Software Design Document

for

<Outlet Ordering Distribution System>

Version 1.0

Prepared by
< NURUL IZZATI MAISARAH BINTI MOHAMMAD PADZIL, 2023696408
NUR SYAFIKA ALYA BINTI MOHD ZAMRI AZHAR, 2023823848
NUR ATHIRAH BINTI MANSOR, 2023479376
NURUL ATHIRAH BINTI ASMADI, 2023862756>

<Team no.7 - FIVERS>

<22 JULY 2024>

Revision History

Name	Date	Reason For Changes	Version

Contents

1.0 Introduction

1.1 Purpose

Facing the fast increase of Smootea, a beverage store that has convenient store located in Kajang as its headquarter and has branches in Shah Alam and Kuantan, this Software Design Description (SDD) document focus on describe the architecture design of the Outlet Ordering Distribution System. This system was developed to address Smootea's current discrete stock request system by providing an original approach to addressing the problem. It is worth mentioning that prior to the invention of stock requests they used to take so much time preparing and submitting the forms through messaging applications like WhatsApp. Understanding these inefficiencies, the Outlet Ordering Distribution System seeks to provide an organized system for the management of stocks, in a routinized manner for both the HQ and outlet supervisors. This SDD outlines the different aspects of the system design and implementation such as the design justification, packages required and how they depend on one another as well as the interactions between the various parts of the system. It can help to define clear and detailed requirements towards the creation, sustaining and evolution of the system. As stated earlier, this document intends to provide Smootea with a technically sound and effective solution that will help it further develop its stock management processes.

1.2 Scope

The Outlet Ordering Distribution System is an innovative system built for Smootea, a growing beverage store with three locations: Kajang (HQ), Shah Alam, and Kuantan. The software is designed to substitute the manual stock request procedure, which previously required outlet supervisors to submit forms over WhatsApp, by providing a creative and efficient way. The primary objective is to assist both headquarters and outlet supervisors by providing a centralized platform for easy stock application placing and tracking. For headquarters, the system allows for efficient stock management, such as analyzing, creating, updating, and removal of stocks, as well as accepting or rejecting applications. Furthermore, HQ gains the ability to manage delivery through designated couriers and track the status of previous orders and application history. Outlet supervisors, on the other hand, benefit from simplified stock requests, order monitoring, as well as access to past order information. This software helps Smootea's overall business strategy through improving operational effectiveness and accuracy in stock management, resulting in a flexible and adaptable organizational structure.

1.3 References

Book Glinz, M., Loenhoud, H. van, Staal, S., & Bühne, S. (n.d.). Handbook for the Cpre Foundation Level according to the Ireb Standard (First Release, Vol. 1.0.0).

John W. Satzinger, Robert B. Jackson, Stephen D. Burd. System Analysis and Design in a Changing World (7th edition) .

Website Bandakkanavar, R. (2023, May 8). Software Requirements Specification document with example.

Krazytech. *https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database*

1.4 Document Structure

The Outlet Ordering Distribution System comes as a standalone and independent software system developed exclusively for Smootea, a beverage store with three separate locations. Unlike an easy upgrade or component of an existing system, this new approach solves an important issue in Smootea's operations which is the manual stock request procedure. Previously relying on a form and WhatsApp uploads, the innovation automates and transforms the stock ordering process. It highlights Smootea's commitment to performance effectiveness and adaptability to market demands. The proposed approach not only eliminates the workload for both headquarters and outlet supervisors, but it also indicates a change toward modern, efficient business procedures.

The system has two primary components which is the HQ interface and the outlet supervisor interface. These interfaces are closely connected, generating an integrated system that enables stock request, management, and delivery. The structure of the system provides a smooth flow of data between headquarters and individual outlets, stimulating effective communication and cooperation. This standalone character of the Outlet Ordering Distribution System highlights its significance as a key tool in Smootea's aim to achieve efficient operations and customer satisfaction.

1.5 Definitions, Acronyms and Abbreviations

Convention	**Description**
Font Size	The header will include a font size of 18. The sub header is going to have a font size of 14. Font size 11 will be used for writing written material.
Font Type	The font used for the header, sub header, and written content is Arial.
Bold	It will be used to highlight headers and sub headers, as well as focusing on some contents titles.
SRS	Stands for System Requirement Specification
DB	Stands for database
SSD	Stands for system sequence diagram
HQ	Stands for headquarters
CRUD	Create, Update and Delete

1.6 System Overview

Functionality

The Outlet Ordering Distribution System (OODS) is designed to streamline and automate the stock management process for Smootea, a growing bubble tea store with multiple locations. The system replaces the manual, time-consuming process of stock requests done through forms and messaging apps like WhatsApp. The key functionalities of the system include:

1. Stock Request and Management:

- Outlet supervisors can request stocks by adding their desired stocks into cart and proceed cart checkout.
- Headquarters (HQ) can manage these requests by accepting or rejecting them and arranging deliveries.

2. Stock Control:

- HQ can create, update, and delete stock items in the system.
- Both HQ and outlet supervisors can view stock details and track delivery status.

3. Order Tracking:

- Users can monitor the status of stock orders, view past order histories and track the delivery process.

Context

The development of this system is in response to the rapid growth of Smootea, which has outlets in Kajang, Shah Alam, and Kuantan. The existing manual stock request method was inefficient and prone to errors. The new system aims to:

- Increase operational efficiency.
- Reduce the time and effort required for stock management.
- Improve accuracy in stock handling and tracking.
- Enhance communication and coordination between HQ and the outlets.

Design

The system has two main user interfaces: one for the HQ and another for outlet supervisors. Key design elements include:

1. HQ Interface:

- Allows HQ staff to manage inventory, approve/reject stock requests, and handle deliveries.
- Provides tools for tracking past orders and managing user access.

2. Outlet Supervisor Interface:

- Enables supervisors to submit stock requests, track order statuses, and view historical data.

3. User Classes and Characteristics:

- HQ Admins: Have high privileges, manage inventory, and perform administrative tasks.
- Outlet Supervisors: Have limited privileges, focus on requesting and tracking stocks.

4. Operating Environment

- The system requires a stable internet connection and is designed to run on Windows 10 or above, with specific hardware requirements outlined for optimal performance.

Background Information

A more effective stock management system was required as Smootea expanded in order to meet its operational demands. The manual processes used in the old system were starting to become inefficient. By offering a consolidated platform for all stock management tasks, the new automated system solves these problems and guarantees more efficient operations and resource management.

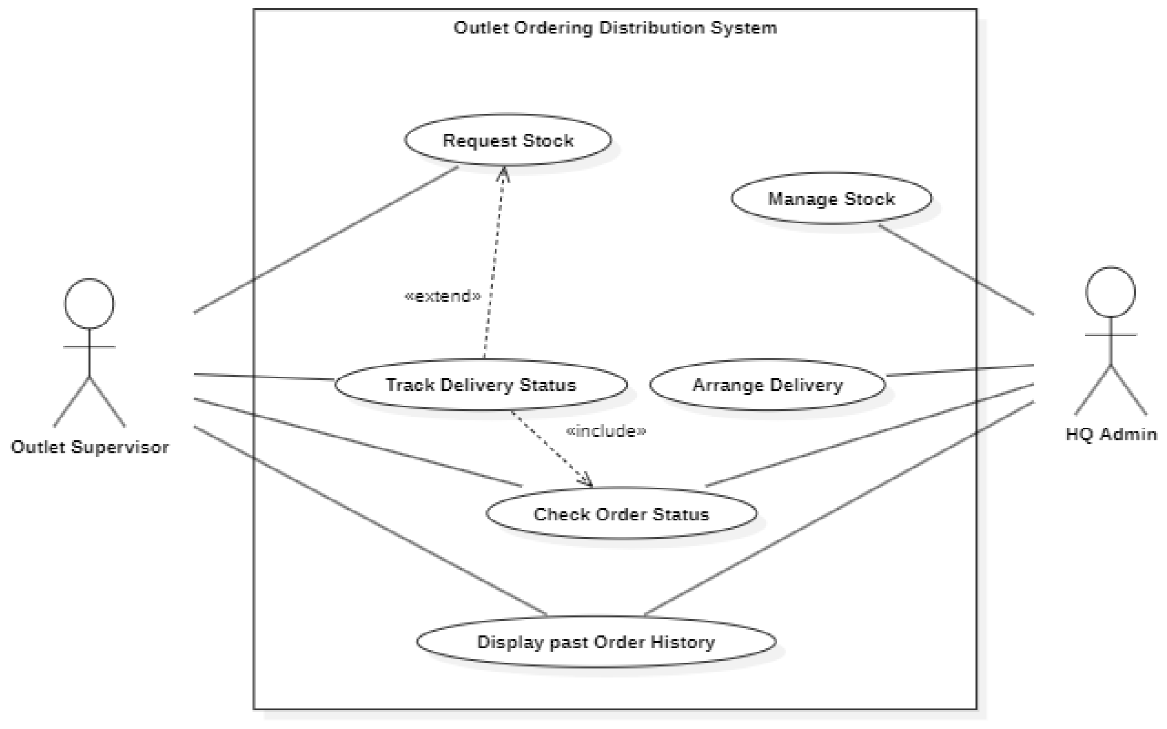


Figure 1.0 Use Case Diagram

1. **UCD100 Request Stock**

- The Outlet Supervisor can request stock from the system.

2. **UCD101 Manage Stock**

- The HQ Admin can manage (CRUD) stocks in the system.

3. **UCD102 Track Delivery Status**

- The Outlet Supervisor can track the status of deliveries for their requested stocks.

4. **UCD103 Arrange Delivery**

- The HQ Admin can arrange deliveries for requested stock.

5. **UCD104 Check Order Status**

- The Outlet Supervisor check the approval status of orders either pending, rejected or approved.

- HQ Admin can give approvals to the requested order either pending, rejected or approved

6. **UCD105 Display Past Order History**

- The Outlet Supervisor and HQ Admin can display the history of past orders made through the system.

2.0 System Architecture

2.1 Architectural Description

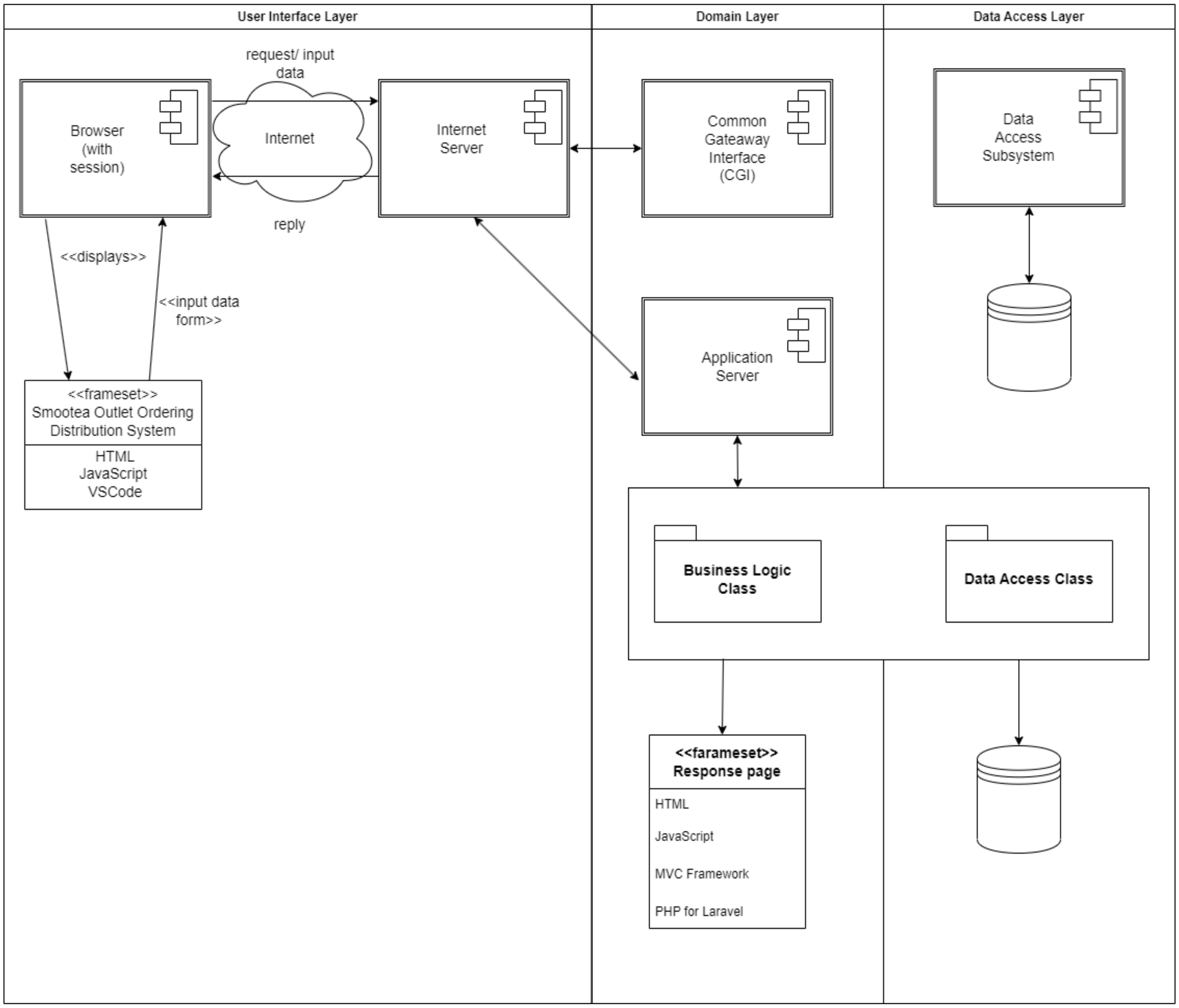


Figure 2.0

Smootea Outlet Ordering Distribution System is specifically laid down based on a three-layer architecture model. This model separates the system into three main layers: These are User Interface Layer, the Domain Layer and the Data Access Layer. Every level has certain tasks, thus, the overall data is divided and organized quite well and that is why it is easier to maintain, scale and make necessary changes in an application that follows this pattern.

First, the User Interface Layer receives the Browser (with session), through which users work with the system. It deals with the user's incoming requests and input data and formulates and presents output data forms. Then, the Internet will serve as the tool through which the browser and the Internet Server interact; the later taking requests from the browser, processing them to provide the appropriate responses. Concerning the <<frameset>> includes the HTML for the layout, JavaScript for the interactivity and VSCode for the development of the user interface, whereas <<frameset Response >> page involves displaying the content based on the actions performed by the users using HTML, JavaScript, MVC Framework, and PHP for Laravel The User Interface Layer deals with presenting the layout to the Outlet Supervisors and the HQ Admins. It receives entries from a user and shows the result generated at its end. This layer interacts with the Domain Layer in order to get or send data following activities performed by the user.

The next model is the Common Getaway Interface located in the Domain Layer; it interacts with the Internet Server and transfers info from/to the suitable application server. Also, Application Server is responsible for the management of application's business logic and processes the incoming requests while interacting with Business Logic Class and Data Access Class. Business Logic Class – This part carries the major operation of the system where most business activities like tracking an order, stock control, order delivery arrangement and lots of other related activities is channeled. The Domain Layer involves core components of the system as well as the business processes of an organization. It accepts a request from the User Interface Layer, computes or does something in response and accesses the Data Access Layer to read or write data. This layer guarantees the right implementation of the business rules and required workflows.

Finally, Data Access Layer has Data Access Subsystem helping it to access the database to perform queries and return or update data. Data Access Class can be defined as the assemblage that accesses the data residing in the database and different from business logic. With respect to Database (MySQL), engaging in the collection and storage of all related data of the system including, but not limited to, stock data, orders, and order status. Again, it improves performance because the frequently requested information is stored in Cache. Some of the subclasses of Data Access Layer implementation include the responsibility of writing to and reading from the database. It offers a disguise for data operation in the Domain Layer so that the actual database structure is not revealed. This layer guarantees the proper management of data and its safe reception by the required person.

2.2 Design Rationale

Smootea has chosen the three-layered architectural model for the Smootea Outlet Ordering Distribution System for the following reasons including modularity, maintainability, scalability, and lastly, the reduction of complication. Such a division of the software allows the independent work on different layers: User Interface, Domain, and Data Access, and when changing one layer, it will not harm the others. This modularity makes it easier to debug and test the system, as is necessary when creating a system meant to automate certain key processes involved in the stock management of various establishments.

Contingent problems or trade-off decisions that may be encountered when deciding on the solution are performance, the system's complexity, and its potential for further development. Performance optimization by caching as well as efficient access patterns are a byproduct of VSF's three-layer architecture. Although this architecture complicates the process of coordinating the interactions between layers it separates the responsibilities of the program well, making it manageable. Some of the compromises which were made included issues like the overhead that would be associated with having many layers together with the problem of latency which is associated with most inter layer communications was deemed fine because of the benefits that was likely to be associated with the long run in terms of maintainability and scalability.

Other architectures including the deployment diagram and the network diagram could have been adopted but they were not selected. A deployment model that is quite relevant to the physical deployment of the software on the platform of hardware would not satisfy the needs of the logical structure of the issue in this situation. Likewise, a network diagram, which focuses on network structure and connection between the physical entities, is not sufficient to reveal the details necessary for application logic and data processing. Thus, the three-layer architecture was selected as the one that distributes business logic, data storage, and the interaction between the application and the user fairly as appropriate for achieving the project objectives.

