



SCHOOL OF TECHNOLOGY

BACHELOR OF SCIENCE IN INFORMATION AND TECHNOLOGY

FINAL YEAR PROJECT II: BIT 04205

KCAVIBES E-COMMERCE WEB APPLICATION

BY

AWANZI HASSAN MUNENE (21/04609)

PRESENTED TO: Prof. PATRIC OGAO

SUBMISSION DATE: 14/03/2024

1.0 Introduction.....	2
2.0 Purpose.....	3
3.0 SCOPE.....	4
4.0 SYSTEM DESIGN AND CONSTRAINTS.....	5
5.0 Design Goals and Objectives.....	6
6.0 Objectives.....	7
7.0 Overview of Document.....	7
8.0 System Architecture.....	8
Introduction.....	8
9.0 The Client-Server Model.....	9
10.0 Design Approach: Modularity.....	10
11.0 Architectural Design.....	12
Two-Tier Architectural Design.....	12
Data Flow and Retrieval.....	13
12.0 Use Case Diagrams.....	14

Actors.....	14
Customer (Buyer):.....	14
Vendor (Seller):.....	14
Administrator:.....	14
13.0 Use Case Diagram.....	15
14.0 Activity Diagrams.....	16
New vendor Account Request Activity.....	16
Admin Login Activity.....	18
15.0 Class Diagram.....	19
Vendor Class.....	19
Product Class.....	19
Customer Class.....	20
Order Class.....	20
16.0 Deployment Diagram.....	21
Web Server.....	21
Application Server.....	21
Database Server.....	21
LAN (Local Area Network).....	21
17.0 DATABASE DESIGN.....	22
18.0 NORMALIZATION.....	23
19.0 Entity-Relationship Diagram (ERD).....	27
20.0 Entities.....	27
21.0 User Interface Design.....	28
Customer-Facing Interfaces.....	28
22.0 Vendor Interfaces.....	29
23.0 Common Interfaces.....	29
24.0 User Experience (UX) Principles.....	29
Vendor Login Form.....	30
Vendor Dashboard.....	30
Components Available.....	32
25.0 Conclusion.....	33

1.0 Introduction

The software design document is a document to provide documentations, which will be used to aid in software development by providing the details for how the software should be built.

Within the software design development are narrative and graphical documentations of the software design for the project including use case models, sequence diagrams, class diagrams, activity diagrams and other supporting requirements information.

This software design specification provides the design details of Kcavibes e-commerce.

In the systems design we are concerned with producing appropriate design which results in a good quality information system that is;

- Provide the correct function of the users.
- Reliable

- Secure
- Well integrated with other systems
- Easy to use
- Rapid in retrieving data and moving between different screen views of the data.

In the software engineering context, design focuses on four major areas of concern, data, architecture, interfaces and components

2.0 Purpose

The primary purpose of this Software Design Specification (SDS) is to serve as a comprehensive guide for the design of our Multi-Vendor E-commerce Web Application. This document is intended to provide a detailed and clear description of the system's design, enabling the software development team to proceed with a profound understanding of what needs to be built and how it should be constructed.

Specifically, this section of the SDS aims to fulfill the following objectives:

Design Clarity: To provide a clear and unambiguous representation of the software design, ensuring that all stakeholders have a common understanding of the project's scope, features, and architecture.

Development Roadmap: To establish a structured roadmap for the development team, outlining the design considerations, components, and their interconnections, thus facilitating efficient and organized software development.

Quality Assurance: To serve as a benchmark for evaluating the design against a predefined set of quality criteria, thereby ensuring that the resulting system is well-designed and upholds the desired standards of excellence.

Communication Tool: To act as a communication tool between different stakeholders, including developers, project managers, testers, and system users, facilitating effective collaboration and alignment of goals.

In essence, this SDS is a vital document that bridges the gap between conceptualization and implementation, laying the foundation for the creation of a robust and user-friendly Multi-Vendor E-commerce Web Application.

3.0 Scope

The scope of this Software Design Specification (SDS) encompasses the design and development of a Multi-Vendor E-commerce Web Application, leveraging the MERN stack (MongoDB, Express, React, Node.js) to fulfill the specific requirements of the project. This SDS is intended to guide the creation of a base-level system, functioning as a proof of concept, to demonstrate the feasibility of building a comprehensive, large-scale production system.

Key points within the scope of this SDS are as follows:

Base-Level System: This document focuses on the design of the initial system, emphasizing core functionality and architectural considerations necessary for the Multi-Vendor E-commerce Web Application.

Integration: The system is designed to seamlessly integrate with other pre-existing systems, ensuring compatibility and interoperability to meet business requirements.

Technological Stack: The basic architecture of the system follows the client/server paradigm, with the MERN stack forming the backbone of the application. MongoDB is chosen for database storage, Express for the backend, React for the frontend, and Node.js for server-side scripting.

Administrative Access: A designated administrator will have full access rights to the Multi-Vendor E-commerce Web Application, allowing them to make essential changes as needed. Administrative responsibilities encompass various aspects, such as modifying system entries, managing data collected from vendors, and configuring user access and permissions.

The purpose of this SDS is to provide a comprehensive framework for designing and building the Multi-Vendor E-commerce Web Application, ensuring that it aligns with your project's specific goals and objectives. This document serves as a foundational resource to guide the development team in constructing a reliable and functional system.

4.0 System Design and Constraints

The system design for our Multi-Vendor E-commerce Web Application is influenced by a range of factors, including user requirements, environmental constraints, and the choice of the MERN (MongoDB, Express, React, Node.js) stack as our technological foundation.

Key considerations and constraints within this system design are as follows:

User Requirements Specification: The system design is intrinsically bound to the User Requirements Specification (URS) derived from meticulous system analysis. The URS serves as a blueprint for defining the required functionalities and features that must be seamlessly integrated into the design.

Hardware and Software Environment: The design is also influenced by the existing hardware and software environment. Specific environmental constraints include:

- o **Hardware Limitations:** In some cases, the system may require an increase in memory or computing resources beyond the capabilities of the current hardware infrastructure.
- o **Data Integration:** Ensuring the smooth integration of data from pre-existing systems into the new Multi-Vendor E-commerce Web Application is a critical consideration.
- o **Interfacing Challenges:** The integration and interfacing with other applications or external services may pose challenges that need to be addressed, considering the complex web of interactions in a multi-vendor e-commerce ecosystem.
- o **Choice of Database:** The decision to use MongoDB as the database management system is grounded in the need for flexibility, scalability, and a NoSQL approach, although it may involve significant development efforts.

Testing and User Engagement: The system design includes provisions for an internal testing phase. The challenge here lies in ensuring that the system undergoes rigorous testing without overburdening users with extensive testing requirements. Providing clear guidance to users during the testing phase is essential to obtain valuable feedback and make necessary adjustments.

In summary, the system design is shaped by a blend of user requirements, technological choices, and environmental constraints. It is essential to strike a balance between the functionality envisioned in the URS and the practical considerations of working within the existing hardware and software environment. By adopting the MERN stack, we aim to create a flexible and scalable architecture that aligns with the goals of the Multi-Vendor E-commerce Web Application.

