Warehouse Stock Management System

1.Project Overview:

1.1 Introduction:

Businesses face inventory and warehouse management challenges as they seek effective solutions for product monitoring, order processing, and stock level maintenance.To solve these issues, the Warehouse Stock Management System (WSMS) project includes features that enhance the user experience, expedite procedures, and enable informed decision-making.Warehouse Stock Management System is a software tool that helps firms manage warehouse operations more efficiently.It provides real-time visibility into commodity and inventory levels.

1.2 Problem Statement:

Manual procedures and disconnected technologies present substantial issues, including inaccurate inventory tracking, delayed order fulfillment, and limited visibility into overall business performance. The executive team recognizes the critical need for a comprehensive warehouse management solution that can streamline operations, increase productivity, and help the firm grow.We have identified the project's key stakeholders and their specific interests**:**

- Warehouse managers: Managers must have real-time visibility into inventory quantities, storage locations, and general warehouse activities. This will enable them to make educated decisions to optimize operations and proactively resolve any issues that arise.
- Warehouse workers: Employees require an intuitive system that can assist them through the numerous workflows, from receiving and putting away to order selection, packaging, and shipping. This will enable them to fulfill orders more correctly and effectively while keeping precise inventory records.
- Operations executives: Operations executives demand strong reporting and business intelligence capabilities to monitor key performance measures, identify areas for improvement, and make data-driven choices to improve overall company operations.

- Logistics Providers: Our client's logistics partners require seamless interaction with the warehouse management system in order to obtain up-to-date order and shipping data. This will allow them to make more timely and precise deliveries, enhancing the client experience.
- Customers: Customers demand accurate order fulfillment, quick shipment, and transparent order tracking. Meeting these needs will be critical to sustaining a positive brand reputation and consumer loyalty.

1.3 Problem Solution:

To overcome these problems and meet stakeholder expectations, we propose creating a centralized, automated, and integrated warehouse stock management system.

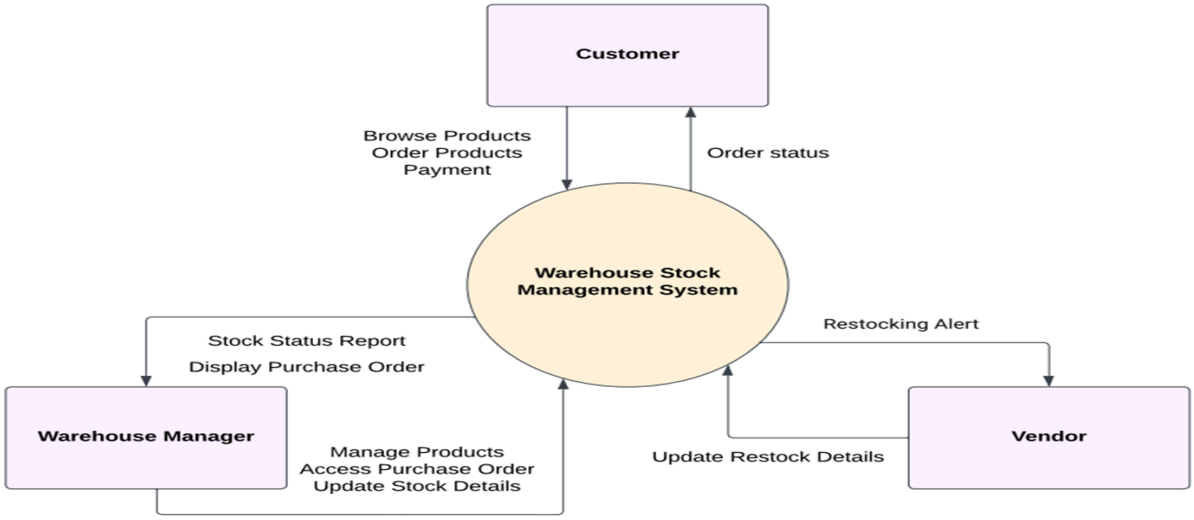
- Inventory Management: The system will give real-time tracking of inventory levels, locations, and statuses, allowing the firm to maintain appropriate stock levels and reduce stock-outs.
- Warehouse Operations: We will increase overall operational efficiency by streamlining essential warehouse activities such as receiving, put-away, order picking, packing, and shipping.
- Integration: The solution will work smoothly with the client's existing enterprise resource planning (ERP), order management, and logistics systems, enabling end-to-end visibility and coordination throughout the supply chain.
- Reporting and Analytics: Comprehensive reporting and business intelligence capabilities will be built to give operations leaders with data-driven insights and aid in strategic decision-making.
- User Interface: Simple web-based and mobile interfaces will be created enabling warehouse employees, managers, and customers to access information and complete their jobs.

By deploying a stock management system, our client will be able to handle their present operational difficulties while also positioning themselves for future development and success in the highly competitive online retail sector.