2.3 Decomposition Description

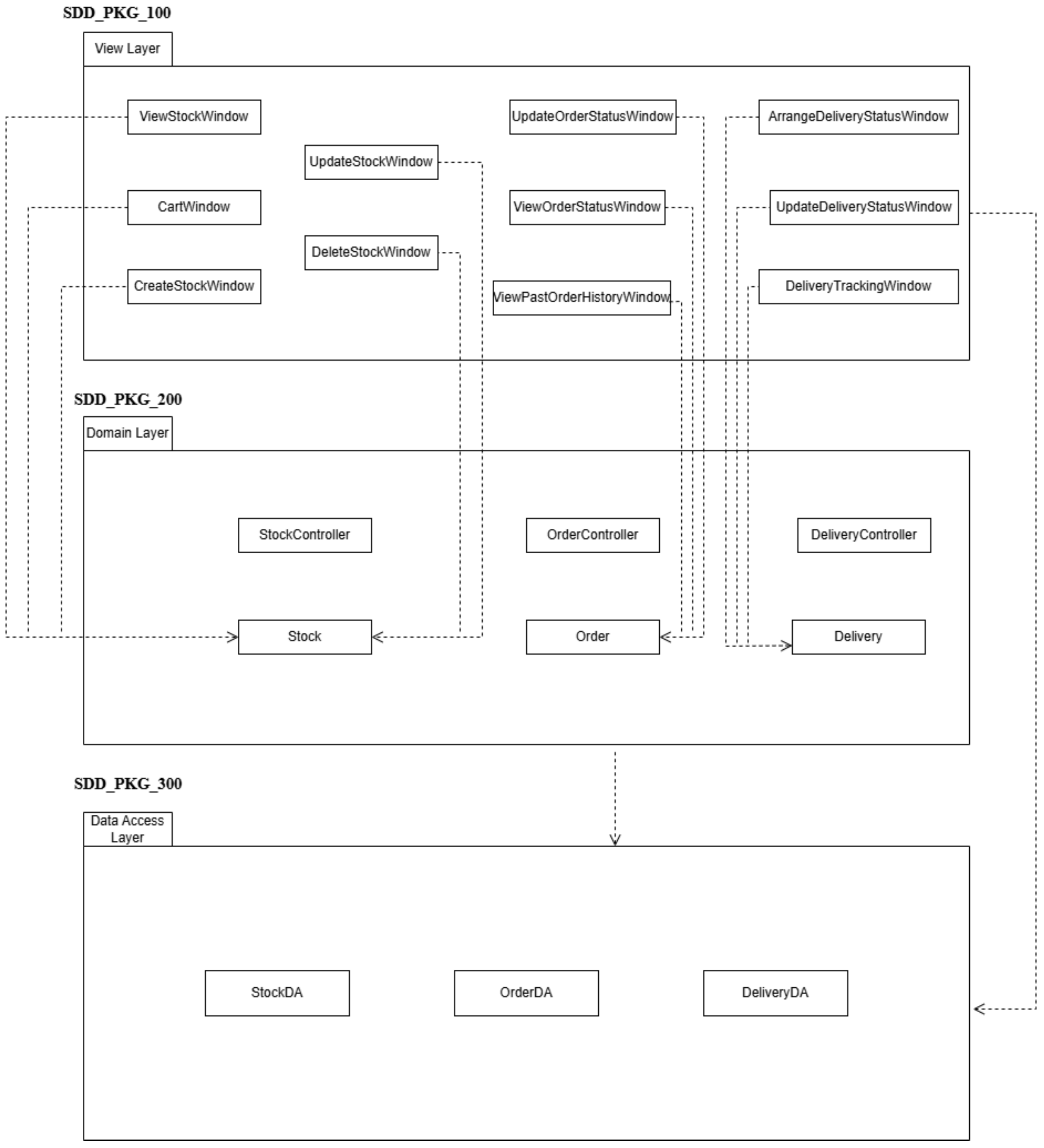


Figure 2.1 Package Diagram

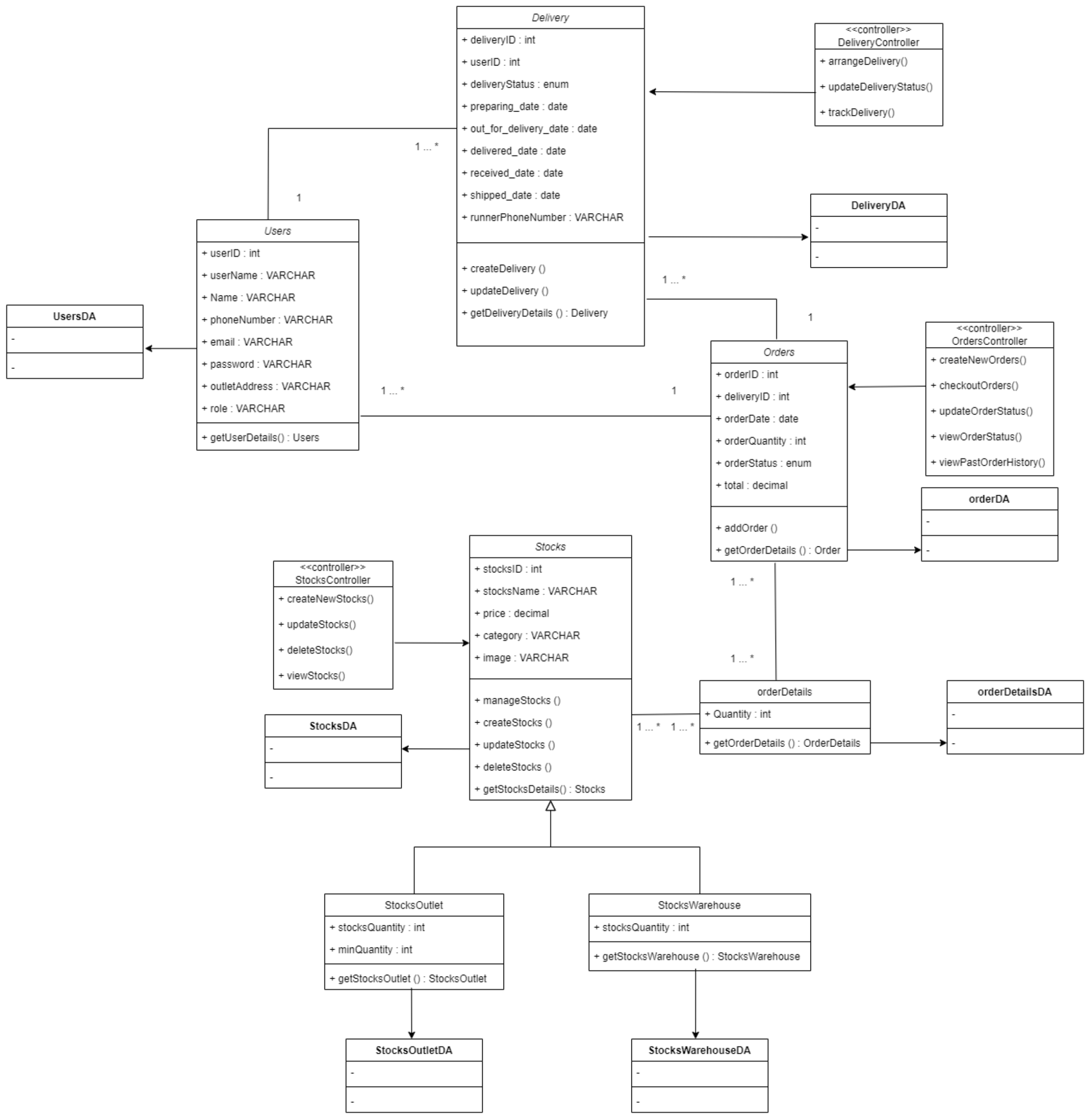


Figure 2.2 Detailed Design Class Diagram

3.0 Data Design

3.1 Database Description

The Outlet Ordering Distribution System (OODS) employs MySQL version 8.0 as its database management system (DBMS) software to store and manage all its data. Several tables are included in the OODS database to handle different system functions and relationships between entities. These tables include "Users" to store user details, such as user IDs, usernames, emails and passwords. The "Stocks" table stores all inventory details, including product IDs, names, quantities and other relevant attributes. "Deliveries" is used to track delivery statuses, including delivery IDs, order IDs, delivery dates, statuses and other relevant attributes. The "OrderDetails" table stores detailed information about each order, such as order IDs, product IDs, quantities, prices and other relevant attributes. The "Orders" table stores order information, including order IDs, user IDs, order dates, total amounts, statuses and other relevant attributes. "StockOutlet" manages the relationship and inventory levels between outlets and the central system, including outlet IDs, stock IDs, quantities and other relevant attributes. Lastly, "StockWarehouse" manages the relationship and inventory levels between warehouses and the central system, including warehouse IDs, stock IDs, quantities and other relevant attributes. The data dictionary specifies the data type and length for each attribute in these tables.

3.2 Mapping of Problem Domain Objects to Relational Tables

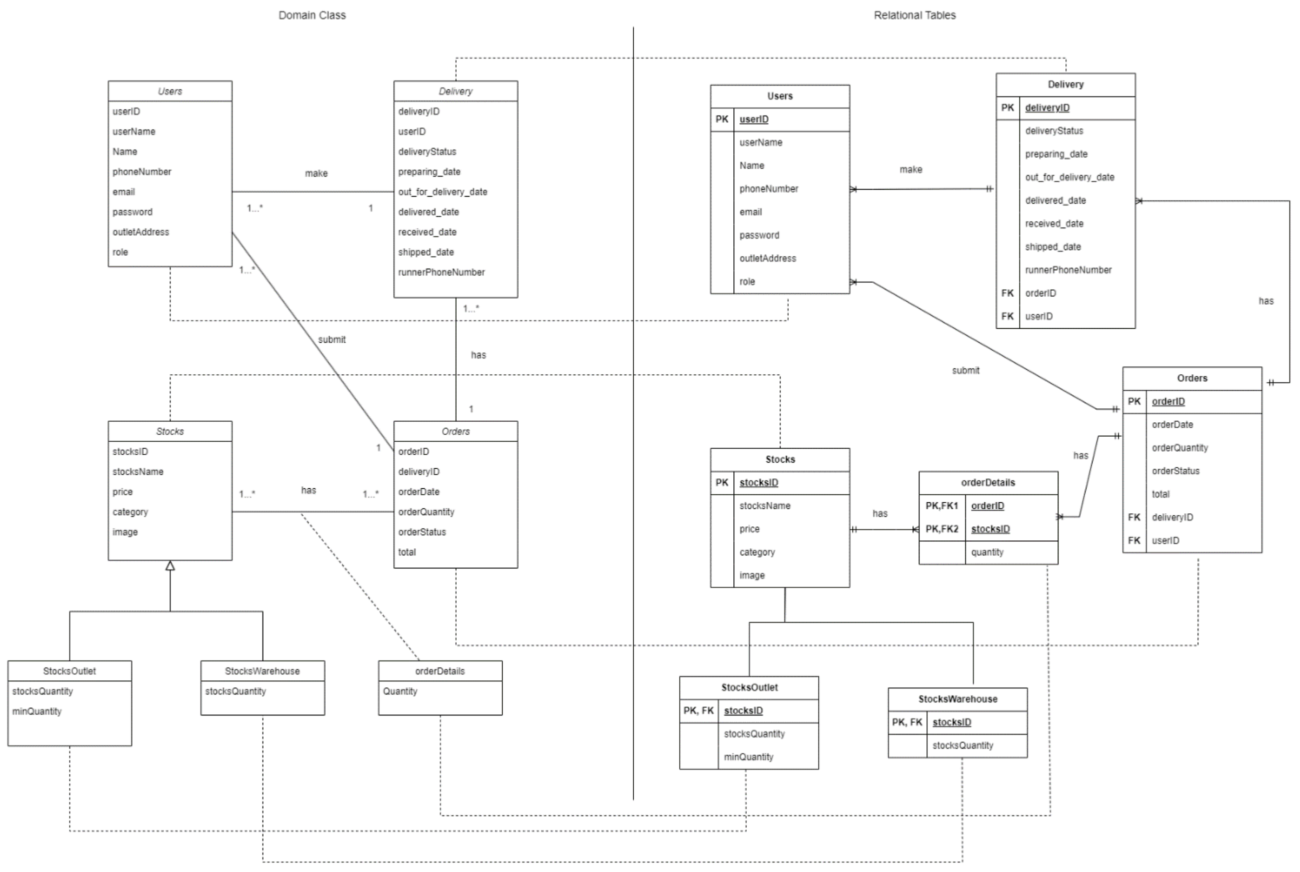


Figure 3.0 Mapping of Problem Domain Objects to Relational Tables

3.3 Data Dictionary

Users

Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
userID	ID of the user	bigint(20)	auto_increment	N	-	Y	Y
userName	Username of the user	varchar(100)	-	N	-	Y	N
name	Name of the user	varchar(255)	-	Y	null	N	N
phoneNumber	Phone number of the user	varchar(11)	-	N	-	Y	N
email	Email of the user	varchar(255)	-	N	-	Y	Y
Email_verified_at	Email verified date	timestamp	-	Y	null	N	N
password	Password of the user	varchar(100)	-	N	-	Y	N
outletAddress	Outlet address of the user	varchar(255)	-	N	-	Y	N
role	Role of the user	varchar(255)	-	N	user	Y	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

Stocks

Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
stocksID	ID of the stock	bigint(20)	auto_increment	N	-	Y	Y

stocksName	Name of the stock	varchar(200)	-	N	-	Y	N
price	Price of the stock	decimal(10,2)	-	N	-	Y	N
category	Category of the stock	varchar(50)	-	N	-	Y	N
image	Image of the stock	varchar(255)	-	Y	null	N	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

Stock Outlet							
Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
stocksID	ID of the stock in outlet	bigint(20)	stocks.stocksID, ON UPDATE RESTRICT, ON DELETE CASCADE	N	-	Y	N
stocksQuantity	Quantity of the stock in outlet	int(11)	-	N	-	Y	N
minQuantity	Minimum quantity of the stock in outlet	int(11)	-	N	-	Y	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

Stock Warehouse							
Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
stocksID	ID of the stock in warehouse	bigint(20)	stocks.stocksID, ON UPDATE RESTRICT,	N	-	Y	N

			ON DELETE CASCADE				
stocksQuantity	Quantity of stock in warehouse	int(11)	-	N	-	Y	N

M=Mandatory? U=Unique?, Y= Yes, N = No

Deliveries							
Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
deliveryID	ID of the delivery	bigint(20)	auto_increment	N	-	Y	Y
userID	ID of the user	bigint(20)	-> users.userID, ON UPDATE CASCADE, ON DELETE CASCADE	Y	null	N	Y
deliveryStatus	Status of the delivery	enum('Preparing for Delivery', 'Shipped', 'Out for Delivery', 'Delivered')	-	Y	null	N	N
preparing_date	Preparing date	date	-	Y	null	N	N
out_for_delivery_date	Out for delivery date	date	-	Y	null	N	N
delivered_date	Delivered date	date	-	Y	null	N	N
received_date	Received date	date	-	Y	null	N	N
shipped_date	Shipped date	date	-	Y	null	N	N

runnerPhoneNumber	Phone number of the runner	varchar(15)	-	Y	null	N	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

Orders

Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U
orderID	ID of the order	bigint(20)	auto_increment	N	-	Y	Y
deliveryID	ID of the delivery	bigint(20)	-> deliveries.deliveryID, ON UPDATE RESTRICT, ON DELETE CASCADE	Y	null	N	N
userID	ID of the user	bigint(20)	-> users.userID, ON UPDATE CASCADE, ON DELETE CASCADE	N	-	Y	N
orderDate	Date for the order	date	-	N	-	Y	N
orderQuantity	Quantity of the order	int(11)	-	N	-	Y	N
orderStatus	Status of the order	enum('Approved', 'Pending', 'Rejected')	-	N	-	Y	N
total	Total	decimal(10,2)	-	N	-	Y	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

Order Details

Attribute Name	Description	Type	Additional Information	Nullable	Default Value	M	U

orderID	ID of the order	bigint(20)	-> orders.orderID, ON UPDATE CASCADE, ON DELETE CASCADE	N	-	Y	N
stocksID	ID of the stock	bigint(20)	-> stocks.stocksID, ON UPDATE CASCADE, ON DELETE CASCADE	N	-	Y	N
quantity	Quantity	int(11)	-	N	-	Y	N

M=Mandatory?, U=Unique?, Y= Yes, N = No

4.0 Component Design

4.1 Package Identifier

Identifier	Package
SDD_PKG_100	View Layer
SDD_PKG_200	Domain Layer
SDD_PKG_300	Data Access Layer

4.1.2 Package Purpose

A Reference Back to The Requirements Specification

The creation and functions of each package are based on the requirements outlined in the requirements specification document, specifically the Software Requirements Specification version 1.4. This document, along with stakeholders' requirements, various websites for supplementary information and consultations with lecturers for advice regarding the project, ensures that each component is designed to meet the specific needs and performance standards necessary for the system’s success.

Rationale For Creation of The Package:

Each package has been created to address particular functional and performance needs, ensuring that the system operates efficiently and effectively. Below are the rationales for each package:

- **View Layer (SDD_PKG_100):** This package is created to handle user interactions and ensure a seamless user experience. It provides the necessary interfaces for users to input data and interact with the system. The View Layer meets the requirements for user input handling, data display and updating input data.
- **Domain Layer (SDD_PKG_200):** The Domain Layer is responsible for processing business rules and logic. It abstracts complex business processes and ensures that the system adheres to the specified business rules. This package meets the requirements for business logic processing, data abstraction, and improving the readability of the View Layer.
- **Data Access Layer (SDD_PKG_300):** This package is essential for managing the system’s data operations. It establishes connections to the database, maintains these connections and

handles CRUD operations. The Data Access Layer meets the requirements for data source creation, database connectivity, data manipulation and user request handling.

4.1.3 Package Function

- *What does the package do?*

PACKAGE	PACKAGE FUNCTION
View Layer (SDD_PKG_100)	• To present data that users can read or create on specific windows • To modify the input data • To manage user interactions • To show specific forms for user data input and system retrieval • To upload data through designated windows
Domain Layer (SDD_PKG_200)	• To represent actual entities in the outlet stock distribution system • To process business rules for use cases • To utilize straightforward business logic • To merge data and abstract logic from a complex database to enhance the readability of the ViewLayer
Data Access Layer (SDD_PKG_300)	• To establish a data source • To configure a database connection • To sustain data connectivity • To perform CRUD (Create, Read, Update, Delete) operations on data • To enable data search functionality • To handle user requests during sessions

Table 4.1 Package Functions

4.1.4 Package Dependencies

Every package might be dependent on some other packages in the system. These dependencies define the factors that connect one package to the other and the limitations placed on one package by another. The View Layer (SDD_PKG_100) uses the function of the Domain Layer (SDD_PKG_200) for data controlling and business logic calculations. It is required that it should always be together with the Domain Layer to work effectively. Much like the implementation of the Business Logic Layer, the Domain Layer (SDD_PKG_200) has a dependency on the Data Access Layer (SDD_PKG_300) during data access/mutation activities. When it comes to CRUD operations, and database connections, this responsibility lies on the Data Access Layer. SDD_PKG_300 Data Access Layer is the core component of the entire system because the layer will directly work with the database. Disruptions or challenges within this layer can impact the rest of the layers of the given supply chain.

4.1.5 Package Components

PACKAGES	**CLASSSES**
View Layer	ViewStockWindow CartWindow CreateStockWindow UpdateStockWindow DeleteStockWindow UpdateOrderStatusWindow ViewOrderStatusWindow ViewPastOrderHistoryWindow ArrangeDeiveryStatusWindow UpdateDeliveryStatusWindow DeliveryTrackingWindow

Domain Layer	StockController OrderController DeliveryController Stock Order Delivery
Data Access Layer	StockDAO OrderDAO DeliveryDAO

4.1.5.1 Class Identifier

Packages	Classes
View Layer (SDD_PKG_100)	ViewStockWindow (SDD_PKG_101) CartWindow (SDD_PKG_102) CreateStockWindow (SDD_PKG_103) UpdateStockWindow (SDD_PKG_104) DeleteStockWindow (SDD_PKG_105) UpdateOrderStatusWindow (SDD_PKG_106) ViewOrderStatusWindow (SDD_PKG_107) ViewPastOrderHistoryWindow (SDD_PKG_108) ArrangeDeliveryStatusWindow (SDD_PKG_109) UpdateDeliveryStatusWindow (SDD_PKG_110) DeliveryTrackingWindow (SDD_PKG_111)

Domain Layer (SDD_PKG_200)	StockController (SDD_PKG_201) OrderController (SDD_PKG_202) DeliveryController (SDD_PKG_203 Stock (SDD_PKG_204) Order (SDD_PKG_205) Delivery (SDD_PKG_206)
Data Access Layer (SDD_PKG_300)	StockDAO (SDD_PKG_301) OrderDAO(SDD_PKG_302) DeliveryDAO (SDD_PKG_303)

4.1.5.2 Class **Purpose**

Class	Purpose
ViewStockWindow (SDD_PKG_101)	This class is for displaying stock items for users to view and manage. Users can see details such as stock ID, name, current quantity, and price. It provides options to update stock quantities or add items to a cart for requesting new stock.
CartWindow (SDD_PKG_102)	The purpose of this class it manages the cart for adding stock items before making a request. Users can adjust quantities, remove items, clear the cart, and proceed to checkout to request stock from HQ. The rationale is it facilitates the process of requesting stock by allowing outlet supervisors to compile and manage their requests efficiently.
CreateStockWindow (SDD_PKG_103)	The purpose of this class is it allows user to input stock name, quantity, price, category, and upload an image to add a new stock.
UpdateStockWindow (SDD_PKG_104)	This class allows users to update details of existing stock items. This includes changing the stock name, quantity, price, and category. It enables HQ admin to maintain

	accurate and up-to-date stock information which is essential for effective inventory management
DeleteStockWindow (SDD_PKG_105)	This class is to ensures that HQ admin can remove outdated or discontinued items from the inventory, keeping the stock list current.
UpdateOrderStatusWindow (SDD_PKG_106)	The purpose of this class is it allows HQ admin to change the status of orders to reflect their current state, such as pending, approved, or rejected.
ViewOrderStatusWindow (SDD_PKG_107)	This class Provides transparency and visibility into the status of orders, allowing users to track the progress of their requests.
ViewPastOrderHistoryWindow (SDD_PKG_108)	The purpose of this class is it enables both HQ admin and outlet supervisors to review past transactions, which is useful for auditing and reference purposes.
ArrangeDeliveryStatusWindow (SDD_PKG_109)	This class is for arranging deliveries. HQ admin can assign delivery details to orders, including delivery ID, status, and date.
UpdateDeliveryStatusWindow (SDD_PKG_110)	This class allows HQ admin to change the delivery status and update the delivery date.
DeliveryTrackingWindow (SDD_PKG_111)	Provides a comprehensive view of delivery progress and enabling users to track deliveries from dispatch to received.
StockController (SDD_PKG_201)	This class is to Manages operations related to stock items, including creating, updating, and deleting stock.It will interacts with ViewStockWindow, UpdateStockWindow, CreateStockWindow, and DeleteStockWindow to process user inputs and manage stock data
OrderController (SDD_PKG_202)	This class will handles operations related to order processing, such as viewing order statuses, updating orders, and managing order history.It will interact with UpdateOrderStatusWindow, ViewOrderStatusWindow, and ViewPastOrderHistoryWindow to update and display order data.
DeliveryController (SDD_PKG_203	This class will Manages delivery-related operations, including arranging deliveries and updating delivery

	statuses. It will interact with ArrangeDeliveryStatusWindow, UpdateDeliveryStatusWindow, and DeliveryTrackingWindow to handle delivery processes.
Stock (SDD_PKG_204)	This class represents stock item entities, holding data such as stock ID, name, quantity, and price.
Order (SDD_PKG_205)	This class represents order entities, holding data such as order ID, date, status, and items ordered.
Delivery (SDD_PKG_206	This class represents delivery entities, holding data such as delivery ID, status, date, and associated orders.
StockDAO (SDD_PKG_301)	The purpose of this class is to facilitates CRUD operations for stock items in the database, including creating, reading, updating, and deleting stock data.
OrderDAO(SDD_PKG_302)	This class will facilitate CRUD operations for orders in the database, including creating, reading and updating order data.
DeliveryDAO (SDD_PKG_303)	This class will facilitate CRUD operations for deliveries in the database, including creating, reading and updating delivery data

4.1.5.3 Class function

- *What does the class do?*

View Layer		
Class	Pseudocode	Method

ViewStockWindow (SDD_PKG_101)	BEGIN
	1. Display ViewStockWindow
	END	

CartWindow (SDD_PKG_102)	BEGIN
	1. Display CartWindow
	END	

CreateStockWindow (SDD_PKG_103)	BEGIN
	1. Display CreateStockWindow
	END	

UpdateStockWindow (SDD_PKG_104)	BEGIN
	1. Display UpdateStockWindow
	END	

DeleteStockWindow	BEGIN	

(SDD_PKG_105)	1. Display DeleteStockWindow END	

UpdateOrderStatusWindow
(SDD_PKG_106)
	BEGIN
	1. Display UpdateOrderStatusWindow
	END	

ViewOrderStatusWindow
(SDD_PKG_107)
	BEGIN
	1. Display ViewOrderStatusWindow
	END	

ViewPastOrderHistoryWindow
(SDD_PKG_108)
	BEGIN
	1. Display ViewPastOrderHistoryWindow
	END	

ArrangeDeliveryStatusWindow
(SDD_PKG_109)
	BEGIN
	1. Display ArrangeDeliveryStatusWindow
	END	

UpdateDeliveryStatusWindow
(SDD_PKG_110)
	BEGIN
	1. Display UpdateDeliveryStatusWindow
	END	

DeliveryTrackingWindow
(SDD_PKG_111)
	BEGIN
	1. Display DeliveryTrackingWindow
	END	

Domain Layer		
Class	Pseudocode	Method

StockController
(SDD_PKG_201)
	BEGIN
	1. Retrieve stock data from html.
	2. Display stocks's data.
	