5.0 Design Goals and Objectives

The paramount objective of our Multi-Vendor E-commerce Web Application is to design a system that fulfills the diverse and dynamic needs of the platform, fostering its growth, functionality, and quality. In this section, we outline the specific design goals and objectives that guide our software development process, considering the utilization of the MERN stack.

Functionality and Feature-Rich: Our primary goal is to create a Multi-Vendor E-commerce Web Application that is rich in features and functionality. This includes robust vendor management, product listings, order processing, and a seamless user experience, ensuring that all essential e-commerce operations are seamlessly supported.

Quality-Centric: Quality is paramount in software design, and it serves as the foundation of our entire development process. Quality assurance encompasses not only the reliability and robustness of the application but also factors like security, performance, and scalability.

Translating User Requirements: Software design acts as a bridge between user requirements and the final product. Our objective is to accurately translate the diverse needs of vendors, customers, and administrators into a cohesive and user-friendly system. This involves the careful mapping of business requirements to technical solutions.

Foundation for Software Engineering: The design is intended to serve as the cornerstone for all subsequent software engineering and maintenance activities. It sets the stage for coding, testing, deployment, and future enhancements. Therefore, the design must be well-structured, adaptable, and maintainable.

Stability and Adaptability: A stable system is one that can accommodate changes, whether they are minor updates or significant expansions. We aim to design a system that can evolve alongside the changing demands of the e-commerce landscape, providing room for growth and adaptability.

Testing and Quality Assessment: The design emphasizes thorough testing and quality assessment, enabling us to identify issues early in the software engineering process. This proactive approach minimizes the risk of building an unstable system that may fail when subjected to changes or testing late in the development process.

In conclusion, our design goals and objectives are rooted in the fundamental principles of delivering a high-quality, functional, and adaptable Multi-Vendor E-commerce Web Application. By adhering to these objectives and leveraging the power of the MERN stack, we aim to create a system that meets the expectations of vendors, customers, and administrators alike.

6.0 Objectives

The design of our Multi-Vendor E-commerce Web Application is guided by a set of clear and crucial objectives tailored to the needs of our platform. These objectives are framed to address key aspects of the system:

Security: Security is paramount. Our system design aims to incorporate comprehensive security measures that restrict access to authorized users only. This includes user authentication, data encryption, and measures to safeguard sensitive information such as payment details.

Ease of Use: User-friendliness is a top priority. We are committed to designing a system that is intuitive and easy to use for all stakeholders, including vendors, customers, and administrators. A clean and straightforward user interface, along with clear navigation, is integral to achieving this objective.

Flexibility and Adaptability: The e-commerce landscape is dynamic. Our design objective is to ensure the system's flexibility, enabling it to adapt to changing requirements without undue complexity. Future enhancements and modifications should be accommodated with ease, allowing the system to evolve alongside business needs.

By aligning our design objectives with these principles, we aim to create a Multi-Vendor E-commerce Web Application that is secure, user-friendly, and adaptable to the evolving demands of the e-commerce market.

7.0 Overview of Document

The Software Design Specification for our Multi-Vendor E-commerce Web Application is organized into several sections, each serving a distinct purpose in detailing the design and structure of the system. These sections and their respective subsections provide a comprehensive overview of the document:

Architectural Design: This section delves into the architectural design of the application. It defines the design entities that collaboratively work to perform various functions within the system. Each design entity is accompanied by an abstract description outlining the services it provides to the rest of the system. Additionally, each design entity is further expanded to reveal a set of lower-level design operations that collaborate to fulfill their designated services.

File and Data Structure Design: Here, we address the design of the file and data structures used in the application. This includes outlining how data is organized, stored, and accessed. Given the MERN stack's focus on data management, this section plays a crucial role in optimizing data storage and retrieval mechanisms.

Normalization: Normalization is a key design activity that enhances the logical storage of data within the database. This section discusses the strategies and techniques employed to ensure that data is structured efficiently, eliminating redundancy and reducing anomalies. In the context of your application, data integrity and database optimization are of utmost importance.

Conclusion: The concluding section of the System Design Document provides a summary of how the system is designed to fulfill user requirements. It ties together the design aspects discussed in the preceding sections and sets the stage for the development and implementation phases.

These sections collectively form the foundation of our Multi-Vendor E-commerce Web Application's software design, ensuring that we meet our objectives and deliver a high-quality, secure, and adaptable system

8.0 System Architecture

Introduction

Software architecture serves as the foundational blueprint for our Multi-Vendor E-commerce Web Application. Just as a structural architect carefully plans and designs a building, a software architect must meticulously analyze requirements, select the appropriate components, and

navigate the challenges throughout the development cycle. Drawing upon our experiences, we're well-equipped to strategize, leverage domain knowledge, and harness various tools and technologies to create a robust and scalable e-commerce platform.

Our architecture aims to:

Analyze Requirements: Thoroughly dissect the requirements of our multi-vendor e-commerce platform. This entails understanding the diverse needs of vendors, customers, and administrators, as well as the functionalities that must be provided.

Component Selection: Choose the right components that constitute the MERN stack (MongoDB, Express.js, React, Node.js) to create a flexible and powerful architecture. These components have been selected for their ability to support real-time interactions, responsive user interfaces, and robust server-side logic.

Strategic Planning: Develop a strategic plan for the execution cycle, considering various phases of development, testing, and deployment. This includes a well-defined roadmap for feature implementation and system scaling.

Domain Knowledge: Leverage our deep understanding of the e-commerce domain to address unique challenges and ensure that the architecture aligns with industry best practices.

Technological Expertise: Utilize our expertise in various technologies and tools to make informed choices during the design and development process. This enables us to anticipate and resolve technical issues effectively.

Our system architecture is designed to create a scalable, secure, and responsive multi-vendor e-commerce platform, offering seamless interactions for users and supporting the complex functionalities required in this domain.

9.0 The Client-Server Model

The client-server architectural model is fundamental to our Multi-Vendor E-commerce Web Application. It delineates the distribution of data and processing among various system components. In this model, we have identified the following major components:

Stand-Alone Servers: These servers act as dedicated service providers, offering specialized functions to other sub-systems within our application. Examples of these servers include:

Product Servers: Handling product catalog and inventory management.

User Authentication Servers: Responsible for user account management, authentication, and authorization.

Payment Processing Servers: Managing secure payment transactions.

Notification Servers: Handling real-time notifications to users and vendors.

Clients: Clients represent various sub-systems, each with distinct responsibilities. These clients are self-contained and may run independently. They include:

Vendor Dashboard: Empowering vendors to manage their product listings, orders, and customer interactions.

Customer Portal: Providing customers with the ability to browse, search, and purchase products.

Administrator Control: Allowing administrators to oversee the entire platform, manage users, and address issues.

Search and Recommendation Engine: Offering personalized product recommendations based on user behavior.

Payment Gateway Interface: Enabling secure payment processing and transaction management.

Network Infrastructure: The network acts as the bridge connecting clients to servers, facilitating communication and data exchange. This network infrastructure can be both internal, within the system, and external, enabling connections with external services and APIs.

