TRƯỜNG ĐẠI HỌC FPT

Requirement & Design Specification

FreshFood Online System (FOS)

Subject: SWD392

Version: 1.0

– Hanoi, August 2024 –

Record of Changes

Version	Date	A* M, D	In charge	Change Description
V1.0	1/8	A	Huectm3	

*A - Added M - Modified D - Deleted

Contents

I. Requirement Specification

I.1 Problem description

Purpose: Build a website to introduce and allow online ordering for FreshFood, a clean food store. The website allows customers in the vicinity to view and order online, and store employees to manage website information and handle order and delivery operations. The system requirements are described as follows:

1. The store sells various product categories. When the store sells a new product category, warehouse staff are responsible for entering the product category information into the system. The staff can update or delete the product category information if there are errors or changes.
2. A category of product will have many different products. When the store sells a new product, warehouse staff enter the product information into the system. If the product information is incorrect or changes, warehouse staff can update or delete the product information.
3. The store imports products from various suppliers. When there is a new supplier, warehouse staff enter the supplier information into the system. If the supplier information is incorrect or changes, warehouse staff can update or delete the data. A supplier can provide many different products.
4. When products are imported into the store, warehouse staff are responsible for entering the products into the store and saving the import bill information into the system. If there are errors in the import bill information, the system allows warehouse staff to update or delete the import bill information according to business rules.
5. The store needs to monitor the remaining quantity of each product in the warehouse based on their respective import batches. The selling price for each product is determined by the specific batch from which it originated. To encourage the sale of products from earlier imports especially those priced higher than newer arrivals or with significant stock still available the store can implement discount policies for these items and may delay the sale of newly imported products. In this scenario, warehouse staff will be responsible for applying discounts to the selling prices of products from previous import batches.
6. When customers want to purchase products, they can visit the website to browse product information, which can be organized by groups such as newly imported products, best sellers, promotional items, and product categories. Customers can also use the search function to find products by name. If a customer is interested in a product, they can view

detailed information about it. While browsing, if they decide to make a purchase, they can add the product to their cart. Besides viewing product information, customers can also read articles available on the website.

7. Customers who want to check out need to log in first. Customers can log in using their Google or Facebook accounts. When customers log in for the first time, the system requires them to enter information such as Full Name, Phone Number, Email, and Address. The system will verify the customer's phone number. A customer can have multiple delivery addresses, with the most recent purchase address being the default address. Customers can view and edit their information if there are any errors.
8. Customers can view the list of orders they have placed along with detailed information for each order. For orders that have not yet been delivered, customers have the option to cancel the order. For delivered orders, customers can provide feedback, and they also have the option to reorder the same items.
9. Sales staff can view detailed order information for orders of customers. For first-time orders, sales staff will call to confirm the order, print the order, pack the order, and deliver the orders to customers. After successful delivery, staff need to confirm the completion of the delivery process or report unsuccessful delivery.
10. To effectively promote product information on the website, staff can update news or articles about new products. If there are errors in the information, staff can correct or delete the article. Additionally, staff can disable outdated news or re-display previously hidden articles.
11. To facilitate the inventory activities for products, discarding products out of date, calculating revenue, etc., warehouse staff can perform the statistics functions according to different requirements.
12. All staff must log in before performing any functions. If store staff are absent/busy, the store owner can perform the functions of the staff. When the store hires new staff, the store owner creates an account for them. If a staff member quits their job, the store owner can revoke their access rights.

I.2 Major Features

I.2.1 Features for customers are as follows

- F01: Viewing products: View product list by category, view new products, view Best Seller Products, View Sale Products, Search Products by Name, View Product Detail.

- F02: Shopping Cart Management: Add product to cart, view cart, Edit Cart (Change Quantity, Remove Items)
- F03: Order Placement: add product to cart, view cart, edit cart, Checkout
- F04: Customer authentication: Register/Login (Google/Facebook), manage profile (View personal information, update profile).
- F05: My order history: View order history, Cancel order, Send feedback, View order detail, ReOrder, Delete Order.
- F06: Viewing news: View news, Search news

