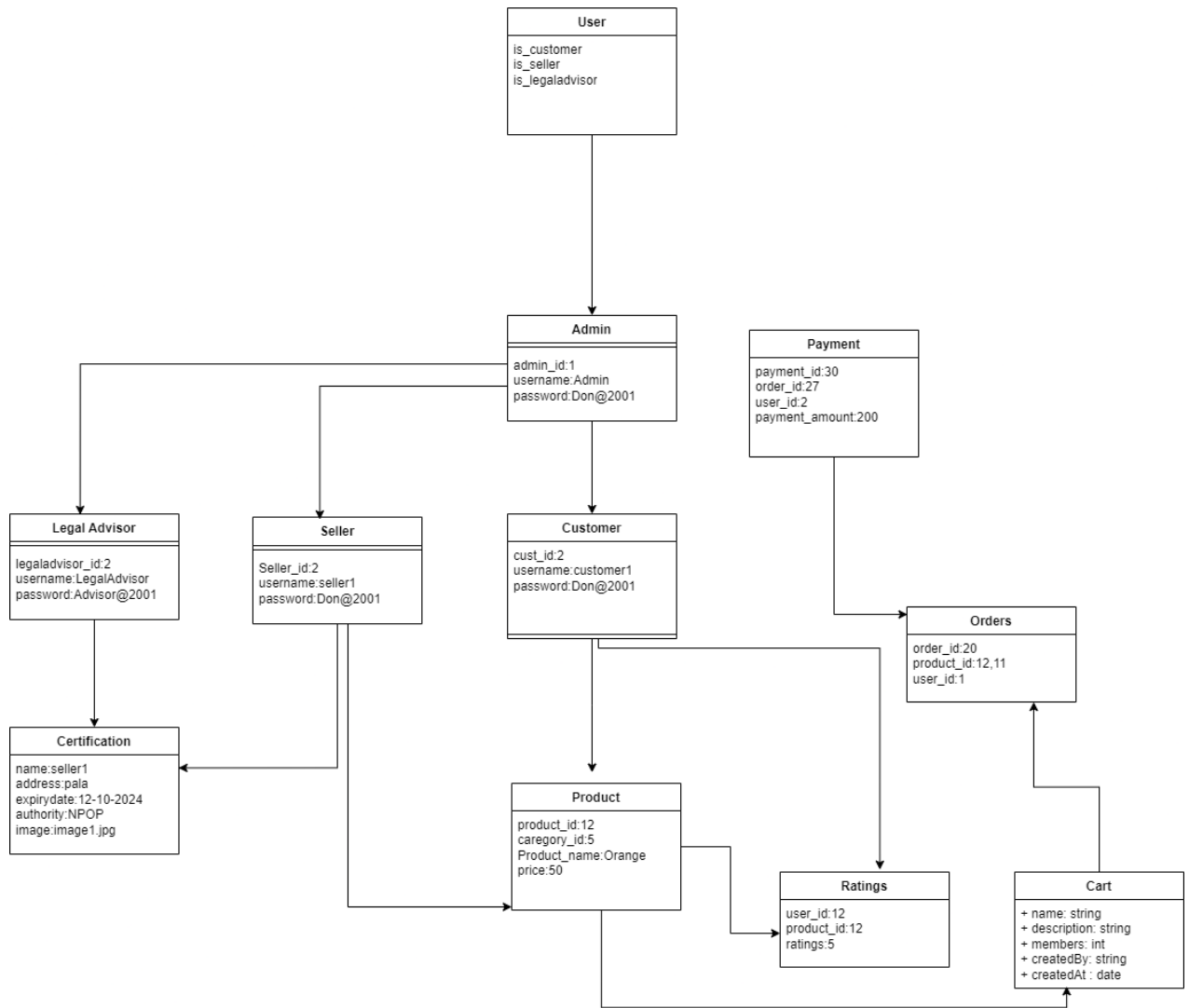


Fig 6: Object Diagram for Eco Hie

4.2.7 Component Diagram

A UML component diagram serves as a visual blueprint for showcasing how diverse parts come together to build intricate software systems. This diagram is a powerful tool for illustrating the inner workings of complex systems with numerous components. Developers can utilize component diagrams to gain a clear picture of a software system's internal structure and how these individual components collaborate to achieve specific tasks.

The key elements of a component diagram include:

- **Component:** These are like self-contained units of functionality within a system, offering interfaces for interactions with other components. They appear as rectangles with the component's name inside.
- **Interface:** Think of these as contracts specifying a set of methods that components can use to communicate. They're depicted as circles with the interface's name inside.
- **Port:** Ports serve as points of interaction between a component and its environment or other components. They're typically small squares located on a component's boundary.
- **Connector:** Connectors link two components, facilitating communication and data exchange. They're shown as lines with optional markings and labels.
- **Dependency:** This relationship indicates that one component relies on another for functionality or implementation. It's represented by a dashed line with an arrow pointing from the dependent component to the independent one.
- **Association:** Associations signify connections or links between components. They're illustrated as lines connecting two components, with optional indicators for direction, multiplicity, and roles.
- **Provided/Required Interface:** Components can either provide interfaces for others to use (represented by lollipops) or require interfaces from other components to operate correctly (depicted as half-circles).

Component diagrams excel at modeling software system architectures, aiding in the detection of potential design issues and enhancements. Additionally, they're effective tools for conveying a system's structure and behavior to stakeholders, including developers and project managers.

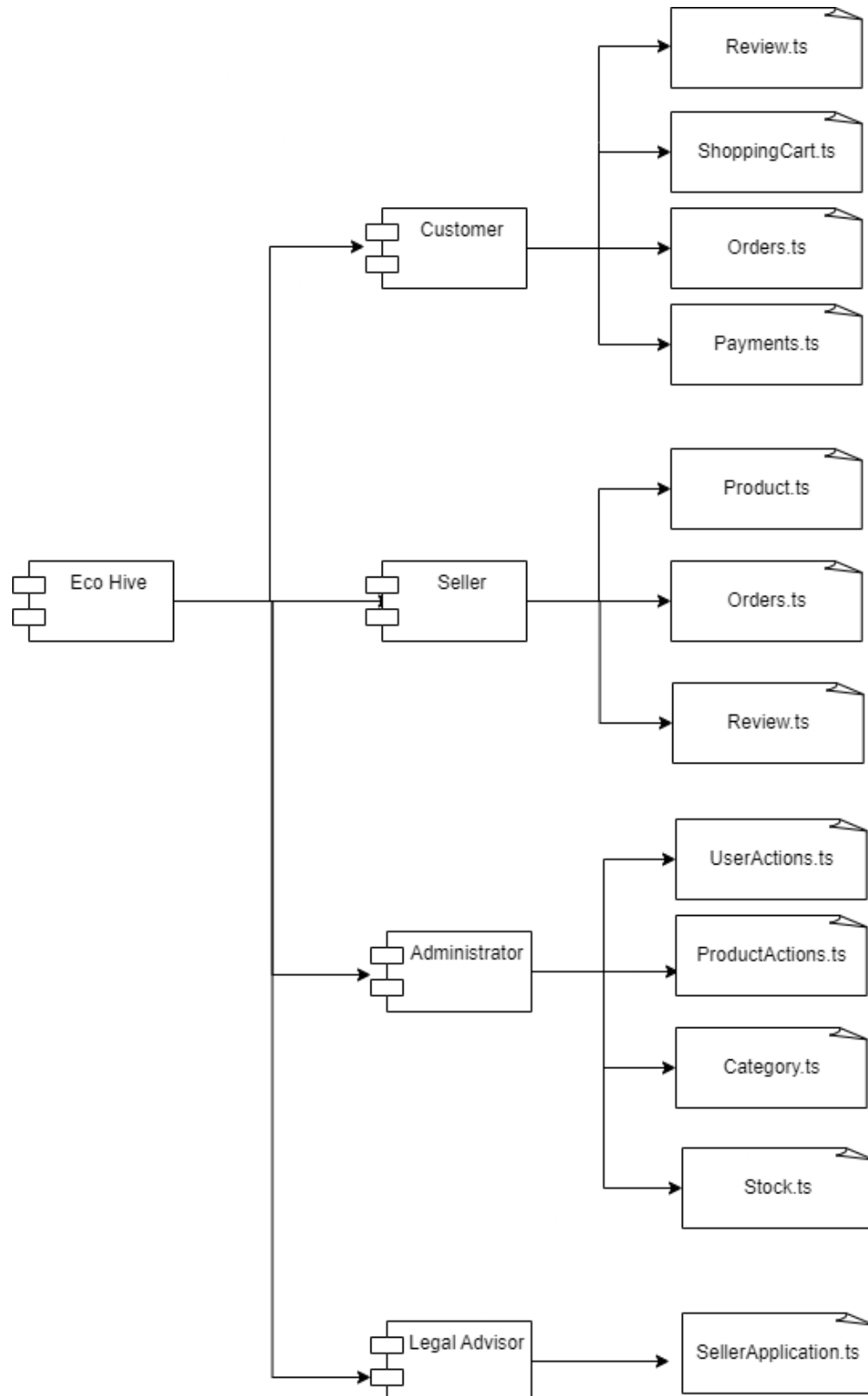


Fig 7: Component Diagram for Eco Hive

4.2.8 Deployment Diagram

A deployment diagram, a type of UML diagram, zeros in on the tangible hardware used for software deployment. It offers a static snapshot of a system's deployment, featuring nodes and their connections. This diagram serves to link the software's architecture to the physical hardware setup, detailing how the software operates on various nodes. Communication paths illustrate how nodes interact. Unlike other UML diagrams that delve into a system's logical components, the deployment diagram spotlights hardware configuration.

Key components of a deployment diagram include:

- **Node:** These are physical or virtual machines where components or artifacts are deployed. They're depicted as boxes with the node's name inside.
- **Component:** These are software elements responsible for specific functions or services. They appear as rectangles with the component's name inside.
- **Artifact:** Artifacts represent physical data pieces used or generated by a component. They're depicted as rectangles with the artifact's name inside.
- **Deployment Specification:** This describes how components or artifacts are placed on nodes, including details like location, version, and configuration parameters.
- **Association:** Associations represent relationships between nodes and components or artifacts, signifying deployment dependencies. They're shown as lines connecting the two elements, with optional details on direction, multiplicity, and roles.
- **Communication Path:** These paths depict connections between nodes, like network links or communication channels. They're shown as lines with optional labels and markers.

Deployment diagrams are invaluable for visualizing a system's physical structure and identifying potential deployment challenges or bottlenecks. They also assist in planning deployment strategies and optimizing hardware resource utilization.

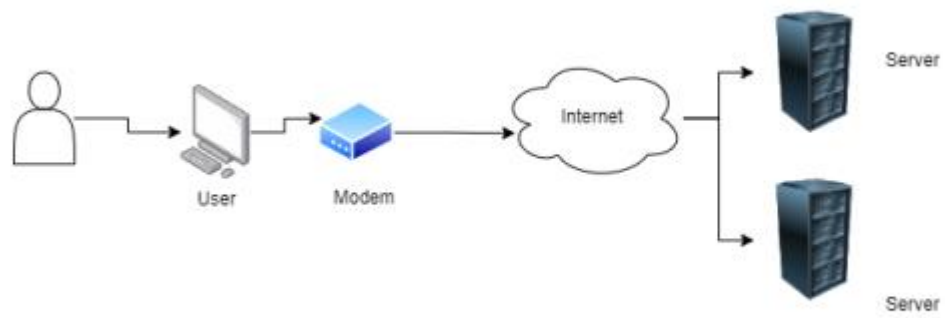


Fig 8: Deployment Diagram for Eco Hive

4.2.9 Collaboration Diagram

A collaboration diagram is a visual tool for illustrating relationships between objects within a system. While it conveys the same information as a sequence diagram, it uses a different approach. Instead of mapping the flow of messages between objects, a collaboration diagram provides a snapshot of the object structure within the system. These diagrams are grounded in object-oriented programming, where objects possess various attributes and are interconnected. Essentially, collaboration diagrams offer a visual representation of a system's object architecture.

Key elements in a collaboration diagram include:

- **Objects:** Objects are symbolized with their names and class (with an underline), separated by a colon. In a collaboration diagram, objects represent instances of a class and specify both the instance's name and class. Not every class requires an object representation, and a single class may have multiple objects. Naming objects is essential for distinguishing them from one another.
- **Actors:** Actors play a significant role in collaboration diagrams as they initiate interactions. Each actor is identified by a name and a specific role. In the diagram, one actor starts the use case.
- **Links:** Links are instances of associations connecting objects and actors. They illustrate the relationships between objects through which messages are exchanged. Links are depicted as solid lines, aiding objects in navigating to other objects.
- **Messages:** Messages signify the communication between objects, carrying information and being labeled with a sequence number. These messages are conveyed as labeled arrows near the links, sent from the sender to the receiver. Message direction must be specific, and the receiver should understand the message.