	If value method is create new stock,
	1. Create store access object and store object.
	2. Store new stock's information.
	3. Set stock status to newly added status.
	
	If value method is update stock
	1. Create update
		store()
		
		StockDAO()

	stock object.
	2. Store updated stock's information.
	3. Set stock status to updated status.
	
	If value method is delete stock
	1. Create delete stock object.
	2. Remove stock's information.
	3. Confirm stock deletion.
	END
		update()
		
		delete()

OrderController (SDD_PKG_202)	BEGIN
	Retrieve data from html.
	
	If value method is create order,
	1. Create order data access object and create object.
	2. Store new order information.
	3. Set order status to created.
	
	If value method is view order,
	1. Retrieve order information.
	2. Display order information.
	
	If value method is update order information,
	1. Create order
		OrderDAO()
		
		create()

	status update object.
	2. Store updated order status.
	3. Set order status to be updated.
	
	If value method is delete order,
	1. Create order delete object.
	2. Delete order information.
	3. Confirm order deletion.
	
	If value method is checkout,
	1. Create checkout object.
	2. Set order to checked out.
	
	If the value method is view order details,
	1. Create order details object.
	2. Retrieve data from order's information.
	3. Display order details.
	END
		view()
		
		update()
		
		delete()
		
		checkout()
		
		orderDetails()

DeliveryController (SDD_PKG	BEGIN Retrieve data from html.	

_203	If the value method is tracking delivered order,
	1. Create tracking delivery object and delivery data access object .
	2. Retrieve delivery tracking data.
	3. Display delivery tracking status.
	
	If the value method is update delivery status,
	1. Create delivery status update object.
	2. Store updated delivery status.
	3. Set delivery status to updated status.
	
	If the value method is delete delivery,
	1. Create delivery deletion object.
	2. Delete delivery information.
	3. Confirm delivery deletion.
	END
		DeliveryDAO()
		
		track()
		
		update()
		
		delete()

Stock (SDD_PKG _204)	BEGIN
	
	Stock(stocksID, stocksName,price, category, image)
	
	getStocksID()
	Return object's stocksID
	
	getStocksName()
	Return object's
		Stock(stocksID, stocksName,price, category, image)
		
		setStocksID(stocksID)
		
		getStocksID()
		
		setStocksName(stocks Name)
		
		setPrice(price)

	stocksName
	
	getPrice()
	Return object's price
	
	getQuantity()
	Return object's quantity
	
	setStocksID(stocksID)
	1. Receive stocksID from parameter
	2. Set object's stocksID with stocksID value
	
	setStocksName(stocksName)
	1. Receive stocksName from parameter
	2. Set object's stocksName with stocksName value
	
	setPrice(price)
	1. Receive price from parameter
	2. Set object's price with price value
	
	setQuantity(quantity)
	1. Receive quantity from parameter
	2. Set object's quantity with quantity value
	
	END
		getPrice()
		
		getStocksName()
		
		setQuantity(quantity)
		
		getQuantity()

Order (SDD_PKG_205)	BEGIN
	
	Order(orderID, deliveryID, userID, orderDate, orderQuantity, orderStatus, total)
	
	getOrderID()
	Return object's orderID
	
	getDeliveryID()
	Return object's deliveryID
		Order(orderID, deliveryID, userID, orderDate, orderQuantity, orderStatus, total)
		
		getOrderID()
		
		setOrderID(String orderID)
		
		getDeliveryID()
		
		setDeliveryID(String deliveryID)

	getUserID()
	Return object's userID
	
	getOrderDate()
	Return object's ordeDate
	
	getOrderQuantity()
	Return object's orderQuantity
	
	getOrderStatus()
	Return object's orderStatus
	
	getTotal()
	Return object's total
	
	setOrderID(String orderID)
	1. Receive orderID from parameter
	2. Set object's orderID with orderID value
	
	setDeliveryID(String deliveryID)
	1. Receive deliveryID from parameter
	2. Set object's deliveryID with deliveryID value
	
	setUserID(String userID)
	1. Receive userID from parameter
	2. Set object's userID with userID value.
	
	setOrderDate(String orderDate)
	1. Receive orderDate from parameter
	2. Set object's orderDate with orderDate value.
	
	setOrderQuantity(int orderQuantity)
		getUserID()
		
		setUserID(String userID)
		
		getOrderDate()
		
		setOrderDate(String orderDate)
		
		getOrderQuantity()
		
		setOrderQuantity(int orderQuantity)
		
		getOrderStatus()
		
		setOrderStatus(String orderStatus)
		
		getTotal()
		
		setTotal(double total)

	1. Retrieve orderQuantity from parameter. 2. Set object's orderQuantity with orderQuantity value. setOrderStatus(String orderStatus) 1. Retrieve orderStatus from parameter. 2. Set object's orderStatus with orderStatus value. setTotal(double total) 1. Retrieve total from parameter. 2. Set object's total with total value. END	

Delivery (SDD_PKG _206)	BEGIN
	
	Delivery(deliveryID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
	
	getDeliveryID()
	Return object's deliveryID
	
	getDeliveryStatus()
	Return object's deliveryStatus
	
	getPreparingDate()
	Return object's preparingDate
	
	getOutForDeliveryDate()
	Return object's outForDeliveryDate
		Delivery(deliveryID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
		
		getDeliveryID()
		
		setDeliveryID(String deliveryID)
		
		getDeliveryStatus()
		
		setDeliveryStatus(String deliveryStatus)
		
		getPreparingDate()
		
		setPreparingDate(String preparingDate)
		
		getOutForDeliveryDate()
		
		setOutForDeliveryDat

	getDeliveredDate()
	Return object's deliveredDate
	
	getReceivedDate()
	Return object's receivedDate
	
	getShippedDate()
	Return object's shippedDate
	
	getRunnerPhoneNumber()
	Return object's runnerPhoneNumber
	
	setDeliveryID(String deliveryID)
	1. Receive deliveryID from parameter
	2. Set object's deliveryID with deliveryID value
	
	setDeliveryStatus(String deliveryStatus)
	1. Retrieve deliveryStatus from parameter.
	2. Set object's deliveryStatus with deliveryStatus value.
	
	setPreparingDate(String preparingDate)
	1. Receive preparingDate from parameter.
	2. Set object's preparingDate with preparing Date value.
	
	setOutForDeliveryDate(String outForDeliveryDate)
	1. Retrieve outForDeliveryDate from parameter
	2. Set object's
		e(String outForDeliveryDate)
		
		getDeliveredDate()
		
		setDeliveredDate(String deliveredDate)
		
		getReceivedDate()
		
		setReceivedDate(String receivedDate)
		
		getShippedDate()
		
		setShippedDate(String shippedDate)
		
		getRunnerPhoneNumber()
		
		setRunnerPhoneNumber(String runnerPhoneNumber)

	outForDelivery Date with outForDelivery Date value. setDeliveredDate(String deliveredDate) 1. Retrieve deliveredDate from parameter. 3. Set object's deliveredDate with deliveredDate value. setReceivedDate(String receivedDate) 1. Retrieve receivedDate from parameter. 4. Set object's receivedDate with receivedDate value. setShippedDate(String shippedDate) 1. Retrieve shippedDate from parameter. 2. Set object' shippedDate with shippedDate value. setRunnerPhoneNumber(String runnerPhoneNumber) 5. Receive runnerPhoneNumber from parameter. 6. Set object's runnerPhoneNumber with runnerPhoneNumber value. END	

Data Access Layer		
Class	Pseudocode	Method

StockDAO (SDD_PKG _301)	BEGIN findStockItem((bigint(20) stockID) 1. Create connection, statement, resultset variable 2. Establish connection with database 3. Execute SQL statement to get stock item details based on stockID and assign it to resultset variable 4. Set id, name, quantity and price derived from resultset data to stock object 5. Return stock object 6. Catch exception 7. Display exception createStockItem(Stock stock) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to insert new stock item details into stock table 4. Return ‘SUCCESS’ 5. Catch exception 6. Display exception	findStockItem ((bigint(20) stockID): Stock createStockItem (Stock stock): String updateStockItem (Stock stock): String deleteStockItem ((bigint(20) stockID): bigint(20)

	updateStockItem(Stock stock) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to update stock item details in stock table 4. Return ‘UPDATE SUCCESS’ 5. Catch exception 6. Return ‘UPDATE FAILED’ deleteStockItem(bigint(20) stockID) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to delete stock item based on stockID from stock table 4. Return ‘DELETE SUCCESS’ 5. Catch exception 6. Return ‘DELETE FAILED’ END	
OrderDAO(SDD_PKG	BEGIN	findOrder ((bigint(20)

_302)	findOrder(bigint(20) orderID) 1. Create connection, statement, resultset variable 2. Establish connection with database 3. Execute SQL statement to get order details based on orderID and assign it to resultset variable 4. Set id, date, and status derived from resultset data to order object 5. Return order object 6. Catch exception 7. Display exception createOrder(Order order) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to insert new order details into order table 4. Return 'SUCCESS' 5. Catch exception 6. Display exception updateOrder(Order order) 1. Create connection,	stockID): Order createOrder(Order order): String updateOrder(Order order): String deleteOrder((bigint (20) orderID): bigint(20)

	prepared statement variable 2. Establish connection with database 3. Execute SQL statement to update order details in order table 4. Return ‘UPDATE SUCCESS’ 5. Catch exception 6. Return ‘UPDATE FAILED’ deleteOrder(bigint(20) orderID) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to delete order based on orderID from order table 4. Return ‘DELETE SUCCESS’ 5. Catch exception 6. Return ‘DELETE FAILED’ END	

DeliveryDAO (SDD_PKG_303)	BEGIN
	
	findDelivery(bigint(20) deliveryID)
		FindDelivery ((bigint(20) deliveryID): Order
		
		createDelivery

	1. Create connection, statement, resultset variable 2. Establish connection with database 3. Execute SQL statement to get delivery details based on deliveryID and assign it to resultset variable 4. Set id, orderID, deliveryDate, and status derived from resultset data to delivery object 5. Return delivery object 6. Catch exception 7. Display exception createDelivery(Delivery delivery) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to insert new delivery details into delivery table 4. Return ‘SUCCESS’ 5. Catch exception 6. Display exception updateDelivery(Delivery delivery)	(Delivery delivery): String updateDelivery (Delivery delivery): String deleteDelivery((big int(20) deliveryID): bigint(20)

	1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to update delivery details in delivery table 4. Return 'UPDATE SUCCESS' 5. Catch exception 6. Return 'UPDATE FAILED' deleteDelivery(bigint(20) deliveryID) 1. Create connection, prepared statement variable 2. Establish connection with database 3. Execute SQL statement to delete delivery based on deliveryID from delivery table 4. Return 'DELETE SUCCESS' 5. Catch exception 6. Return 'DELETE FAILED' END	

4.1.5.4 Class subordinates

Parent Table	Child Table	Foreign Key
users	deliveries	userID
	orders	
orders	orderDetails	orderID
stocks	orderDetails	stocksID
	stocksOutlet	
	stocksWarehouse	
deliveries	orders	deliveryID

4.1.5.5 Class Dependencies

The Design Class Diagram illustrates the interdependence between classes, regardless of whether they are in the same package. This diagram displays the relationships between classes, showing how classes in the view layer depend on classes in the domain and how classes in the domain depend on classes in the data access layer.

4.1.5.6 Interfaces

The control and data flow from one class to another can be seen in the Figure 2.2 Detailed Design Class Diagram, Decomposition Description 2.3.

4.1.5.7 Data

All data that is being used can be referred to in the Data Dictionary in Section 3, Data Dictionary 3.3.

5.0 Human Interface Design (Screens)

5.1 Overview of the User Interface

General Functionality of the System from the User's Perspective

- **System Overview:**
  - The system automates the stock ordering process, replacing the manual method. It allows outlet supervisors to request stock and HQ staff to manage and approve these requests. The system also includes tracking for stock delivery and inventory management.
- **User Interfaces:**
  - **Outlet Supervisor Interface:** This interface enables outlet supervisors to request stock, track delivery status and view past order history.
  - **HQ Interface:** This interface allows HQ staff to manage stock, approve requests, arrange deliveries and view past orders history.

How Users Will Use the System to Complete Expected Features and Feedback

1. **Login and Authentication:**

- **Process:** Users log in using their credentials.
- **Feedback:** Users gain access to the system with correct credentials or receive an error message if the credentials are incorrect.

2. **View Stock:**

- **Process:** Outlet supervisors navigate to the 'View Stock' section, input the required quantities and add them to the cart to submit the request.
- **Feedback:** The system displays a success message when the desired stocks are added to the cart. If there are errors (e.g., invalid quantities), the system displays an appropriate error message.

3. **Cart:**

- **Process:** Outlet supervisors navigate to the 'Cart' section and click on 'Checkout.'
- **Feedback:** A success message is displayed upon successful checkout.

4. **View Order Status:**

- **Process:** Users can see the status of their orders as pending, rejected or approved and can view the order details for each specific order.
- **Feedback:** The system displays the current status of orders and detailed order information.

5. **Track Delivery Status:**

- **Process:** Users can track the delivery status of their stock orders in real-time through the 'Track Delivery' section.
- **Feedback:** The system displays the current status of deliveries. If the system cannot find the delivery status, it displays nothing on the list of the page.

6. **Update Stock Information:**

- **Process:** HQ staff can update stock information, including adding new items, updating quantities or removing obsolete items.
- **Feedback:** The system confirms the update and displays the updated one.

7. **Manage Stocks:**

- **Process:** HQ admin can edit and delete stocks in inventory, with a confirmation prompt for deletions ("Are you sure?"). HQ Admins can also add new stocks by inserting details and clicking the 'Add Stock' button.
- **Feedback:** The system displays a success message upon successful addition, update or deletion of stocks. For deletions, a success message appears if confirmed and completed.

8. **Order Approval (HQ Admin):**

- **Process:** Admins can approve orders, changing their status to pending, rejected, or approved and view order details made by supervisors.
- **Feedback:** A success message appears when the order status is updated. When admins tick the order status and click on 'Arrange Delivery,' they are directed to the 'Arrange Delivery' page.

9. **Arrange Delivery (HQ Admin):**

- **Process:** Admins update the delivery status and date, and view order details on the 'Arrange Delivery' page.
- **Feedback:** A success message is displayed when the delivery status is updated.

10. **Delivery Tracking (Outlet Supervisor):**

- **Process:** Outlet Supervisors can track their orders on the 'Delivery Tracking' page.
- **Feedback:** When the status is updated to 'Order Received', a success message is displayed. Admins can see the order as successfully delivered.

11. **View Past Order History:**

- **Process:** Outlet Supervisors and HQ admin can view the history of past orders, including details such as order dates, quantities and delivery statuses.
- **Feedback:** The system provides a detailed view of past orders and shown nothing in the list if no records are found.

5.2 Screen Images

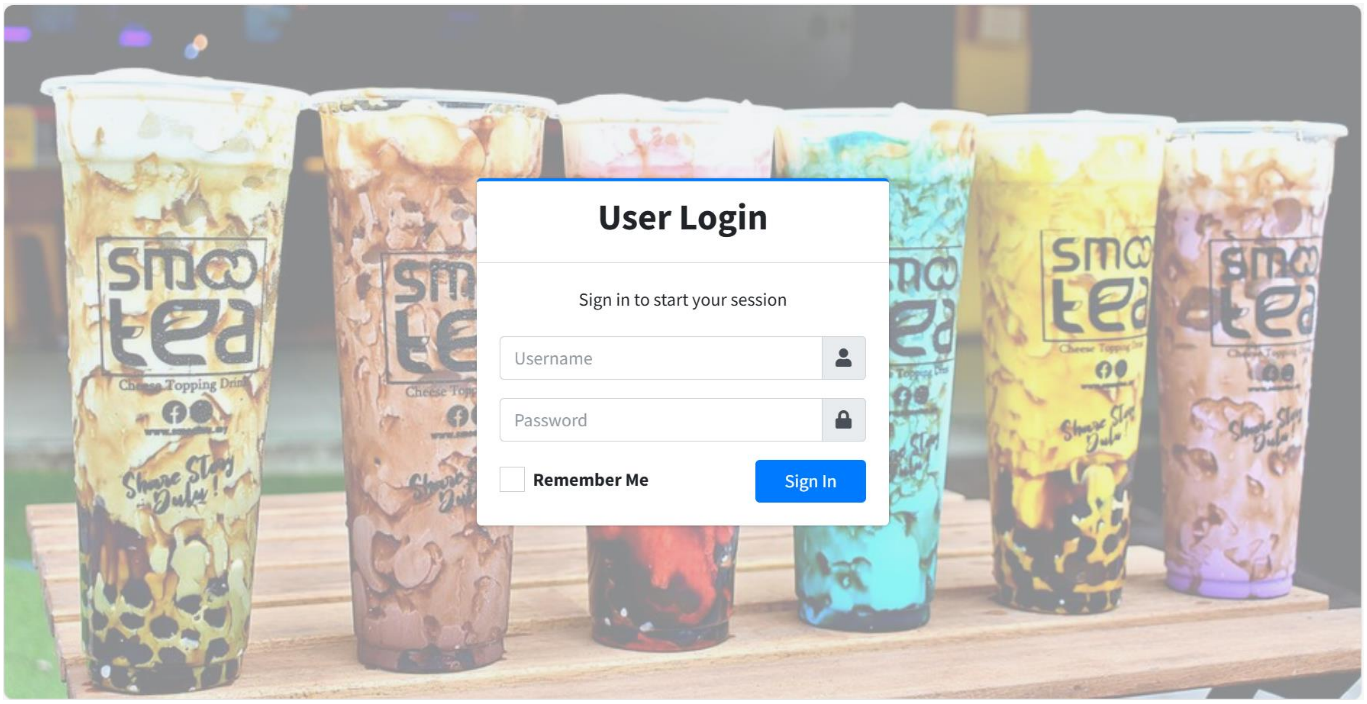


Figure 5.2.1: Login Page

1. *Admin Interface Screen*

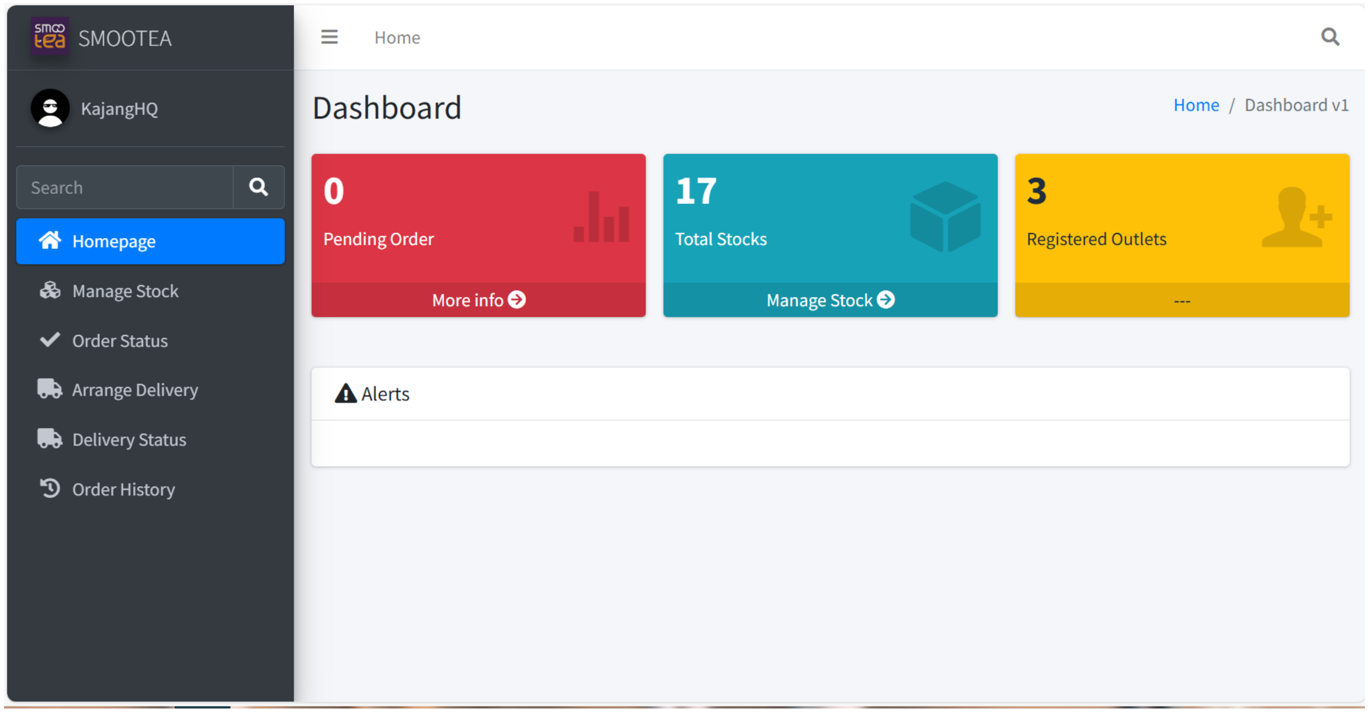
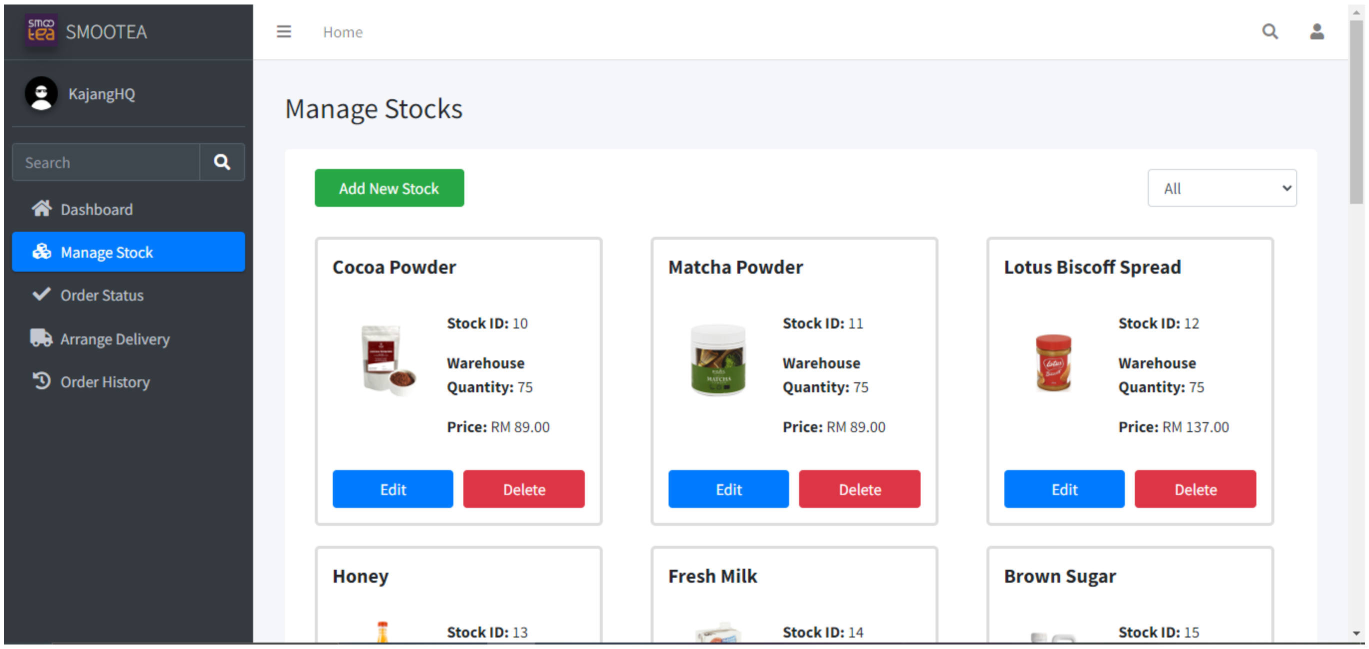