Key characteristics of this model include:

Client Knowledge: Clients must be aware of the available servers and the specific services they offer. This knowledge is crucial for directing requests to the appropriate server.

Server Agnosticism: Servers do not need detailed information about the identity of clients or how many clients are in operation. They focus on providing services in response to incoming requests, ensuring a more decoupled and scalable system.

Remote Access: Clients access services provided by servers through remote access, enabling distributed processing and data retrieval.

By adopting the client-server architectural model, we aim to build a robust, scalable, and efficient e-commerce platform that can handle the diverse needs of vendors, customers, and administrators. This model allows us to distribute processing, ensuring a responsive and reliable system.

10.0 Design Approach: Modularity

Modularity is at the core of our design approach, ensuring that our software is divided into distinct, named, and addressable components called modules. These modules are integrated to fulfill the diverse requirements of our multi-vendor e-commerce platform. Our design approach revolves around the following key principles:

Dividing Complex Systems: Complex software systems can be challenging to understand and manage. By breaking the system into modules, we simplify the design and development process, making it more comprehensible and manageable.

Control Paths: Monolithic software often involves a myriad of intertwined control paths. Such complexity can lead to errors and difficulties in debugging and maintaining the system. By adopting a modular approach, we minimize control path complexity, making it easier to troubleshoot and enhance the system.

Reference Span: In monolithic systems, the reference span—the extent to which different components interact—is extensive, leading to tightly coupled code. Modularity reduces this reference span, fostering loose coupling and improving system flexibility.

Variable Management: A large number of variables in monolithic software can create confusion and increase the likelihood of bugs. Modular design encapsulates variables within individual modules, reducing naming conflicts and enhancing code organization.

Scalability: Modular systems are inherently more scalable. New features, components, or services can be added without significantly impacting existing modules. This flexibility is vital for accommodating future enhancements.

Maintenance and Testing: Modularity simplifies software maintenance. Modules can be isolated and tested independently, streamlining the debugging and testing processes.

In the context of the MERN stack, our modularity approach translates into the following:

Front-End (MERN): We divide the front-end into modular components, each responsible for specific UI elements, pages, or features. React components and Redux for state management ensure modularity and maintainability.

Back-End (MERN): Node.js and Express enable modular routing and middleware, while MongoDB facilitates the organization of data into collections, further enhancing modularity.

Microservices: For specific functionalities, we may employ microservices, which are inherently modular. Each microservice is responsible for a well-defined task, ensuring clean separation of concerns.

APIs and Integration: Our API design adheres to RESTful principles, enabling the creation of modular endpoints for data exchange between various system components.

Third-Party Integrations: Integrations with third-party services or vendor APIs are implemented as modular components, ensuring ease of maintenance and scalability.

This design approach empowers us to build a system that is not only comprehensible and manageable but also adaptable to the evolving needs of a multi-vendor e-commerce platform. Modularity is the cornerstone of our architecture, fostering scalability, maintainability, and efficient development.

11.0 Architectural Design

In the architectural design of our Multi-Vendor E-commerce Web Application using the MERN stack, we establish the structural framework of the system, defining the program structure and data structure. Our architectural design is tailored to create a robust, scalable, and modular system to support the functionalities of a multi-vendor e-commerce platform.

Two-Tier Architectural Design

We adopt a **two-tier architectural design** for our system. In this design, the application is divided into two primary tiers or layers: the client-side and the server-side. Each tier has specific responsibilities and functions:

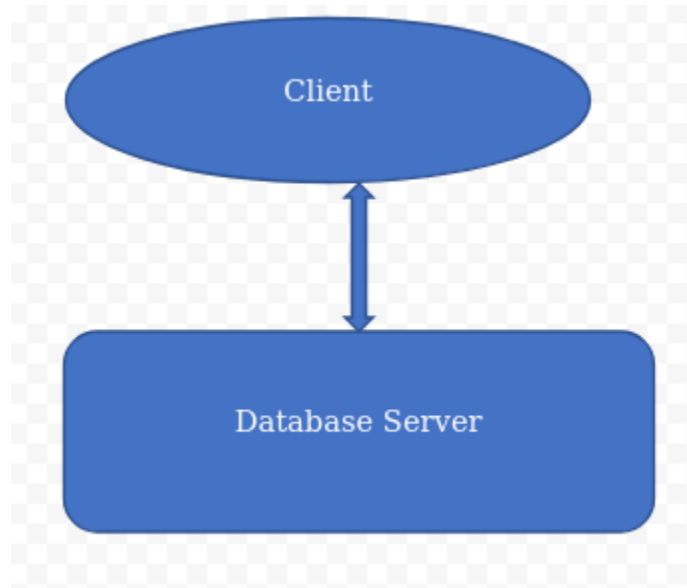
Client-Side (Front-End):

1. This tier encompasses the user interface (UI) and presentation logic of our application.
2. It's responsible for rendering web pages, handling user interactions, and presenting data to users.
3. We use the MERN stack for this tier, comprising **MongoDB, Express.js, React, and Node.js**.
4. React components are used to build modular and reusable UI elements, ensuring a dynamic and responsive user interface.
5. The Redux library is utilized for state management, promoting a clear and maintainable application structure.

Server-Side (Back-End):

1. This tier manages application logic, business processes, and data storage.
2. It includes the server, which responds to client requests and interacts with the database for data retrieval and storage.
3. Our back-end architecture utilizes Node.js and Express.js, providing a scalable and efficient server environment.

4. Data is stored in a **MongoDB database**, which is known for its flexibility and scalability, making it suitable for e-commerce platforms.
5. Express.js handles routing, middleware, and API endpoints, ensuring data is processed and served efficiently.



Data Flow and Retrieval

Data flow within our application is carefully structured to optimize performance and reliability. Here's how data flows through the system:

User Interaction: Users interact with the front-end interface, making requests, searching for products, adding items to the cart, and completing transactions.

Front-End Processing: The React components on the client-side process user input and make requests to the back-end server using HTTP requests.

Back-End Logic: The Express.js back-end server receives and processes these requests. It handles business logic, user authentication, and data manipulation.

Database Interaction: To retrieve and store data, the back-end interacts with the MongoDB database using **MongoDB Query/Retrieval**. MongoDB's NoSQL design is well-suited for the dynamic data structures typical of e-commerce systems.

Data Response: The server sends back the requested data or confirmation of actions to the front-end, which updates the user interface accordingly.

By employing this two-tier architecture and optimizing data flow, our multi-vendor e-commerce platform can efficiently handle user interactions, product catalog management, order processing, and more.

This architectural design lays the foundation for our system's development, enabling us to build a scalable and responsive e-commerce platform.

12.0 Use Case Diagrams

A **use case diagram** is a visual representation that illustrates the interactions between various actors (users or external systems) and the system itself, highlighting the primary use cases or functionalities. Use case diagrams help in understanding the system's behavior from a user's perspective and serve as a foundation for more detailed design and development.