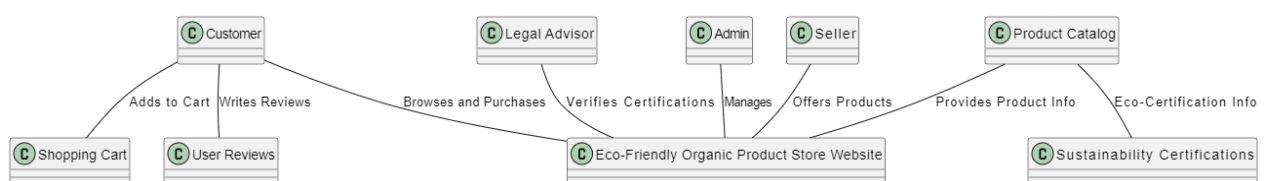


Fig 9: Collaboration Diagram for Eco Hive

4.3 USER INTERFACE DESIGN USING FIGMA

1. Form Name: Registration Page

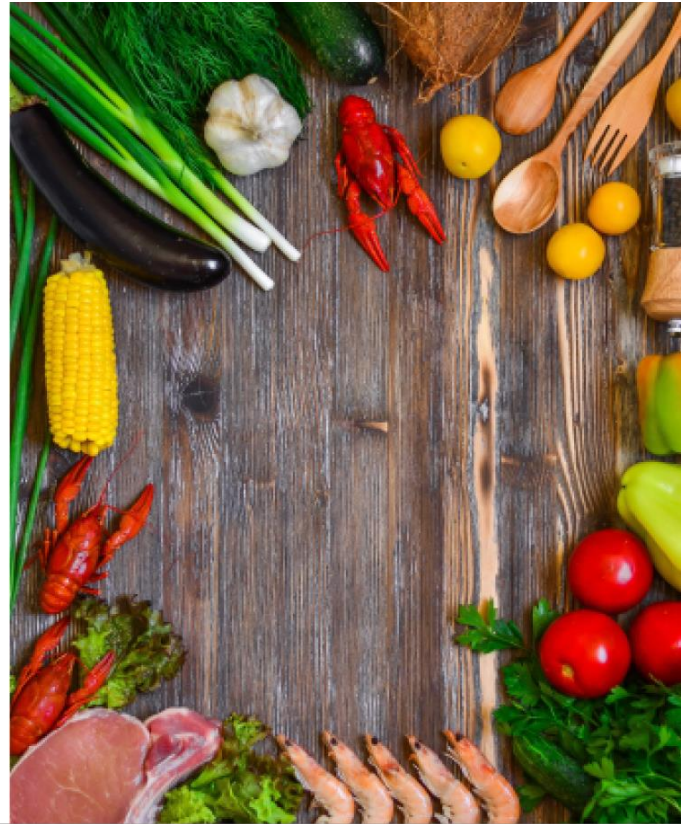
Create an account

Connecting you with nature's pure essence.

Create account

 Sign up with Google

Already have an account ? [Log in](#)



2. Form Name: Login Page

Login

Welcome back Please enter your details .

[Forgot password ?](#)

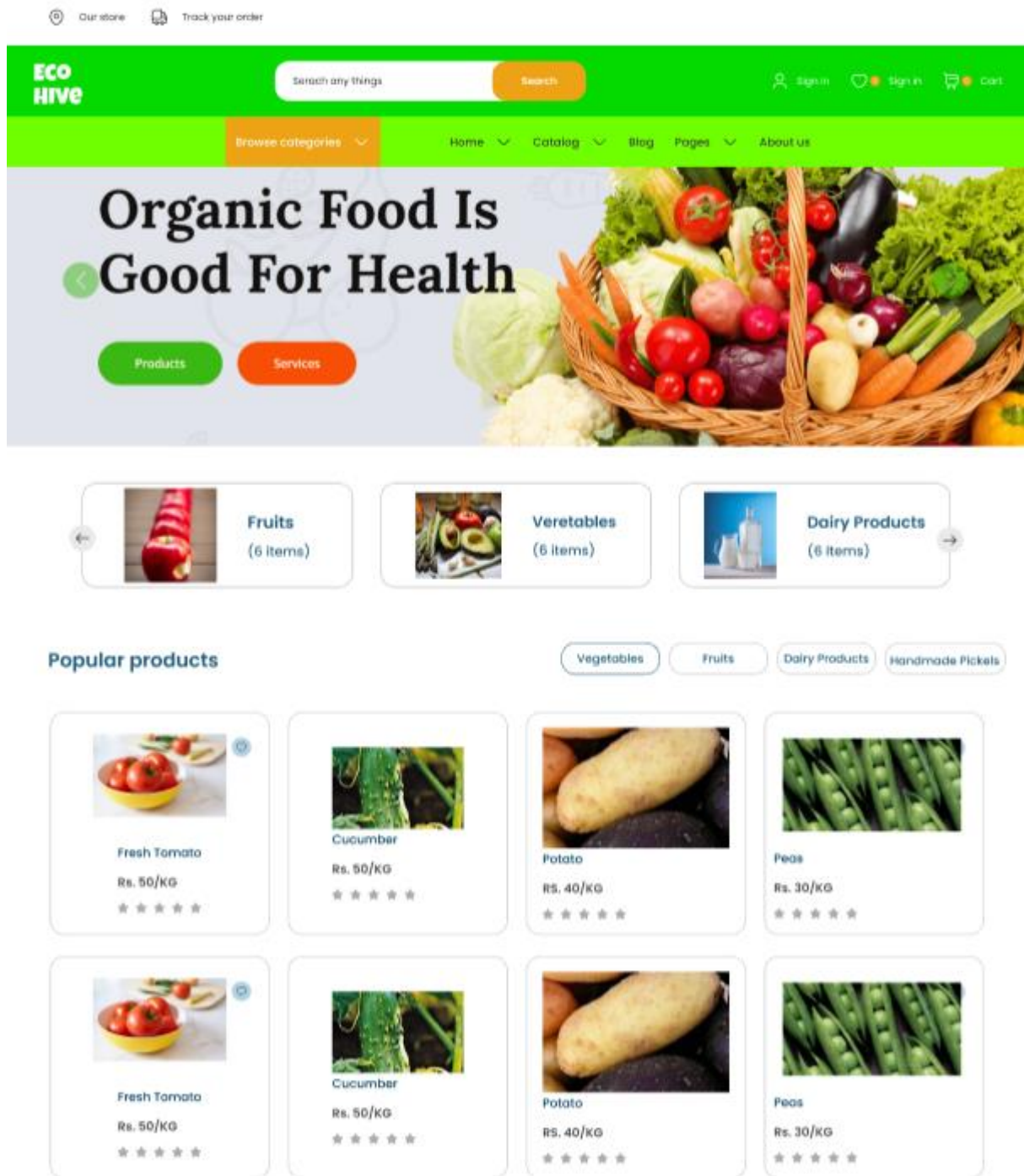
Login

 Login with Google

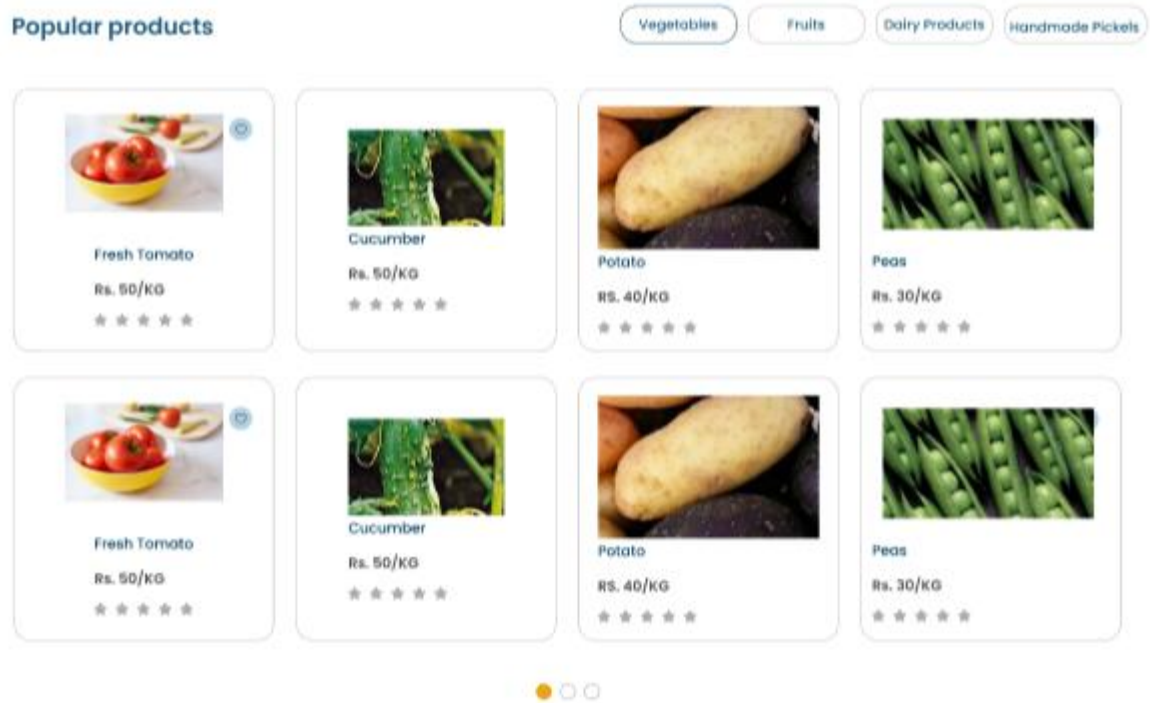
Don't have an account? [Sign up](#)



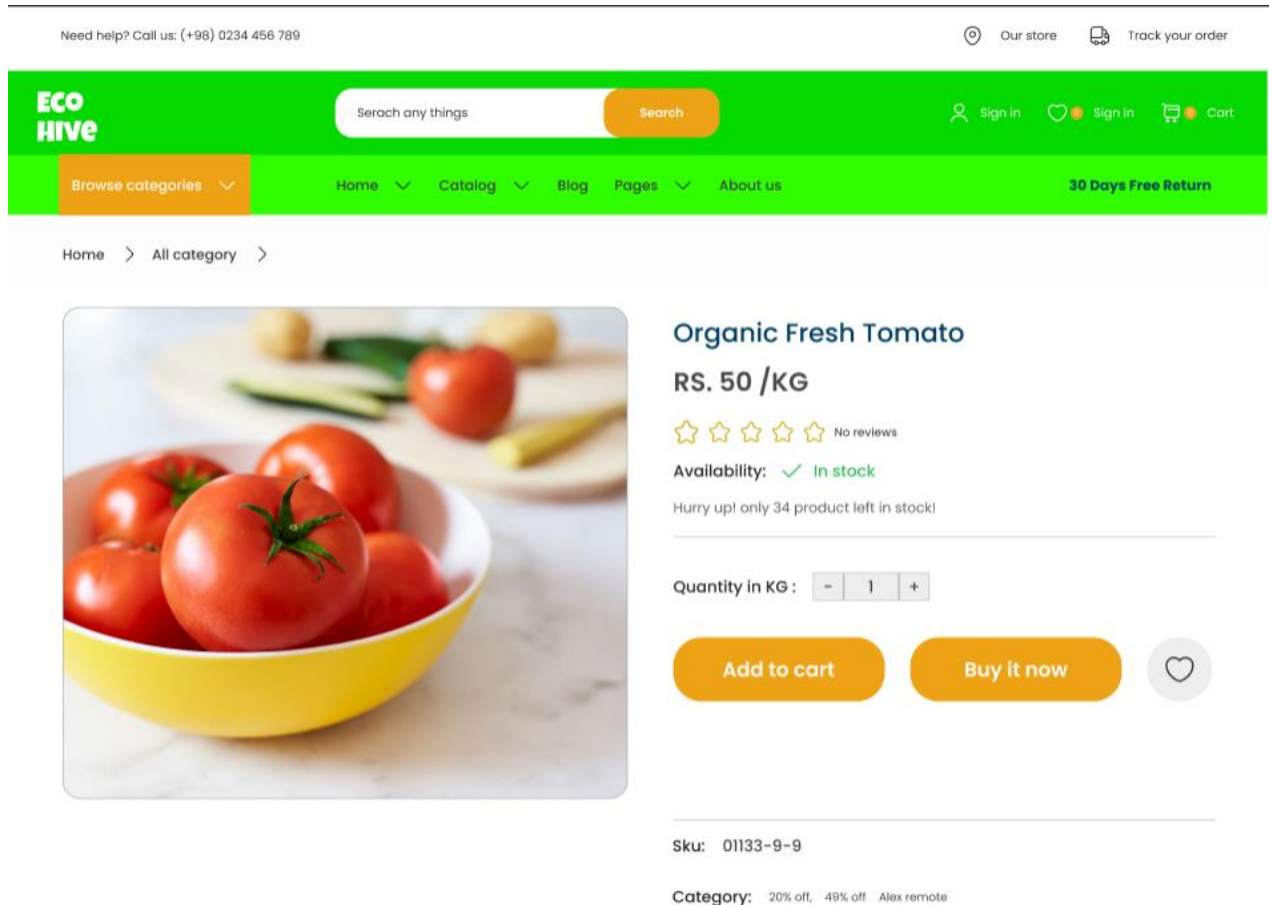
3. Form Name: Home Page



4. Form Name: Product View Page



5. Form Name: Single Product View Page



6. Form Name: Cart Page

Need help? Call us: (+98) 0234 456 789

Our store Track your order

Eco Hive

Search any things Search



Sign in Sign in Cart

Browse categories

Home Catalog Blog Pages About us

30 Days Free Return

Home > All category >

Product	Price	Quantity	Subtotal
 Tomato	50/KG	<div>- 1 +</div>	50 ×
 Cucumber	30	<div>- 1 +</div>	30 ×

Continue shopping

Update cart

Clear cart

Cart total

Subtotal \$ 23,20

Enter coupon code Apply

County

Total amount \$ 23,20

Proceed to ckeckout

4.4 DATABASE DESIGN

A database is a structured repository of information, meticulously organized to facilitate easy access, efficient management, and periodic updates. Ensuring the security of data is a fundamental priority in any database system. The process of designing a database typically unfolds in two distinct stages. In the initial phase, user requirements are meticulously gathered to craft a database that seamlessly aligns with these requirements, ensuring clarity and precision. This step is often referred to as information-level design, conducted independently of any specific Database Management System (DBMS). In the subsequent stage, the design is translated from an information-level design into a concrete design tailored to a particular DBMS, which will serve as the foundation for constructing the system. This stage is commonly termed physical-level design, taking into account the specific characteristics and requirements of the chosen DBMS.