Figure 5.2.1.1: Dashboard Page



Cocoa Powder
Stock ID: 10
Warehouse Quantity: 75
Price: RM 89.00

Matcha Powder
Stock ID: 11
Warehouse Quantity: 75
Price: RM 89.00

Lotus Biscoff Spread
Stock ID: 12
Warehouse Quantity: 75
Price: RM 137.00

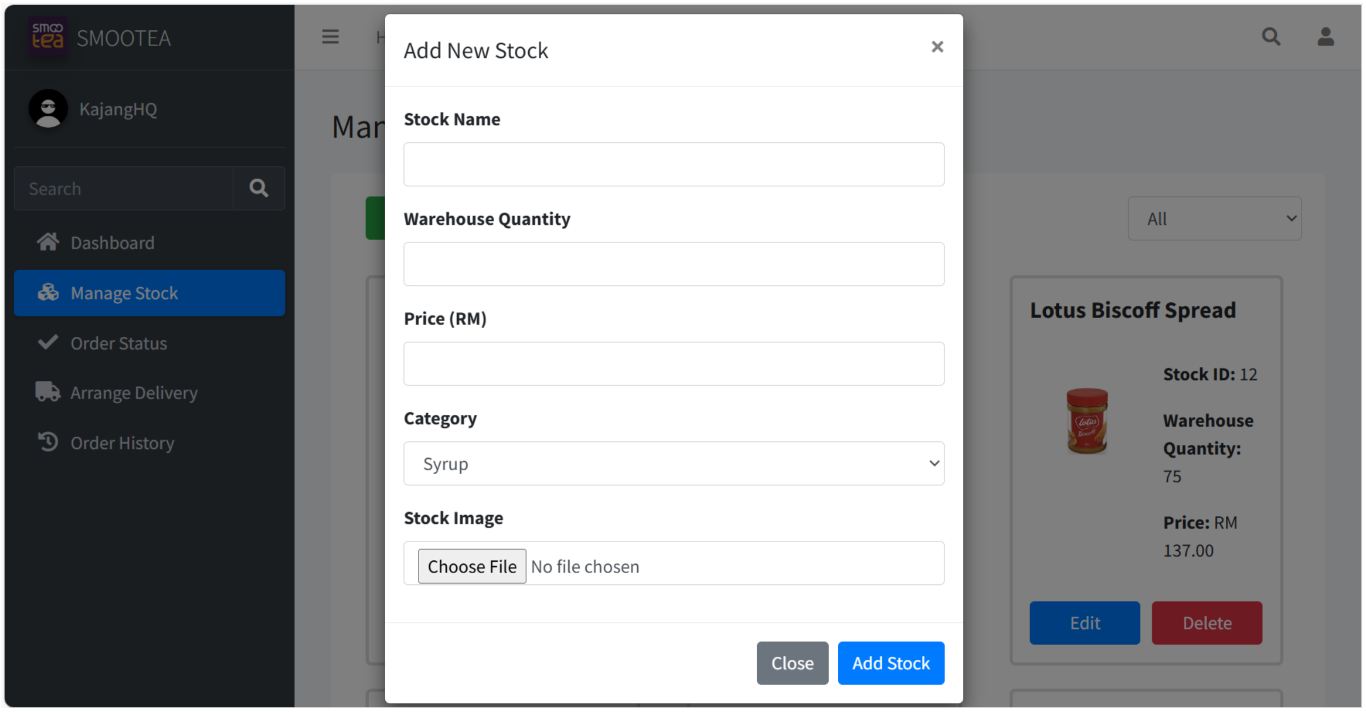
Honey
Stock ID: 13

Fresh Milk
Stock ID: 14

Brown Sugar
Stock ID: 15

Figure 5.2.1.2: Manage Stocks Page

- This screen allows admin to manage the stock items in the inventory. Admin can add new stock items, edit existing ones, or delete them. Each stock item card displays the item name, stock ID, warehouse quantity, and price. Buttons for editing and deleting are provided for each item.



Add New Stock

Stock Name

Warehouse Quantity

Price (RM)

Category: Syrup

Stock Image: Choose File No file chosen

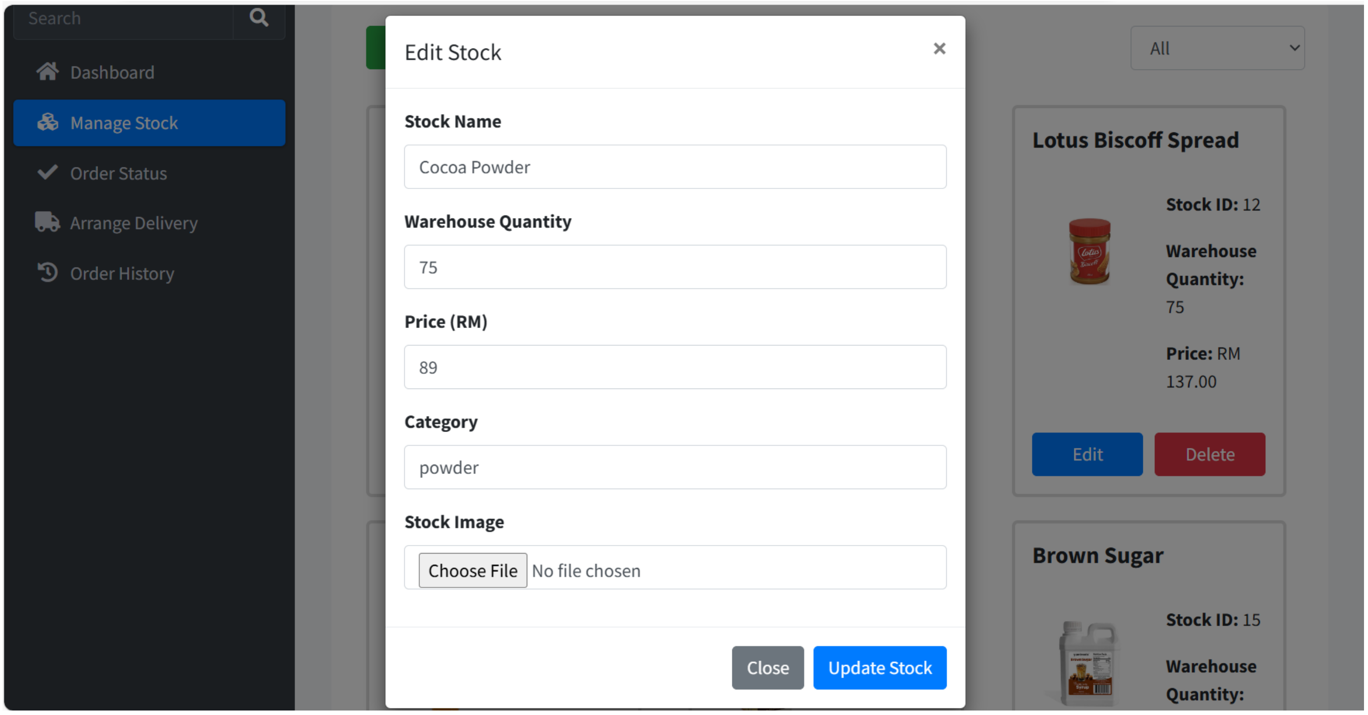
Lotus Biscoff Spread

Stock ID: 12

Warehouse Quantity: 75

Price: RM 137.00

Figure 5.2.1.3: Manage Stocks Page (Add New Stock)



Edit Stock

Stock Name: Cocoa Powder

Warehouse Quantity: 75

Price (RM): 89

Category: powder

Stock Image: Choose File No file chosen

Lotus Biscoff Spread

Stock ID: 12

Warehouse Quantity: 75

Price: RM 137.00

Brown Sugar

Stock ID: 15

Warehouse Quantity:

Figure 5.2.1.4: Manage Stocks Page (Edit Stock)

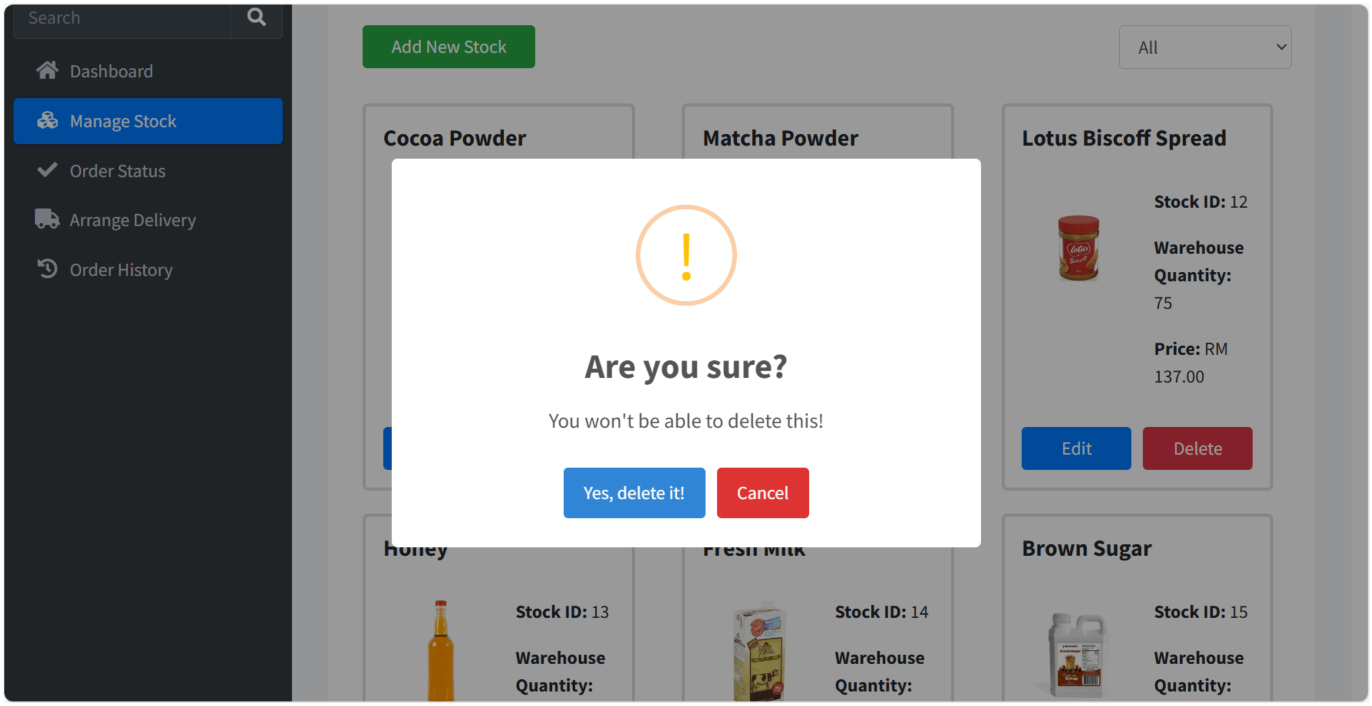
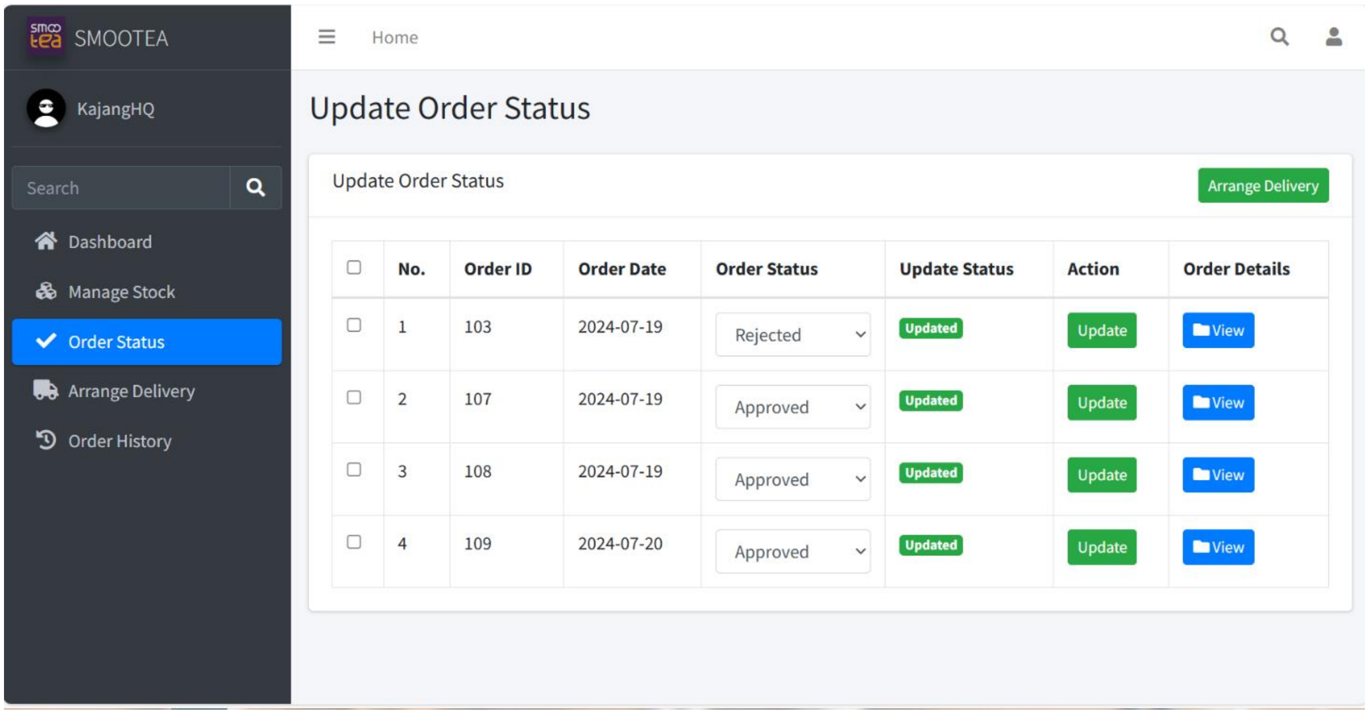


Figure 5.2.1.5: Manage Stocks Page (Delete Stock)



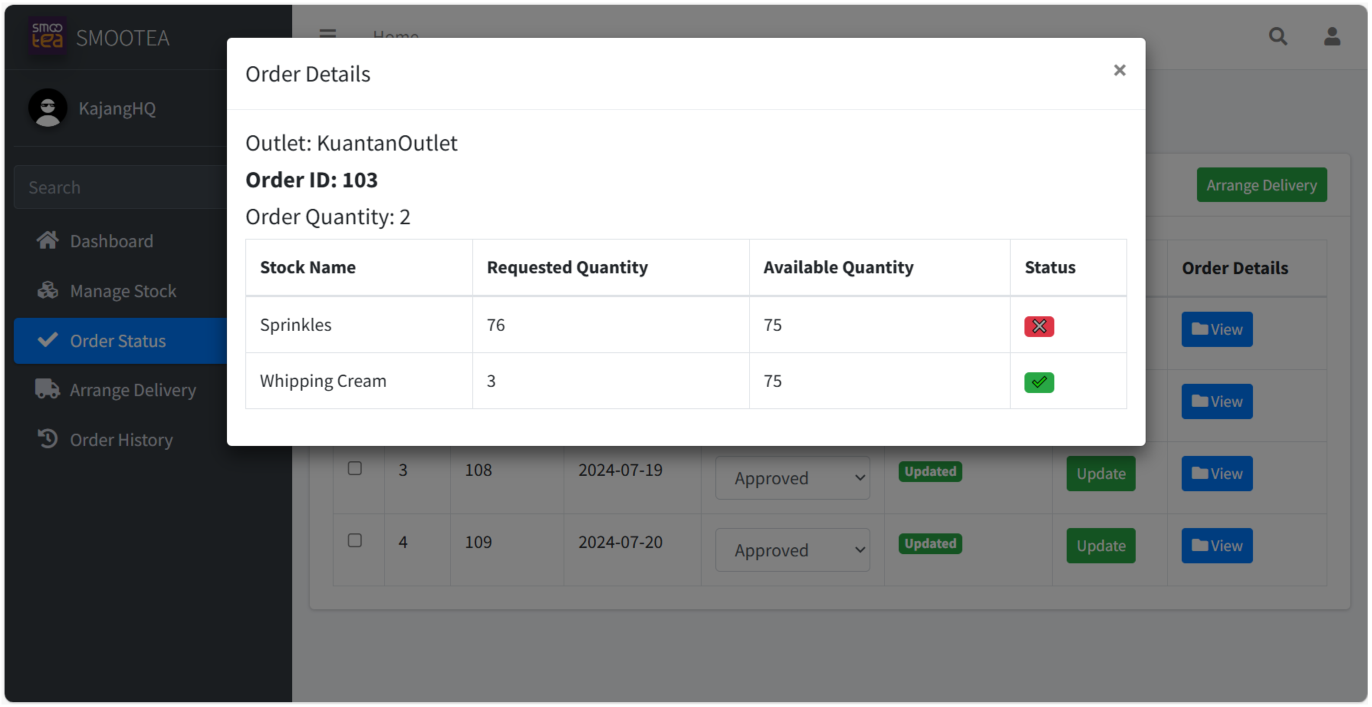
Update Order Status

Update Order Status

☐	No.	Order ID	Order Date	Order Status	Update Status	Action	Order Details
☐	1	103	2024-07-19	Rejected	Updated	Update	View
☐	2	107	2024-07-19	Approved	Updated	Update	View
☐	3	108	2024-07-19	Approved	Updated	Update	View
☐	4	109	2024-07-20	Approved	Updated	Update	View

Figure 5.2.1.6: Update Order Status Page

- HQ admin can update the status of outlet orders using this interface. With order list table showing columns for order ID, order date, order status, update status, actions, and order details. Using a dropdown menu, admin can modify the order status. Then, by clicking the "Update" button, the changes take effect. Clicking the 'View' button provides complete information on every order.