Actors

In our Multi-Vendor E-commerce Web Application, we have identified several key actors who interact with the system. These actors play distinct roles and have specific responsibilities within the application:

Customer (Buyer):

A customer, often referred to as a "Buyer," is a user who visits the platform to browse, search for products, add items to their cart, and complete the purchase process.

Vendor (Seller):

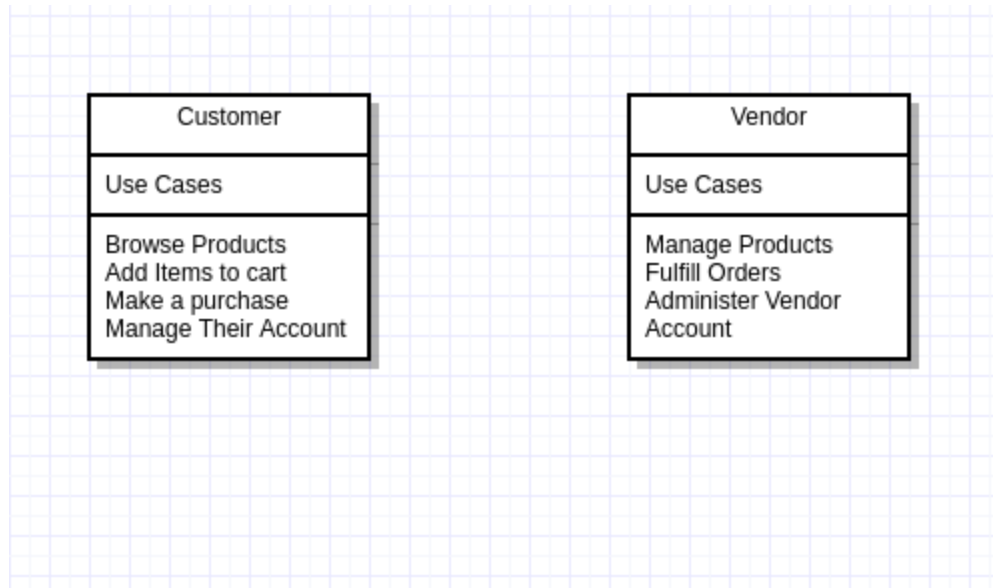
Vendors are users who register on the platform to sell their products. They can list, manage, and update their product listings, as well as fulfill orders from customers.

Administrator:

The administrator is responsible for overseeing the overall system, including account management, platform settings, and ensuring the smooth operation of the application. They have a higher level of access and control compared to other users.

13.0 Use Case Diagram

The use case diagram below depicts the interactions between two primary actors: **Customer (Buyer)** and **Vendor (Seller)**. This diagram provides a high-level overview of the key functionalities associated with these actors.



In this diagram:

- The **Customer (Buyer)** actor can perform actions like **Browsing Products, Adding Items to Cart, Making a Purchase, and Managing Their Account.**
- The **Vendor (Seller)** actor can perform actions such as **Managing Products, Fulfilling Orders, and Administering Their Vendor Account.**

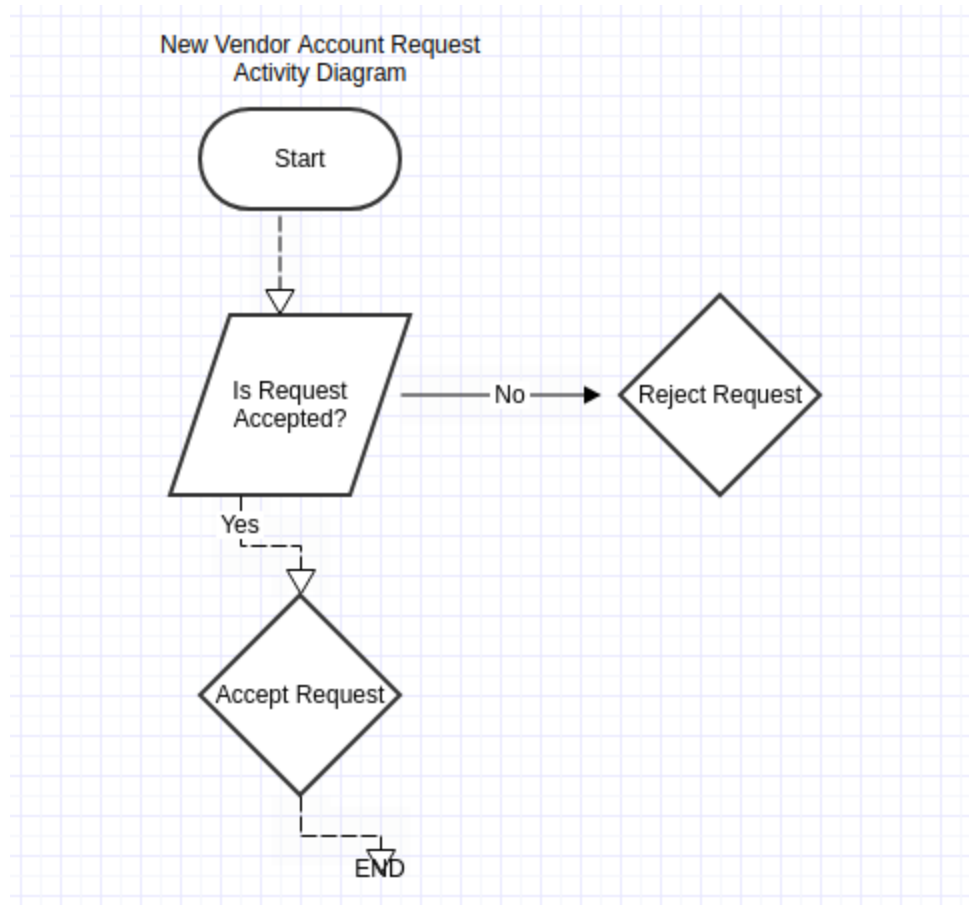
The use case diagram serves as a reference for understanding the interactions and responsibilities of different actors within the application. As we proceed with the design and development of the system, we will further detail the specific use cases and their associated functionality.

14.0 Activity Diagrams

In our Multi-Vendor E-commerce Web Application, **activity diagrams** are essential tools for modeling the dynamic aspects of the system. These diagrams illustrate the flow of control or the sequence of activities that occur during various processes. They help visualize and specify the steps in a computational process, whether sequential or concurrent.