*I.2.2 **Features for staff are as follows***

- F07: Product Category Management: Add, Edit, delete, Sort, Search Product Category
- F08: Product Management: Add, Edit, delete Product, Search, Sort
- F09: Supplier Management: Add, Edit, delete Supplier, Search, Sort
- F10: Import Bill Management: Record Product Import (Add, Edit, Delete Import Bill), Search, sort.
- F11: Order Processing: view order list, View order Details, Print Order, confirm order, confirm Order for Packing, Confirm Delivery Successful, Confirm Returned Order, Handle Customer Feedback
- F12: Discounts Manage: Create Discounts, Edit Discounts, Delete Discount
- F13: News Management: View new details, Add news, Edit News, Delete News, Disable news, Enable news.
- F14: User Management: Login of Staff, Create staff, assign staff permission, revoke staff permission
- F15: Inventory Statistics: Statistics on expired products, statistics on out-of-stock products Use case diagrams

I.3 System context

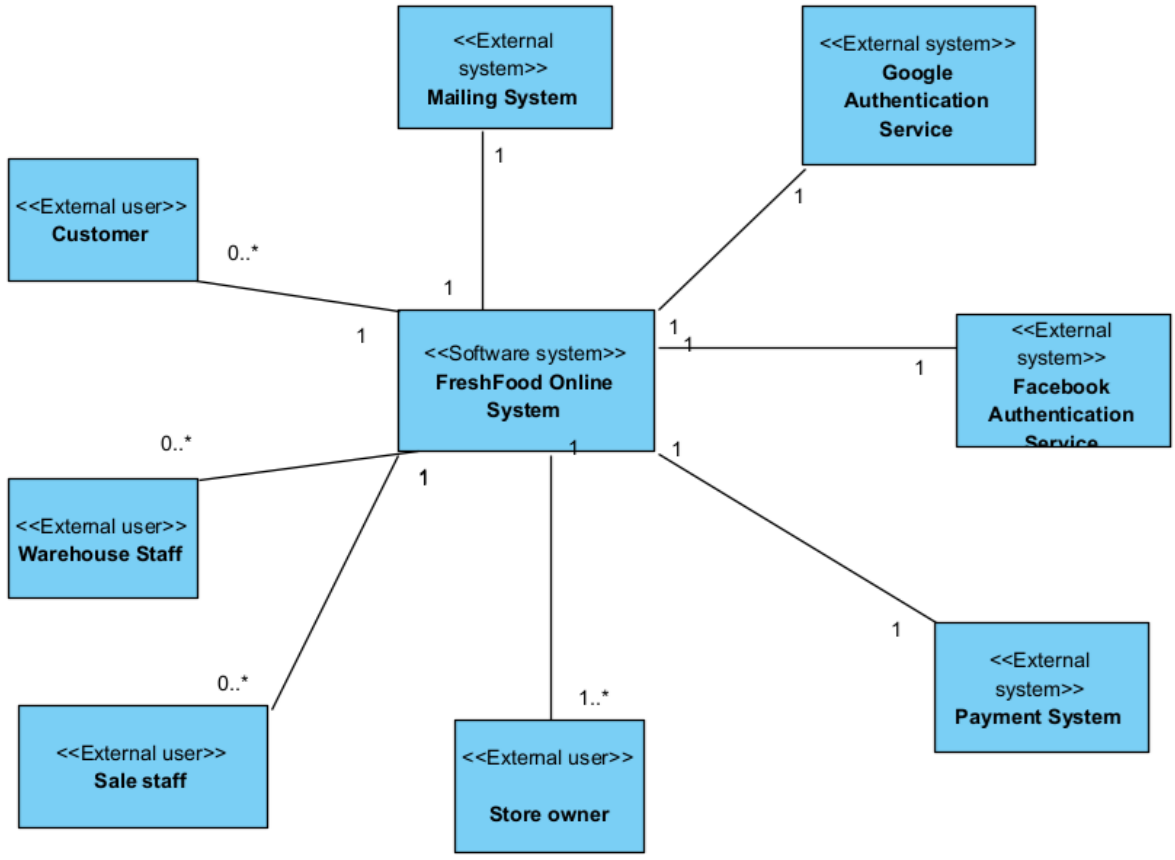

Figure I-1 System context is modeled by a class diagram

Figure 1-1 shows the system context of the FreshFood Online System, illustrating the interactions between the system and its various entities including external users and external systems.