Order Details

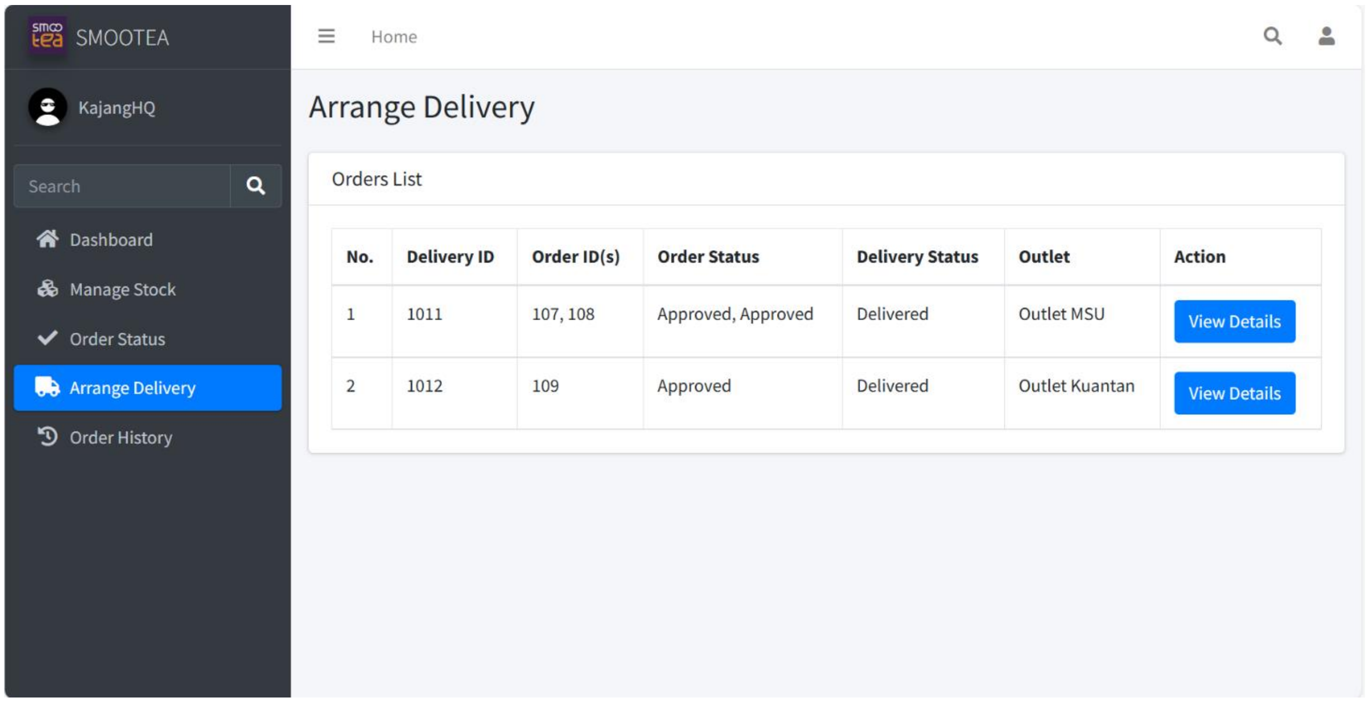
Outlet: KuantanOutlet

Order ID: 103

Order Quantity: 2

Stock Name	Requested Quantity	Available Quantity	Status
Sprinkles	76	75	✗
Whipping Cream	3	75	✓

Figure 5.2.1.7: Update Order Status Page (Display Order Details)



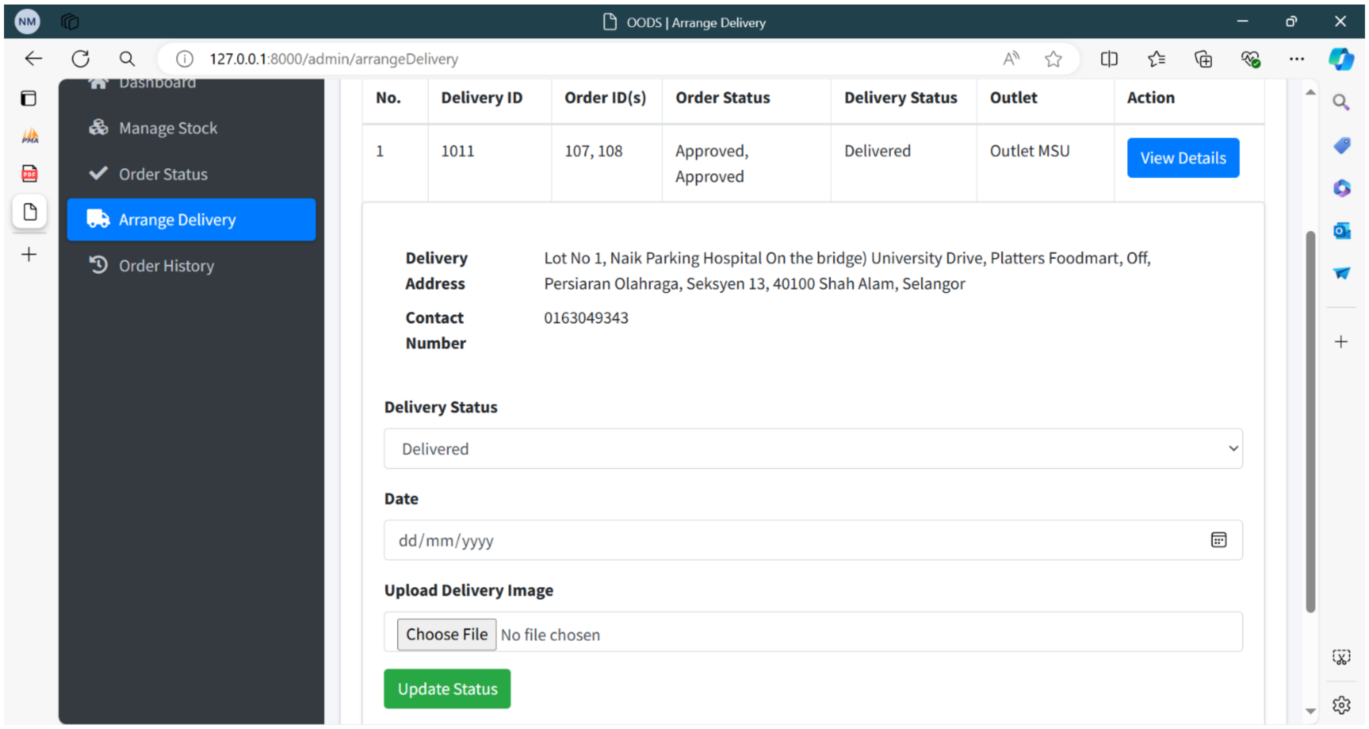
Arrange Delivery

Orders List

No.	Delivery ID	Order ID(s)	Order Status	Delivery Status	Outlet	Action
1	1011	107, 108	Approved, Approved	Delivered	Outlet MSU	View Details
2	1012	109	Approved	Delivered	Outlet Kuantan	View Details

Figure 5.2.1.8: Arrange Delivery Page

- This screen enables the HQ admin to arrange deliveries for approved orders. It displays a list of deliveries with columns for delivery ID, associated order IDs, order status, delivery status, outlet, and an action button. The 'View Details' button expands the view to show detailed delivery information.



No.	Delivery ID	Order ID(s)	Order Status	Delivery Status	Outlet	Action
1	1011	107, 108	Approved, Approved	Delivered	Outlet MSU	View Details

Delivery Address	Lot No 1, Naik Parking Hospital On the bridge) University Drive, Platters Foodmart, Off, Persiaran Olahraga, Seksyen 13, 40100 Shah Alam, Selangor
Contact Number	0163049343

Delivery Status: Delivered

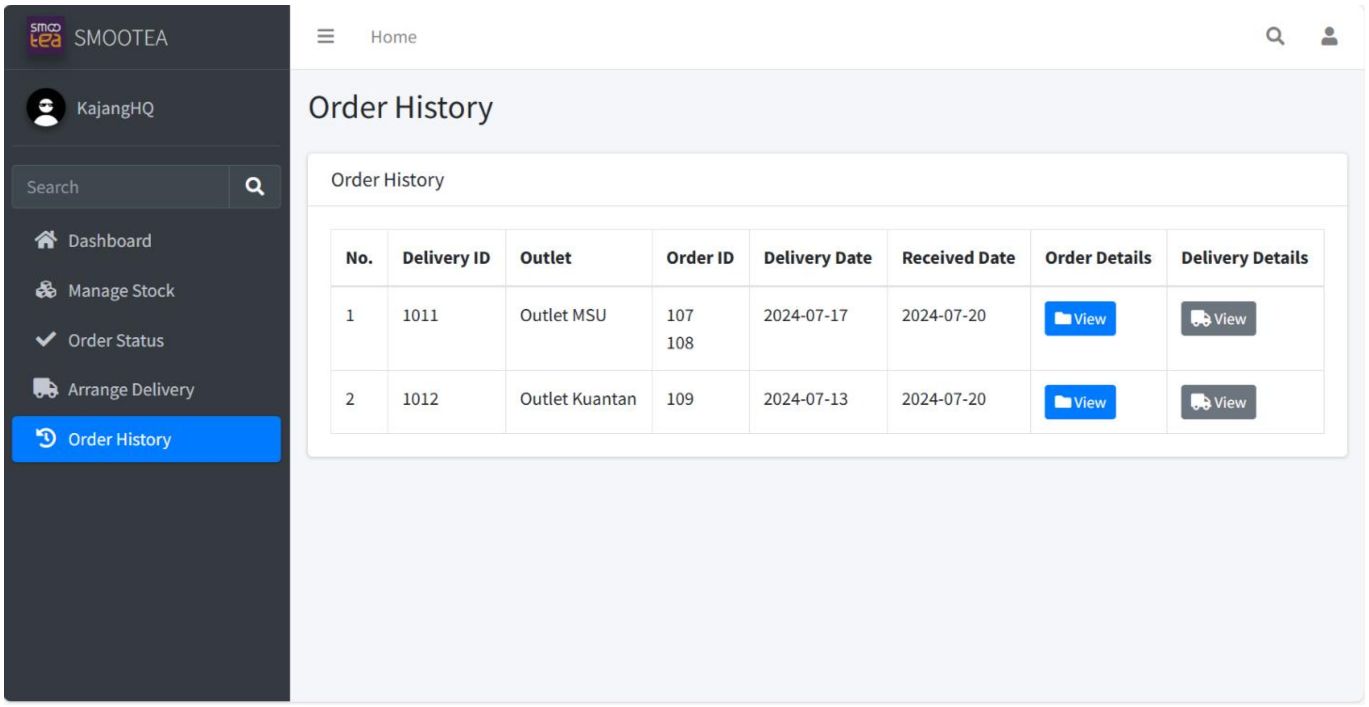
Date: dd/mm/yyyy

Upload Delivery Image: Choose File No file chosen

Update Status

Figure 5.2.1.9: Arrange Delivery Page (Display Delivery Details Expandable Table)

- This expanded view provides detailed information about a specific delivery. It includes the delivery address, contact number, delivery status dropdown, date picker, and an option to upload a delivery image. The 'Update Status' button saves the updated delivery details.



Order History

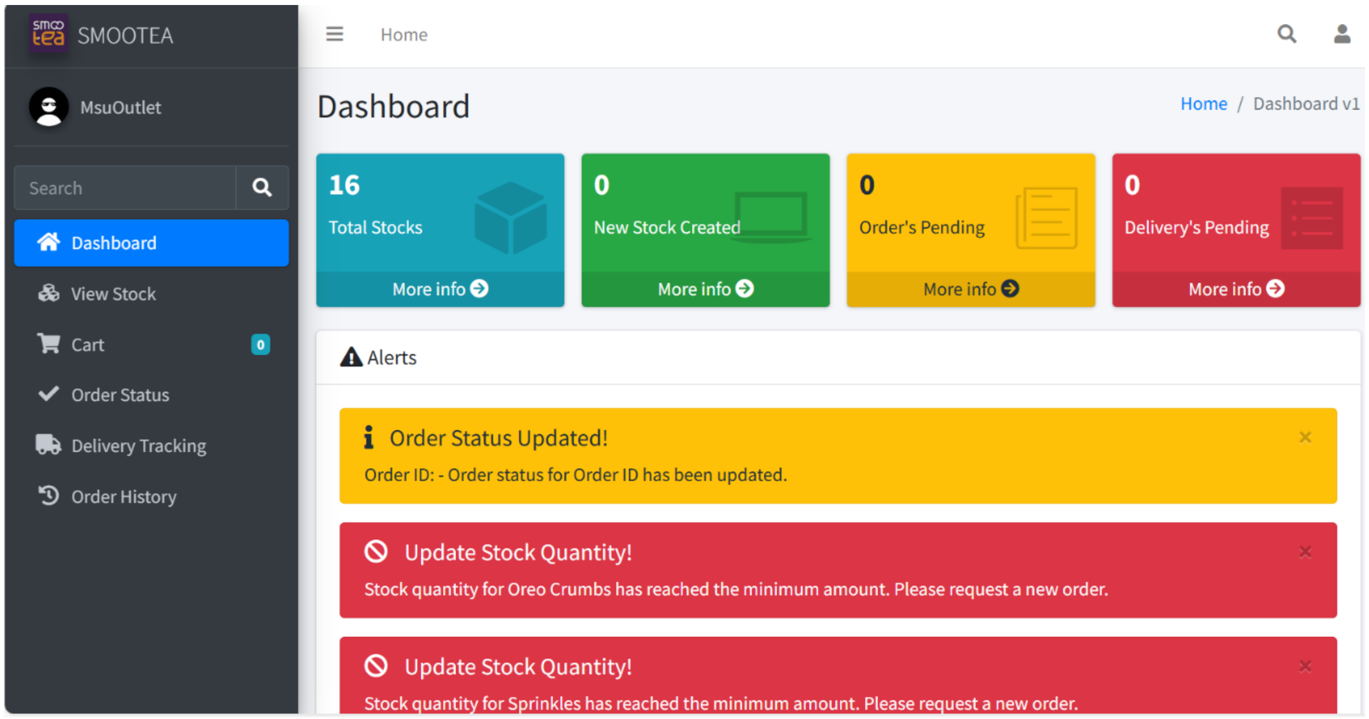
Order History

No.	Delivery ID	Outlet	Order ID	Delivery Date	Received Date	Order Details	Delivery Details
1	1011	Outlet MSU	107 108	2024-07-17	2024-07-20	View	View
2	1012	Outlet Kuantan	109	2024-07-13	2024-07-20	View	View

Figure 5.2.1.5: Order History Page

- This screen allows both the HQ admin and outlet supervisor to view past orders. It lists deliveries with details such as delivery ID, outlet, order ID, delivery date, received date, and actions to view order and delivery details.

2. Outlet Supervisor Interface Screen

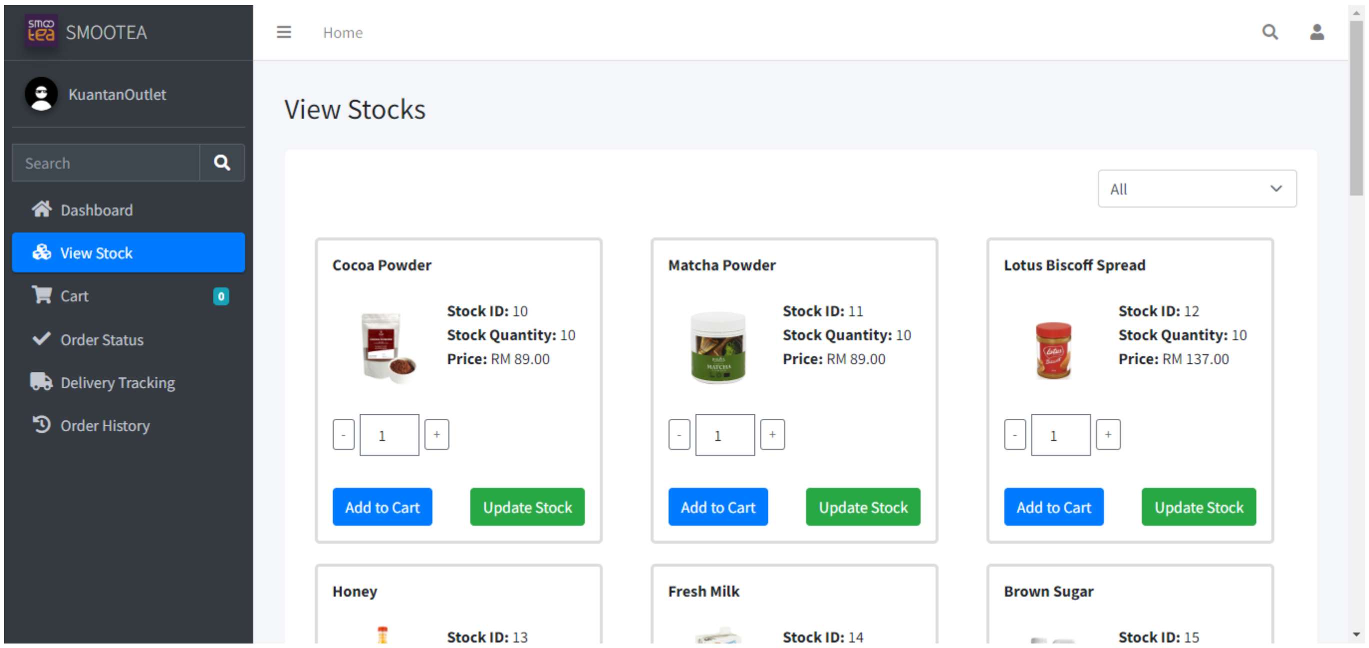


Order Status Updated!
Order ID: - Order status for Order ID has been updated.

Update Stock Quantity!
Stock quantity for Oreo Crumbs has reached the minimum amount. Please request a new order.

Update Stock Quantity!
Stock quantity for Sprinkles has reached the minimum amount. Please request a new order.

Figure 5.2.2.1: Dashboard for outlet supervisor page



Cocoa Powder
Stock ID: 10
Stock Quantity: 10
Price: RM 89.00

Matcha Powder
Stock ID: 11
Stock Quantity: 10
Price: RM 89.00

Lotus Biscoff Spread
Stock ID: 12
Stock Quantity: 10
Price: RM 137.00

Honey
Stock ID: 13

Fresh Milk
Stock ID: 14

Brown Sugar
Stock ID: 15

Figure 5.2.2.2: View Stocks Page

- This screen allows the outlet supervisor to view and manage stock levels. It displays stock items with details such as stock ID, name, quantity, and price. The supervisor can update stock quantities, add items to the cart for requesting new stock, and view detailed information about each item.

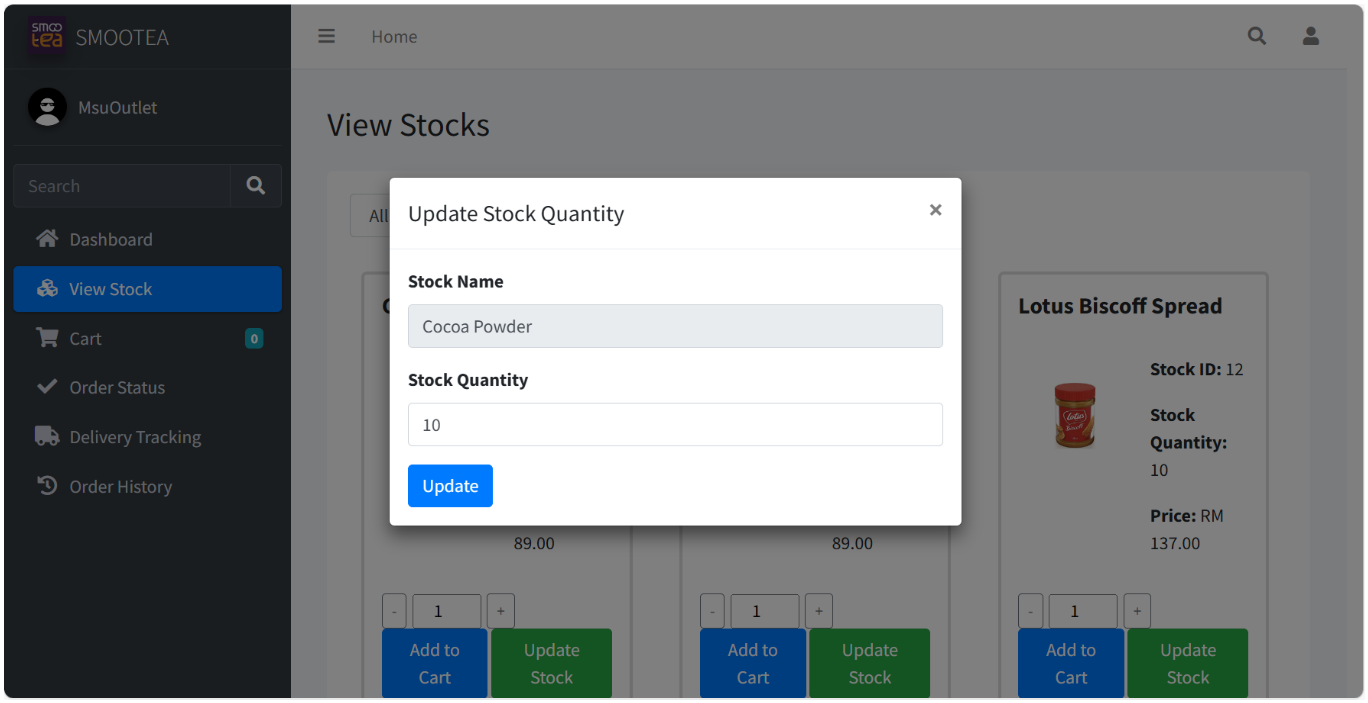
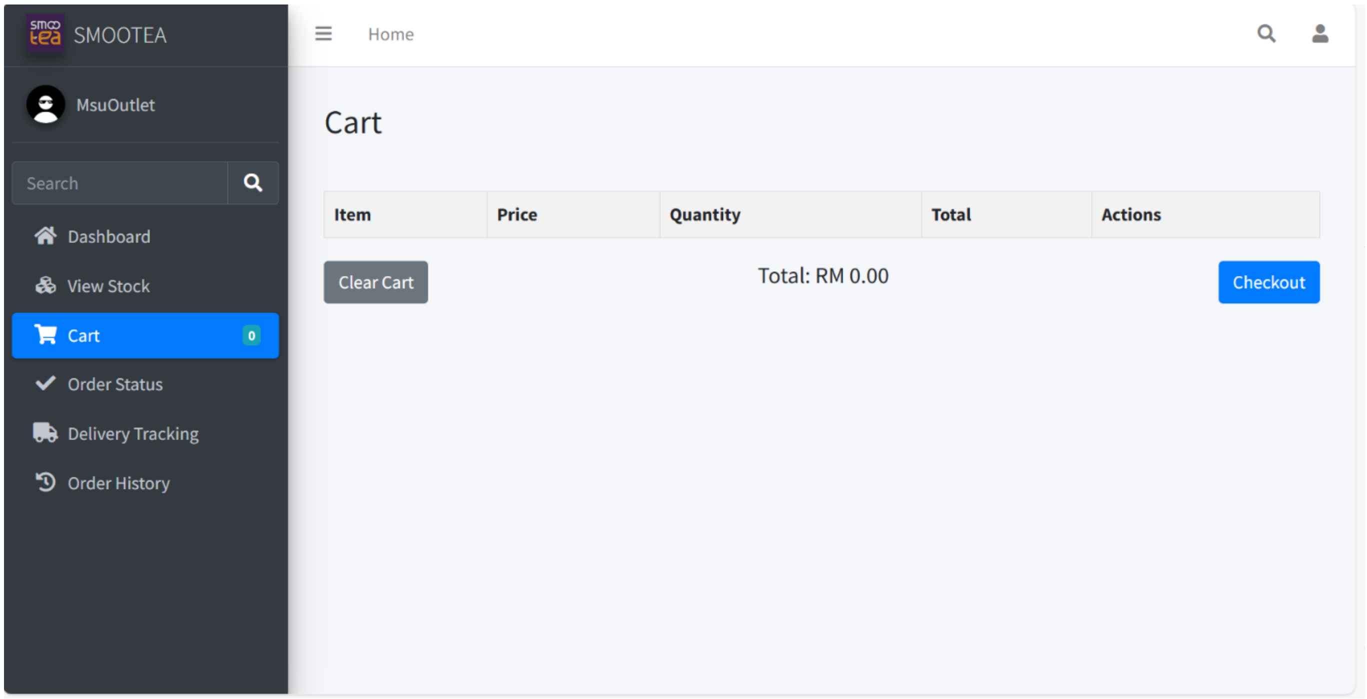