New vendor Account Request Activity

The activity diagram below represents the activities involved when a new student requests an account on the platform:

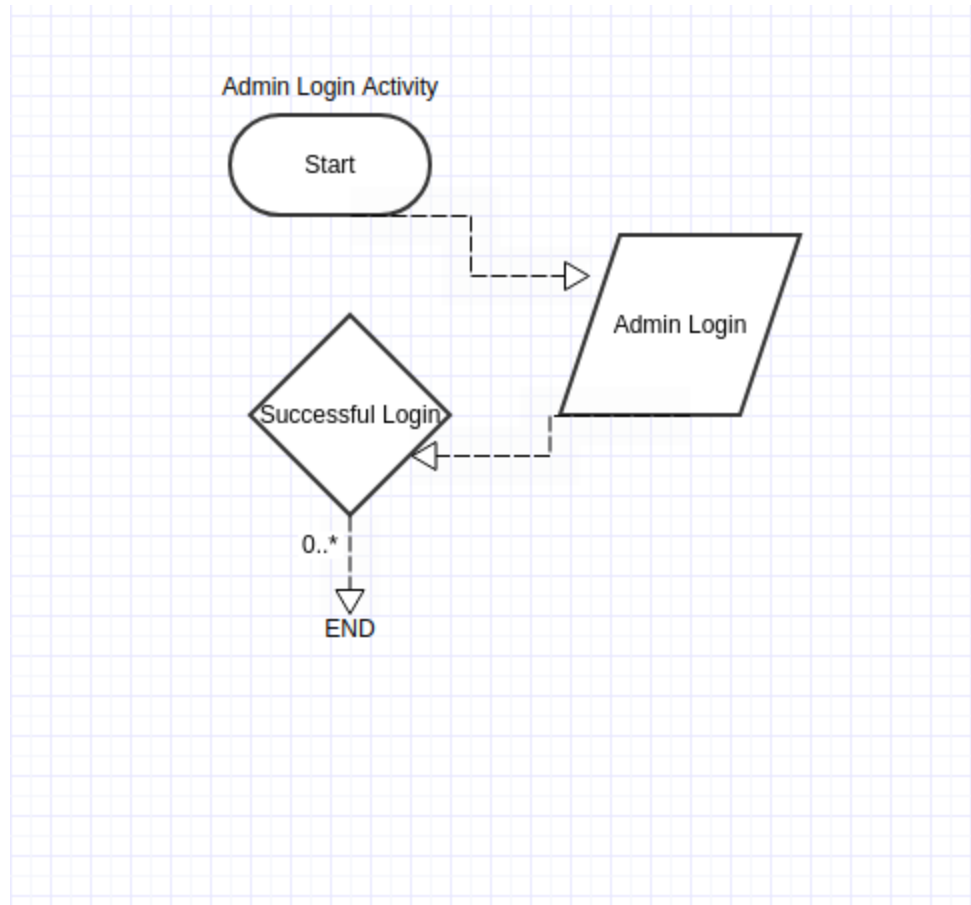


In this diagram:

- **Start** marks the beginning of the activity.
- The process involves two possible paths:
 - If the request is **Accepted**, the flow leads to the **Accept** activity.
 - If the request is **Rejected**, the flow leads to the **Reject** activity.
- The process ends with the **End** activity.

Admin Login Activity

The following activity diagram depicts the activities involved when an admin logs in to the system:



In this diagram:

- The process starts with the **Start** activity.
- The admin's login process is shown, and upon successful login, the flow leads to the **End** activity, indicating the completion of the login process.

These activity diagrams provide an overview of the sequential steps and control flow for these specific scenarios. They can serve as a starting point for more detailed process modeling and, eventually, for the implementation of these activities in the software

15.0 Class Diagram

In a multi-vendor e-commerce web application, you would typically have the following key classes:

Vendor Class

The Vendor class represents the vendors or sellers using the platform to list and manage their products. It may include attributes like:

- VendorID: Vendor's unique identifier
- VendorName: Name of the vendor
- Products[]: An array or list of products offered by the vendor

Product Class

The Product class represents the products available in the e-commerce platform. It may include attributes such as:

- ProductID: Unique product identifier
- ProductName: Name of the product
- Description: Product description
- Price: Product price
- VendorID: ID of the vendor selling the product
- Category: Product category or type
- Rating: Product rating
- Reviews[]: An array of reviews for the product

Customer Class

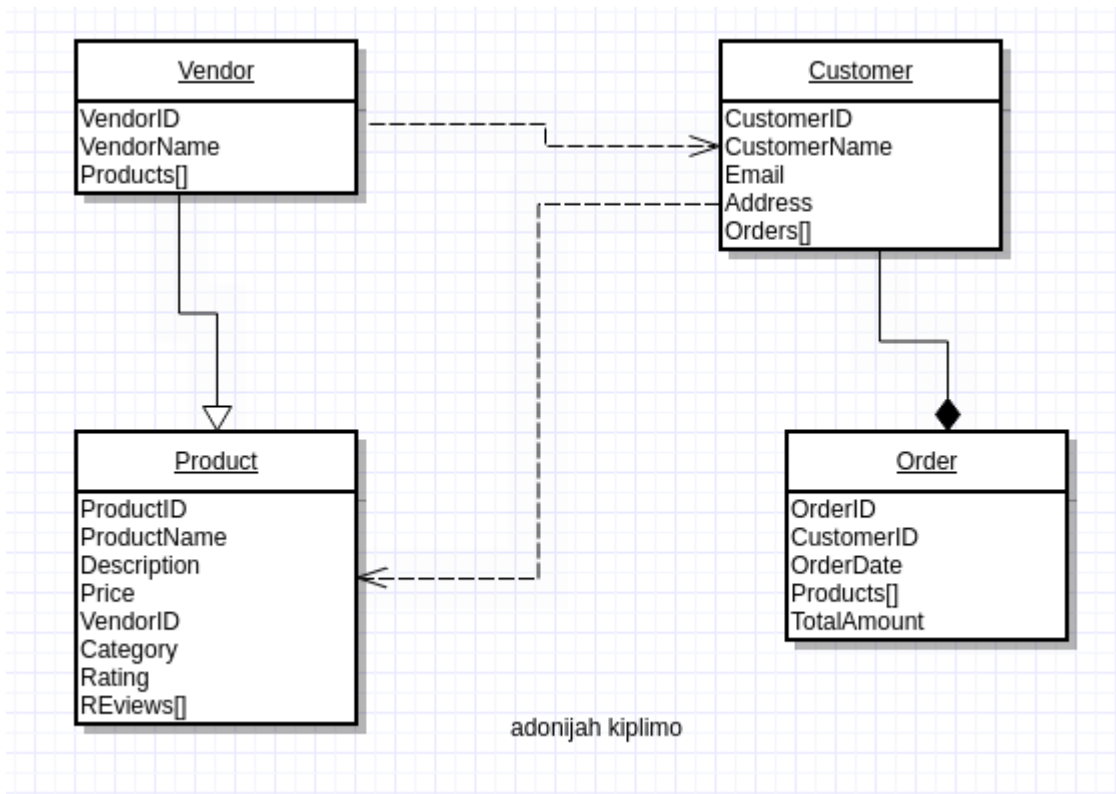
The Customer class represents the users (buyers) of the e-commerce platform. It may include attributes like:

- CustomerID: Customer's unique identifier
- CustomerName: Customer's name
- Email: Customer's email address
- Address: Customer's shipping address
- Orders: An array of orders placed by the customer

Order Class

The Order class represents individual orders made by customers. It may include attributes such as:

- OrderID: Unique order identifier
- CustomerID: ID of the customer who placed the order
- OrderDate: Date and time of the order
- Products: A list of products included in the order
- TotalAmount: Total order amount



16.0 Deployment Diagram

In a basic deployment scenario, you would typically have the following components:

Web Server

The web server hosts the application logic and user interface. It serves web pages to users, handles user interactions, and communicates with the application server and database. This component is typically responsible for rendering web pages to end-users.