In parallel with the system design, there exists the crucial task of database design, which pursues two primary objectives: upholding data integrity and achieving data independence.

4.4.1 Relational Database Management System (RDBMS)

A Relational Database Management System (RDBMS) is a widely used database system that organizes data into tables, making it easy to establish connections with other stored datasets. These tables can hold extensive datasets, ranging from hundreds to millions of rows, where each row is formally referred to as a "tuple," a column heading as an "attribute," and the table as a "relation." Within a relational database, multiple tables coexist, each having a distinct name, and every row within a table represents a set of related values.

In the realm of a relational database, relationships between tables are predefined to ensure the integrity of both referential and entity relationships. A "domain," denoted as 'D,' is a collection of indivisible atomic values, typically defined by selecting a data type from which these values originate. Providing a name for a domain aid in comprehending the nature of the values it encompasses. It's crucial to note that each value within a relation is indivisible and cannot be further subdivided.

Within a relational database, table relationships are established through the use of keys, with the primary key and foreign key holding significant roles. These keys are pivotal in establishing entity integrity, which guarantees that primary keys do not contain null values, and referential integrity, which mandates that each unique foreign key value corresponds to a matching primary key value within the same domain. Additionally, there exist other key types

like super keys and candidate keys, each serving specific purposes.

4.4.2 Normalization

Data is initially grouped together in the simplest manner to ensure that future modifications have minimal impact on data structures. The formal process of data structure normalization aims to eliminate redundancy and enhance data integrity. This is achieved by using the normalization technique, which involves the removal of unnecessary fields and the division of a large table into smaller, more manageable ones. Additionally, it helps prevent anomalies during data insertion, deletion, and updates.

The normalization process relies on two fundamental concepts: keys and relationships. A key is a unique identifier that distinguishes a row in a table. There are two types of keys: primary keys, which are elements or sets of components within a table used to differentiate records within that same table, and foreign keys, which are columns in a table employed to uniquely identify records in other tables. This process continues up to the third normal form.

Normalization, a crucial step in database design, aims to structure data into well-organized tables and columns for user comprehension. This procedure eliminates data redundancy, which can burden computer resources. The key steps in normalization involve structuring the data correctly, selecting suitable names for tables and columns, and using appropriate names for the data. Following these steps results in a more efficient and easily manageable database.

First Normal Form (1NF)

This initial step requires that each attribute in a table should have indivisible values. In simple terms, each piece of data should be atomic. Nested relations or relations within attributes are not allowed. To achieve 1NF, data must be separated into different tables, where data types are consistent within each table. Each table should have a primary key or foreign key as needed. This process eliminates repeating data groups and establishes new relationships for non-atomic attributes. A relation is in 1NF if it adheres to these constraints.

Second Normal Form (2NF)

The second step focuses on ensuring that non-key attributes (attributes are not part of the primary key) depend on the entire primary key, not just a part of it. To achieve 2NF, tables may need to be decomposed, creating new relationships for subkeys and their dependent attributes. The original primary key should maintain its relationship with all attributes that are fully dependent on it.

Third Normal Form (3NF)

This stage mandates that there should be no transitive dependency on the primary key, meaning non-key attributes should not be functionally determined by other non-key attributes. To reach 3NF, relations may need further decomposition to establish new relations that include non-key attributes determining other non-key attributes. This step eliminates dependencies that do not rely solely on the primary key.

4.4.3 Sanitization

Data sanitization involves the crucial task of eliminating any undesirable characters or values from data. In the context of web applications, sanitizing user input is a fundamental practice to bolster security and prevent vulnerabilities. PHP offers a built-in filter extension that simplifies and accelerates the process of sanitizing and validating different types of external input, such as email addresses, URLs, IP addresses, and more. These filters are specifically designed to streamline data sanitization.

For instance, within the PHP filter extension, there's a function that can effectively strip away all characters except letters, digits, and certain permitted special characters (like `!#$%&'*+.=?_`{|}~@.[]`), as defined by a flag. Web applications routinely receive external input from diverse sources, including user submissions through forms, cookies, data from web services, server variables, and results from database queries. It is imperative to sanitize all incoming external input rigorously to ensure its safety and to prevent the infiltration of any malicious code or unwelcome values. This practice is vital for upholding the integrity and security of web applications.

4.4.4 Indexing

An index is a structural component in a database that significantly improves the speed of operations performed on tables. Indexes can be established on one or more columns to expedite swift data retrieval and efficient record ordering. When configuring an index, it's essential to identify the columns used in SQL queries and establish one or more indexes on those columns.

In practical terms, indexes can be perceived as specialized tables that store a primary key or index field along with pointers to each record within the actual table. These indexes remain hidden from users and are exclusively employed by the database search engine to expedite record identification. The creation of indexes in tables is accomplished using the "CREATE INDEX" statement.

It's worth noting that tables with indexes may experience slightly prolonged execution times for "INSERT" and "UPDATE" statements, as the database is required to manage the index values during these operations. Nevertheless, "SELECT" statements executed on these tables exhibit improved performance because the index streamlines the process of locating records.

4.5 TABLE DESIGN

1. Tbl_users_login

Primary key: **loginid**

Foreign key: **loginid** references table **Tbl_users_login**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	loginid	Primary Key	PRIMARY KEY	Primary key for user login
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model

2. Tbl_user

Primary key: **uid**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	uid	INT (10)	PRIMARY KEY	Primary key for user
2	username	Char (150)	NOT NULL	User's username
3	password	Char (128)	NOT NULL	User's password (hashed)
4	first_name	Char (30)	NOT NULL	User's first name
5	last_name	Char (150)	NOT NULL	User's last name
6	email	Email (255)	NOT NULL	User's email address
7	is_active	Boolean	-	User's active status (True/False)
9	is_customer	Boolean	-	User is a customer (True/False)
10	is_seller	Boolean	-	User is a seller (True/False)
11	is_legaladvisor	Boolean	-	User is a legal advisor (True/False)

12	date_joined	DateTime	NOT NULL	User's registration date and time
----	-------------	----------	----------	-----------------------------------

3. Tbl_user_profile

Primary key: **pid**

Foreign key: **user** references table **Tbl_user**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	pid	INT (10)	PRIMARY KEY	Primary key for user profile
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	name	Char (100)	NOT NULL	User's name
4	email	Email (255)	NOT NULL	User's email address
5	profile_picture	Image	NOT NULL	User's profile picture (image)
6	phone_number	Char (20)	NOT NULL	User's phone number
7	address	Text	NOT NULL	User's address

4. Tbl_certification

Primary key: **cid**

Foreign key: **user** references table **Tbl_user**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	cid	INT (10)	PRIMARY KEY	Primary key for certification
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	certification_image	Image	NOT NULL	Certification image (if applicable)
4	first_name	Char (100)	NOT NULL	First name of the certificate holder
5	last_name	Char (100)	NOT NULL	Last name of the certificate holder
6	expiry_date_from	Date	NOT NULL	Expiry date of the certification
7	certification_authority	Char (200)	NOT NULL	Authority that issued the certification

8	phone_number	Char (15)	NOT NULL	Phone number of the certificate holder
9	certification_number	Char (50)	NOT NULL	Certification number (if applicable)
10	address	Char (255)	NOT NULL	Address of the certificate holder
11	timestamp	DateTime	NOT NULL	Timestamp when the certification record was created
12	is_approved	Char (10)	NOT NULL	Certification approval status (Approved/Rejected/Pending)

5. Tbl_seller

Primary key: **sid**

Foreign key: **user** references table **Tbl_user**, **certification** references table **Tbl_certification**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	sid	INT (10)	PRIMARY KEY	Primary key for seller
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	certification (Foreign Key)	Integer	References Tbl_certification	Foreign key to the Certification model

6. Tbl_category

Primary key: **category_id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Category_id	INT (10)	PRIMARY KEY	Primary key for category
2	category_name	Char (100)	NOT NULL	Name of the category
3	category_description	Text	NOT NULL	Description of the category

7. Tbl_product

Primary key: **Product_id**

Foreign key: **category** references table **Tbl_category**, **seller** references table **Tbl_seller**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Product_id	INT (10)	PRIMARY KEY	Primary key for product
2	product_name	Char (100)	NOT NULL	Name of the product
3	product_description	Text	NOT NULL	Description of the product
4	category (Foreign Key)	Integer	References Tbl_category	Foreign key to the Category model
5	product_price	Decimal (10, 2)	NOT NULL	Price of the product
6	product_stock	Integer	NOT NULL	Stock quantity of the product
7	product_image	Image	NOT NULL	Product image (if applicable)
8	seller (Foreign Key)	Integer	References Tbl_seller	Foreign key to the Seller model

8. Tbl_cart

Primary key: **cart_id**

Foreign key: **user** references table **Tbl_users**, **product** references table **Tbl_product**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	cart_id	INT (10)	PRIMARY KEY	Primary key for cart item
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	product (Foreign Key)	Integer	References Tbl_product	Foreign key to the Product model
4	quantity	Positive Integer	NOT NULL	Quantity of the product in the cart
5	date_added	DateTime	NOT NULL	Date and time the item was added

6	price	Decimal (10, 2)	NOT NULL	Price of the cart item
7	image	Image	NOT NULL	Image of the cart item (if applicable)

9. Tbl_product_summary

Primary key: **Summery _id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Summery _id	INT (10)	PRIMARY KEY	Primary key for product summary
2	product_name	Char (100)	NOT NULL	Name of the product
3	total_stock	Integer	NOT NULL	Total stock quantity for the product
4	product_image	Image	NOT NULL	Product image (if applicable)
5	product_description	Varchar(20)	NOT NULL	Description of the product
6	product_price	Decimal (10, 2)	NOT NULL	Price of the product

10. Tbl_billing_details

Primary key: **billing_id**

Foreign key: **user** references table **Tbl_users**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	billing_id	INT (10)	PRIMARY KEY	Primary key for billing details
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	first_name	Char (255)	NOT NULL	First name of the billing recipient
4	last_name	Char (255)	NOT NULL	Last name of the billing recipient

5	state	Char (255)	NOT NULL	State of the billing address
6	street_address	Char (255)	NOT NULL	Street address of the billing address
7	apartment_suite_unit	Char (255)	NOT NULL	Apartment/Suite/Unit (optional)
8	town_city	Char (255)	NOT NULL	Town or city of the billing address
9	postcode_zip	Char (20)	NOT NULL	Postcode/ZIP code of the billing address
10	phone	Char (20)	NOT NULL	Phone number of the billing recipient
11	email	Email (255)	NOT NULL	Email address of the billing recipient

11. Tbl_order_item

Primary key: **Order_item_id**

Foreign key: **order** references table **Tbl_order**, **product** references table **Tbl_product**, **seller** references table **Tbl_seller**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Order_item_id	INT (10)	PRIMARY KEY	Primary key for order
2	order (Foreign Key)	Integer	References Tbl_order	Foreign key to the Order model
3	product (Foreign Key)	Integer	References Tbl_product	Foreign key to the Product model
4	seller (Foreign Key)	Integer	References Tbl_seller	Foreign key to the Seller model
5	quantity	Positive Integer	NOT NULL	Quantity of the product in the order
6	price	Decimal (10, 2)	NOT NULL	Price per unit of the product
7	total_price	Decimal (10, 2)	NOT NULL	Total price for the order item

12. Tbl_order

Primary key: **order_id**

Foreign key: **user** references table **Tbl_users**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	order_id	INT (10)	PRIMARY KEY	Primary key for order
2	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
3	total_price	Decimal (10, 2)	NOT NULL	Total price of the order
4	order_date	DateTime	NOT NULL	Date and time of the order
5	razorpay_order_id	Char (255)	NOT NULL	Razorpay Order ID
6	payment_status	Char (20)	NOT NULL	Payment status (Pending/Successful/Failed)

13. Tbl_review

Primary key: **review_id**

Foreign key: **user** references table **Tbl_users**, product references table **Tbl_product**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	review_id	INT (10)	PRIMARY KEY	Primary key for review
2	product (Foreign Key)	Integer	References Tbl_product	Foreign key to the Product model
3	user (Foreign Key)	Integer	References Tbl_user	Foreign key to the User model
4	rating	Positive Integer	NOT NULL	Rating for the product (e.g., 1 to 5 stars)
5	comment	Text		Optional comment for the review

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing is the process of systematically evaluating a software program to determine if it operates as intended. It typically involves using methods for verification and validation. Validation ensures that the product aligns with its specifications, while verification may encompass reviews, analyses, inspections, and walkthroughs. Static analysis focuses on examining the software's source code to detect issues, whereas dynamic analysis observes its behavior during runtime, collecting data like execution traces, timing profiles, and test coverage details.