1.4 General model

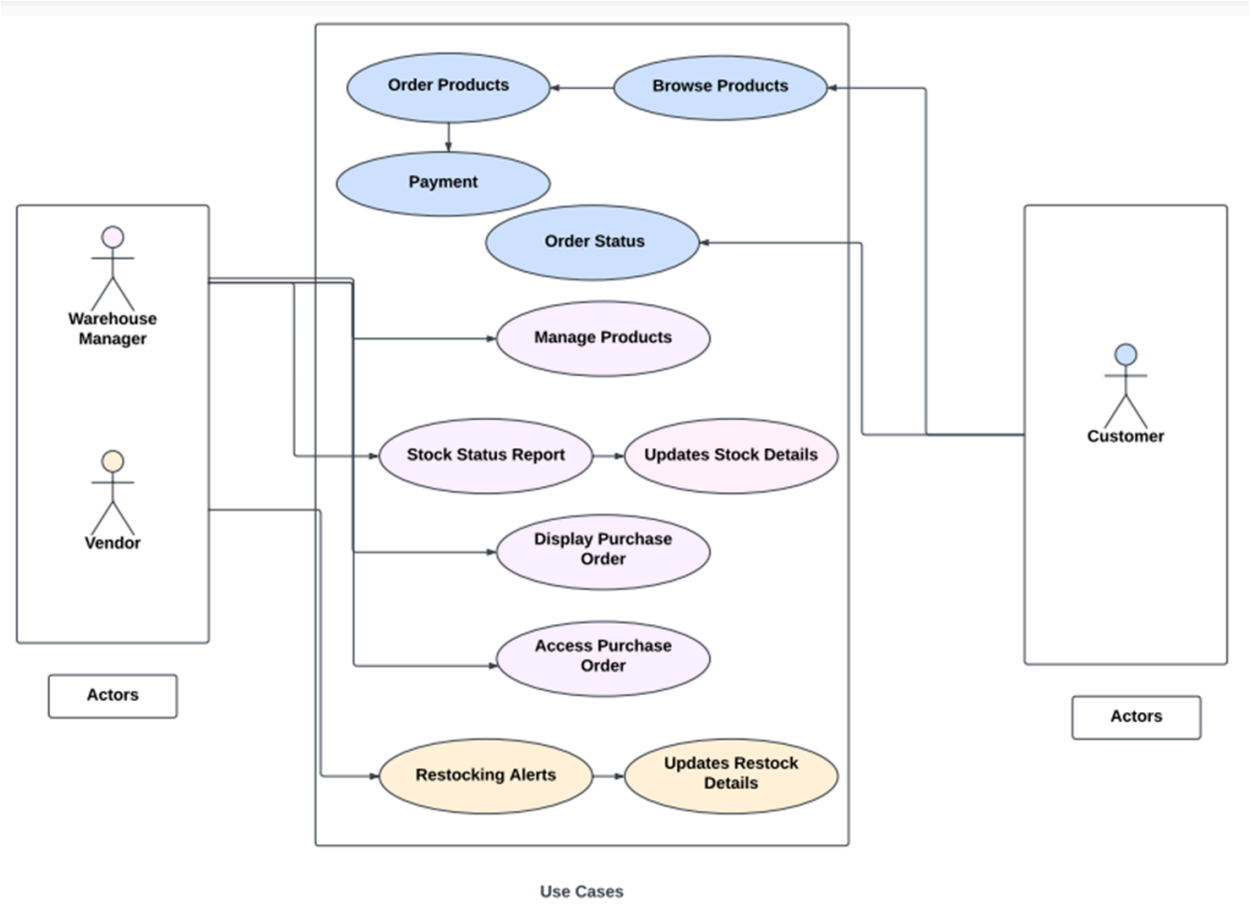
1.4.1 Context diagram

The context diagram gives a high-level perspective of the Warehouse Stock Management System and its connections with major external entities. It depicts the customer engaging with the system to explore, order, and see order status. The Warehouse Manager used the system to handle merchandise, retrieve purchase orders, and update stock information. The Vendor communicates with the system to get restocking notifications and update stock information.This diagram describes the system's boundaries and the flow of information between the system and its stakeholders, allowing for a more examination.



1.4.2 Use case diagram

The use case diagram depicts the Warehouse Stock Management System's primary features and interactions. It illustrates the most common use cases, including Order Products, Browse Products, Manage Products, Display Purchase Order, Access Purchase Order, Stock Status Report, Updates Stock Details, and Restocking Alerts.. These use examples describe the system's main features as well as the interactions between the Warehouse Manager, Vendor, and Customer actors. This graphic gives a brief overview of the system's scope and functioning.



1.4.3 User Stories

Here are some user stories identified:

• Create interfaces for Customer, Manager, and Vendor.

• Create product browsing and ordering functionality for customer.

• Develop product management features.

• Implement order tracking for all customers.

• Create stock status reports for warehouse managers.

• Allow warehouse managers to update stock details.

• Enable restocking details management.

• Develop purchase order management for vendors.

• Implement user-friendly purchase order display.

• Set up automated alerts for low-stock products.

• Optimize for mobile use and conduct usability testing.

•Payment System for customers

2.Architectural Overview:

Model View Controller Architecture:

1. Warehouse Stock Management View:

This view enables warehouse personnel to interact with and manage inventory. This includes:

Display of current stock:

Lists the available products, including their name, number in stock, and location inside the warehouse.

Search and Filtering Functionality:

Customers can search for specific products or filter inventory using parameters like category or location.

Add or Remove Products:

Allows to add new products to the inventory or remove current ones.

Update the Product Information:

Allows to update product characteristics such as number, location, and other pertinent attributes.

Referring to GP_Data_Model:

The View accepts data from the Controller and displays it.

The Model (GP_Data_Model) provides data on goods, quantities, and other inventory details.

2. Order Processing View:

This view handles customer orders and fulfillment. This includes:

Order details:

Displays extensive information about a specific order, including products, quantities, and customer information.

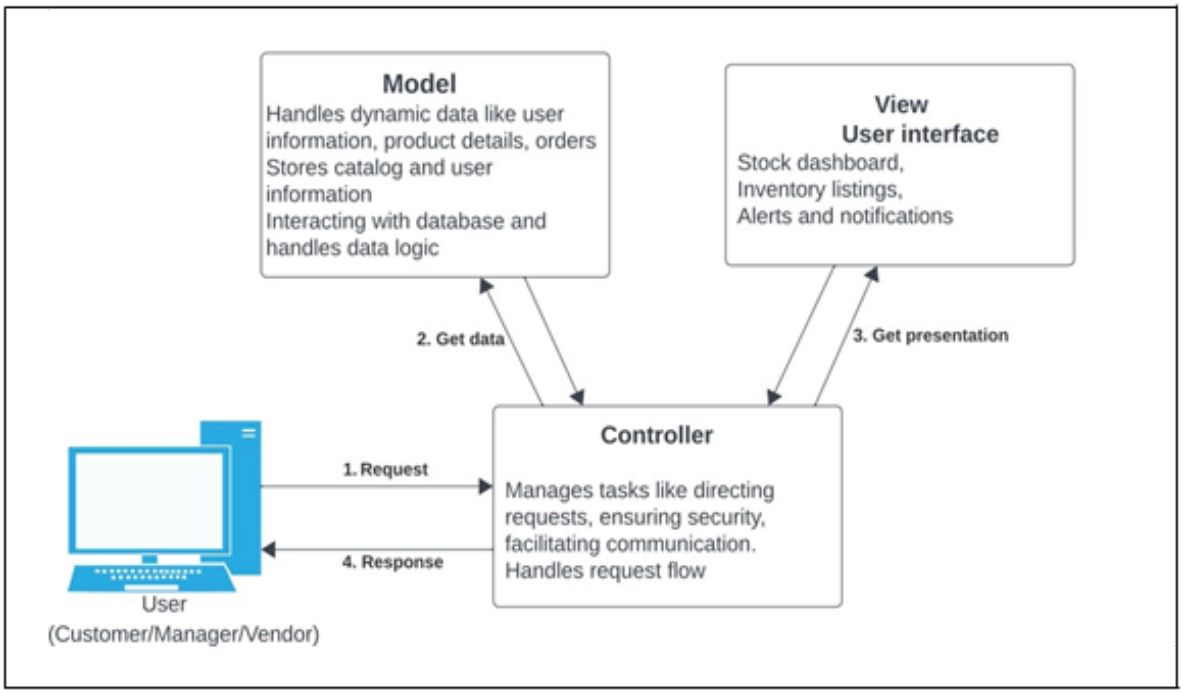
Order Processing Actions:

Allows warehouse staff to update an order's status (processed, shipped) and manage inventory accordingly.

Referring to GP_Data_Model:

The View communicates with the Controller to handle customer actions related to order fulfillment.

The Model (GP_Data_Model) returns data on orders, products, and customer information.