Figure 5.2.2.3: View Stocks Page (Update Stock for Outlet)

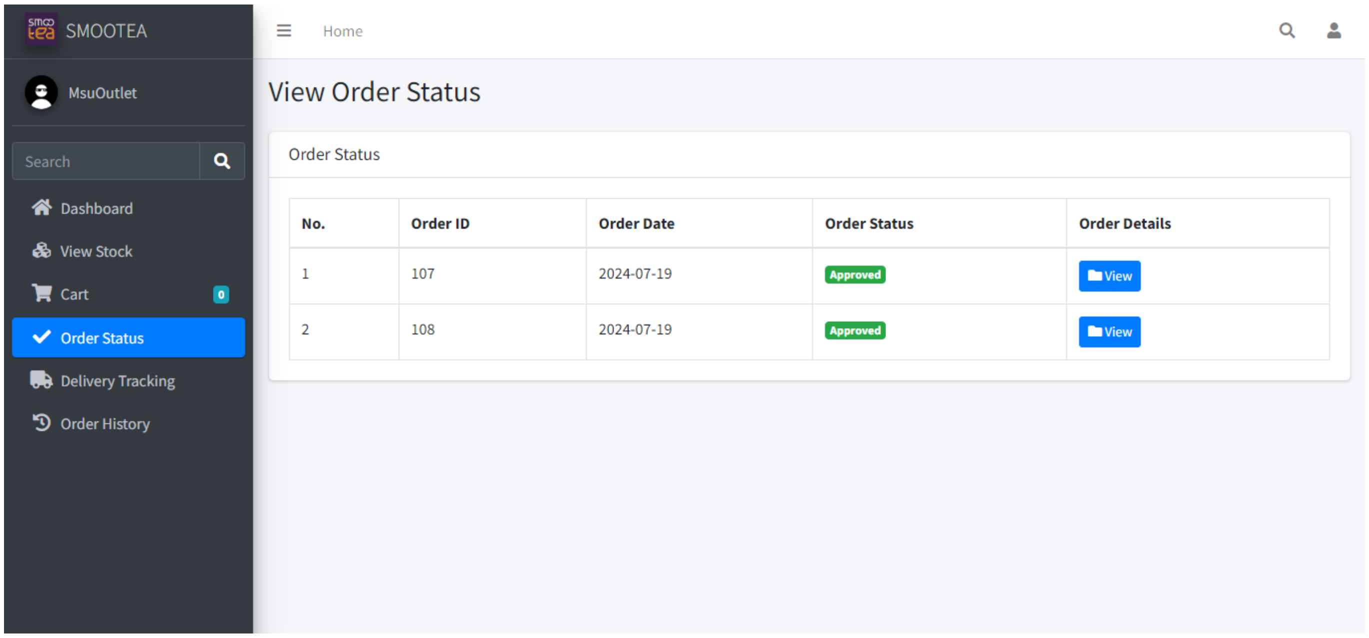


Item	Price	Quantity	Total	Actions

Total: RM 0.00

Figure 5.2.2.4: Cart Page

- This screen allows the outlet supervisor to review items added to the cart before requesting new stock. It displays the item name, price, quantity, and total cost. The supervisor can update the quantity, remove items, clear the cart, and proceed to checkout.



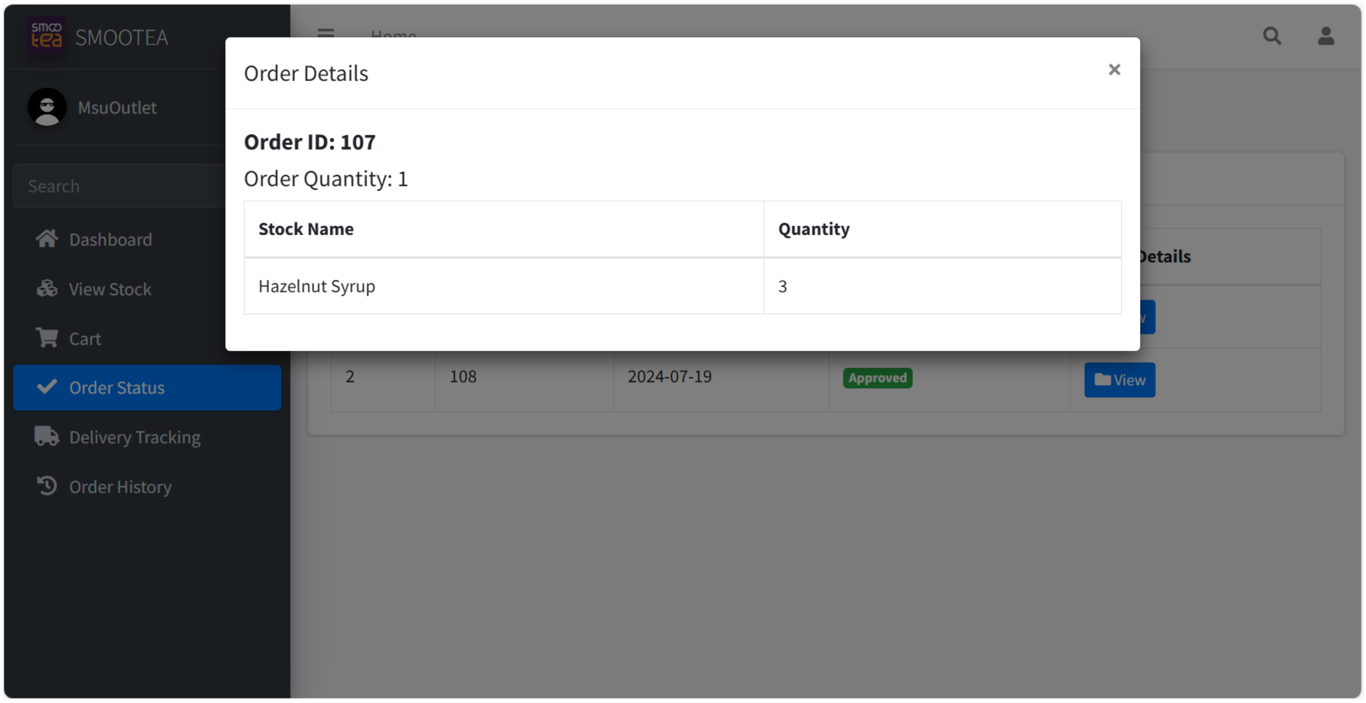
View Order Status

Order Status

No.	Order ID	Order Date	Order Status	Order Details
1	107	2024-07-19	Approved	View
2	108	2024-07-19	Approved	View

Figure 5.2.2.5: Order Status Page

- This screen allows the outlet supervisor to view the status of orders placed. It displays order ID, date, status, and an action button to view detailed order information.



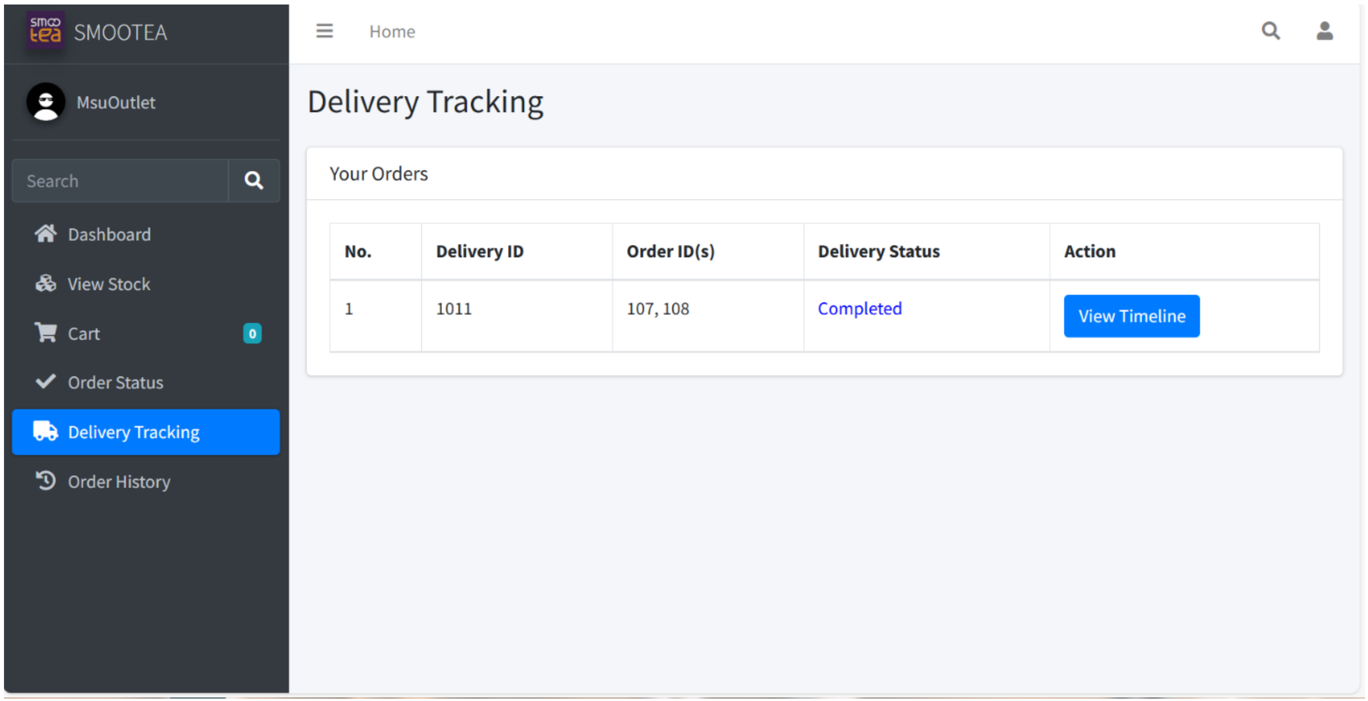
Order Details

Order ID: 107

Order Quantity: 1

Stock Name	Quantity
Hazelnut Syrup	3

Figure 5.2.2.6: Order Status Page (Display Order Details)



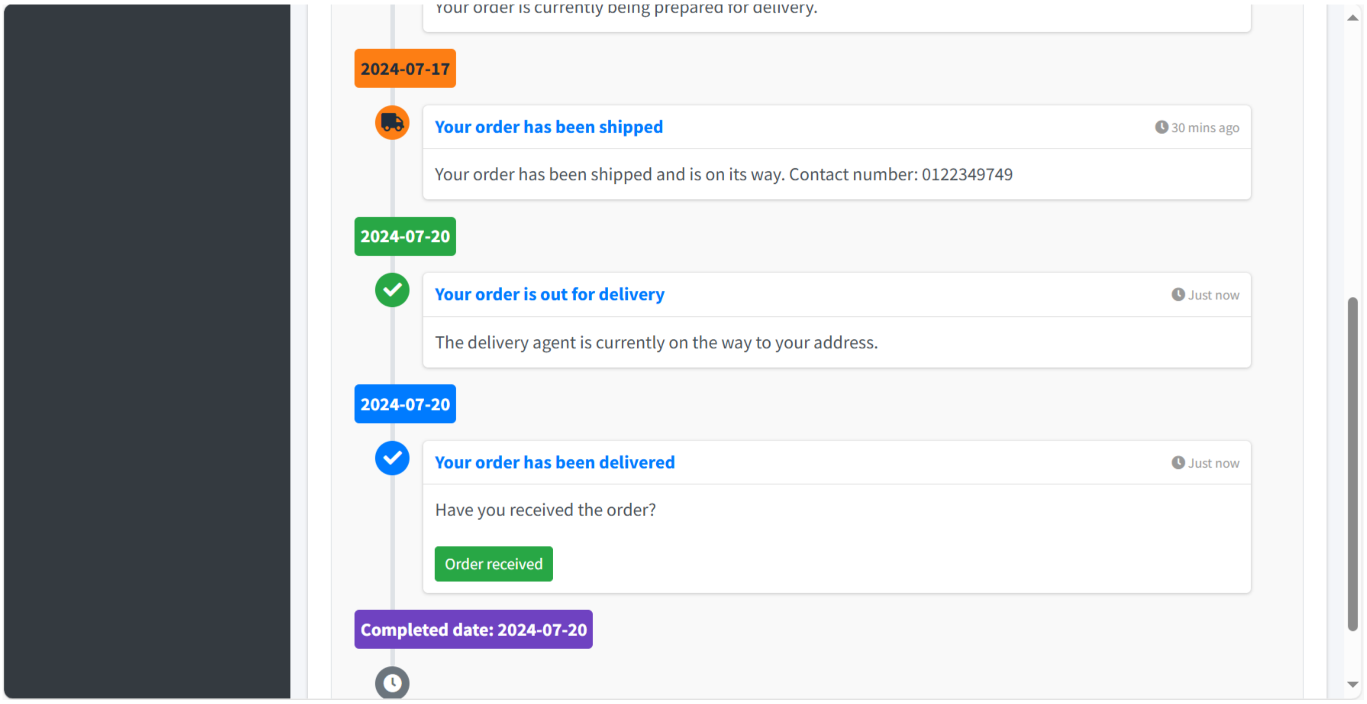
Delivery Tracking

Your Orders

No.	Delivery ID	Order ID(s)	Delivery Status	Action
1	1011	107, 108	Completed	View Timeline

Figure 5.2.2.4: Delivery Tracking Page

- This screen allows the outlet supervisor to track the delivery status of orders. It displays delivery ID, associated order IDs, delivery status, and an action button to view the delivery timeline.



Your order is currently being prepared for delivery.

2024-07-17

Your order has been shipped — 30 mins ago

Your order has been shipped and is on its way. Contact number: 0122349749

2024-07-20

Your order is out for delivery — Just now

The delivery agent is currently on the way to your address.

2024-07-20

Your order has been delivered — Just now

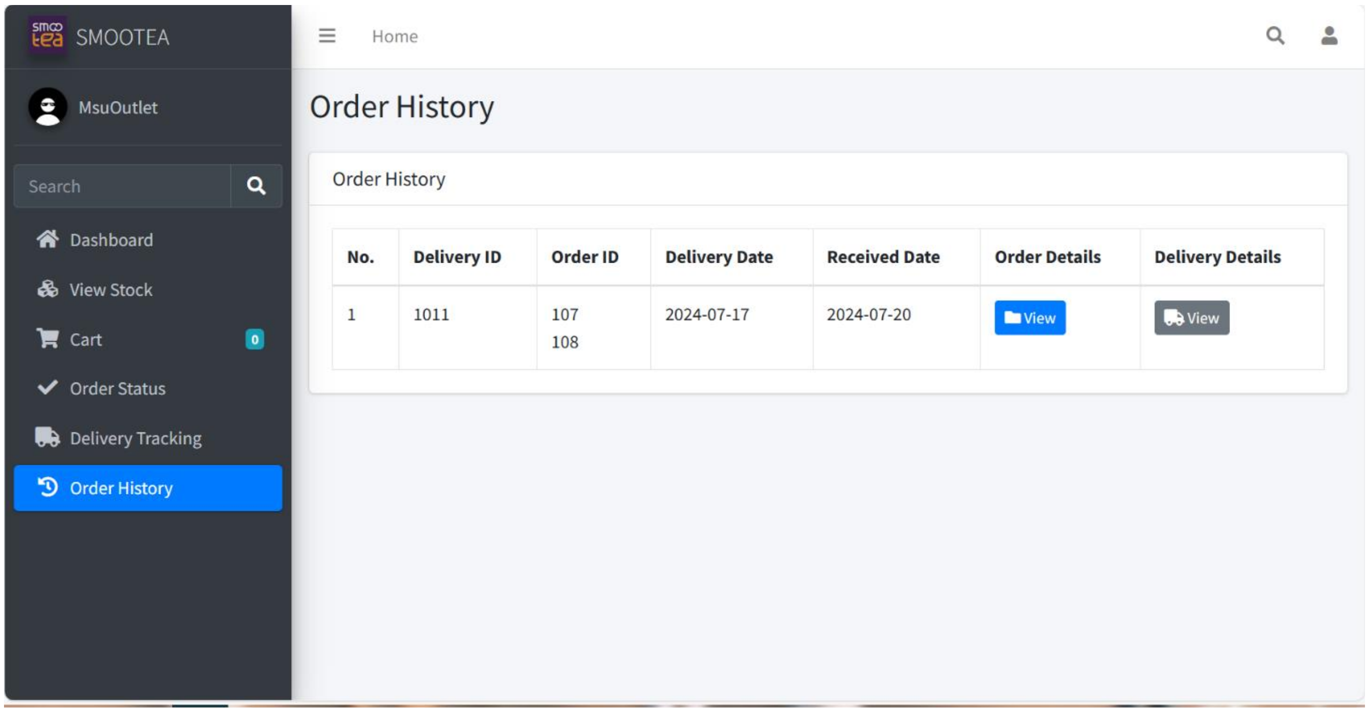
Have you received the order?

Order received

Completed date: 2024-07-20

Figure 5.2.2.5: Delivery Tracking Page (Display details delivery timeline)

- This expanded view shows the detailed timeline of a delivery. It includes various statuses such as preparing, shipped, out for delivery, delivered, and order received, along with their corresponding dates.



Order History

Order History

No.	Delivery ID	Order ID	Delivery Date	Received Date	Order Details	Delivery Details
1	1011	107 108	2024-07-17	2024-07-20	View	View

Figure 5.2.2.6: Order History Page

- This screen allows the outlet supervisor to view the history of orders. It lists delivery ID, associated order IDs, delivery date, received date, and actions to view detailed order and delivery information.

5.3 Screen Objects and Actions

1. HQ Admin Pages

Figure 5.2.1.1 – Manage Stock Page

- Add New Stock Button: Clicking this button (located at the top of Figure 1) opens an Add New Stock Screen pop-up
- Edit Button (on stock item card): Clicking the blue 'Edit' button on any stock item card (shown in Figure 1) opens the Edit Stock Screen pop-up screen to edit the selected stock item.
- Delete Button (on stock item card): Clicking the red 'Delete' button on any stock item card (shown in Figure 1) opens the Delete Confirmation Dialog alert to confirm deletion of the selected stock item.
- Stock Item Cards: Each card (highlighted in Figure 1) displays the details of a stock item including name, ID, quantity, and price. Users can view these details at a glance.

Figure 5.2.1.2 – Update Order Status Page

- Order Table: Displays a list of orders with columns for No., Order ID, Order Date, Order Status, Update Status, Action, and Order Details.
  - Order Status Dropdown: Allows changing the status of an order.
  - Update Button: Applies the new status.
  - Order Details Button: Views detailed information about an order.
  - Arrange Delivery Button: Navigates to the delivery arrangement screen.

Figure 5.2.1.3 – Arrange Delivery Page

- Order List Table: Displays a list of deliveries with columns for No., Delivery ID, Order ID(s), Order Status, Delivery Status, Outlet, and Action.
  - View Details Button: Expands to show detailed delivery information (Figure 5.2.1.4).

Figure 5.2.1.4 - Arrange Delivery Page (display delivery details expandable table)

- Delivery Details Section: Displays detailed information about the delivery address, contact number, and status.
- Delivery Status Dropdown: Allows changing the delivery status.
- Date Picker: Allows selecting the delivery date.
- Update Status Button: Saves the updated delivery status and details.

Figure 5.2.1.5 – Order History page

- Order List Table: Displays past orders with columns for No., Delivery ID, Outlet, Order ID, Delivery Date, Received Date, and Action.
  - View Order Details Button: Opens the Order Details dialog
  - View Delivery Details Button: Opens the Delivery Details dialog

2. Outlet Supervisor Pages

Figure 5.2.2.1 - View Stocks Page

- Stock Item Cards: Display details of each stock item, including stock ID, name, quantity, and price.
    - Quantity Adjuster: Allows adjusting the quantity of the stock item to be added to the cart.
    - Add to Cart Button: Adds the selected quantity of the stock item to the cart for requesting new stock.
    - Update Stock Button: Updates the current quantity of the stock item in the system.

Figure 5.2.2.2 - Cart Page

- Item List: Displays items added to the cart with columns for item name, price, quantity, and total cost.
    - Quantity Field: Allows updating the quantity of each item in the cart.
    - Remove Button: Removes an item from the cart.
    - Clear Cart Button: Empties the entire cart.
    - Checkout Button: Proceeds to request the items in the cart from the HQ.

Figure 5.2.2.3 – Order Status Page

- Order List Table: Displays a list of orders with columns for No., Order ID, Order Date, Order Status, and Order Details.
    - Order Details Button: Views detailed information about each order.