Testing comprises a series of organized activities that commence with individual modules and extend to the integration of the entire computer-based system. The primary goals of testing include the identification of errors and bugs within the software, confirmation that the software adheres to its specifications, and validation of its performance against requirements. Testing can encompass assessments of correctness, implementation efficiency, and computational complexity.

A successful test is one that uncovers previously undetected errors, and a valuable test case holds a high likelihood of revealing such errors. Testing is a critical element in achieving system testing objectives and encompasses various techniques like functional testing, performance testing, and security testing.

5.2 TEST PLAN

A test plan is a comprehensive document that delineates the necessary steps for executing diverse testing methodologies. It serves as a roadmap for the activities that must be carried out during the testing process. Software developers are tasked with crafting computer programs, documentation, and the associated data structures. Their responsibility entails scrutinizing each component of the program to ensure that it aligns with its intended purpose. To mitigate self-assessment biases, it is common practice to establish an independent test group (ITG).

When setting testing objectives, it is essential to express them in measurable terms. This may involve metrics such as mean time to failure, the cost associated with identifying and rectifying defects, the remaining defect density or frequency of occurrences, and the number of test work-hours required for regression tests.

The various tiers of testing encompass:

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

5.2.1 Unit Testing

Unit testing is a software testing approach that concentrates on the validation of individual software components or modules. Its primary aim is to assess the functionality of the smallest units within the software design, ensuring they operate as intended. Typically, unit testing takes a white-box perspective, and it's possible to test multiple components simultaneously. The testing process relies on the component-level design description as a reference, allowing the identification of critical control paths and potential faults within the module's scope.

In the course of unit testing, the focus lies on testing the module's interface to confirm that data effectively enters and exits the software unit under examination. The local data structure is meticulously examined to guarantee that temporarily stored data maintains its integrity throughout the execution of algorithms. Boundary conditions are examined to ensure that all statements within a module have been executed at least once. Additionally, error handling paths are scrutinized to verify the software's capability to handle errors correctly.

Prior to conducting any other testing, it is imperative to ensure the proper flow of data through a module's interface. If data cannot flow in and out of the system correctly, all other tests become inconsequential. Another crucial aspect of unit testing involves a selective evaluation of execution paths to anticipate potential errors and ensure that error-handling mechanisms are in place to redirect or halt processes in the event of an error. Finally, boundary testing is executed to confirm that the software operates flawlessly even under extreme conditions.

5.2.2 Integration Testing

Integration testing is a methodical process that involves not only building the program structure but also conducting tests to detect interface-related issues. The primary goal is to construct a program structure based on the design, incorporating components that have undergone unit testing. Subsequently, the entire program is subjected to testing. Rectifying errors during integration testing can be a formidable task, mainly due to the scale of the overall program, which makes it challenging to pinpoint the sources of errors. It can often be a dynamic process where fixing one set of errors may lead to the emergence of new ones, creating what seems like an unending cycle.

Upon completing unit testing for all modules within the system, these modules are integrated to identify and resolve any inconsistencies in their interfaces. Any disparities in program structures are addressed, ultimately resulting in the creation of a cohesive and unified program structure.

5.2.3 Validation Testing or System Testing

The concluding phase of the testing process entails evaluating the complete software system in its entirety, encompassing all forms, code, modules, and class modules. This phase is typically known as system testing or black box testing. Black box testing places its emphasis on assessing the software's functional requirements. In this approach, a software engineer can establish input conditions that comprehensively examine each program requirement. The key categories of errors that black box testing aims to identify encompass incorrect or absent functions, interface discrepancies, issues in data structures or external data access, performance-related problems, initialization errors, and termination errors.

5.2.4 Output Testing or User Acceptance Testing

User acceptance testing is conducted to verify that the system aligns with the business requirements and fulfills the user's expectations. It's crucial to engage end users throughout the development process to ensure that the software caters to their needs and anticipations. In the course of user acceptance testing, the input and output screen designs are scrutinized using various types of test data. The meticulous preparation of test data plays a vital role in ensuring a thorough assessment of the system. Any discrepancies uncovered during testing are duly addressed and rectified, and these corrections are duly documented for future reference.

5.2.5 Automation Testing

Automation testing is a software testing approach that leverages specialized automated testing software tools to run a suite of test cases. Its primary objective is to validate that the software or

equipment functions precisely as intended. Automation testing serves to identify defects, bugs, and other issues that may surface during the course of product development. While certain types of testing, like functional or regression testing, can be carried out manually, automating the process offers several advantages. Automation testing can be conducted at any time, employing scripted sequences to assess the software. The outcomes are documented, and these results can be compared to previous test runs. Automation developers commonly write code in programming languages such as C#, JavaScript, and Ruby.

5.2.6 Selenium Testing

Selenium is an open-source automated testing framework designed for validating web applications across various browsers and platforms. Selenium facilitates the creation of test scripts in different programming languages, such as Java, C#, and Python. The framework was initially developed by Jason Huggins, an engineer at Thought Works, in 2004, while he was working on a web application that required frequent testing. He introduced a JavaScript program called "JavaScriptTestRunner" to automate browser actions and enhance testing efficiency. Since its inception, Selenium has evolved and is continually improved by a community of contributors.

In addition to Selenium, another widely used tool for automated testing is Cucumber. Cucumber is an open-source software testing framework that supports behavior-driven development (BDD). It enables the creation of executable specifications in a human-readable format called Gherkin. One of the significant advantages of Cucumber is its capability to bridge the communication gap between business stakeholders and technical teams. By using a common language, Cucumber enhances effective communication and collaboration during the testing process. It promotes a shared understanding of requirements and ensures that the developed software aligns with the intended business objectives.

Cucumber can be seamlessly integrated with Selenium to harness the benefits of both tools. Selenium is employed for interacting with web browsers and automating browser actions, while Cucumber provides a structured framework for organizing and executing tests. This combination empowers the creation of end-to-end tests that validate the behavior of web applications across diverse browsers and platforms. It does so using a format that is both business-readable and easily maintainable.

Test Case 1: User Login

Code

```
from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select

class Hosttest(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000'
    def tearDown(self):
        self.driver.quit()
    def test_02_login_page(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(2)
        #login from home seller
        theme=driver.find_element(By.CSS_SELECTOR,"a.nav-link[href='/login.html']")
        theme.click()
        elem = driver.find_element(By.NAME, "username")
        elem.send_keys("seller1")
        elem = driver.find_element(By.NAME, "password")
        elem.send_keys("Don@2001")
        time.sleep(2)
        submit_button=driver.find_element(By.CSS_SELECTOR,
"button#login_button.signinbtn")
        submit_button.click()
        time.sleep(5)
```

Screenshot

```
(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:49225/devtools/browser/5e870990-47ab-44e3-88c7-68a61339e5a3
.
-----
Ran 1 test in 31.368s

OK
Destroying test database for alias 'default'...

(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>
```

Test Report

Test Case 1					
Project Name: Eco Hive					
Login Test Case					
Test Case ID: Test_1			Test Designed By: Don K joseph		
Test Priority(Low/Medium/High):High			Test Designed Date: 06-10-2023		
Module Name: Login Screen			Test Executed By : Ms. Merin Manoj		
Test Title : User Login			Test Execution Date: 06-10-2023		
Description: Verify Registration with valid details					
Pre-Condition :User has valid username and password					
St ep	Test Step	Test Data	Expect ed Result	Actual Result	Status(Pass/ Fail)
1	Navigate to Register page		Dashboa rd should be displaye d	Login page displayed	Pass
2	Provide valid username	Seller6	User should be able to Login	User Logged in and navigate to User Dashboard	Pass
3	Provide valid password	Don @2001			

4	Click on login button				
Post-Condition: User is validated with database and successfully login into account. The Account session details are logged in database					

Test Case 2: Add Product to the store

Code

```

from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select

class Hosttest(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000'
    def tearDown(self):
        self.driver.quit()
    def test_02_login_page(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(2)
        #add product
        browse=driver.find_element(By.CSS_SELECTOR,"a.sidebar-link[href='/addproducts']")
        browse.click()
        time.sleep(5)
        elem = driver.find_element(By.NAME, "product_name")
        elem.send_keys("rambutan")
        time.sleep(2)
        elem = driver.find_element(By.NAME, "product_description")
        elem.send_keys("good natural product")
        time.sleep(2)
        elem = driver.find_element(By.NAME, "product_price")
        elem.send_keys("250")
        time.sleep(2)
        elem = driver.find_element(By.NAME, "product_stock")
        elem.send_keys("20")
        time.sleep(2)

```



```

select_element = Select(driver.find_element(By.NAME, "select_category"))
select_element.select_by_value("5")
time.sleep(2)

file_path = r"C:\Users\donkj\Downloads\organic products\rambutan.jpg"
elem = driver.find_element(By.NAME, "product_image")
elem.send_keys(file_path)
time.sleep(2)
browse=driver.find_element(By.CSS_SELECTOR,"button#add-product-button")
browse.click()
time.sleep(2)

browse=driver.find_element(By.CSS_SELECTOR,"a.btn.btn-
danger[href='/viewaddproduct']")
browse.click()
time.sleep(2)

```

Screenshot

```

(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:49225/devtools/browser/5e870990-47ab-44e3-88c7-68a61339e5a3
.
-----
Ran 1 test in 31.368s

OK
Destroying test database for alias 'default'...

(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>

```

Test report

Test Case 2					
Project Name: Eco Hive					
Add Product Test Case					
Test Case ID: Test_2			Test Designed By: Don K joseph		
Test Priority(Low/Medium/High):High			Test Designed Date: 12-10-2023		
Module Name: Product Screen			Test Executed By : Ms. Merin Manoj		
Test Title : Add product			Test Execution Date: 12-10-2023		
Description: Seller add product to the store					
Pre-Condition :User has to add product					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/Fail)

1	Navigate to Product add page		Dashboa rd should be displaye d	Add product page displayed	Pass
2	Provide valid product name	Orange	User should be able to add Product to the store	Product is added to the store	Pass
3	Provide a product description	Eco friendly organic product			
4	Provide a valid product price	55			
5	Provide valid product stock	10KG			
6	Select Product category	Fruits			
7	Provide an image for the Product				
8	Click on the Add product button				
Post-Condition: The product is added to the Store, the product details are stored in the database					

Test Case 3: Add Product Category

Code

```

from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
class Hosttest(TestCase):
    def setUp(self):

```

```

        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000'

    def tearDown(self):
        self.driver.quit()

    def test_02_login_page(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(2)
        #login from home admin
        theme=driver.find_element(By.CSS_SELECTOR,"a.nav-link[href='/login.html']")
        theme.click()
        elem = driver.find_element(By.NAME, "username")
        elem.send_keys("Admin")
        elem = driver.find_element(By.NAME, "password")
        elem.send_keys("Don@2001")
        time.sleep(2)
        submit_button=driver.find_element(By.CSS_SELECTOR, "button#login_button.signinbtn")
        submit_button.click()
        time.sleep(5)
        browse=driver.find_element(By.CSS_SELECTOR,"a.sidebar-link[href='/addcategory']")
        browse.click()
        time.sleep(5)
        elem = driver.find_element(By.NAME, "category_name")
        elem.send_keys("dairyproducts")
        time.sleep(2)
        elem = driver.find_element(By.NAME, "category_description")
        elem.send_keys("dirty products like nuts etc")
        time.sleep(2)
        browse=driver.find_element(By.CSS_SELECTOR,"button.btn.btn-primary")
        browse.click()
        time.sleep(2)

```

Screenshot

```

(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:49225/devtools/browser/5e870990-47ab-44e3-88c7-68a61339e5a3
.
-----
Ran 1 test in 31.368s

OK
Destroying test database for alias 'default'...

(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>