External Users

1. **Customer**: The system's end-user who can browse products, add products to the cart, place orders, and rate orders.
2. **Warehouse Staff**: Responsible for managing product information, entering products into the warehouse, and managing supplier information.
3. **Sales Staff**: Responsible for processing orders, contacting customers to confirm orders, and managing the delivery process.
4. **Store Owner**: Manages the entire system, including managing staff, tracking revenue, and ensuring system operations.

External Systems

1. **Payment System**: Handles payment transactions when customers make payments via bank transfer or other online payment methods.
2. **Facebook Authentication Service**: Provides login services using customers' Facebook accounts.
3. **Google Authentication Service**: Provides login services using customers' Google accounts.
4. **Mailing System**: Sends notification and confirmation emails to customers, including order confirmations, promotional emails, and other notifications.

I.4 Nonfunction Requirements

NF-01 Maintainability: The system should be designed to support efficient maintenance and updates. This includes providing clear traceability among components and modules through detailed documentation of their dependencies, relationships, and interactions. The code should be modular and well-organized, with comprehensive documentation to facilitate understanding and modifications. This approach ensures that changes can be implemented smoothly and that the system remains adaptable to future needs.

NF-02 Usability: The website must offer a user-friendly interface for easy navigation and transactions.

- Task completion time (find product and complete purchase within 5 minutes).
- Error rate (less than 5% for key tasks).
- User satisfaction (at least 85% score).

NF-03 Portability: The website should be compatible across various browsers and devices.

- Browser compatibility (latest versions of Chrome, Firefox, Safari, Edge).
- Device compatibility (3 screen sizes, 2 operating systems).
- Responsive design (screen widths from 320px to 1920px).

NF-04 Performance: The website should ensure fast performance with quick load times and smooth interactions.

- Page load time (less than 4 seconds for 95% of users).
- Response time (handle 500 users, avg. < 6 seconds).
- Stress testing (200 concurrent users, performance degradation < 20%).

In this example, we will use UML to construct models in software development's requirements, analysis, and design phases. These models are platform-independent and follow the PIM (Platform-Independent Model) approach, ensuring they focus on business logic and functional requirements without being tied to any specific technology platform. In the following

sections, we will present the models for each phase requirements, analysis, and design followed by the mapping of these models to source code.

I.5 Functional requirements

I.5.1 Use case diagrams

Based on described user requirements in Section 1.1 and detailed analysis, the system has been deployed on a web platform, consisting of two primary sites: the Customer Site and the Administration Site. Figure 1-2 presents the Overview Use Case Diagram for the Customer Site, highlighting key functionalities such as viewing product information, reading news, managing cart, managing profile, and Checkout. Figures 1-3 showcase the Overview Use Case Diagram for the Administration Site, covering essential tasks like managing product categories, managing products, processing orders, managing import bills, managing news, managing users, reporting statistics, managing discounts, and managing suppliers. Figures 1-4 to 1-11 show some Detailed Use Case Diagrams, which further break down the specific use cases for both the Customer Site and the Administration Site.

Tables 1-1 to 1-6 show descriptions of use cases Manage Products, Add product, Edit product, Delete product, Disable news, Enable news. These descriptions will be inputs to building analysis models in Section 2.