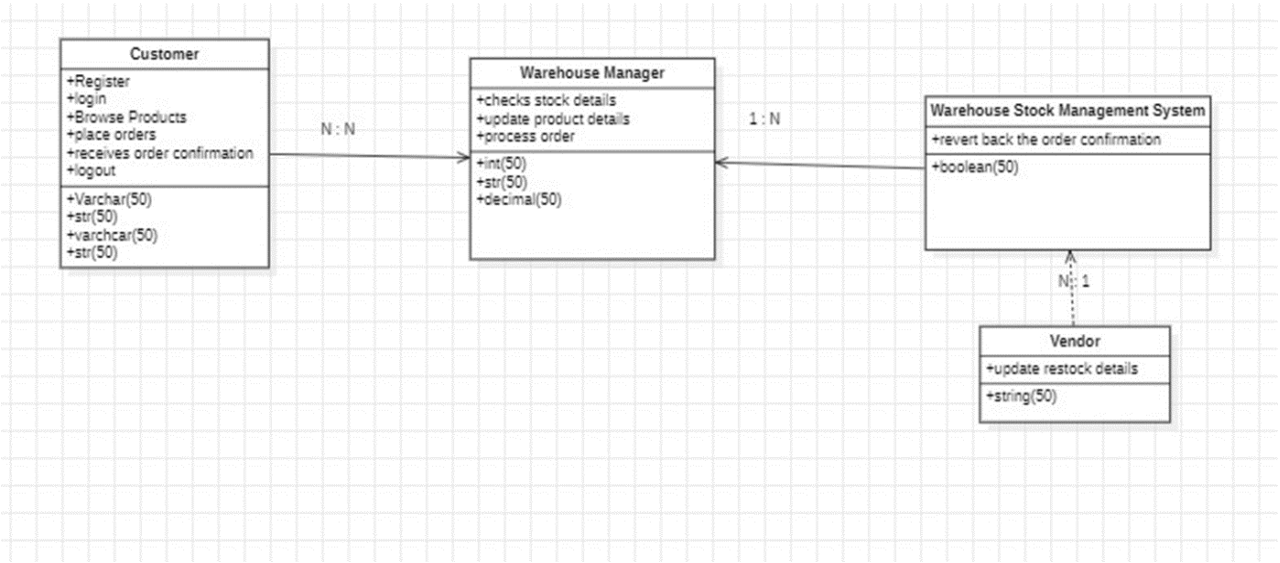


2.1.Subsystem Architecture:

UML Diagrams

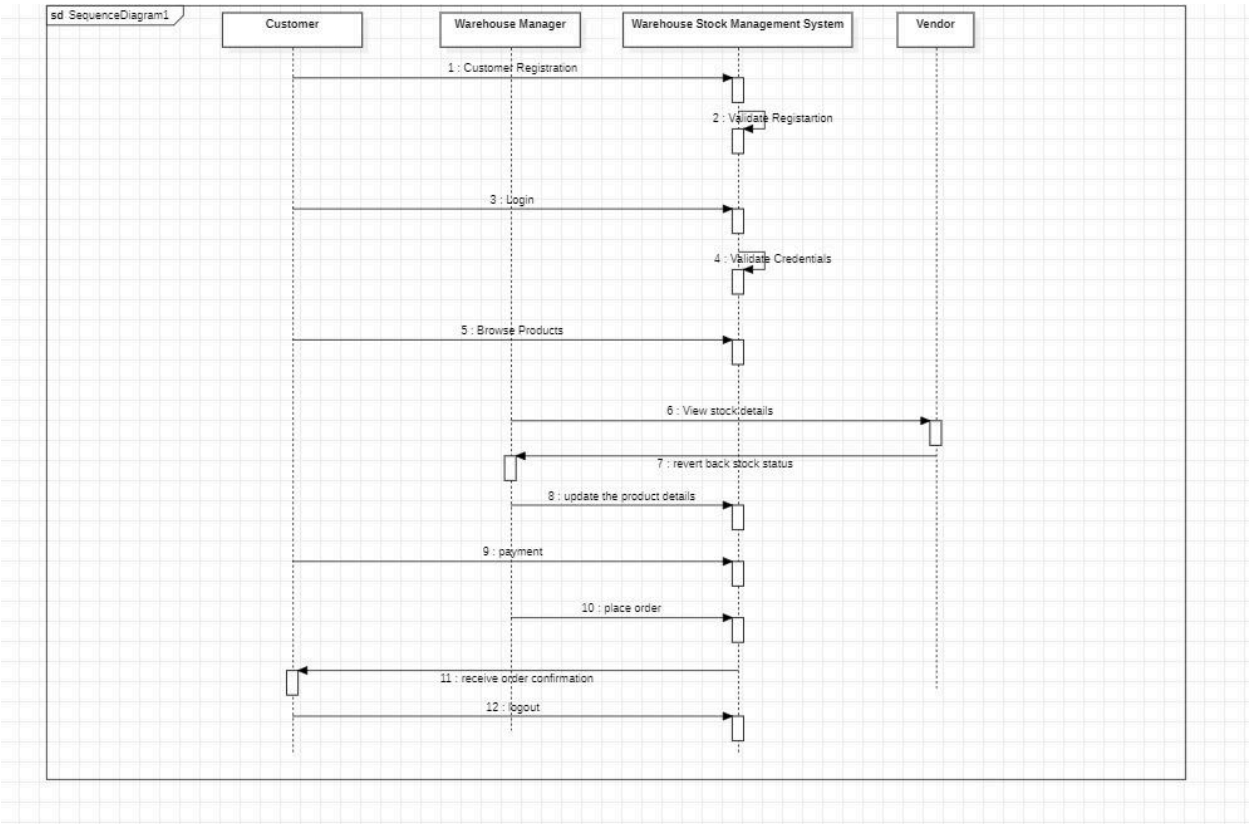
Class Diagram:

The class diagram demonstrates the linkages and interactions that exist between the Warehouse Stock Management System's core components. The important classes are Customer, Warehouse Stock Management System, and Warehouse Manager, each having its own set of properties and methods. This diagram offers a high-level overview of the system's structure as well as data flow among the many entities involved, which is critical for understanding the overall system design and performance.



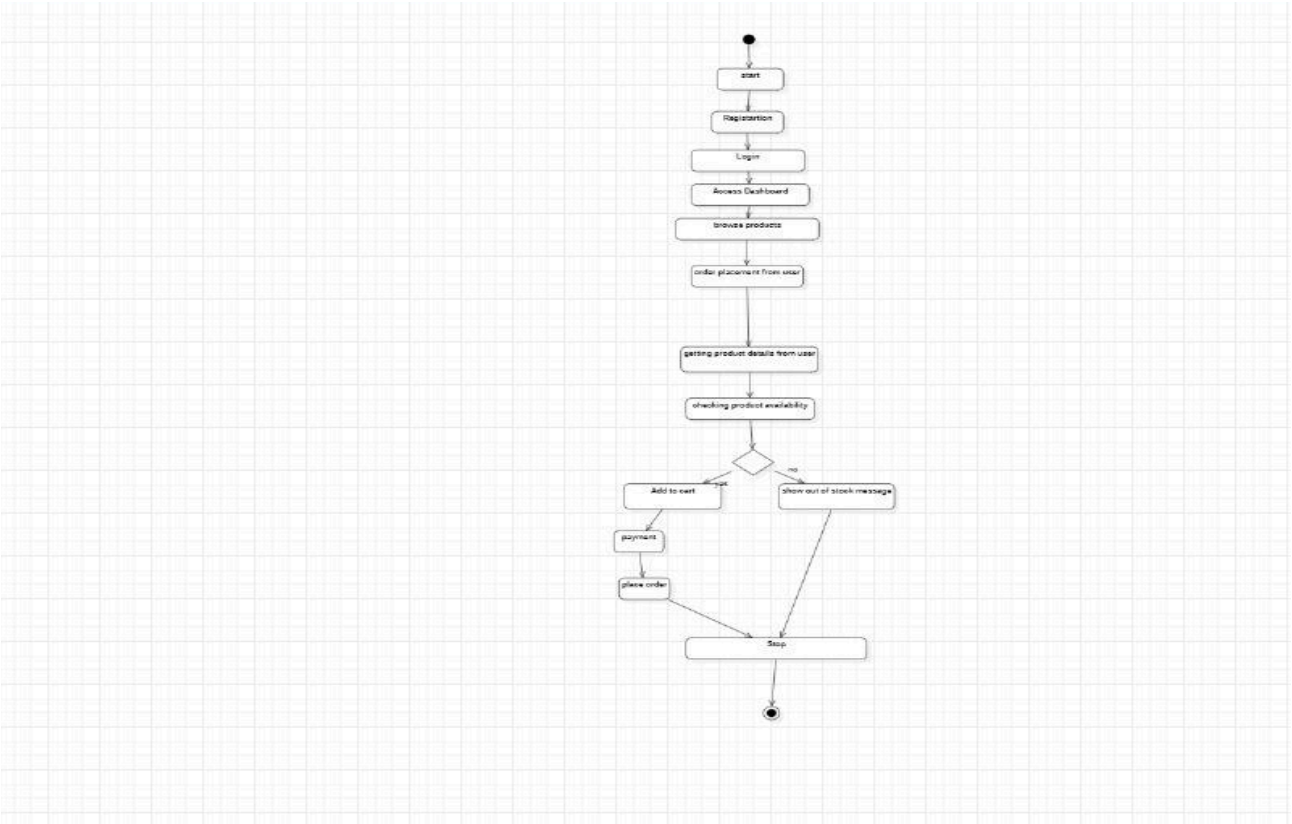
Sequence Diagram:

The Sequence diagram describes the flow of working in a warehouse stock management system. The roles begins with the customer for registration and login, then moves on to the warehouse manager ,warehouse stock management system,vendor for browsing products, adding them to the cart, checking product availability, and completing payment. The procedure culminates with the placing of the order. The figure shows the consecutive flow of the process involved in completing an online purchase of the product from the warehouse stocks.



Activity Diagram:

The activity diagram describes the process of working in a warehouse stock management system. It begins with user registration and login, then moves on to browse products, add them to the cart, check product availability, and complete payment. The procedure culminates with the placing of the order. The figure shows the consecutive stages involved in completing an online purchase of the product.



Package diagram

Warehouse System: The umbrella package that includes all warehouse management features.

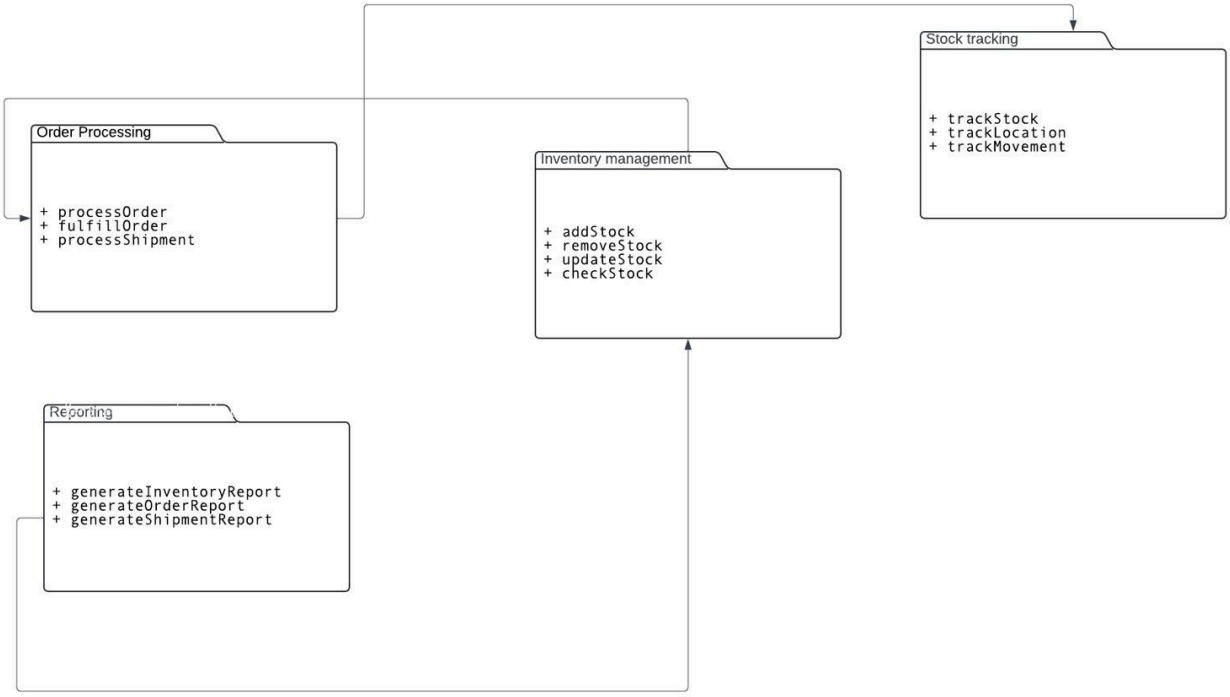
Inventory Management: This package handles the inventory in the warehouse. It manages stock levels, adds new goods, updates amounts, and removes things from inventory.

Order Management: This package manages orders received by the warehouse. It offers features for processing orders, fulfilling orders, tracking order status, and handling returns.

Reporting Module: Provides reporting capabilities for the warehouse stock management system. It enables customers to produce multiple statistics about inventory levels, order fulfillment rates, shipment statuses, and other essential key metrics

Each package encompasses related functionality, increasing modularity and maintainability throughout the system. The interactions and dependencies between these

programs are included into the overall functioning of the warehouse stock management system.



2.2.Dependency Architecture

Database ER Diagram

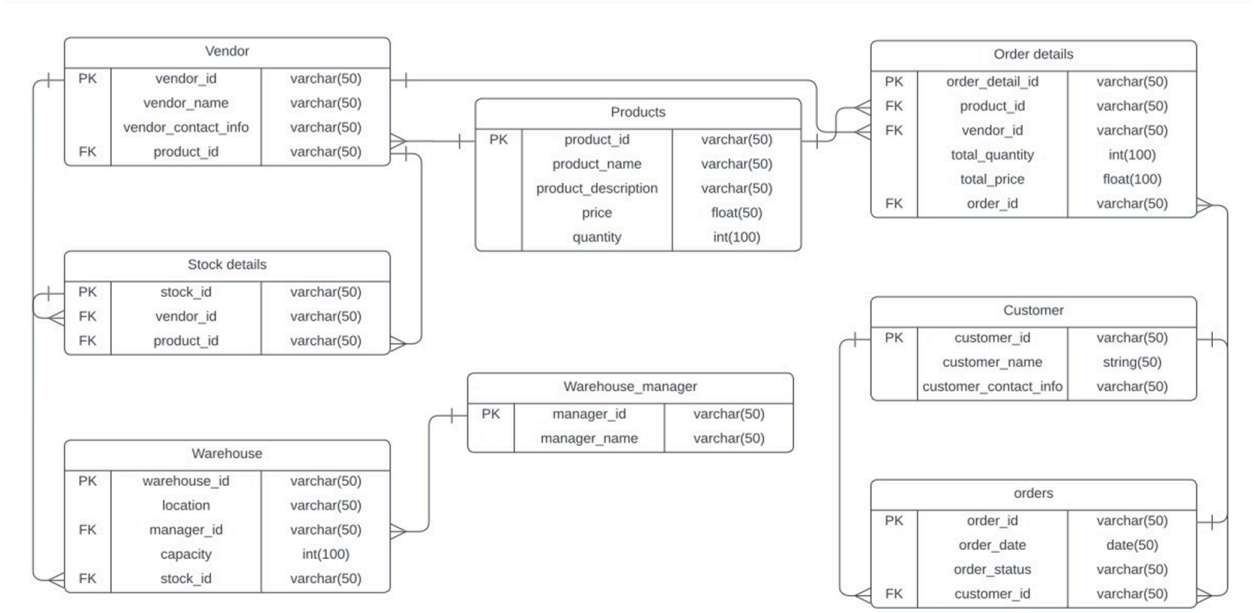
A dependency Architecture for a warehouse stock management system has four layers: application-specific, application-generic, middleware, and system-software.