```

Test report

Test Case 3					
Project Name: Eco Hive					
Add Product Category Test Case					
Test Case ID: Test_3			Test Designed By: Don K joseph		
Test Priority(Low/Medium/High) :High			Test Designed Date: 16-10-2023		
Module Name: Category Screen			Test Executed By : Ms. Merin Manoj		
Test Title : Add Product Category			Test Execution Date: 16-10-2023		
Description: Admin adds Product Categories					
Pre-Condition :Admin has to add Product Category					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/Fail)
1	Navigate to Category add page		Dashboar d should be displaye d	Add category page displayed	Pass
2	Provide valid Category name	Fruits			
3	Provide a Category descripti on	Fruits, the taste of nature organic	User should be able to add Product Category	Product category is added to the store	Pass
4	Clicks on Add Category button				
Post-Condition: The product category is added to the Store, The category details are stored in the database					

Test Case 4: Add to Cart

Code

```
from django.test import TestCase
from selenium import webdriver
```

```

from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
class Hosttest(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000'

    def tearDown(self):
        self.driver.quit()

    def test_02_login_page(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(2)
#Add to cart
        theme=driver.find_element(By.CSS_SELECTOR,"a.nav-link[href='/login.html']")
        theme.click()
        elem = driver.find_element(By.NAME, "username")
        elem.send_keys("customer1")
        elem = driver.find_element(By.NAME, "password")
        elem.send_keys("Don@2001")
        time.sleep(2)
        submit_button = driver.find_element(By.CSS_SELECTOR,
"button#login_button.signinbtn")
        submit_button.click()
        time.sleep(2)
        browse=driver.find_element(By.CSS_SELECTOR,"a#top.nav-link")
        browse.click()
        time.sleep(2)
        driver.execute_script("window.scrollTo(0, 500);")
        browse=driver.find_element(By.CSS_SELECTOR,"a.img-prod")
        browse.click()
        driver.execute_script("window.scrollTo(0, 500);")
        time.sleep(2)
        browse=driver.find_element(By.CSS_SELECTOR,"button.add-to-cart-
button[name='buy_now']")
        browse.click()
        time.sleep(5)

```

Screenshot

```
(venv) C:\Users\donkj\OneDrive\Desktop\project\ecohive>python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:51103/devtools/browser/89a4d98e-f323-4a6d-abf6-5fe762a42feb
.
-----
Ran 1 test in 23.104s

OK
Destroying test database for alias 'default'...
```

Test report

Test Case 4					
Project Name: Eco Hive					
Add to Cart Test Case					
Test Case ID: Test_4			Test Designed By: Don K joseph		
Test Priority(Low/Medium/High):High			Test Designed Date: 16-10-2023		
Module Name: Prdouct Screen			Test Executed By : Ms. Merin Manoj		
Test Title : Add Product to Cart			Test Execution Date: 16-10-2023		
Description: Add Product to Cart					
Pre-Condition :Add product to Cart					
St ep	Test Step	Test Data	Expect ed Result	Actual Result	Status(Pass/ Fail)
1	Navigate to Product page		Dashboar d should be displayed	Single product page displayed	Pass
2	Clicks on Add to Cart button		The Item should be added to cart	The item is added to cart	Pass
Post-Condition: The product is added to Cart, it is saved in the database					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

The implementation phase of a project marks the transformation of the design into a fully functional system. It stands as a critical juncture in ensuring the success of the new system, as it necessitates building user confidence in its effective and accurate operation. Of great importance during this phase are user training and the creation of comprehensive documentation. The conversion process may happen alongside user training or at a later stage. Implementation encompasses the process of turning a freshly revised system design into an operational and functional system. In this stage, the user department shoulders the primary workload, experiences the most substantial changes, and wields the greatest influence over the existing system. If implementation is poorly planned or lacks control, it can lead to confusion and disorder. Whether the new system is entirely novel, replaces an existing manual or automated system, or modifies an existing one, a well-executed implementation is essential to align with the organization's needs.

System implementation encompasses all the activities necessary to transition from the old system to the new one. The new system can only be implemented after thorough testing confirms that it operates according to the specified requirements. System personnel assess the feasibility of the new system. Implementation demands substantial effort in three key areas: education and training, system testing, and changeover. The implementation phase calls for meticulous planning, exploration of system and constraints, and the development of methods to achieve a smooth transition.

6.2 IMPLEMENTATION PROCEDURES

Software implementation involves the process of installing software in its intended environment and verifying that it meets its intended purpose while functioning correctly. In certain organizations, the software development project may be initiated by someone who won't be using the software personally. At the initial stages, doubts about the software may arise, but it's crucial to prevent resistance from building up.

This can be achieved by:

- Ensuring that active users are well-informed about the advantages of the new system, thus building their confidence in the software.
- Offering proper guidance to users to ensure they feel at ease when using the application.

It's important for users to understand that the server program needs to be operational on the server before they can access the system. Without the server object running, the desired processes will not occur.

6.2.1 User Training

User training is a crucial phase intended to get users ready for testing and transitioning to a new system. To realize the anticipated advantages of a computer-based system, it's vital for individuals involved to have confidence in their roles within the new system. As systems become increasingly complex, the necessity for training becomes more pronounced. Through user training, individuals gain knowledge on tasks such as data entry, handling error messages, querying databases, and utilizing routines for generating reports and executing other essential functions. This ensures that users are well-prepared and capable of effectively utilizing the system.

6.2.2 Training on the Application Software

Following the provision of fundamental computer awareness training, it becomes crucial to deliver training on the new application software to the users. This training should encompass the fundamental principles of using the new system, including screen navigation, screen design, available on-screen assistance, potential data entry errors, corresponding validation checks for each entry, and methods to rectify entered data. Furthermore, the training should encompass user or group-specific information that is essential for effective system or module utilization. It's worth noting that this training may vary among distinct user groups and hierarchical levels.

6.2.3 System Maintenance

The maintenance phase is a crucial aspect of the software development cycle, as it is the time when the software is actually put to use and performs its intended functions. Proper maintenance is essential to ensure that the system remains functional, reliable, and adaptable to changes in the system environment. Maintenance activities go beyond simply identifying and fixing errors or bugs in the system. It may involve updates to the software, modifications to its functionalities, and enhancements to its performance, among other things. In essence, software maintenance is an ongoing process that requires continuous monitoring, evaluation, and improvement of the system to meet changing user needs and requirements.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In summary, the "Eco-Friendly Organic Product Store Website" project is a promising venture that seeks to meet the growing demand for eco-friendly products. With user-friendly features for Administrators, Customers, Sellers, and Legal Advisors, it provides a convenient platform for sustainable shopping.

Key functionalities such as user registration, a product catalog, a shopping cart, user reviews, and sustainability certifications enhance the user experience and promote eco-conscious consumer choices. Technologically, we employ modern web development tools to ensure a responsive and secure platform.

Our economic feasibility analysis confirms the project's viability, with costs aligning with revenue projections, promising a positive return on investment. For the future, we plan to expand with advanced certification systems, a mobile app, and increased seller participation to stay ahead of eco-friendly trends.

In conclusion, the "Eco-Friendly Organic Product Store Website" project is not just a website; it's a commitment to a greener world. We invite everyone to join us on this journey towards a more sustainable future.

7.2 FUTURE SCOPE

- **Advanced Certification Systems:** To enhance trust and transparency, consider implementing advanced eco-certification systems. This could involve partnerships with recognized environmental certification bodies, providing users with even more reliable information about product sustainability.
- **Mobile Application:** Develop a dedicated mobile app to cater to the growing mobile user base. Mobile apps can provide a more convenient shopping experience, push notifications, and better user engagement.
- **Global Expansion:** Expand the platform to serve a broader international audience. This will involve localization, multi-language support, and a global network of eco-friendly sellers, further promoting sustainable practices worldwide.

- **User-Generated Content:** Encourage users to contribute eco-friendly content. This can include user-generated product reviews, sustainability tips, and community forums, fostering an engaged eco-conscious community.
- **Analytics and Personalization:** Implement advanced data analytics and AI to provide personalized product recommendations to users based on their preferences and past behavior. This can enhance the user experience and drive more sales.
- **Sustainable Shipping and Packaging:** Collaborate with logistics partners to offer sustainable shipping options and promote eco-friendly packaging practices, reducing the project's carbon footprint.
- **Educational Initiatives:** Introduce educational resources and initiatives to raise awareness about sustainable living and the importance of eco-friendly products. This can include blogs, webinars, and partnerships with environmental organizations

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- PankajJalote, “Software engineering: a precise approach”
- Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009
- Ken Schwaber, Mike Beedle, Agile Software Development with Scrum, Pearson (2008)
- Roger S Pressman, “Software Engineering”
- IEEE Std 1016 Recommended Practice for Software Design Descriptions

WEBSITES:

- seednet.gov.in
- www.w3schools.com
- www.bootstrap.com
- <https://chat.openai.com/chat>
- www.jquery.co

CHAPTER 9

APPENDIX

9.1 Sample Code

1. Login

```
{% load static %}
{% load socialaccount %}

<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <link rel="stylesheet" href="{% static 'css/login.css' %}">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css">
</head>

<body onload="noBack();">
  <nav class="Logo">
    <div class="container">
      
    </div>
  </nav>

  <div class="signin">
    <div id="signintext">
      <p id="signintext1">Green, Clean, Serene.</p>
      <h2 id="signintext3">SIGN IN</h2>
    </div>
    <form class="signin-form" method="post">
      {% csrf_token %}
      {% comment %} <input type="text" class="signinelement" placeholder="User
Name" required name="username"> {% endcomment %}
      <input type="text" class="signinelement" name="{{ form.username.name }}"
name="username" id="{{ form.username.id_for_label }}" placeholder="User Name"
required>
      {% comment %} <input type="password" class="signinelement"
placeholder="password" required name="password"> {% endcomment %}
      <input type="password" class="signinelement" name="{{ form.password.name }}"
name="password" id="{{ form.password.id_for_label }}" placeholder="Password"
required><br><br>
      <span class="mb-0 text-muted" style="margin-left: 60px; color:red;">
        {% if msg %}
          {{ msg | safe }}
        {% endif %}
      </span>
      <p id="signinelement2"><a href="password_reset" style="color: #27350F; text-
decoration: none">Forgot password?</a></p>
      <div class="icon-container">
        <a href="{% provider_login_url 'google' %}?next="/"><span id="icon" class="fab
fa-google" style="color: red"></span></a>
        {% comment %} <a href="#"><span id="icon" class="fab fa-facebook"
```



```

style="color: blue"></span></a> <!-- Facebook icon placeholder -->
    <a href="#"><span id="icon" class="fab fa-twitter" style="color:
#1DA1F2"></span></a> <!-- Twitter icon placeholder --> {% endcomment %}
</div>
<button class="signinbtn" id="login_button" type="submit">Login</button>

<p id="signinelement3">Don't have an account? <a href="register.html"
style="color: #27350F; text-decoration: none">Signup</a></p>

</form>
<div class="signinimg">
    
</div>
</div>
<script>
    window.history.forward();
    function noBack() {
        window.history.forward();
    }
</script>
</body>
</html>

```

2. Apply For Certification Approval

```

{% extends 'sellerdash/base.html' %}
{% load static %}
{% block content %}
<div class="container">
    <div class="card">
        <div class="card-body">
            <h5 class="card-title fw-bold mb-4">Certification Approval</h5>
            {{ existing_certification.is_approved }}
            {% if existing_certification %}

                {% if existing_certification.is_approved == 'pending' %}
                <p class="certification-message">You have already entered your Licence details.</p>

                {% elif existing_certification.is_approved == 'approved' %}
                <p class="certification-message">Your Application is approved.</p>
                {% elif existing_certification.is_approved == 'rejected' %}
                <p class="certification-message">Your Application is rejected.</p>
                {% else %}
                <p> Not working </p>
            {% endif %}
            {% else %}
                <!-- Display the certification form -->
                <form method="post" enctype="multipart/form-data" class="certification-form"
id="certification-form">
                    {% csrf_token %}

```

```

<div class="form-group">
  <!-- Display the 'first_name' field -->
  <label for="{{ form.first_name.id_for_label }}">First Name:</label>
  <input type="text" id="{{ form.first_name.id_for_label }}" name="{{
form.first_name.name }}" class="form-control" required>
  <span class="text-danger" id="first-name-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'last_name' field -->
  <label for="{{ form.last_name.id_for_label }}">Last Name:</label>
  <input type="text" id="{{ form.last_name.id_for_label }}" name="{{
form.last_name.name }}" class="form-control" required>
  <span class="text-danger" id="last-name-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'expiry_date_to' field -->
  <label for="{{ form.expiry_date.id_for_label }}">Expiry Date :</label>
  <input type="date" id="{{ form.expiry_date_from.id_for_label }}" name="{{
form.expiry_date_from.name }}" class="form-control" required>
  <span class="text-danger" id="expiry-date-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'certification_authority' field -->
  <label for="{{ form.certification_authority.id_for_label }}">Certifying Body:</label>
  <input type="text" id="{{ form.certification_authority.id_for_label }}" name="{{
form.certification_authority.name }}" class="form-control" required>
  <span class="text-danger" id="certification-authority-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'phone_number' field -->
  <label for="{{ form.phone_number.id_for_label }}">Phone Number:</label>
  <input type="text" id="{{ form.phone_number.id_for_label }}" name="{{
form.phone_number.name }}" class="form-control" required>
  <span class="text-danger" id="phone-number-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'certification_number' field -->
  <label for="{{ form.certification_number.id_for_label }}">Certification
Number:</label>
  <input type="text" id="{{ form.certification_number.id_for_label }}" name="{{
form.certification_number.name }}" class="form-control" required>
  <span class="text-danger" id="certification-number-error"></span>
</div>

<div class="form-group">
  <!-- Display the 'certification_number' field -->
  <label for="{{ form.address.id_for_label }}">Full Address:</label>
  <textarea id="{{ form.address.id_for_label }}" name="{{ form.address.name }}"
class="form-control" required></textarea>
  <span class="text-danger" id="address-error"></span>
</div>