Figure 5.2.2.4 – Delivery Tracking Page

- Delivery List Table: Displays a list of deliveries with columns for No., Delivery ID, Order ID(s), Delivery Status, and Action.
    - View Timeline Button: Expands to show the detailed delivery timeline (Figure 5.2.2.4).

Figure 5.2.2.5 - Delivery Tracking Page (Display details delivery timeline)

- Timeline Section: Displays the detailed timeline of a delivery, including various statuses (preparing, shipped, out for delivery, delivered, order received) and corresponding dates.

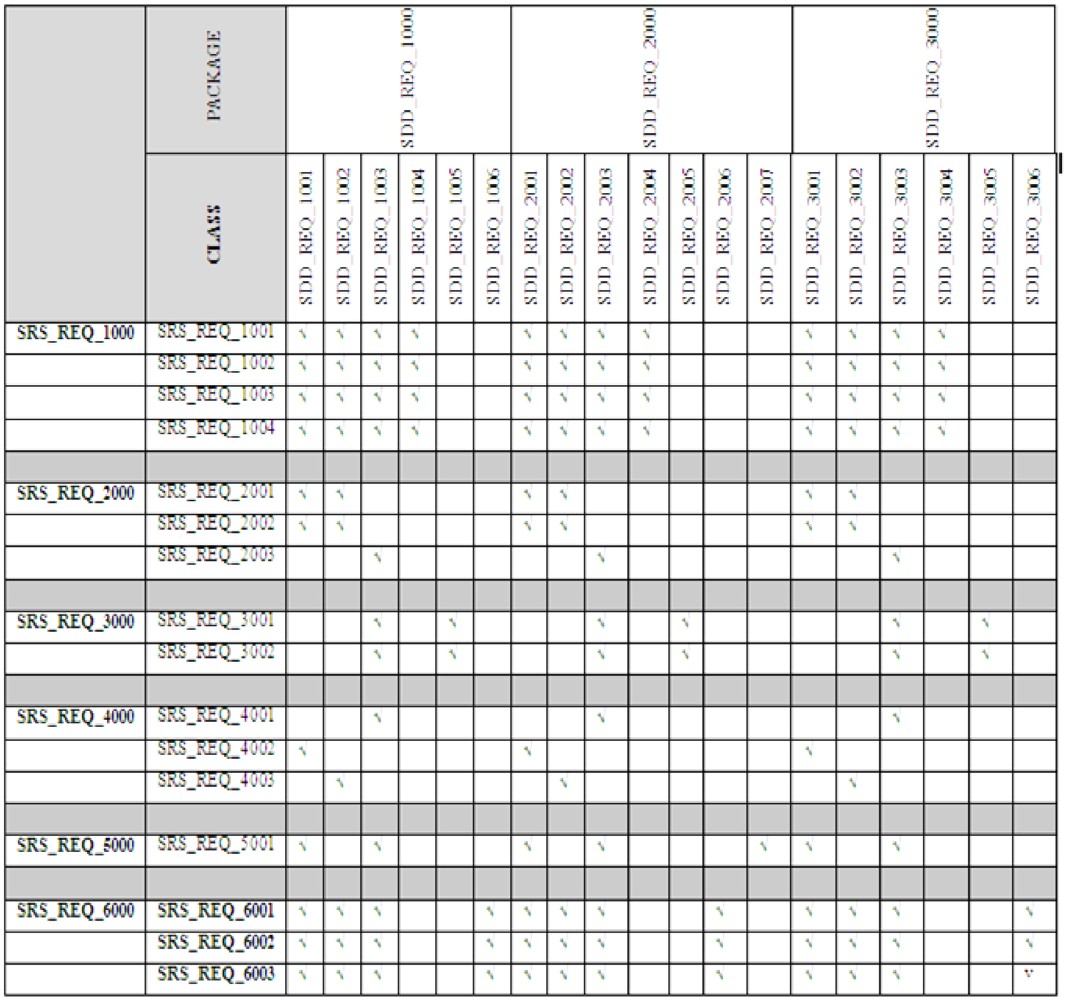
Figure 5.2.2.6 – Order History Page

- Order List Table: Displays past orders with columns for No., Delivery ID, Order ID, Delivery Date, Received Date, and Action.
    - View Order Details Button: Opens the Order Details dialog (Figure 9).
    - View Delivery Details Button: Opens the Delivery Details dialog

5.4 Report Formats

<Not Applicable>

6.0 Traceability Requirements Matrix



	PACKAGE	SDD_REQ_1000						SDD_REQ_2000							SDD_REQ_3000					
	CLASS	SDD_REQ_1001	SDD_REQ_1002	SDD_REQ_1003	SDD_REQ_1004	SDD_REQ_1005	SDD_REQ_1006	SDD_REQ_2001	SDD_REQ_2002	SDD_REQ_2003	SDD_REQ_2004	SDD_REQ_2005	SDD_REQ_2006	SDD_REQ_2007	SDD_REQ_3001	SDD_REQ_3002	SDD_REQ_3003	SDD_REQ_3004	SDD_REQ_3005	SDD_REQ_3006
SRS_REQ_1000	SRS_REQ_1001	✓	✓	✓	✓			✓	✓	✓	✓				✓	✓	✓	✓		
	SRS_REQ_1002	✓	✓	✓	✓			✓	✓	✓	✓				✓	✓	✓	✓		
	SRS_REQ_1003	✓	✓	✓	✓			✓	✓	✓	✓				✓	✓	✓	✓		
	SRS_REQ_1004	✓	✓	✓	✓			✓	✓	✓	✓				✓	✓	✓	✓		
SRS_REQ_2000	SRS_REQ_2001	✓	✓					✓	✓						✓	✓				
	SRS_REQ_2002	✓	✓					✓	✓						✓	✓				
	SRS_REQ_2003			✓						✓							✓			
SRS_REQ_3000	SRS_REQ_3001			✓		✓				✓		✓					✓		✓	
	SRS_REQ_3002			✓		✓				✓		✓					✓		✓	
SRS_REQ_4000	SRS_REQ_4001			✓						✓							✓			
	SRS_REQ_4002	✓						✓							✓					
	SRS_REQ_4003		✓						✓							✓				
SRS_REQ_5000	SRS_REQ_5001	✓		✓				✓		✓				✓	✓		✓			
SRS_REQ_6000	SRS_REQ_6001	✓	✓	✓			✓	✓	✓	✓			✓		✓	✓	✓			✓
	SRS_REQ_6002	✓	✓	✓			✓	✓	✓	✓			✓		✓	✓	✓			✓
	SRS_REQ_6003	✓	✓	✓			✓	✓	✓	✓			✓		✓	✓	✓			✓

7.0 Resources Estimates

Operating Environment

Hardware

Item	Description
Central Processing Unit (CPU)	AMD Athlon Silver 3050U with Radeon Graphics 2.30 GHz
Graphic Processing Unit (GPU)	AMD Radeon(TM) Graphics
Solid State Driver (SSD)	WDC PC SN530 SDBPNPZ-256G-1002
Random Access Memory (RAM)	4GB DDR4 on board
Power Supply	ø4.0, 45W AC Adapter, Output: 19V DC, 2.37A, 45W, Input: 100~240V AC 50/60Hz universal
Keyboard	PC/AT Enhanced PS/2 Keyboard (101/102-Key)
Mouse	CLIPtec Wireless Silent Optical Mouse DWM237

Table 7.1 Hardware Description

Operating System

Item	Description
Window 10 and above	Version 2202 and above

Table 7.2 Operating System Description

Software

Item	Description
Visual Studio Code Builder	Version 1.90
MySQL WorkBench in database	Version 8.036
Internet Browser	Google Chrome
Antivirus/Firewall	Avast Secure Antivirus

Table 7.3 Software Description

Design And Implementation Constraints

Types of Constraints	Constraints
Network	▪ Inconsistent internet connection ▪ Inadequate bandwidth due to large data transfers.

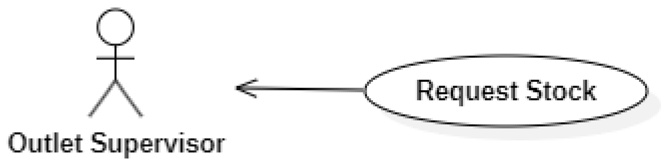
Software	▪ Inconsistent data formats, APIs or security standards needed specific skills or alternatives. ▪ Various kinds of operating system support for specific technologies or tools used by the system. ▪ Recurring license costs associated with the usage of third-party software
Hardware	▪ Limited processing speed or memory on devices at outlets could affect the system's performance, responsiveness, and its ability to deal with complex tasks. ▪ Devices with outdated or restricted input/output options can have an influence on the user experience and efficiency of data entry. ▪ Devices with a short battery life or high heat production require extra battery management or cooling solutions.
Tools and Database	▪ Complex queries or significant reporting requirements might demand a more advanced database solution ▪ Restricted scalability or usefulness of the selected database technology.

Computer Resources Required for Operating the Software

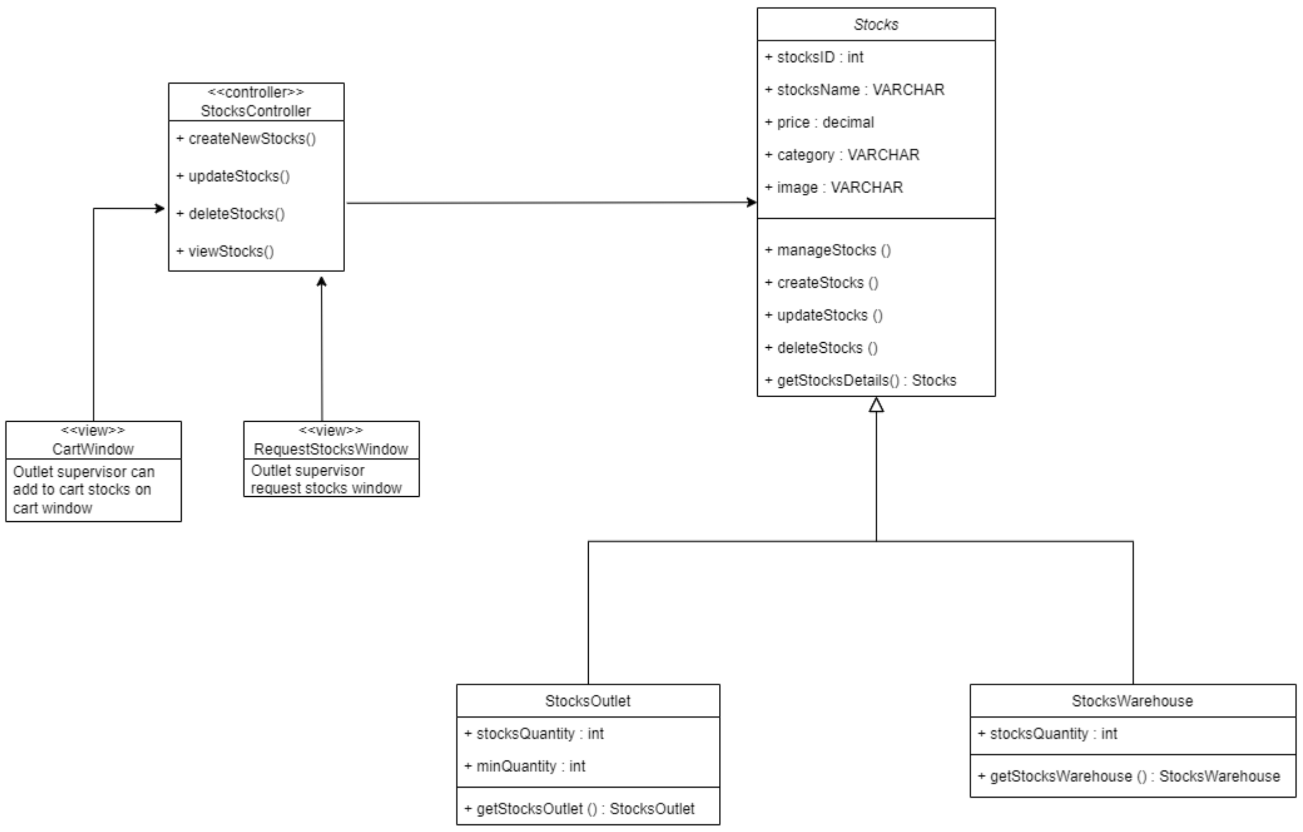
Category	Minimum Requirements	Recommended Requirements
Server Hardware		
Processor	AMD Athlon Silver 3050U with Radeon Graphics 2.30 GHz	AMD Athlon Silver 3050U with Radeon Graphics 2.30 GHz or higher
Memory (RAM)	16 GB	32 GB or higher
Storage	500 GB SSD	1 TB SSD with RAID configuration for redundancy
Network	1 Gbps Ethernet	10 Gbps Ethernet for high availability and faster data transfer
Client Hardware		
Processor	Intel Core i3 or equivalent	Intel Core i5 or higher
Memory (RAM)	4 GB	8 GB or higher
Storage	100 GB HDD or SSD	256 GB SSD
Display	1366 x 768 resolution	1920 x 1080 resolution or higher
Network	10 Mbps internet connection	100 Mbps or higher for seamless operation
Server Software		
Operating System	Windows Server 2016	Windows Server 2019
Database Management System (DBMS)	MySQL 5.7	MySQL 8.0
Web Server	Apache 2.4	Apache 2.4 or higher
Application Server	Apache 2.4	Apache 2.4 or higher
Client Software		
Operating System	Windows 10 or macOS Mojave	Windows 10 (latest updates) or macOS Catalina
Web Browser	Google Chrome 79 or Firefox 72	Latest version of Google Chrome, Firefox or Microsoft Edge
Other Software	PDF Reader (e.g., Adobe Acrobat Reader)	Office Suite (e.g., Microsoft Office 2016 or later)
Network Requirements		
Internal Network	1 Gbps LAN for internal communications	10 Gbps LAN for high-speed data transfer
Internet Connection	10 Mbps for basic operations	100 Mbps or higher for optimal performance, especially for remote access and cloud-based operations

8.0 Appendices

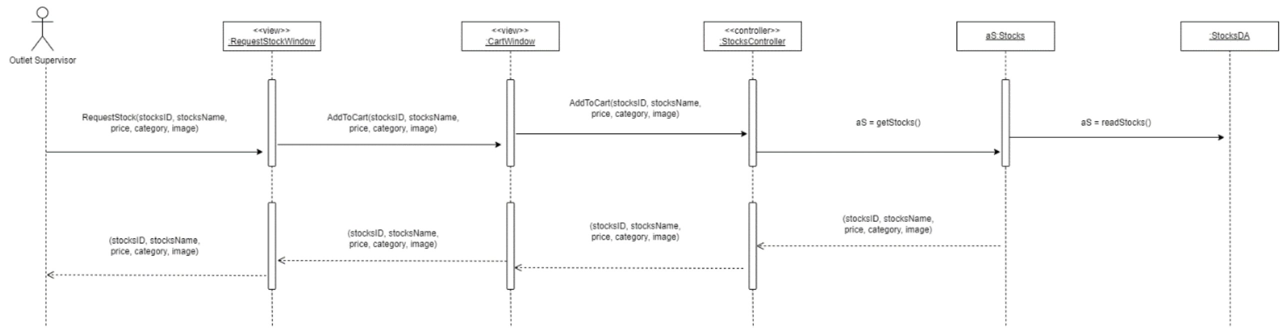
8.1 Use Case 1 – Request Stock



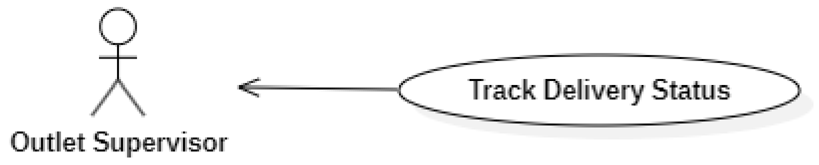
8.1.1 Detailed Class Diagram



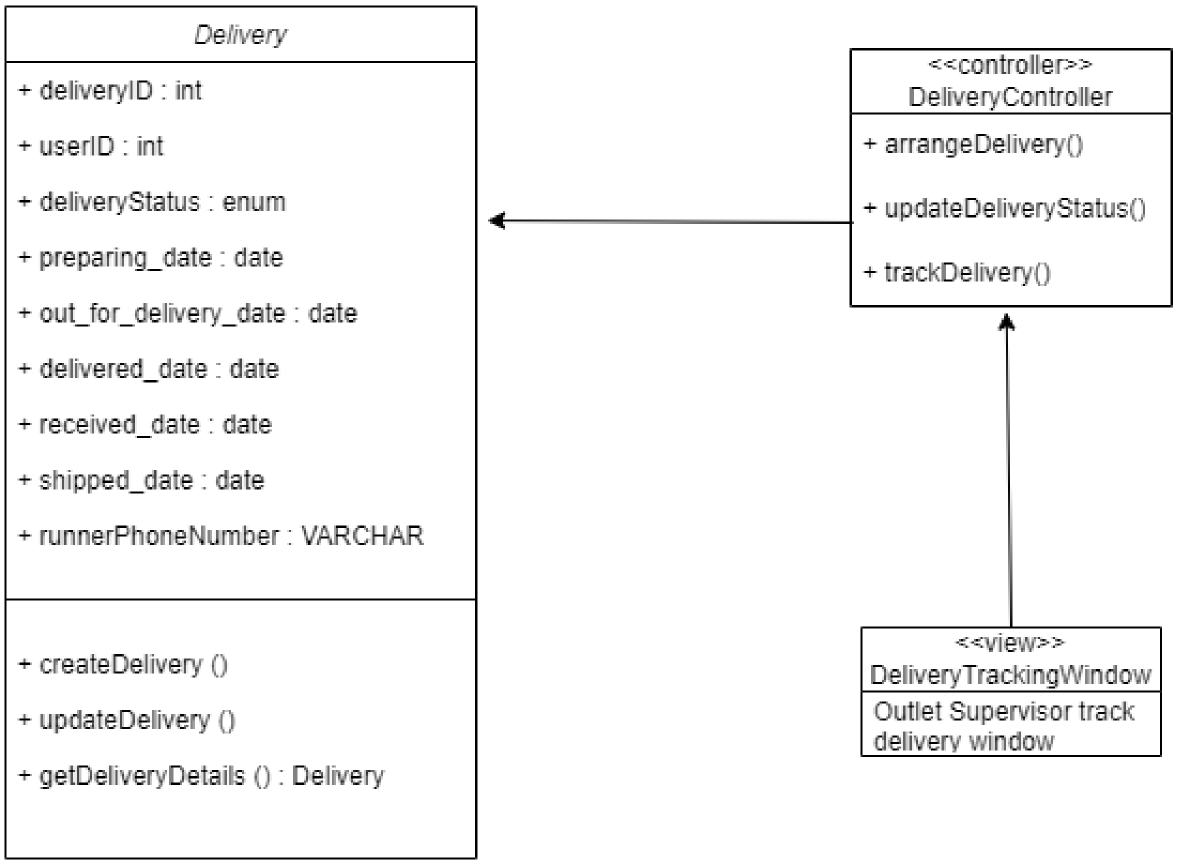
8.1.2 Multilayer Sequence Diagram


Outlet Supervisor
<<view>> :RequestStockWindow
<<view>> :CartWindow
<<controller>> :StocksController
aS:Stocks
:StocksDA
RequestStock(stocksID, stocksName, price, category, image)
AddToCart(stocksID, stocksName, price, category, image)
AddToCart(stocksID, stocksName, price, category, image)
aS = getStocks()
aS = readStocks()
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)