Application-particular Layer: This layer includes software components specific to the warehouse stock management system. It includes capabilities like as inventory management, order processing, delivery tracking, and reporting.

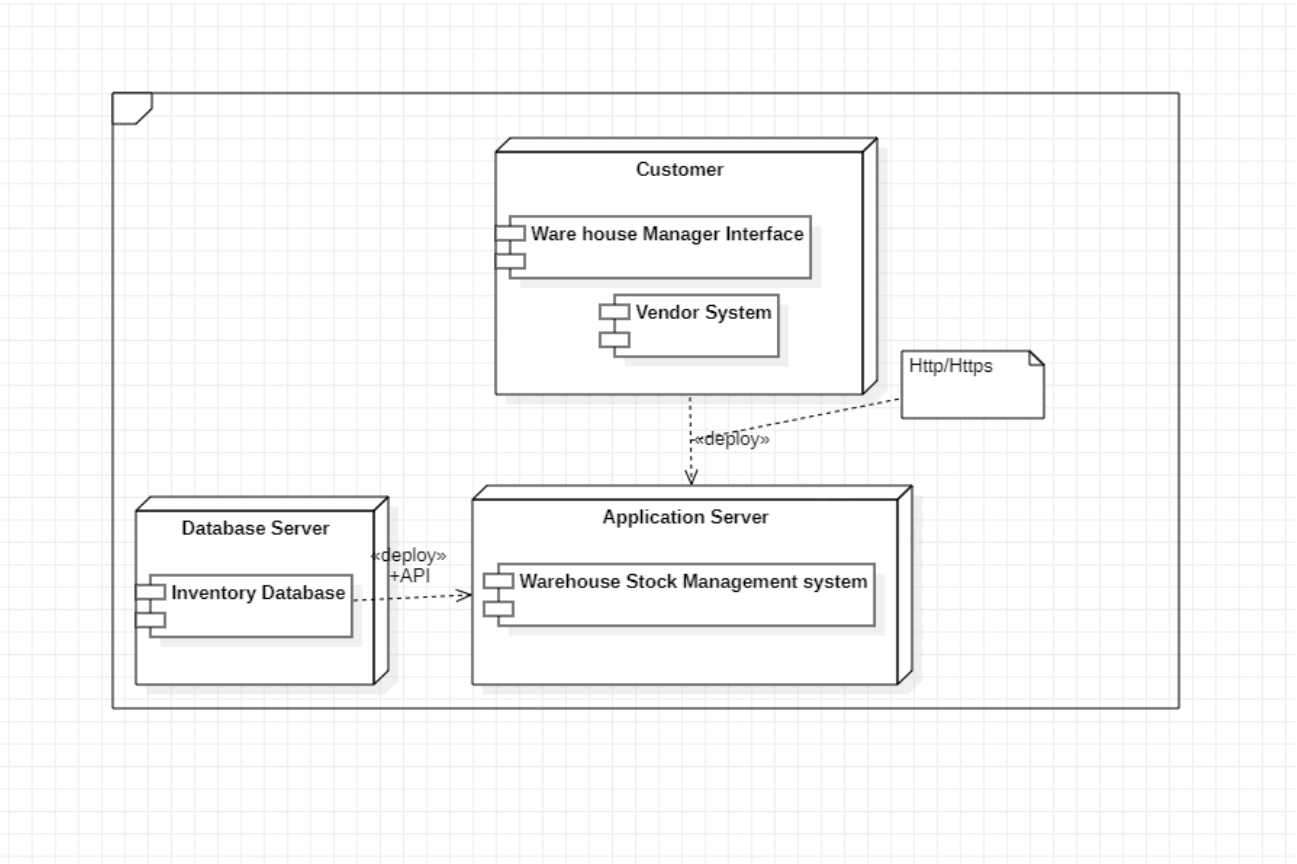
Application-Generic Layer: This layer contains reusable components and libraries that are not specific to the warehouse management system but are used to build and support its functionality. It may include modules that handle user authentication, database access, logging, and error handling.

Middleware Layer: This layer connects the application-specific and system-software layers. Middleware components like as message brokers, API gateways, and integration frameworks facilitate communication between different parts of the system.

System-Software Layer: This layer includes the underlying software and infrastructure that allows the warehouse stock management system to operate. It contains the operating system, database management system, web server, and other system-level software components required by the program to function.



2.3.Deployment Architecture:



The main components of this deployment diagram are:

- Web/Mobile Client: This is the client-side interface that allows the customer to interact with the system.
- The main Warehouse Stock Management System and the Warehouse Manager interface are hosted on the Application Server.
- Database Server: This server houses the Inventory Database, which maintains all product, order, and inventory information.
- Vendor System: This is the external system used by vendors to communicate with the Warehouse Stock Management System.

Communication between these components is facilitated by the following protocols:

- HTTP/HTTPS enables secure web-based communication between web/mobile clients and application servers.
- JDBC (Java Database Connectivity) links the application server with the database server.

- API integration between the application server and the vendor system is used to communicate data and make modifications.
- This deployment diagram shows how the different software components and hardware infrastructure are organized to support the Warehouse Stock Management System and its interactions with stakeholders such as customers, warehouse managers, and vendors.

2.4 Persistent Data storage:

The "Persistent Data Storage" section provides overview of the data to be stored in the warehouse stock management system, along with the rationale for storing each type of data. Additionally, it explains how the data will be stored and represented within the system.

1. Data Types:
- Inventory Data: Information on stocked items including SKU, description, quantity, location, and status.
- Order Data: Details of customer orders such as order ID, customer information, items ordered, quantity, and delivery status.
- Transactional Data: Records of warehouse transactions encompassing receiving, putaway, picking, packing, and shipping activities.
- Configuration Data: Settings and configurations pertinent to warehouse operations like storage locations, equipment setup, and system parameters.

2. Reasons for Data Storage:
- Operational Efficiency: Storing diverse datasets facilitates smoother management of warehouse operations, aiding tasks like inventory tracking and order processing.
- Decision Making: Access to historical and real-time data enables better decision-making regarding inventory management and resource allocation.
- Regulatory Compliance: Accurate records of transactions and inventory movements ensure compliance with regulatory requirements and audit standards.
- System Integrity: Persistent storage of configuration and event data ensures system reliability and simplifies troubleshooting and maintenance.