```

```

    <div class="form-group">
      <!-- Display the 'certification_image' field -->
      <label for="{{ form.certification_image.id_for_label }}">Copy of
Certification:</label>
      <input type="file" accept=".img,.jpg,.png/*" id="{{ {
form.certification_image.id_for_label }}" name="{{ form.certification_image.name }}"
class="form-control" required>
      <span class="text-danger" id="certification-image-error"></span>
    </div>

    <div class="btn-container">
      <button type="submit" id="submit-button" class="btn btn-primary">Submit
Certification Request</button>
    </div>
  </form>
  {% endif %}
</div>
</div>
</div>
</div>
<script>
  document.addEventListener("DOMContentLoaded", function () {
    const form = document.getElementById("certification-form");
    const firstNameInput = document.querySelector(
      "#certification-form input[name='first_name']"
    );
    const lastNameInput = document.querySelector(
      "#certification-form input[name='last_name']"
    );
    const expiryDateInput = document.querySelector(
      "#certification-form input[name='expiry_date_from']"
    );
    const certificationAuthorityInput = document.querySelector(
      "#certification-form input[name='certification_authority']"
    );
    const phoneNumberInput = document.querySelector(
      "#certification-form input[name='phone_number']"
    );
    const certificationNumberInput = document.querySelector(
      "#certification-form input[name='certification_number']"
    );
    const certificationImageInput = document.getElementById(
      "{{ form.certification_image.id_for_label }}"
    );
    const certificationImageError = document.getElementById(
      "certification-image-error"
    );

    form.addEventListener("submit", function (e) {
      let valid = true;

      // Validation for First Name
      const firstNameValue = firstNameInput.value.trim();
      if (!/^[A-Za-z]+$/.test(firstNameValue)) {
        document.getElementById("first-name-error").textContent =

```

```
        "Only alphabetic characters are allowed.";
        valid = false;
    } else {
        document.getElementById("first-name-error").textContent = "";
    }

    // Validation for Last Name
    const lastNameValue = lastNameInput.value.trim();
    if (!/^[A-Za-z]+$/.test(lastNameValue)) {
        document.getElementById("last-name-error").textContent =
            "Only alphabetic characters are allowed.";
        valid = false;
    } else {
        document.getElementById("last-name-error").textContent = "";
    }

    // Validation for Expiry Date (from today to any day)
    const currentDate = new Date();
    const selectedDate = new Date(expiryDateInput.value);
    if (selectedDate < currentDate) {
        document.getElementById("expiry-date-error").textContent =
            "Expiry date must be today or later.";
        valid = false;
    } else {
        document.getElementById("expiry-date-error").textContent = "";
    }

    // Validation for Certification Authority
    const certificationAuthorityValue = certificationAuthorityInput.value.trim();
    if (!/^[A-Za-z]+$/.test(certificationAuthorityValue)) {
        document.getElementById("certification-authority-error").textContent =
            "Only alphabetic characters are allowed.";
        valid = false;
    } else {
        document.getElementById("certification-authority-error").textContent = "";
    }

    // Validation for Phone Number
    const phoneValue = phoneNumberInput.value.trim();
    if (!/^(\+91-0)?[6-9]\d{9}$/.test(phoneValue)) {
        document.getElementById("phone-number-error").textContent =
            "Invalid phone number (10 digits required).";
        valid = false;
    } else {
        document.getElementById("phone-number-error").textContent = "";
    }

    // Validation for Certification Number (assuming only digits are allowed)
    const certificationNumberValue = certificationNumberInput.value.trim();
    if (!/^\d+$/.test(certificationNumberValue)) {
        document.getElementById("certification-number-error").textContent =
            "Only numeric characters are allowed.";
        valid = false;
    } else {
        document.getElementById("certification-number-error").textContent = "";
    }
}
```

```
    }

    // Validation for Certification Image
    const certificationImageValue = certificationImageInput.value.trim();
    const allowedExtensions = ["jpg", "jpeg", "png", "img"];
    const fileExtension = certificationImageValue
        .split(".")
        .pop()
        .toLowerCase();
    if (allowedExtensions.indexOf(fileExtension) === -1) {
        certificationImageError.textContent =
            "Please select a valid image file (jpg, jpeg, png, or img).";
        certificationImageInput.value = ""; // Clear the input field
        valid = false;
    } else {
        certificationImageError.textContent = "";
    }

    if (!valid) {
        e.preventDefault(); // Prevent form submission if there are validation errors
    }
});

// Add keyup event listeners for real-time validation
firstNameInput.addEventListener("keyup", function () {
    const firstNameValue = firstNameInput.value.trim();
    if (!/^[A-Za-z]+$/.test(firstNameValue)) {
        document.getElementById("first-name-error").textContent =
            "Only alphabetic characters are allowed.";
    } else {
        document.getElementById("first-name-error").textContent = "";
    }
});

lastNameInput.addEventListener("keyup", function () {
    const lastNameValue = lastNameInput.value.trim();
    if (!/^[A-Za-z]+$/.test(lastNameValue)) {
        document.getElementById("last-name-error").textContent =
            "Only alphabetic characters are allowed.";
    } else {
        document.getElementById("last-name-error").textContent = "";
    }
});

expiryDateInput.addEventListener("keyup", function () {
    const currentDate = new Date();
    const selectedDate = new Date(expiryDateInput.value);
    if (selectedDate < currentDate) {
        document.getElementById("expiry-date-error").textContent =
            "Expiry date must be today or later.";
    } else {
        document.getElementById("expiry-date-error").textContent = "";
    }
});
```

```

certificationAuthorityInput.addEventListener("keyup", function () {
    const certificationAuthorityValue = certificationAuthorityInput.value.trim();
    if (!/^[A-Za-z]+$/.test(certificationAuthorityValue)) {
        document.getElementById("certification-authority-error").textContent =
            "Only alphabetic characters are allowed.";
    } else {
        document.getElementById("certification-authority-error").textContent = "";
    }
});

phoneNumberInput.addEventListener("keyup", function () {
    const phoneNumberValue = phoneNumberInput.value.trim();
    if (!/^(\\+91-|0)?[6-9]\\d{9}$/.test(phoneNumberValue)) {
        document.getElementById("phone-number-error").textContent =
            "Invalid phone number (10 digits required).";
    } else {
        document.getElementById("phone-number-error").textContent = "";
    }
});

certificationNumberInput.addEventListener("keyup", function () {
    const certificationNumberValue = certificationNumberInput.value.trim();
    if (!/^\\d+$/.test(certificationNumberValue)) {
        document.getElementById("certification-number-error").textContent =
            "Only numeric characters are allowed.";
    } else {
        document.getElementById("certification-number-error").textContent = "";
    }
});

certificationImageInput.addEventListener("change", function () {
    const certificationImageValue = certificationImageInput.value.trim();
    const allowedExtensions = ["jpg", "jpeg", "png", "img"];
    const fileExtension = certificationImageValue
        .split(".")
        .pop()
        .toLowerCase();
    if (allowedExtensions.indexOf(fileExtension) === -1) {
        certificationImageError.textContent =
            "Please select a valid image file (jpg, jpeg, png, or img).";
        certificationImageInput.value = ""; // Clear the input field
    } else {
        certificationImageError.textContent = "";
    }
});
</script>
{% endblock %}

```

3. Approve Certification by Legal Advisor

```

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Legal Administrator Dashboard</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
<div class="container mt-4">
<div class="d-flex justify-content-between align-items-center mb-3">
<div>
<p>Welcome, { { request.user.username } }</p>
</div>
<div>
<p><a href="{ % url 'logout' % }" class="btn btn-danger">Logout</a></p>
</div>
</div>
<div id="indiv">
<div class="row mb-3">
<div class="col-md-4">
<label for="status-filter" class="filter-label">Filter by status:</label>
<select id="status-filter" class="filter-select">
<option value="all">All</option>
<option value="approved">Approved</option>
<option value="rejected">Rejected</option>
<option value="pending">Pending</option>
</select>
</div>
<div class="col-md-4">
<form class="search-form" id="searchform">
<div class="input-group">
<label for="name" class="search-label">Search by name:</label>
<input type="text" id="name" name="name" class="form-control search-input">
<div class="input-group-append">
<button type="submit" class="btn btn-primary search-button">Search</button>
</div>
</div>
</form>
</div>
</div>
<div>
<h2>Legal Administrator Dashboard</h2>
<p id="no-application-message" style="display: none;">No matching applications</p>
<table class="table application-list">
<thead>
<tr>
<th>Application ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>Expiry Date</th>

<th>Certification Authority</th>
<th>Phone Number</th>

```

```

        <th>Certification Number</th>
        <th>File</th>
        <th>Action</th>
    </tr>
</thead>
<tbody>
    { % for application in seller_applications % }
    <tr>
        <td>{{ application.id }}</td>
        <td>{{ application.first_name }}</td>
        <td>{{ application.last_name }}</td>
        <td>{{ application.expiry_date_from }}</td>

        <td>{{ application.certification_authority }}</td>
        <td>{{ application.phone_number }}</td>
        <td>{{ application.certification_number }}</td>
        <td>
            { % if application.certification_image % }
            <a href="{{ application.certification_image.url }}" target="_blank" class="btn btn-
primary">View File</a>
            { % else % }
            No File Available
            { % endif % }
        </td>
        <td id="status-cell-{{ application.id }}">
            <div class="approval-status">
                { % if application.is_approved == 'pending' % }
                <form method="post" action="{ % url 'approve_certification' application.id % }">
                    { % csrf_token % }
                    <button class="btn btn-success approve-btn" type="submit">Approve</button>
                </form>
                <form method="post" action="{ % url 'reject_certification' application.id % }">
                    { % csrf_token % }
                    <button class="btn btn-danger reject-btn" type="submit">Reject</button>
                </form>
                { % elif application.is_approved == 'approved' % }
                <span class="text-success">Approved</span>
                <form method="post" action="{ % url 'reject_certification' application.id % }">
                    { % csrf_token % }
                    <button class="btn btn-danger reject-btn" type="submit">Reject</button>
                </form>
                { % elif application.is_approved == 'rejected' % }
                <span class="text-danger">Rejected</span>
                <form method="post" action="{ % url 'approve_certification' application.id % }">
                    { % csrf_token % }
                    <button class="btn btn-success approve-btn" type="submit">Approve</button>
                </form>
                { % else % }
                <p>Not working</p>
                { % endif % }
                <div class="approval-message"></div>
            </div>
        </td>
    </tr>
    { % endfor % }

```



```

</tbody>
</table>
</form>
</div>
<script>
document.addEventListener('DOMContentLoaded', function () {
  const input = document.querySelector('#name');
  const roleFilter = document.querySelector('#status-filter');
  const rows = document.querySelectorAll('.application-list tbody tr');
  const noUserMessage = document.querySelector('#no-application-message');

  input.addEventListener('input', filterUsers);
  roleFilter.addEventListener('change', filterUsers);

  function filterUsers() {
    const searchTerm = input.value.toLowerCase();
    const selectedRole = roleFilter.value;
    let found = false;

    rows.forEach(function (row) {
      const username = row.querySelector('td:nth-child(2)').textContent.toLowerCase();
      const role = row.querySelector('td:nth-child(7)').textContent.toLowerCase();

      if ((selectedRole === 'all' || role.includes(selectedRole)) &&
        (searchTerm.length === 0 || username.includes(searchTerm))) {
        row.style.display = "";
        found = true;
      } else {
        row.style.display = 'none';
      }
    });

    if (found) {
      noUserMessage.style.display = 'none';
    } else {
      noUserMessage.style.display = 'block';
    }
  }
});
</script>
</body>
</html>

```

4. Add Category

```

{% extends 'admindash/base.html' %}
{% load static %}
{% block content %}
<div class="container-fluid">
  <div class="card">
    <div class="card-body">
      <h5 class="card-title fw-semibold mb-4">Add Product Category</h5>
      <script>
        function validateForm() {
          const category_nameInput = document.getElementById('category_name');