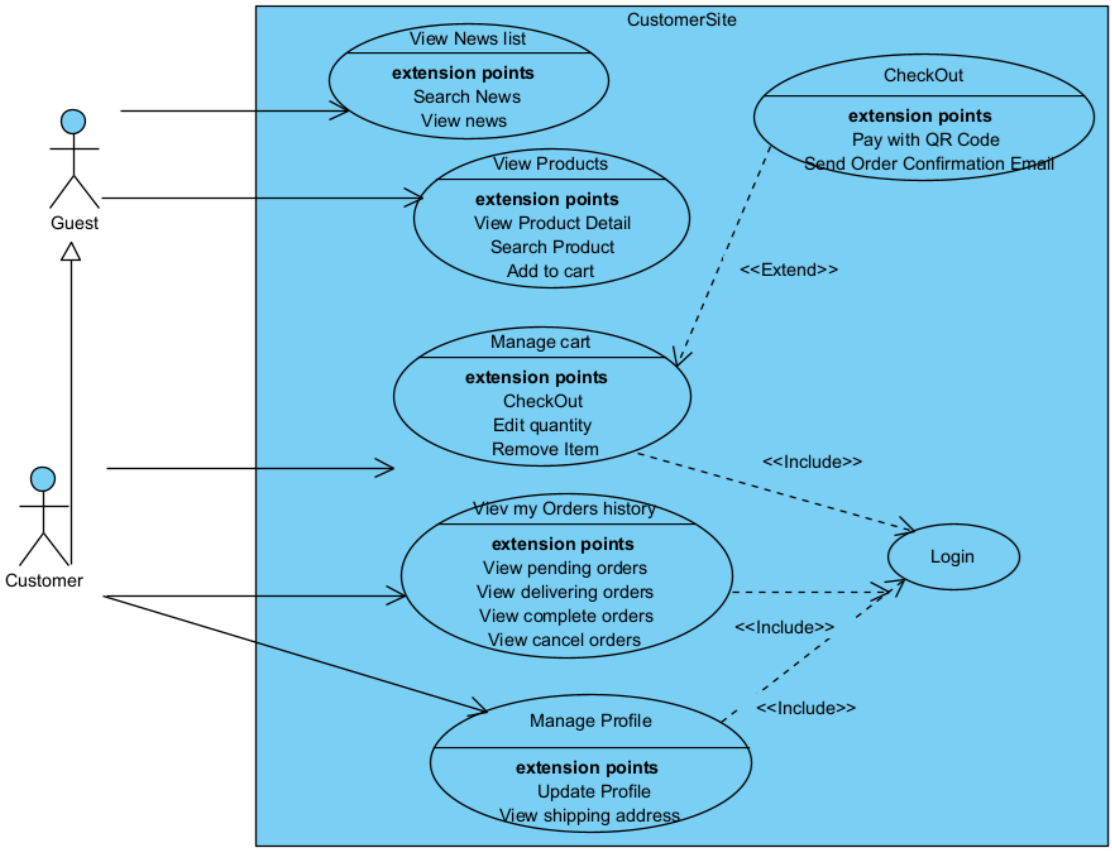

Figure I-2 Overview Use Case Diagram of Customer Site

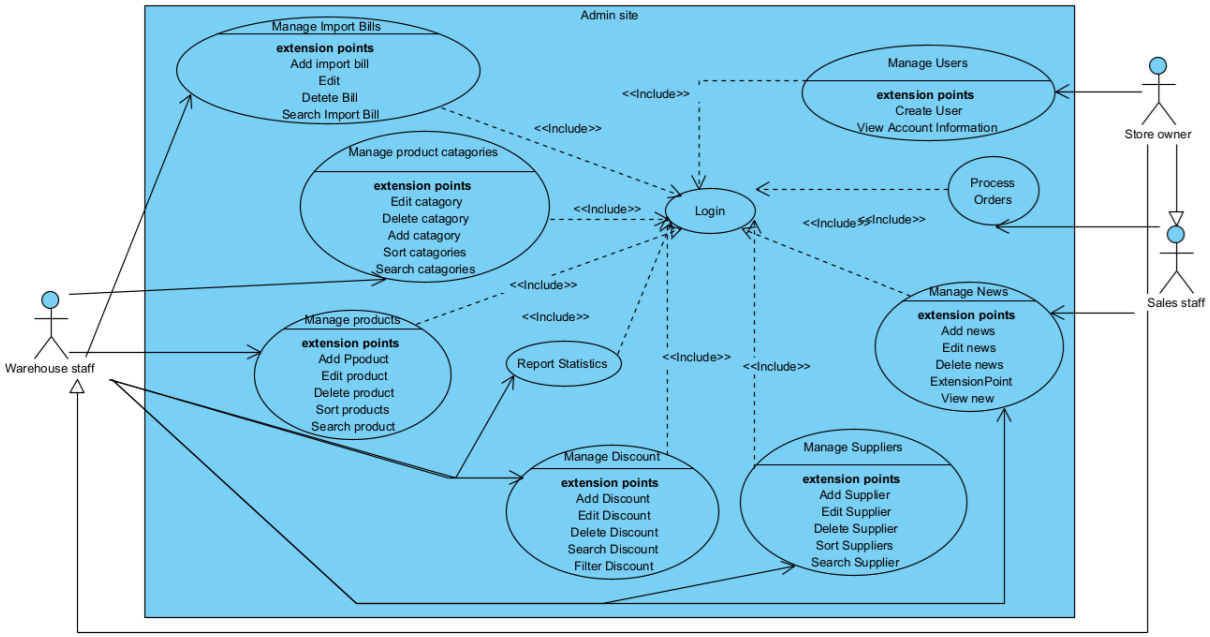

Figure I-3 Overview Use Case