Application Server

The application server contains the core business logic of the e-commerce platform. It handles processes like user authentication, product management, order processing, and communication between the web server and the database.

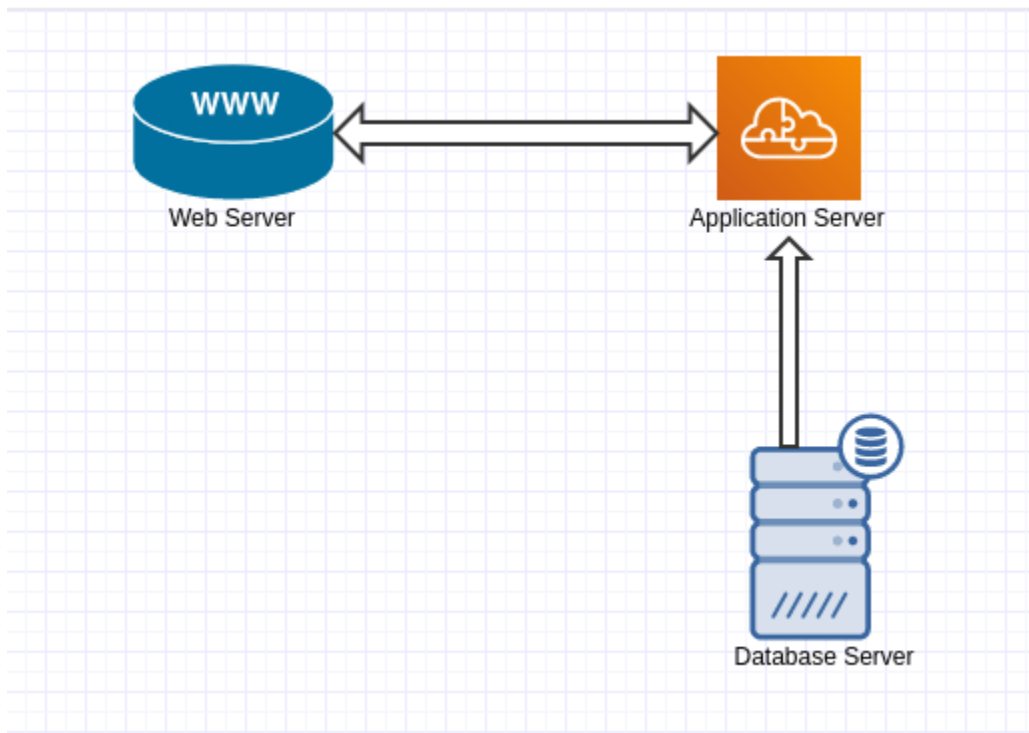
Database Server

The database server stores and manages data used by the application, such as user profiles, product listings, order history, and reviews. The database server communicates with the application server to retrieve and update data.

LAN (Local Area Network)

The Local Area Network (LAN) represents the network infrastructure that connects these hardware components. It ensures that data can flow between the web server, application server, and database server. This network is vital for enabling communication between different components of the system.

This is a simplified deployment diagram, and in a real-world scenario, you might have load balancers, multiple web servers, redundancy for fault tolerance, and other elements for a robust and scalable deployment.



17.0 Database Design

Introduction

A database is a fundamental component of a multi-vendor e-commerce web application, and it plays a critical role in structuring and managing data to ensure consistency and accessibility. In the context of this project using the MERN stack, MongoDB, a NoSQL database, will be utilized as the database management system.

MongoDB is a popular choice for web applications like multi-vendor e-commerce platforms for several reasons:

Flexibility: MongoDB's NoSQL nature allows for the storage of diverse and unstructured data, making it suitable for managing product listings, user profiles, order information, and more without the need for a predefined schema.

Scalability: MongoDB is designed to handle large volumes of data and traffic, ensuring that your e-commerce platform can scale to accommodate a growing number of vendors and customers.

Ease of Integration: MongoDB integrates seamlessly with Node.js and Express.js, the back-end components of the MERN stack. This simplifies the process of data retrieval and manipulation.

Performance: MongoDB is known for its fast read and write operations, which are essential for delivering a responsive and efficient user experience in an e-commerce application.

In the context of a multi-vendor e-commerce application, the database design aims to support atomicity, consistency, isolation, and durability (ACID properties) to ensure data integrity and reliability. This is particularly important because the database is at the core of the application and plays a vital role in managing vendors, products, orders, and user interactions.

The choice of MongoDB as the database system for this project is based on its proven capabilities in handling the data needs of modern web applications. With MongoDB, your e-commerce platform can ensure data consistency and provide scalability, making it a reliable choice for this multi-vendor e-commerce web application

18.0 Normalization

Normalization is a crucial database design technique that optimizes data storage, minimizes redundancy, and helps maintain data integrity. In the context of a multi-vendor e-commerce web application, it's important to organize data efficiently. Below are the tables demonstrating the normalization process:

1NF:

Table 1: Users

UserID	UserName	Password	Email	Role
1	user1	pass1	user1@gmail.com	Customer
2	user2	pass2	user3@gmail.com	admin
3	user3	pass3	user3@gmail.com	vendor

Table 2: Products

ProductID	ProductName	VendorID	Description	Price	
1	ProductA	2	DescriptionA	10000	
2	ProductB	1	DescriptionB	500.00	

Table 3: Orders

OrderID	UserID	OrderDate	Status
1	1	02/11/2023	Delivered
2	2	02/11/2023	Processing

Table 4: Vendors

VendorID	VendorName	Contact
1	Vendor1	vendor1@gmail.com
2	Vendor2	vendor2@gmail.com

2NF:

Table 1: Users

UserID	UserName	Password	Email	Role
1	user1	pass1	user1@gmail.com	Customer
2	user2	pass2	user3@gmail.com	admin
3	user3	pass3	user3@gmail.com	vendor

Table 2: Products

ProductID	ProductName	VendorID	Description	Price	•
1	ProductA	2	DescriptionA	10000	•
2	ProductB	1	DescriptionB	500.00	•

Table 3: Orders

OrderID	UserID	OrderDate	Status
1	1	02/11/2023	Delivered
2	2	02/11/2023	Processing

Table 4: Vendors

VendorID	VendorName	Contact
1	Vendor1	vendor1@gmail.com

2	Vendor2	vendor2@gmail.com
---	---------	-------------------

3NF

Table 1: Users

UserID	UserName
1	user1
2	user2

Table 2: Passwords

UserID	PasswordHash
1	Pass1hash
2	pass2hash

Table 3: Emails

UserID	EmailAddress
1	user1@gmail.com
2	usr2@gmail.com

Table 4: Products

ProductID	ProductName	Description	Price	
1	ProductA	DescriptionA	1000.00	
2	ProductB	DescriptionB	20000.00	