3. Storage Mechanism:

- Database Usage: The system employs a relational database management system (RDBMS) for persistent storage due to advantages like structured data organization, enforced data integrity, efficient querying, and scalability.
- Data Representation: The section delineates the schema design and representation of data entities within the database, including tables for inventory, orders, transactions, configurations, and event logs. It may also address relationships between entities, foreign key constraints, and index optimization for query performance.

In summary, the "Persistent Data Storage" section meticulously delineates the data types, reasons for storage, and the chosen storage mechanism, ensuring clarity and thoroughness in managing the warehouse stock management system's persistent data.

2.5 Global Control Flow:

The section on "Global Control Flow" for the warehouse stock management system offers a thorough explanation of the control flow model employed, justifies its selection, and outlines a suitable strategy for synchronizing control among concurrent activities.

1.Control Flow Model: The system utilizes a hybrid model, combining sequential processing for core warehouse operations with event-driven control for managing user interactions and external events. This choice is grounded in the structured nature of warehouse processes alongside the need for real-time responsiveness to various events.

2. Concurrency and Synchronization: Given the simultaneous occurrence of multiple warehouse operations, the system employs locks, semaphores, and monitors to coordinate access to shared resources, ensuring data consistency without deadlocks and maintaining optimal throughput.

3. System Initialization and Shutdown: The system follows a specific sequence during initialization, establishing connections and configurations. Shutdown procedures ensure the completion of ongoing operations, proper data persistence, and resource release.

4. Event Handling and Coordination: An event-driven approach is adopted to manage asynchronous events effectively, prioritizing and dispatching them to appropriate handlers for timely responses to dynamic warehouse conditions.

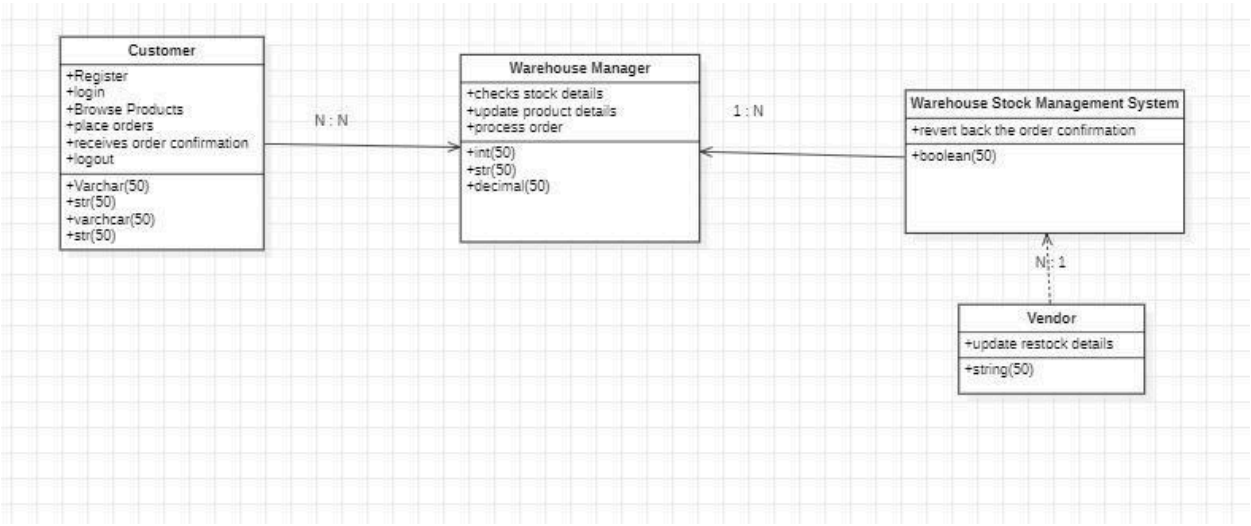
5. Error Handling and Recovery: Robust error handling mechanisms, including error detection, logging, and recovery procedures, minimize disruptions to operations and maintain data integrity in case of exceptions.

6. Scalability and Performance Considerations: To scale efficiently and maintain high performance, the system leverages asynchronous processing, task queuing, and load balancing techniques to distribute workloads across multiple instances or nodes.

In summary, the "Global Control Flow" section provides a comprehensive and logical explanation of the control flow model, its rationale, and its effective management of concurrency and other critical control flow aspects within the warehouse stock management system.

3.System Design

UML class diagram



3.1 Static view semantics:

Customer Class:

Semantics: The Customer class accurately captures the data and behaviors relevant to a system user, including creating an account, perusing the product catalog, and

submitting orders. Semantically appropriate, it should also protect user data privacy.

Completeness: The course covers the procedures for creating an account, placing an order, and registering. If attributes for payment information or a transaction history are required for the system's scope, it might not be complete.

Warehouse Manager:

Semantics: Tasks like processing orders and verifying stock are included in the Warehouse Manager class's definition. Given that it has a direct bearing on warehouse management duties, this is semantically true.

Completeness: The course covers order processing and updating product data, which is essential. To be comprehensive, it might encompass procedures for managing refunds or trades, if relevant.

Warehouse Stock Management System:

Semantics: This class acts as a central ordering confirmation tracking mechanism. It is expected to handle a wider variety of stock and order management-related duties semantically.

Completeness: The system's scope, which includes controlling inventory levels and offering reporting capabilities, may not be fully represented by the static display.

Vendor:

Semantics: The Vendor class's ability to update restock information fulfills the system's semantic need for a supplier.

Completeness: The class may be incomplete if additional functionalities, such as managing orders from the warehouse or handling invoices, are within the scope of the system.

Responsibilities of each class and justification:

- **Customer Class:**

  Responsibilities:

1. Managing personal details (e.g., username, password, address).
2. Browsing products available in the inventory.
3. Placing orders and keeping track of the order status.
4. Receiving order confirmations.

   Justification:

   The Customer class is decomposed to focus solely on the customer's interactions with the system. This design limits the class to user-specific actions, promoting the principle of single responsibility and ensuring that the class is not overloaded with system-level concerns.

- **Warehouse Manager:**

  Responsibilities:

  1. Keeping track of inventory details, including product details and stock levels.
  2. processing and carrying out consumer orders.
  3. Possibly managing exchanges and refunds.

  Justification:

  Serving as an intermediary between the inventory and customer orders, the Warehouse Manager class is the system's controller. It is broken down to handle internal warehouse chores, making a distinct division between operations that interact with customers and those that do not.

- **Warehouse Stock Management System:**

  Responsibilities:

  1. Taking up the role of the main contact for order processing and inventory control.
  2. keeping track of inventory levels and offering stock update interfaces.
  3. Managing order and inventory-related alerts and notifications.

  Justification: The stock management system's essential functions are contained in this class. Because of its modular design, distinct subsystems

(such as order processing and inventory control) can be created and maintained separately.

- **Vendor:**

  Responsibilities:

  1. provide the warehouse with information regarding goods resupply.
  2. overseeing deliveries and shipments to the warehouse.

  Justification:

  The Vendor class is meant to capture the communication between the warehouse and its providers. Its primary duty is supplying chain management, which is unrelated to internal warehouse management and customer relations.