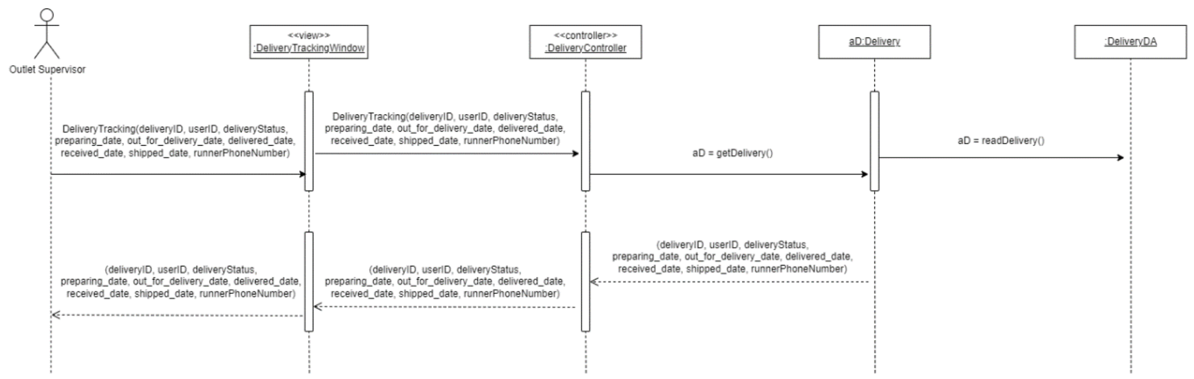
8.2 Use Case 2 – Track Delivery Status



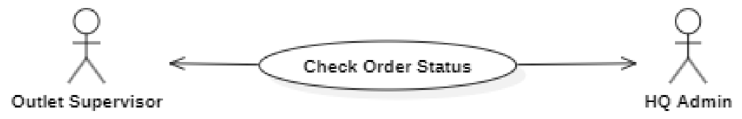
8.2.1 Detailed Class Diagram



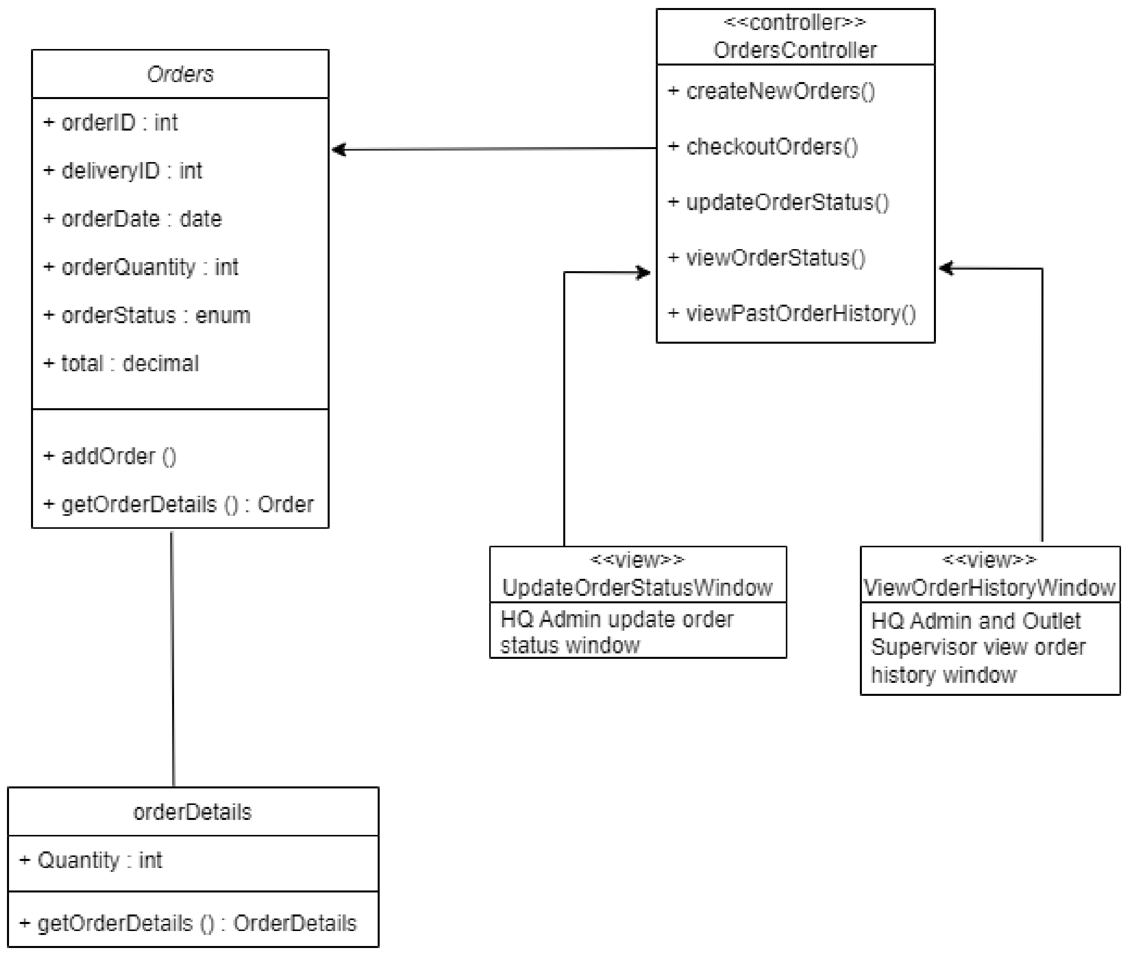
8.2.2 Multilayer Sequence Diagram


Outlet Supervisor
<<view>> :DeliveryTrackingWindow
<<controller>> :DeliveryController
aD:Delivery
:DeliveryDA
DeliveryTracking(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
DeliveryTracking(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
aD = getDelivery()
aD = readDelivery()
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)

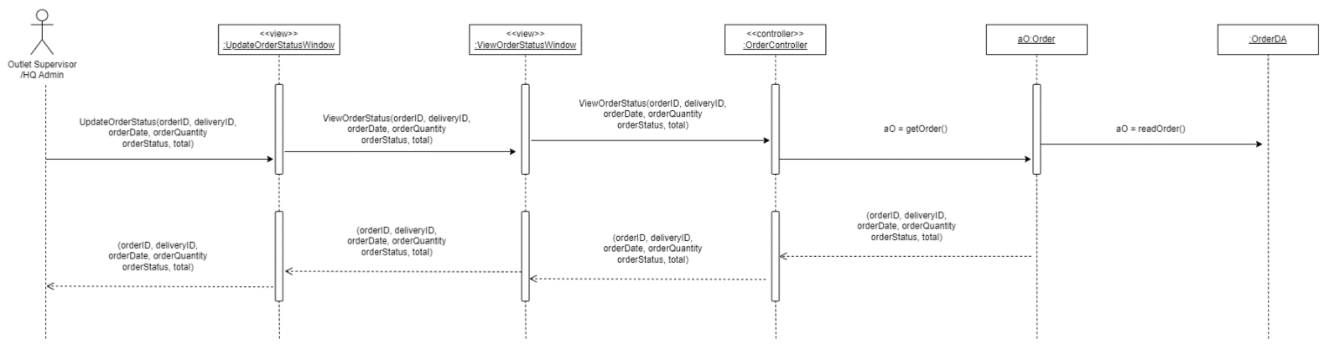
8.3 Use Case 3 – Order Status



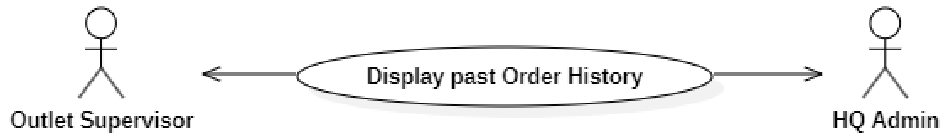
8.3.1 Detailed Class Diagram



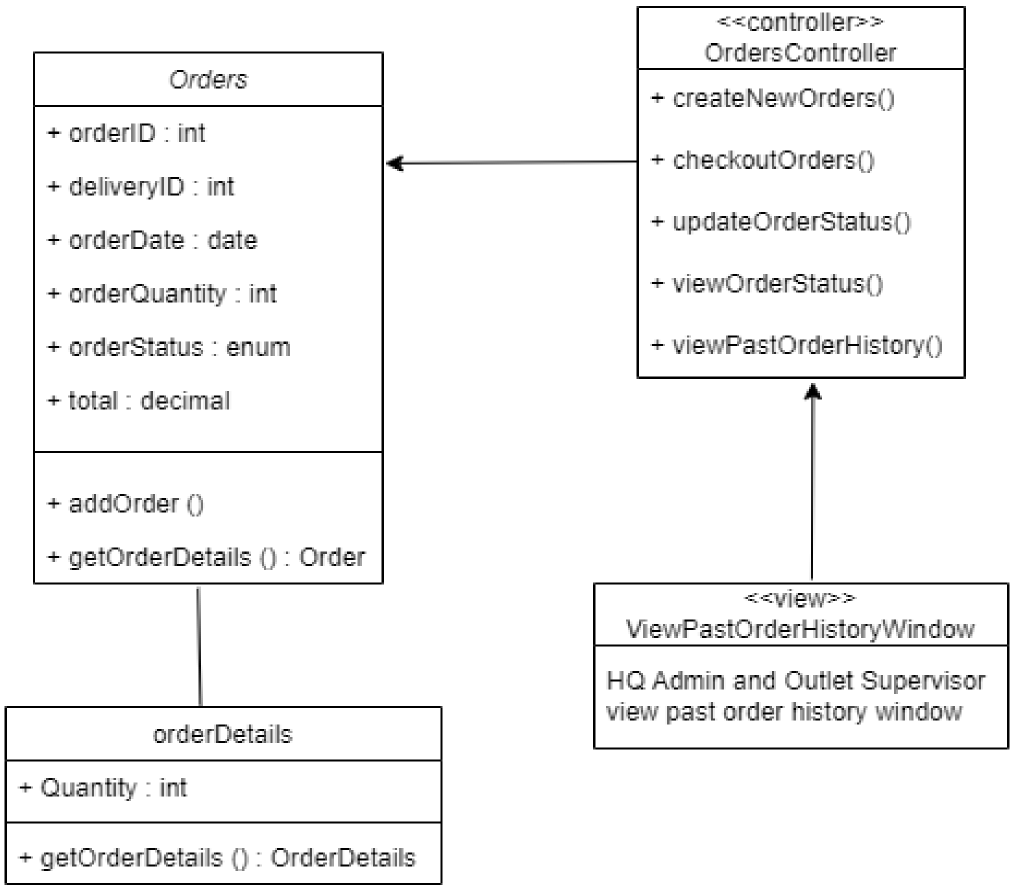
8.3.2 Multilayer Sequence Diagram


Outlet Supervisor /HQ Admin
<<view>> :UpdateOrderStatusWindow
<<view>> :ViewOrderStatusWindow
<<controller>> :OrderController
aO:Order
:OrderDA
UpdateOrderStatus(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
ViewOrderStatus(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
ViewOrderStatus(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
aO = getOrder()
aO = readOrder()
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)

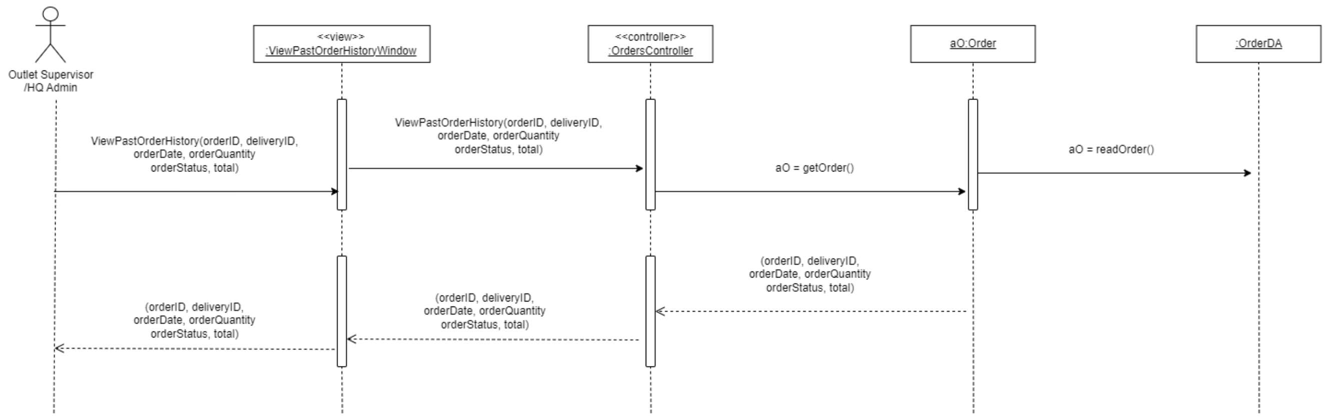
8.4 Use Case 4 – Display past Order History



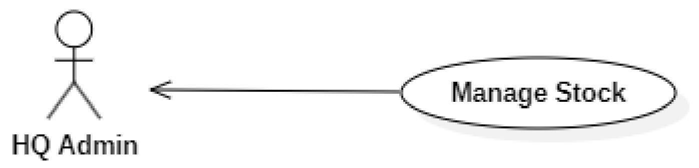
8.4.1 Detailed Class Diagram



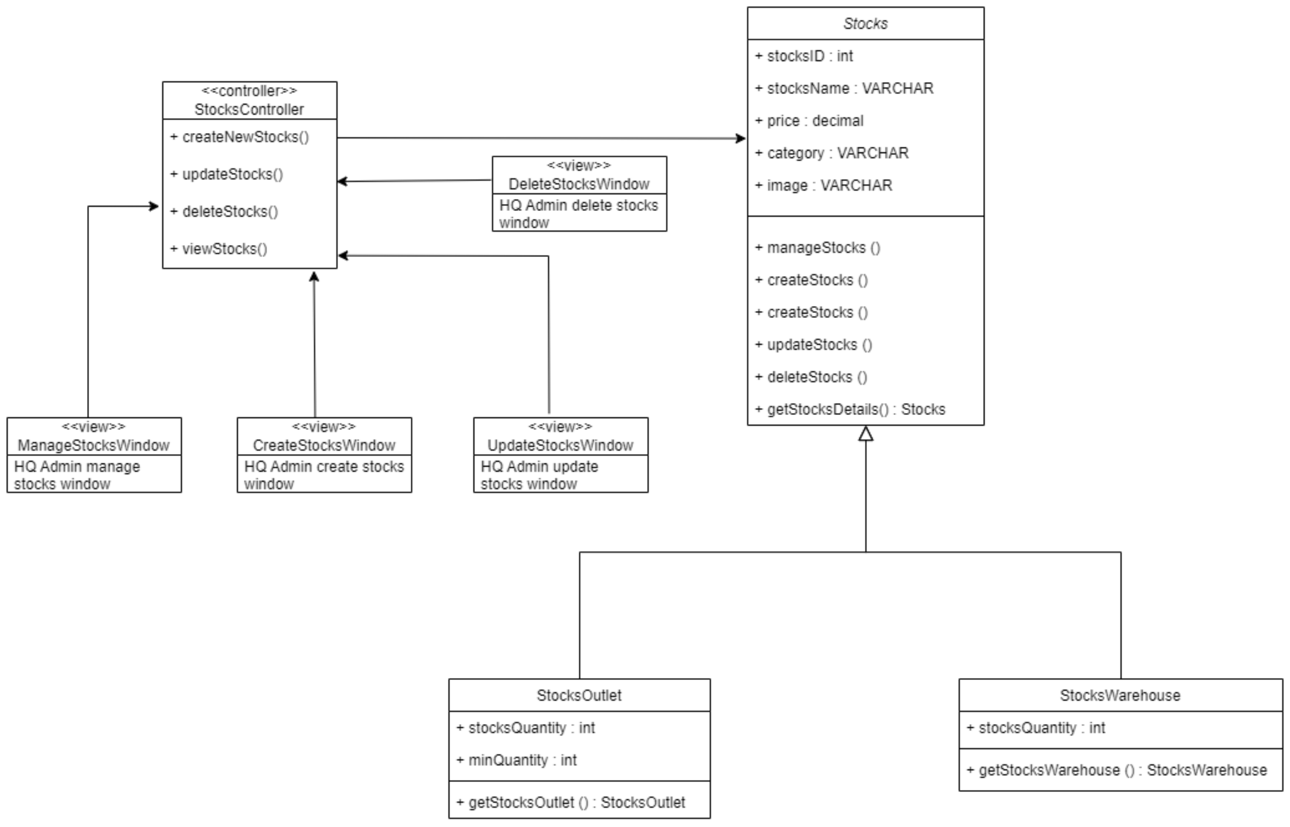
8.4.2 Multilayer Sequence Diagram


Outlet Supervisor /HQ Admin
<<view>> :ViewPastOrderHistoryWindow
<<controller>> :OrdersController
aO:Order
:OrderDA
ViewPastOrderHistory(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
ViewPastOrderHistory(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
aO = getOrder()
aO = readOrder()
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)
(orderID, deliveryID, orderDate, orderQuantity orderStatus, total)

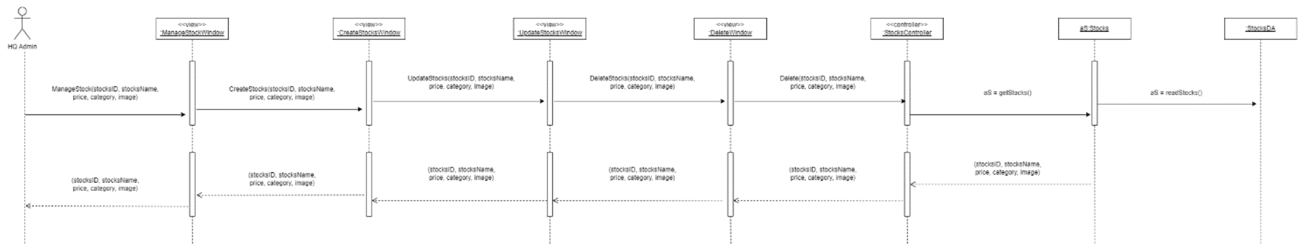
8.5 Use Case 5 – Manage Stock



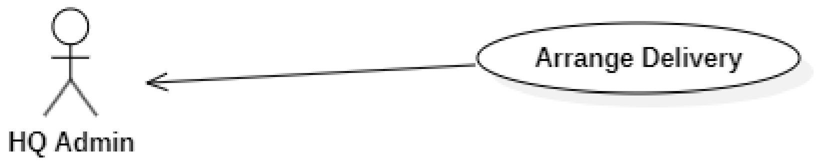
8.5.1 Detailed Class Diagram



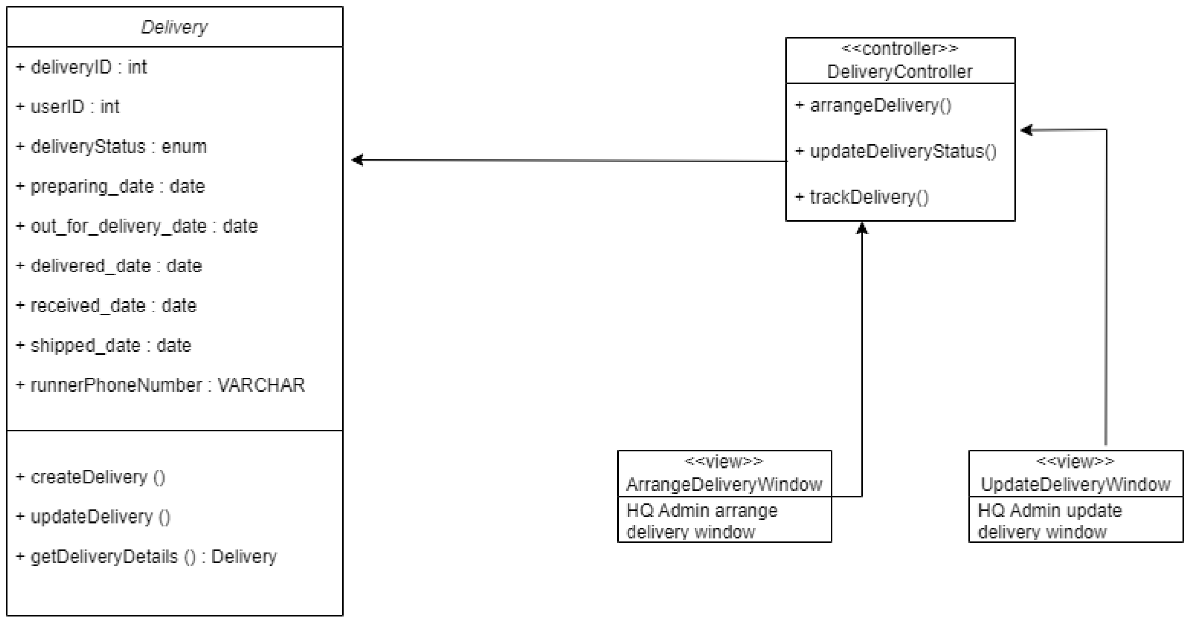
8.5.2 Multilayer Sequence Diagram


HQ Admin
<<view>>
:ManageStockWindow
<<view>>
:CreateStocksWindow
<<view>>
:UpdateStocksWindow
<<view>>
:DeleteWindow
<<controller>>
:StocksController
aS:Stocks
:StocksDA
ManageStock(stocksID, stocksName, price, category, image)
CreateStocks(stocksID, stocksName, price, category, image)
UpdateStocks(stocksID, stocksName, price, category, image)
DeleteStocks(stocksID, stocksName, price, category, image)
Delete(stocksID, stocksName, price, category, image)
aS = getStocks()
aS = readStocks()
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)
(stocksID, stocksName, price, category, image)

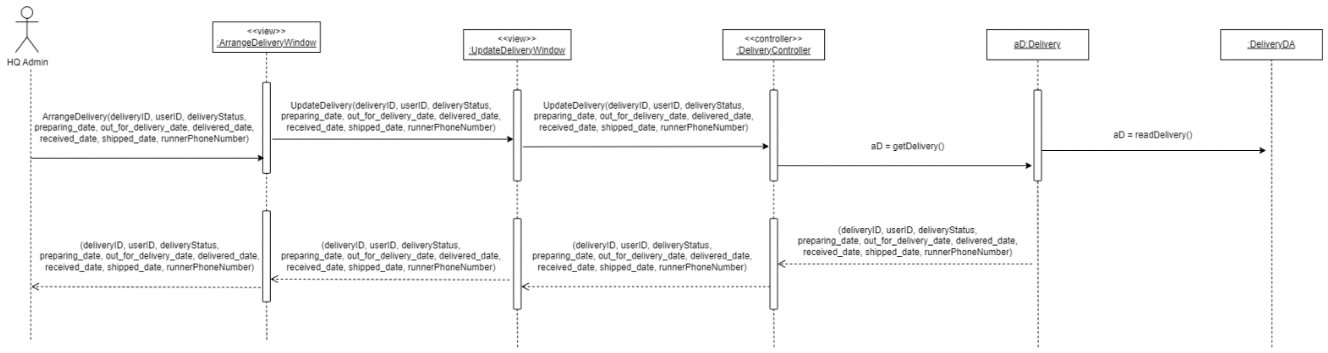
8.6 Use Case 6 – Arrange Delivery



8.6.1 Detailed Class Diagram



8.6.2 Multilayer Sequence Diagram


HQ Admin
<<view>> :ArrangeDeliveryWindow
<<view>> :UpdateDeliveryWindow
<<controller>> :DeliveryController
aD:Delivery
:DeliveryDA
ArrangeDelivery(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
UpdateDelivery(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
UpdateDelivery(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
aD = getDelivery()
aD = readDelivery()
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)
(deliveryID, userID, deliveryStatus, preparing_date, out_for_delivery_date, delivered_date, received_date, shipped_date, runnerPhoneNumber)