Table 5. Vendors

VendorID	VendorName
1	Vendor1
2	Vendor2

Table 6. Contacts

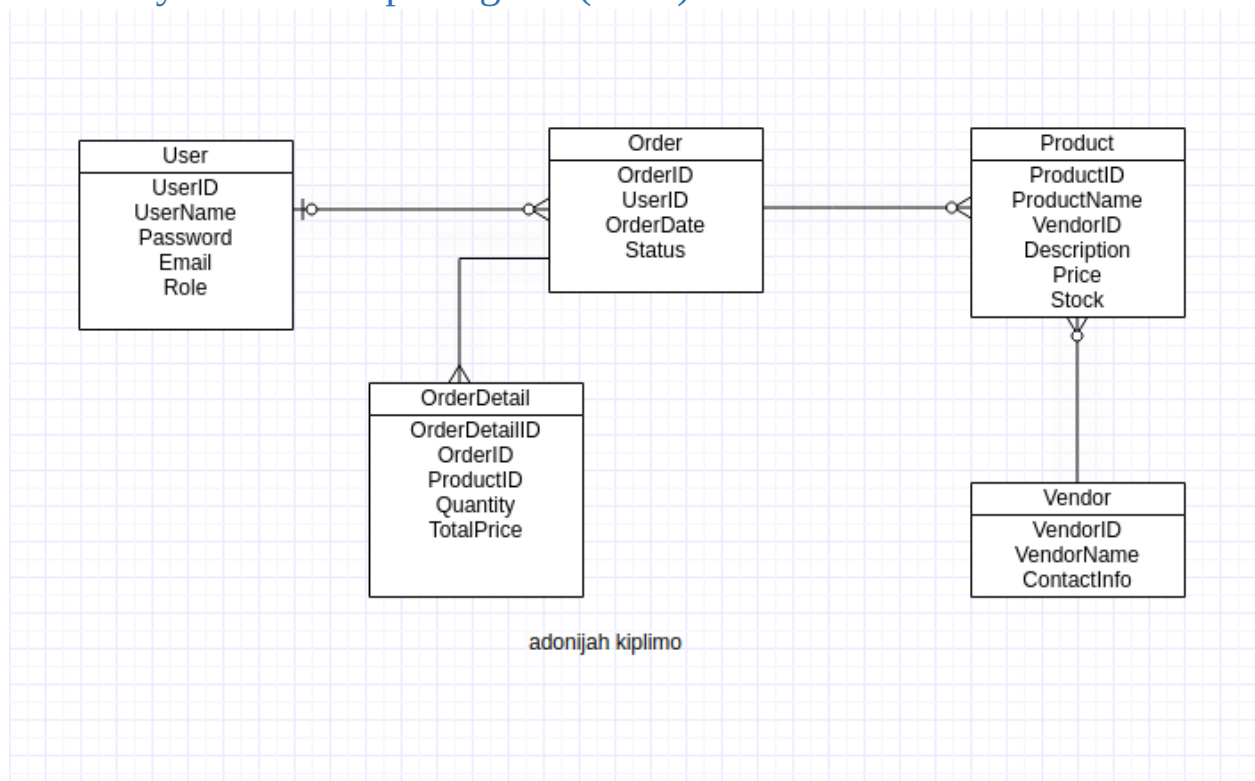
VendorID	ContactType	OrderDate
1	Email	vendor1@gmail.com
2	Email	vendor2@gmail.com

Table 7. Orders

OrderID	UserID	OrderDate
1	1	02/11/2023
2	2	02/11/2023

This normalization ensures that data is efficiently organized, and there's minimal data redundancy, reducing data anomalies and improving data integrity. This structure allows for easy scalability and maintenance of your multi-vendor e-commerce web application's database.

19.0 Entity-Relationship Diagram (ERD)



20.0 Entities

User

Attributes: UserID (Primary Key), Username, Password, Email, Role

Vendor

Attributes: VendorID (Primary Key), VendorName, ContactInfo

Product

Attributes: ProductID (Primary Key), ProductName, VendorID (Foreign Key), Description, Price, Stock

Order

Attributes: OrderID (Primary Key), UserID (Foreign Key), OrderDate, Status

OrderDetail

Attributes: OrderDetailID (Primary Key), OrderID (Foreign Key), ProductID (Foreign Key), Quantity, TotalPrice

In this simplified ERD:

- Users can be customers, vendors, or administrators (differentiated by the "Role" attribute).
- Vendors can add products, and customers can place orders.
- Products belong to vendors, and the "VendorID" attribute establishes this relationship.
- Orders are placed by users and have associated order details that list the products in the order.

This ERD represents the core entities and relationships in a multi-vendor e-commerce application. Depending on your specific requirements, you may expand the ERD with additional entities and relationships as needed. It's essential to align your ERD with the functionalities and features your application will provide to its users.

21.0 User Interface Design

Customer-Facing Interfaces

Website Layout: The web application will feature a user-friendly and responsive design, accessible through standard web browsers on desktop and mobile devices.

Product Listings: Customers can browse products with detailed descriptions, images, and pricing. They can filter and search for products.

Shopping Cart: Customers can add items to their shopping cart, view the cart's content, and proceed to checkout.

Checkout Process: An intuitive checkout process with form fields for shipping address, payment details, and order summary.

User Profiles: Registered customers can manage their profiles, view order history, and save payment/shipping information for faster checkout.

Product Reviews and Ratings: Customers can read and write product reviews and give ratings.

Wishlist: Users can add items to a wishlist for future purchase.

Authentication: Secure login and registration systems for customers.

22.0 Vendor Interfaces

Vendor Dashboards: Vendors can manage their product listings, view orders, and access sales reports.

Product Management: Vendors can add, edit, or remove product listings, including product descriptions, prices, and stock levels.

Order Management: Vendors can view and fulfill orders.

23.0 Common Interfaces

Admin Dashboard: Administrators can oversee the entire system, manage users, vendors, and products, and view analytics.

Search Functionality: A powerful search system for users and vendors to find products or orders.

Notification System: Users, vendors, and administrators receive notifications for order updates, product reviews, and system events.

Security Features: Secure payment processing and data protection measures.

Feedback and Support: Users can contact customer support or provide feedback.

24.0 User Experience (UX) Principles

User-Friendly Navigation: An intuitive menu and site structure.

Responsive Design: Ensuring a seamless experience on various devices.

Consistent Design: Uniform fonts, colors, and layout for a cohesive user experience.