```

```

const category_nameError = document.getElementById('category_name_error');
const category_nameValue = category_nameInput.value.trim();

const category_descriptionInput = document.getElementById('category_description');
const category_descriptionError = document.getElementById('category_description_error');
const category_descriptionValue = category_descriptionInput.value.trim();

let isValid = true;

// Validate Category Name
if (category_nameValue === "") {
    category_nameError.textContent = 'Category Name cannot be empty.';
    isValid = false;
} else if (!/^[A-Za-z]+$/.test(category_nameValue)) {
    category_nameError.textContent = 'Category Name should contain only alphabetic
characters.';
    isValid = false;
} else {
    category_nameError.textContent = "";
}

// Validate Category Description
if (!/^[A-Za-z\s]*$/.test(category_descriptionValue)) {
    category_descriptionError.textContent = 'Category Description should contain only
alphabetic characters and spaces.';
    isValid = false;
} else {
    category_descriptionError.textContent = "";
}

return isValid;
}
</script>

<form method="post" enctype="multipart/form-data" onsubmit="return validateForm()">
  {% csrf_token %}
  <div class="form-group" style="margin-top: 30px;">
    <label for="category_name">Category Name:</label>
    <input type="text" id="category_name" name="category_name" class="form-control"
required>
    <span id="category_name_error" class="text-danger"></span>
  </div>
  <div class="form-group" style="margin-top: 10px;">
    <label for="category_description">Category Description:</label>
    <textarea id="category_description" name="category_description" class="form-
control"></textarea>
    <span id="category_description_error" class="text-danger"></span>
  </div>
  <button type="submit" class="btn btn-primary" style="margin-top: 30px;">Create
Category</button>
</form>

</div>
</div>
</div>

```

```
{%endblock% }
```

5. Add Product

```
{%extends 'sellerdash/base.html'% }
{% load static % }
{%block content% }
    <div class="container-fluid">
        <div class="card swing">
            <div class="card-body">
                {% if error_message % }
                <div class="alert alert-danger">
                    {{ error_message }}
                </div>
                {% endif % }
                <h5 class="card-title fw-semibold mb-4">Add Products</h5>
                {% if certification_status == 'approved' % }

                    <form method="post" enctype="multipart/form-data" onsubmit="return validateForm()">
                        {% csrf_token % }
                        <div class="form-group mb-3" style="margin-top: 30px;">
                            <label for="product_name">Product Name:</label>
                            <input type="text" id="product_name" name="product_name" class="form-control"
oninput="validateProductName()" >
                            <span id="product_name_error" class="text-danger"></span>
                        </div>
                        <div class="form-group mb-3" style="margin-top: 10px;">
                            <label for="product_description">Product Description:</label>
                            <textarea id="product_description" name="product_description" class="form-control"
rows="4" oninput="validateProductDescription()"></textarea>
                            <span id="product_description_error" class="text-danger"></span>
                        </div>
                        <div class="form-group mb-3" style="margin-top: 10px;">
                            <label for="product_price">Product Price:</label>
                            <input type="text" id="product_price" name="product_price" class="form-control"
oninput="validateProductPrice()">
                            <span id="product_price_error" class="text-danger"></span>
                        </div>
                        <div class="form-group mb-3" style="margin-top: 10px;">
                            <label for="product_stock">Enter Stock(in KG):</label>
                            <input type="text" id="product_stock" name="product_stock" class="form-control"
oninput="validateProductstock()">
                            <span id="product_stock_error" class="text-danger"></span>
                        </div>
                        <div class="form-group mb-3" style="margin-top: 10px;">
                            <label for="select_category">Select Category:</label>
                            <select id="select_category" name="select_category" class="form-control" required>
                                <option value="">Select Category</option>
                                {% for category % }
                                    <option value="{{ category.id }}">{{ category.category_name }}</option>
                                {% endfor % }
                            </select>
                            <span id="select_category_error" class="text-danger"></span>
                        </div>
                    </form>
                {% endif % }
            </div>
        </div>
    </div>
```

```

        <div class="form-group mb-3" style="margin-top: 10px;">
            <label for="product_image">Category Image:</label>
            <input type="file" accept=".img,.jpg,.png/*" id="product_image"
name="product_image" accept="image/*" oninput="validateProductImage()">
            <span id="product_image_error" class="text-danger"></span>
        </div>
        <input type="hidden" name="product_form" value="1">
        <button type="submit" style="margin-top: 30px;" class="btn btn-primary pulse" id="add-
product-button">Add Product</button>
    </form>
    { % elif certification_status == 'pending' % }
    <p class="certification-message">You need an approved certification to add products.</p>
    { % elif certification_status == 'rejected' % }
        <p class="certification-message">Your certification application has been rejected.</p>
    { % else % }
    <p><a href="{ % url 'addproducts' % }" class="btn btn-danger">Go Back</a></p>
    { % endif % }

</div>
</div>
</div>
<script>
function validateProductName() {
    const productNameInput = document.getElementById('product_name');
    const productNameError = document.getElementById('product_name_error');
    const productNameValue = productNameInput.value.trim(); // Removes leading and trailing
spaces

    // Remove spaces from the input value
    const productNameWithoutSpaces = productNameValue.replace(/\s+/g, "");

    if (!/^[A-Za-z]+$/.test(productNameWithoutSpaces)) {
        productNameError.textContent = 'Product Name should contain only alphabetic
characters.';
    } else {
        productNameError.textContent = "";
    }

    // Set the input field value to the one without spaces
    productNameInput.value = productNameWithoutSpaces;
}

function validateProductDescription() {
    const productDescriptionInput = document.getElementById('product_description');
    const productDescriptionError = document.getElementById('product_description_error');
    const productDescriptionValue = productDescriptionInput.value.trim();

    if (productDescriptionValue === "") {
        productDescriptionError.textContent = 'Product Description is required.';
    } else {
        productDescriptionError.textContent = "";
    }
}

```

```
function validateProductPrice() {
    const productPriceInput = document.getElementById('product_price');
    const productPriceError = document.getElementById('product_price_error');
    const productPriceValue = productPriceInput.value.trim();

    if (isNaN(productPriceValue)) {
        productPriceError.textContent = 'Product Price should be a numeric value.';
    } else {
        productPriceError.textContent = "";
    }
}

function validateProductStock() {
    const productStockInput = document.getElementById('product_stock');
    const productStockError = document.getElementById('product_stock_error');
    const productStockValue = productStockInput.value.trim();

    if (isNaN(productStockValue)) {
        productStockError.textContent = 'Stock (in KG) should be a numeric value.';
    } else if (parseFloat(productStockValue) < 1) {
        productStockError.textContent = 'Stock must be atleast 1 KG';
    } else {
        productStockError.textContent = "";
    }
}

function validateProductImage() {
    const productImageInput = document.getElementById('product_image');
    const productImageError = document.getElementById('product_image_error');
    const productImageValue = productImageInput.value.trim();

    if (productImageValue === "") {
        productImageError.textContent = 'Product Image is required.';
    } else {
        productImageError.textContent = "";
    }
}

function validateForm() {
    validateProductName();
    validateProductDescription();
    validateProductPrice();
    validateProductStock();
    validateProductImage();

    // Check if there are any validation errors
    const errorElements = document.querySelectorAll('.text-danger');
    for (const errorElement of errorElements) {
        if (errorElement.textContent !== "") {
            return false; // Prevent form submission if there are validation errors
        }
    }

    return true; // Submit the form if there are no validation errors
}
```

```
</script>
{%endblock%}
```

6. Shop

```
{%extends 'base.html'%}
{% load static %}
{%block content%}
<!-- END nav -->

<div class="hero-wrap hero-bread" style="background-image: url({% static 'images/bg_1.jpg'
%});">
  <div class="container">
    <div class="row no-gutters slider-text align-items-center justify-content-center">
      <div class="col-md-9 ftco-animate text-center">
        <p class="breadcrumbs"><span class="mr-2"><a
href="index.html">Home</a></span> <span>Products</span></p>
        <h1 class="mb-0 bread">Products</h1>
      </div>
    </div>
  </div>
</div>

<section class="ftco-section">
<div class="container">
<div class="row justify-content-center">
  <div class="col-md-10 mb-5 text-center">
    <ul class="product-category">
      <li><a href="{% url 'shop' %}" >All</a></li>
      <li><a href="{% url 'category_vegetables' %}" >Vegetables</a></li>
      <li><a href="{% url 'category_fruits' %}" >Fruits</a></li>
    </ul>
  </div>
</div>
<div class="row">
  {% for product in products %}
  <div class="col-md-6 col-lg-3 ftco-animate">
    <div class="product {% if product.product_stock <= 0 %} out-of-stock{% endif %}">
      <a href="{% url 'product_single' product.id %}" class="img-prod">
        <div class="img-container">
          
        </div>
        {% if product.product_stock > 0 %}
        <span class="status in-stock">In Stock</span>
        {% else %}
        <span class="status out-of-stock">Out of Stock</span>
        {% endif %}
        <div class="overlay"></div>
      </a>
      <div class="text py-3 pb-4 px-3 text-center">
        <h3><a href="{% url 'product_single' product.id %}">{{ product.product_name
}}</a></h3>
        <p>{{ product.seller }}</p>
      <div class="d-flex">
```



```

<div class="cart-list">
  <table class="table">
    <thead class="thead-primary">
      <tr class="text-center">
        <th>Product</th>
        <th>Product name</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Total</th>
        <th>Remove from Cart</th>
      </tr>
    </thead>
    <tbody>
      {% if cart_items %}
      {% for cart_item in cart_items %}
      <tr class="text-center">
        <td class="product-thumbnail">
          
        </td>
        <td class="product-name">
          <h3>{{ cart_item.product.product_name }}</h3>
        </td>
        <td class="price">RS {{ cart_item.product.product_price }}</td>
        <td class="quantity">
          <form action="{{ url 'update_cart_item' cart_item.id }}" method="post">
            {% csrf_token %}
            <input type="number" name="quantity" value="{{ cart_item.quantity }}"
min="1" class="quantity-input">
            <button type="submit" class="btn btn-update-quantity">
              <i class="fas fa-sync"></i> <!-- FontAwesome refresh icon -->
            </button>
          </form>
        </td>
        {% if messages %}
        {% for message in messages %}
        <div class="alert alert-danger text-center">{{ message }}</div>
        {% endfor %}
        {% endif %}
        <td class="total">RS {{ cart_item.price }}</td>
        <input type="hidden" name="total_price" value="{{ total_price }}">
        <td class="remove-from-cart">
          <form action="{{ url 'remove_from_cart' cart_item.id }}" method="post">
            {% csrf_token %}
            <button type="submit" class="btn btn-remove-from-cart"
id="deleteButton">
              <i class="fas fa-trash"></i> <!-- FontAwesome trash icon -->
            </button>
          </form>
        </td>
      </tr><!-- END TR-->
      {% endfor %}
      {% else %}
      <tr>

```



```

        <td colspan="6" class="text-center">Your cart is empty.</td>
    </tr>
    {% endif %}
</tbody>
</table>
</div>
<div class="cart-total" style="align:center;">
    <h6>Total: RS {{ total_price|floatformat:2 }}</h6>
</div>
</div>
</div>
<div class="row justify-content-end">
    {% if cart_items %}
    <div class="col-lg-4 mt-5 cart-wrap ftco-animate">
        <div class="cart-total mb-3">
            <h3>Cart Totals</h3>
            <p class="d-flex">
                <span>Subtotal</span>
                <span>RS {{ total_price|floatformat:2 }}</span>
            </p>
            <p class="d-flex">
                <span>Delivery</span>
                <span>RS 0.00</span>
            </p>
            <p class="d-flex">
                <span>Discount</span>
                <span>RS 0.00</span>
            </p>
            <hr>
            <p class="d-flex total-price">
                <span>Total</span>
                <span id="total-amount">RS {{ total_price|floatformat:2 }}</span>
            </p>
        </div>
        <!-- Link to proceed to checkout -->
        <p><a href="{% url 'checkout' %}" class="btn btn-primary py-3 px-4">Proceed to
Checkout</a></p>
    </div>
    {% endif %}
</div>
</div>
</section>
<script>
    // Function to handle the delete action
    function handleDelete() {

        // Display a confirmation dialog
        var confirmed = window.confirm('Are you sure you want to delete?');

        // Check if the user confirmed
        if (confirmed) {

        } else {
            event.preventDefault();
            // User canceled the deletion

```

```

        alert('Deletion canceled.');
```

```

    }
}
```

```
// Add an event listener to the button to call the function
```

```
document.getElementById('deleteButton').addEventListener('click', handleDelete);
```

```
</script>
```

```
{%endblock% }
```

Views.py

#cart

```
@login_required
```

```
def cart_view(request):
```

```
    # Assuming you have user authentication and each user has a unique cart
```

```
    user = request.user
```

```
    # Retrieve the user's cart items
```

```
    cart_items = Cart.objects.filter(user=user)
```

```
    cart_item_count = cart_items.count()
```

```
    # Calculate the total price of items in the cart
```

```
    total_price = sum(cart_item.product.product_price * cart_item.quantity for cart_item in
cart_items)
```

```
    context = {
```

```
        'cart_items': cart_items,
```

```
        'total_price': total_price,
```

```
        'cart_item_count': cart_item_count,
```

```
    }
```

```
    return render(request, 'cart.html', context)
```