Diagram of Admin Site

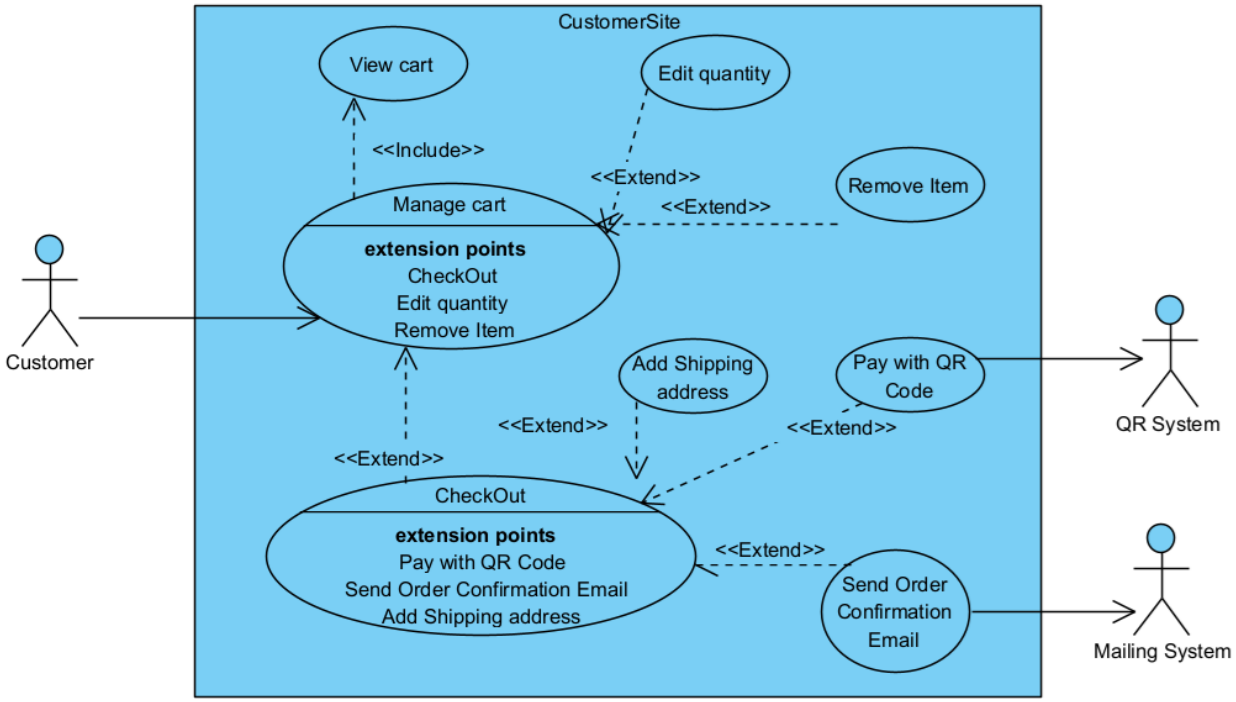

Figure I-4 Detailed Use Case Diagram for Managing Cart

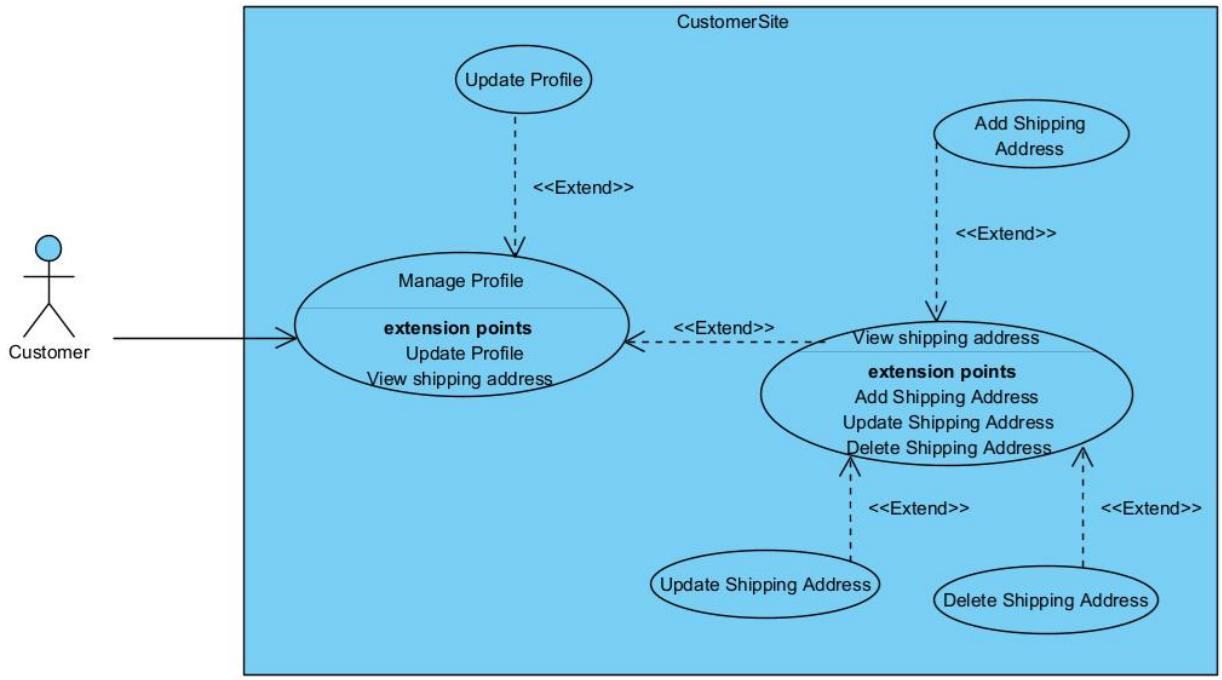