Relationships between classes:

- **Customer to Warehouse Manager (N:N Relationship):**
  This relationship, while not typical in many warehouse systems, might indicate a business model where personalized customer service is paramount. In such a model, customers may directly interact with multiple warehouse managers to address specific needs or concerns. This can be seen as a customer-centric approach that prioritizes direct communication and tailored services, enhancing the customer experience.

- **Warehouse Manager to Warehouse Stock Management System (1:N Relationship):**

  A single warehouse manager may interact with several subsystems or functionalities of the Warehouse Stock Management System. This relationship is logical, as it reflects the diverse roles a manager plays, from checking stock details to updating product details and processing orders. It showcases a system that efficiently distributes various tasks to the warehouse manager, ensuring a streamlined workflow within the warehouse operations.

- **Warehouse Stock Management System to Vendor (N:1 Relationship):**

  The system is designed to aggregate restock details from multiple vendors into a centralized database. This N:1 relationship is both logical and efficient, as it allows for a consolidated view of inventory updates, enabling the system to maintain accurate and up-to-date stock levels. It's a reasonable design choice that facilitates better inventory management and vendor coordination.

3.2 Static View Quality:

Design quality:

Customer

The Customer entity appears to have a high degree of cohesiveness because its tasks are well-defined and targeted. It includes every feature associated with a customer's engagement with the system, including order processing, login, and registration procedures. The Customer class exhibits a high degree of coherence, suggesting that the system is well built to comprehend and execute customer-related tasks.

Warehouse Manager

The Warehouse Manager has a clearly defined set of duties and is efficiently built to manage order processing and stock-related information. A warehouse manager may be able to effectively manage several tasks or interactions with the stock system since the class is made to have a one-to-many relationship with the warehouse stock management system. The Warehouse Manager's job responsibilities are clearly defined, which indicates a good internal cohesion strategy and efficient stock management.

Warehouse Stock Management System

The Warehouse Stock Management System is unique in that it functions as a centralized hub that links several system components, including managing order confirmations and liaising with vendors and warehouse managers. Its layout seems to effectively control information flow, guaranteeing that orders are processed

accurately and that stock levels are current. Order confirmations can be reversed by the system, which suggests a strong mechanism for transactional integrity and dependability.

Vendor

The Vendor entity's streamlined process for updating replenishment information demonstrates its narrow and focused focus. This focus on a single task exemplifies the high cohesion concept, enabling a vendor module that is simple to comprehend and manage.

Coupling and Relationships

The N:N, 1:N, and N:1 relationship between entities point to a system that is built to manageably couple disparate elements while managing complicated interactions. For instance, the N:N relationship between the warehouse manager and the customer suggests that the system is adaptable enough to support several clients corresponding with several warehouse managers without producing overly strict dependencies.

Highly Cohesive Classes/Modules

Customer:

There is a lot of cohesiveness in this class. Its primary function is to facilitate customer actions such as ordering, browsing, logging in, and registering. The techniques are precise and solely applicable to the acts of the customer.

Warehouse Manager:

Another example of a well-designed module is the Warehouse Manager. As the single point of control for all inventory-related operations, it oversees carrying out tasks directly associated with warehouse management, such as updating product data and verifying stock levels.

Vendor:

One of the main characteristics of this very cohesive module is that it has a single duty, updating replenishment details. By concentrating on vendor interactions with

the stock system, it reduces the possibility that the class would grow excessively large with irrelevant features.

Loosely Coupled Design

N:N The warehouse manager's relationship with the customer:

This many-to-many interaction suggests a design that facilitates communication between numerous warehouse managers and customers. This may indicate that the system is meant to be flexible, able to expand in response to an increase in users without requiring modifications to the underlying classes.

1:N Warehouse Manager and Warehouse Stock Management System Relationship:

The stock management system handles direct inventory management duties instead of piling the warehouse manager with too many of them. This division of responsibilities suggests a shift toward loose coupling, which makes inventory management easier to improve and modify without requiring major adjustments to the warehouse manager's class.

N:1 Vendor-Warehouse Stock Management System Relationship:

This suggests that even while numerous providers can update the stock, they do it via a single mechanism. By establishing a single point of contact for stock updates, this relationship can preserve the accuracy and consistency of stock data without placing undue reliance on any one vendor or the stock records themselves.

3.3 Static view syntax:

Syntactical Elements:

Class Names:

In accordance with the rule for class representation, each class name "Customer," "Warehouse Manager," "Vendor," and "Warehouse Stock Management System" is capitalized and accurately describes the entities within the system.

Attributes:

Each class's attributes are listed alongside the relevant data types, using standard syntax. This shows that each class's properties have been declared correctly.

Methods:

In accordance with UML syntax guidelines, methods are listed with leading plus signs denoting their visibility (public in this case). They use the () convention, using the proper data types for parameters when they are present.

Relationships:

The diagram illustrates relationships between the classes with standard UML notation:

- N:N Relationship: Represented with a line and 'N:N' multiplicity, showing a many-to-many association between 'Customer' and 'Warehouse Manager'.
- 1:N Relationship: Indicated by a line connecting 'Warehouse Manager' to 'Warehouse Stock Management System' with '1:N' multiplicity, showing a one-to-many association.
- N:1 Relationship: A line with 'N:1' multiplicity between 'Vendor' and 'Warehouse Stock Management System', illustrating a many-to-one association.

Multiplicity:

The number of instances of one class that can be connected to a single instance of another is indicated by the multiplicity notation, which is appropriately positioned at the endpoints of association lines.

Directionality:

In UML, the direction of a link or dependency is usually indicated by arrowheads. According to UML standards, the arrows in this diagram indicate a dependency or

navigability direction from "Warehouse Manager" to "Warehouse Stock Management System" and from "Vendor" to "Warehouse Stock Management System."

Data types and visibility:

Customer:

Despite their unusual names, attributes like strVarchar(50) and int(50) do indicate types and imply suggest visibility (public by default with the plus sign).

Warehouse Manager:

Listed characteristics with types like int(50) and strVarchar(50) give a clear indication of the data types and visibility, just like in the Customer class.

Warehouse Stock Management System:

It's a little strange that an attribute of type boolean(50) is listed booleans normally don't have a length specifier, but the visibility is mentioned and the purpose is apparent.

Vendor:

The class displays a property of type string(50), which complies with UML guidelines for data type and visible representation.

At least one attribute for each class is listed along with its type. While the naming convention is different, this is a good example of how classes' types are consistent and how visibility is indicated, indicating that the diagram is meant to be understood by system users. The goal of maintaining the static view uncluttered and concentrating on the most important properties necessary for comprehending the system at this level of abstraction may also be reflected in the diagram's attribute representation simplicity.

Every method in the class diagram has visibility, as shown by the plus symbol (+), which stands for public methods. The methods have unambiguous names that facilitate an instant comprehension of the actions linked to the entities. Nevertheless, the graphic does not specifically state what parameters or return types are. The return type and parameters are not shown in this figure; in UML notation, they are often indicated after the method's parentheses.

Despite this, the diagram does a good job of emphasizing the operations that each class is capable of, which is a useful strategy for high-level knowledge of each class's behavior. The method signatures' simplicity may be by design, meant to facilitate communication without getting bogged down in the specifics. This is advantageous in the early stages of design, when the main goal is to get the big picture right.