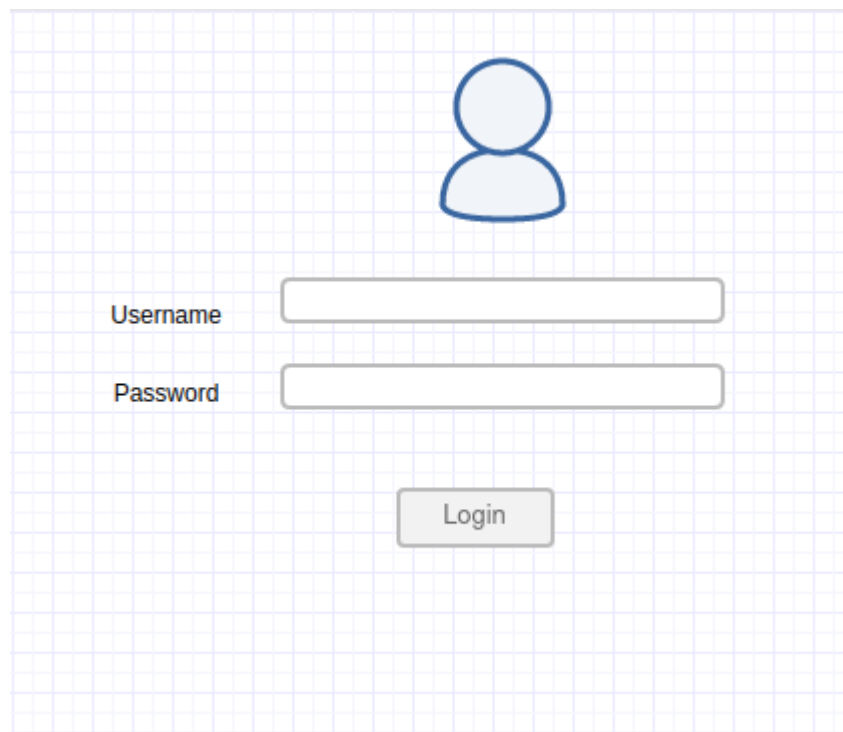
Call-to-Action (CTA) Buttons: Clear, action-oriented buttons for tasks like "Add to Cart" and "Checkout."

Product Images: High-quality images with zoom-in capabilities.

Feedback: Confirmations, error messages, and prompts for user actions.

Vendor Login Form

Login with your vendor credentials:



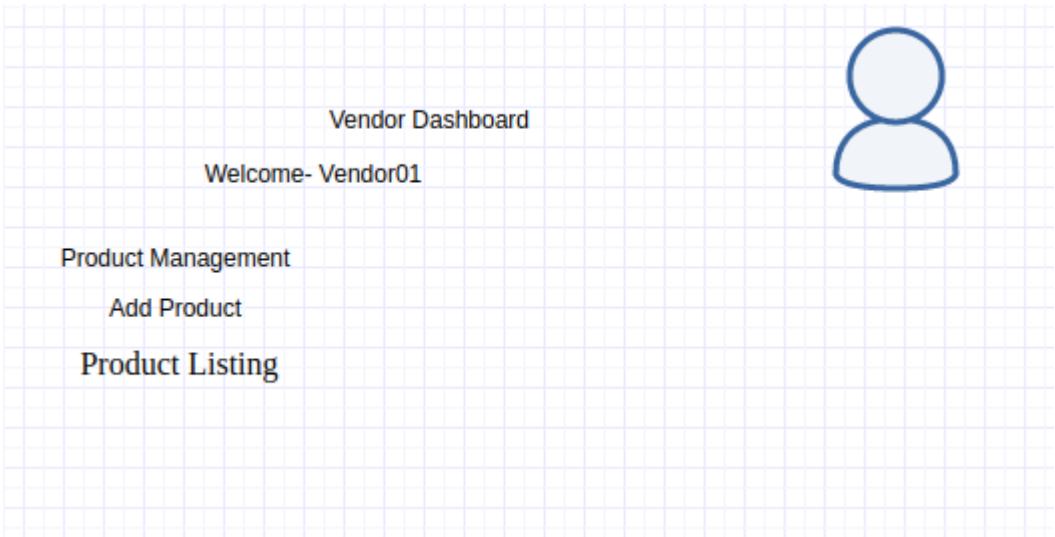
A UI mockup of a vendor login form on a light blue grid background. At the top center is a blue outline icon of a person. Below it are two input fields: the first is labeled "Username" and the second is labeled "Password". Both labels are in a dark blue font. Below the input fields is a gray button with the text "Login" in a dark blue font.

Vendor Dashboard

Welcome, [Vendor Name]!

Product Management

Add New Product



Product Listing

Name	Category	Price	Stock	Actions
[Product 1]	[Category 1]	[Price 1]	[Stock 1]	[Edit] [Delete]
[Product 2]	[Category 2]	[Price 2]	[Stock 2]	[Edit] [Delete]
[Product 3]	[Category 3]	[Price 3]	[Stock 3]	[Edit] [Delete]
[Add Product Button]				

Orders

Recent Orders

Order ID	Customer	Total Amount	Order Date	Status	Actions
[Order 1]	[Customer 1]	[Amount 1]	[Date 1]	[Status 1]	[Details]
[Order 2]	[Customer 2]	[Amount 2]	[Date 2]	[Status 2]	[Details]
[View Details Button]					

Settings

Account Settings

Change Password

Old Password: [Old Password Field]

New Password: [New Password Field]

Confirm Password: [Confirm Password Field]

[Change Password Button]

Logout

[Logout Button]

Components Available

- Add Product: Use this button to add a new product to your store.
- Manage Products: This section allows you to edit, update, or delete your product listings.
- View Orders: Access and process customer orders here.
- Settings: Configure your vendor account settings, including password changes and other preferences.

Software Context

The application is designed with a modular structure, consisting of multiple modules that are interconnected to provide a seamless user experience. These modules include product management, order processing, and settings. They work together to ensure the smooth operation of your multi-vendor ecommerce web application.

Expected Software Response

The web application is anticipated to function smoothly, with minimal errors. Prior to deployment, extensive testing will be conducted by our development team to identify and rectify any potential issues. This rigorous testing process ensures that the software meets high-quality standards before it is delivered to our clients. Our commitment is to provide a reliable and user-friendly platform for both vendors and customers.

Packaging and Installation Guidelines

When it comes to packaging and installation, our goal is to make the process as straightforward as possible for our vendors. We will provide a comprehensive installation package, which includes all the necessary files and documentation required to set up and run the multi-vendor ecommerce web application. This package will typically be in the form of a downloadable archive.

The installation instructions will be provided in an easy-to-follow, step-by-step format to guide vendors through the setup process. We aim to ensure that the application can be effortlessly deployed on various web hosting environments. Additionally, we will include user documentation to assist vendors in making the most of the platform's features.

Should vendors encounter any issues during installation or have questions about using the application, our support team will be readily available to provide assistance and resolve any concerns. Our goal is to facilitate a smooth installation process, so vendors can quickly start utilizing our multi-vendor ecommerce platform to grow their businesses.

25.0 Conclusion

The requirements specification forms a crucial foundation for our design process. This step ensures that the multi-vendor ecommerce web application is closely aligned with the needs and expectations of our users, including vendors and customers alike.

The design phase, as presented in this document, is the next critical step in turning these requirements into a functional and efficient platform. It's our belief that the proposed design will effectively meet the stipulated user requirements and offer an intuitive, seamless, and feature-rich experience.

To further affirm the system's ability to fulfill the outlined user requirements, we will proceed with the development of a comprehensive test plan. This test plan will serve as a systematic tool for validating the functionality, performance, and reliability of the multi-vendor ecommerce web application. We remain committed to delivering a robust, user-friendly, and dependable solution to our clients.

References.

Governors State University

<https://opus.govst.edu/cgi/viewcontent.cgi?article=1079&context=capstones>