#login

```
def user_login(request):
```

```
    form = LoginForm(request.POST or None)
```

```
    msg = None
```

```
    if request.method == 'POST':
```

```
        if form.is_valid():
```

```
            username = form.cleaned_data.get('username')
```

```
            password = form.cleaned_data.get('password')
```

```
            user = authenticate(username=username, password=password)
```

```
        if user is not None:
```

```
            login(request, user)
```

```
            auth_login(request, user)
```

```
            if user.is_customer:
```

```
                # login(request, user)
```

```
                request.session['customer_id'] = user.id
```

```
                request.session['user_type'] = 'customer'
```

```
                return redirect('index')
```

```
            elif user.is_seller:
```

```
                # login(request, user)
```

```
                request.session['seller_id'] = user.id
```

```
                request.session['user_type'] = 'seller'
```

```
                return redirect('index')
```

```
            elif user.is_legaladvisor:
```

```

        print("Redirecting to dashlegal")
        request.session['legaladvisor_id'] = user.id
        request.session['user_type'] = 'legaladvisor'
        return redirect('dashlegal')
    elif user.is_superuser:
        print("Redirecting to dashlegal")
        request.session['admin_id'] = user.id
        request.session['user_type'] = 'admin'
        return redirect('admindash')
    else:
        msg = 'Invalid credentials'
    else:
        msg = 'Error validating form'

    return render(request, 'login.html', {'form': form, 'msg': msg})

```

#Add Product

```

@login_required
def addproducts(request):
    existing_certification = Certification.objects.filter(user=request.user).first()

    if existing_certification:
        certification_status = existing_certification.is_approved
    else:
        certification_status = 'pending' # Set a default value if no certification exists
    if certification_status == 'approved':
        if request.method == 'POST':
            product_name = request.POST.get('product_name')
            formatted_product_name = product_name.capitalize()
            product_description = request.POST.get('product_description')
            select_category_id = request.POST.get('select_category')
            product_price = request.POST.get('product_price')
            product_stock = request.POST.get('product_stock')
            product_image = request.FILES.get('product_image')

            # Retrieve the selected category
            category = Category.objects.get(id=select_category_id)

            # Check if a product with the same name already exists in the selected category
            # Check if the current user has already added a product with the same name in this category
            existing_product = Product.objects.filter(
                product_name=formatted_product_name,
                category=category,
                seller__user=request.user # Filter by the current user
            )

            if existing_product.exists():
                error_message = "You have already added a product with this name in the selected category."
                return render(request, 'sellerdash/addproducts.html', {'error_message': error_message})

            # Retrieve the seller associated with the currently logged-in user
            seller = Seller.objects.get(user=request.user)

```

```
# Create and save the Product instance
product = Product(
    product_name=formatted_product_name,
    product_description=product_description,
    category=category,
    product_price=product_price,
    product_stock=product_stock,
    product_image=product_image,
    seller=seller # Associate the seller with the product
)
product.save()

return redirect('successaddproduct') # Redirect to a success page or your desired destination

categories = Category.objects.all() # Retrieve all Category objects from the database

context = {
    'categories': categories,
    'certification_status': certification_status,
}
# Pass the categories queryset to the template context
return render(request, 'sellerdash/addproducts.html', context)
else:
    return render(request, 'sellerdash/addproducts.html', {'certification_status': certification_status})
```

#Shop

```
def shop(request):
    products = Product.objects.all()

    return render(request, 'shop.html', {'products': products})
```

#Legaladvisor

```
@login_required
def dashlegal(request):
    # Retrieve Certification objects
    seller_applications = Certification.objects.all()

    # Retrieve User roles for each Certification applicant
    user_roles = {}
    for application in seller_applications:
        # Ensure the user associated with the Certification exists
        user = get_object_or_404(User, id=application.user_id)

        # Retrieve user roles
        user_roles[application.id] = {
            'is_admin': user.is_admin,
            'is_customer': user.is_customer,
            'is_seller': user.is_seller,
            'is_legaladvisor': user.is_legaladvisor,
        }
```

```
context = {  
    'seller_applications': seller_applications,  
    'user_roles': user_roles, # Include user roles in the context  
}  
return render(request, 'dashlegal.html', context)
```

9.2 Screen Shots


Login Page

A screenshot of the login page. On the left, a white panel contains the text "Green, Clean, Serene." followed by "SIGN IN". Below this are input fields for "User Name" and "Password", a "Forgot password?" link, a red "G" logo, a green "Login" button, and a "Don't have an account? Signup" link. On the right, a brown paper bag is filled with various fresh vegetables like tomatoes, peppers, and eggplants.

Apply for Certification Page

A screenshot of the "Certification Approval" page. The page has a sidebar on the left with the "Eco Hive Organic Product Store" logo and a menu under "SELLER PAGE" with "Certification" highlighted. The main content area is titled "Certification Approval" and contains several form fields: "First Name:", "Last Name:", "Expiry Date:" (with a date picker), "Certifying Body:", "Phone Number:", "Certification Number:", "Full Address:", and "Copy of Certification:" (with a file upload button). A blue "Submit Certification Request" button is at the bottom right. The top right corner shows a user profile and the text "Welcome, seller7".

Add Product



ECO HIVE
Organic Product Store

SELLER PAGE

Certification


Home Page

SELLER FUNCTIONALITIES

Add Products

View Added Products

View Product orders

Welcome, seller1

Add Products

Product Name:

Product Description:

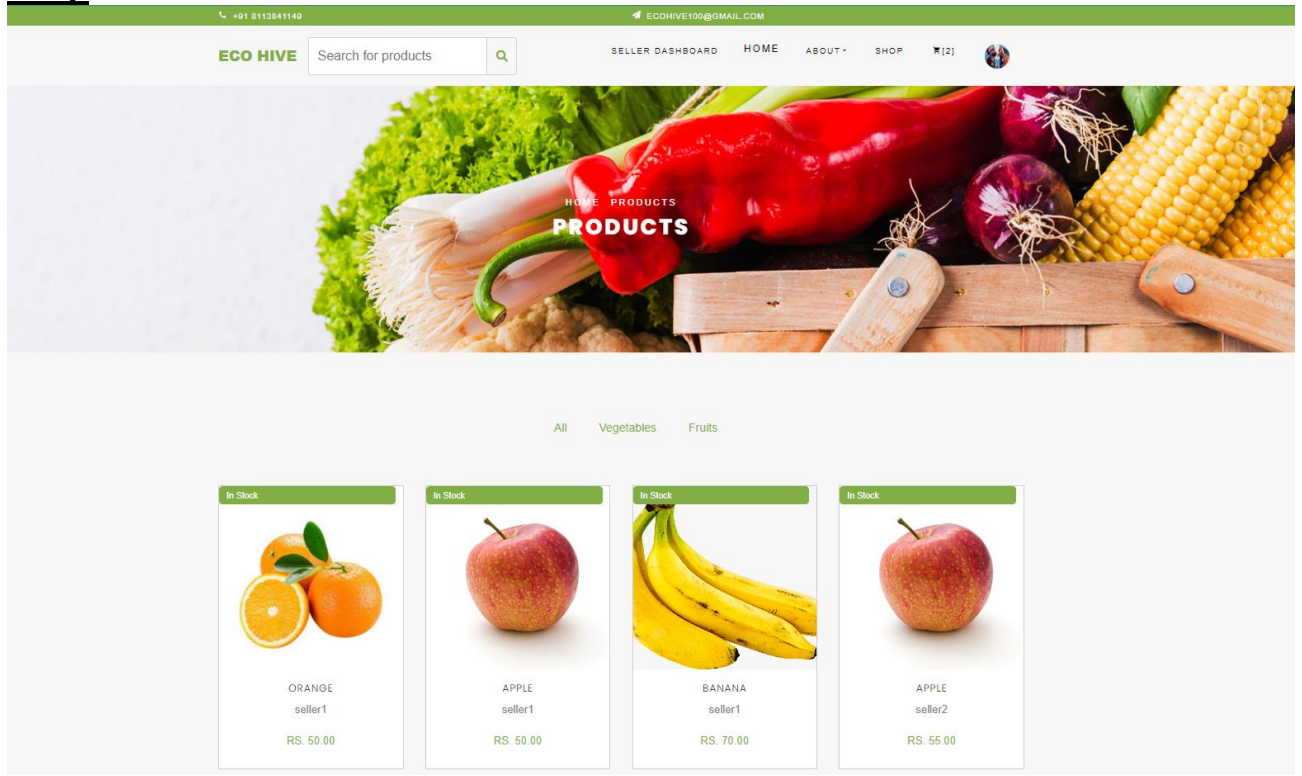
Product Price:

Enter Stock(in KG):

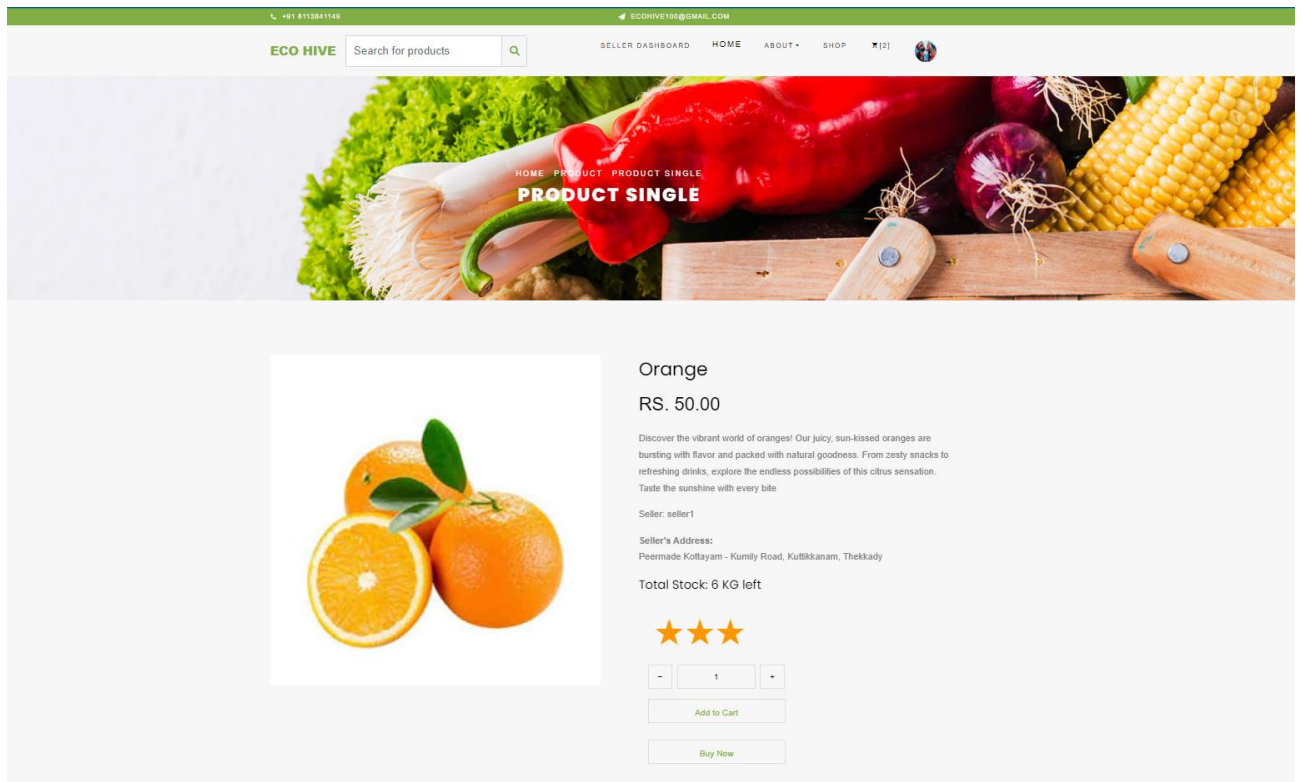
Select Category:

Category Image: No file chosen


Shop









Single Product Page



Cart



Product	Product name	Price	Quantity	Total	Remove from Cart
	Banana	RS 70.00	<input type="text" value="1"/> 	RS 70.00	
	Ginger	RS 112.00	<input type="text" value="2"/> 	RS 224.00	

Total: RS 294.00

Seller Approval

Welcome, Legaladvisor123 [Logout](#)

Filter by status: All Search by name: [Search](#)

Legal Administrator Dashboard

Application ID	First Name	Last Name	Expiry Date	Certification Authority	Phone Number	Certification Number	File	Action
1	seller	one	Nov. 30, 2023	NPOP	09961695895	5845236985412	View File	<div>Approved Reject</div>
2	seller	two	Dec. 14, 2023	USOCA	8113841149	5845236985412	View File	<div>Approved Reject</div>
3	seller	three	Dec. 30, 2023	NPOP	9947161615	8767675467678578	View File	<div>Approved Reject</div>