In terms of association constraints, the diagram accurately places multiplicity, a type of constraint at the endpoints of association lines to convey the nature of the relationship.

- The **N:N relationship** between **Customer** and **Warehouse Manager** implies that multiple customers can interact with multiple warehouse managers.
- The **1:N relationship** between **Warehouse Manager** and **Warehouse Stock Management System** indicates that a single warehouse manager may be associated with multiple instances within the stock management system.
- The **N:1 relationship** between **Vendor** and **Warehouse Stock Management System** shows that multiple vendors can update a single stock management system.

In fact, these multiplicities are limitations that reveal crucial details about the possible relationships between instances of the different classes. They are positioned appropriately and provide helpful information on the cardinality of the associations.

4.Dynamic View

4.1 Dynamic View Semantics

1. The UML sequence diagram provided outlines a series of interactions between a Customer, Warehouse Manager, Warehouse Stock Management System, and Vendor. Based on the diagram and the standard practices of sequence diagrams, let's evaluate the dynamic view semantics:

2. **Customer Registration (Customer to Warehouse Stock Management System)**: This is a common starting point for customer interaction with a system, making it semantically correct and important for system design.
3. **Validate Registration (Warehouse Stock Management System)**: The system validates the customer registration, which is a necessary step to ensure data integrity and security, making this interaction semantically correct.
4. **Login (Customer to Warehouse Stock Management System)**: After registration, the next logical step for a user is to log in to the system, which is meaningful and essential for any secure system.
5. **Validate Credentials (Warehouse Stock Management System)**: The system needs to validate the login credentials, an expected behavior to ensure authorized access.
6. **Browse Products (Customer)**: Browsing products is a core feature of any shopping or warehouse management system, thus it's a semantically correct interaction.
7. **View Stock Details (Warehouse Manager to Warehouse Stock Management System)**: This interaction makes sense as a warehouse manager would need to check stock details to manage inventory.
8. **Revert Back Stock Status (Warehouse Stock Management System to Warehouse Manager)**: The system provides the stock status back to the warehouse manager, which is a practical and necessary interaction for inventory management.

9. **Update Product Details (Warehouse Manager to Warehouse Stock Management System)**: A warehouse manager updating product details in the system is a typical and required interaction.
10. **Payment (Customer)**: In the context of a purchase, payment is a critical step. However, the diagram lacks an interaction line to the system which should be processing the payment, which may be a semantic error.
11. **Place Order (Customer to Warehouse Stock Management System)**: After payment, placing an order is the expected sequence, so this interaction is meaningful.
12. **Receive Order Confirmation (Warehouse Stock Management System to Customer)**: The system should confirm the order to the customer, which is an important interaction for customer satisfaction and clarity.
13. **Logout (Customer)**: Logging out is a security measure and marks the end of a session, which is an appropriate and meaningful interaction.

4.2 Dynamic view Syntax

The interactions inside the warehouse stock management system are accurately shown in the UML sequence diagram that has been given. The steps that are shown adhere to a logical and cohesive order that is consistent with the typical workflows of these kinds of systems.

By incorporating the essential validation procedures for both the registration and login processes, the diagram accurately illustrates the core of a secure system, starting with client registration and guaranteeing that only verified users may

access the system. The layout makes it easier for customers to explore items after the security checks, which is a basic function for any retail or warehousing business.

The integration of warehouse manager functionalities, such as the ability to see and update stock information, is indicative of the practical requirement for continuous inventory control and upkeep of product details. These exchanges are not only significant but also critical to the operational efficiency of the warehouse.

The diagram shows that buying something involves a payment step, which is key. The customer starts the payment, and even though the diagram doesn't show every detail, we know that the system takes care of the actual payment handling. After paying, the customer gets a confirmation, which shows that the system is reliable and can be trusted with money matters.

Then, the customer places their order, and the system confirms it, wrapping up the usual online shopping process. When the customer logs out, it shows they're done and everything is secure, which is really important when dealing with personal information.

In simple terms, the diagram lays out all the steps clearly and every part matters for the system to work well. It shows exactly what's needed for a warehouse management system to run smoothly and how people and the system should interact, which is super important for the system to be easy to use and secure.

we can assert that the major behaviors of the system are described with syntactically correct sequence diagrams:

- The use of arrows throughout the diagram indicates method calls between the different entities (Customer, Warehouse Manager, Warehouse Stock Management System, Vendor), and these arrows are used correctly to depict both synchronous and asynchronous method calls.
- The execution occurrence boxes (narrow rectangles on the lifelines) seem properly aligned with the method calls to which they correspond, indicating where processes start and finish within the system's flow.
- The arrows' start and end points correctly align with the lifelines of the interacting entities, indicating the source and target of each method call, consistent with the conventions of sequence diagrams.
- The messages named along the arrows (e.g., "Customer Registration", "Validate Registration") suggest that these correspond to the interfaces or methods of the classes in the static view, though without the static view for reference, this is assumed to be correct in context.

Judging from the diagram, it's structured correctly and sticks to the rules for UML sequence diagrams, making it clear and precise in showing how the system's parts interact. The activities shown match what you'd expect in a warehouse stock management system, and they're in a sensible order that fits how the system is supposed to work.

5.Design Rationale:

Warehouse Stock Management System (WSMS) project is designed to tackle real-world issues in inventory and warehouse management. The project's intent is

to better the user experience, quicken processes, and facilitate decisions based on accurate data.

This system is built with features that serve the main users—customers, warehouse managers, and vendors. It aims to give a clear view of the current stock levels and movements of goods. The creation of user interfaces, browsing, and ordering systems, as well as inventory management, are all central to the system's functionality and support its purpose.

The mention of a payment system for customers suggests that the project considers essential non-functional aspects like user-friendliness and data security. Insights gained during the project emphasize the significance of designing with the user in mind, ensuring the system can scale, and prioritizing security from the outset.

Adopting agile development shows the project's commitment to refining the system over time, potentially integrating different design options as suggested by ongoing feedback. The project also points to the development still in progress, indicating a willingness to adapt features based on testing results.

Regarding non-functional requirements and design considerations, the document recognizes the importance of ensuring the system works across different browsers, is easy to maintain, monitors its performance effectively, and stays up-to-date with the latest technology. This project values teamwork, good use of version control, and ongoing user input—all contributing to the thought process behind the system's design choices.

1. **Database Design**:

- **Current Design**: A single database might be used for all data storage needs.
- **Alternative**: Using a microservices architecture, each service (customer service, inventory, order management) could have its own database to ensure loose coupling and independent scalability.

2. **Payment Processing**:
   - **Current Design**: The payment system might be directly integrated into the WSMS.
   - **Alternative**: A third-party payment gateway could be used, providing an extra layer of security and offloading the compliance requirements for financial data handling.
3. **User Interface (UI)**:
   - **Current Design**: A traditional web-based UI might be the primary interface for users.
   - **Alternative**: A single-page application (SPA) for a more responsive experience or progressive web app (PWA) to allow offline capabilities could be considered.
4. **Stock Management**:
   - **Current Design**: The WSMS may handle stock updates manually through warehouse manager inputs.
   - **Alternative**: An automated stock management system using IoT devices for real-time tracking and updates could be implemented.