Figure I-5 Detailed Use Case Diagram for Managing Profile

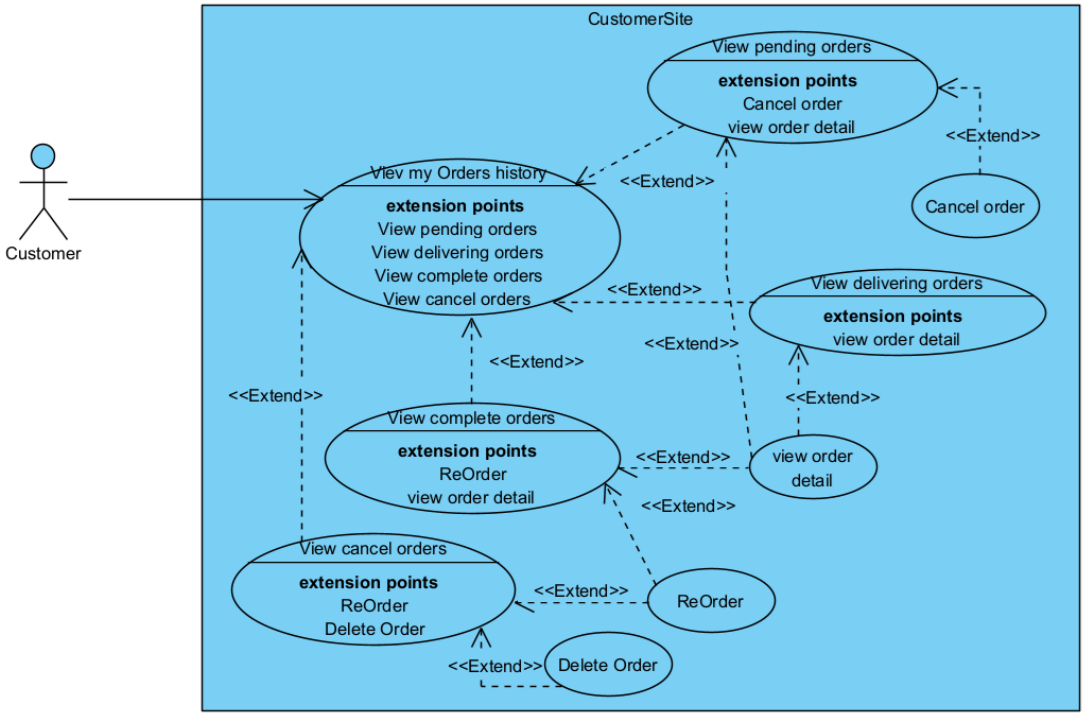

Figure I-6 Detailed Use Case Diagram for View my Orders history

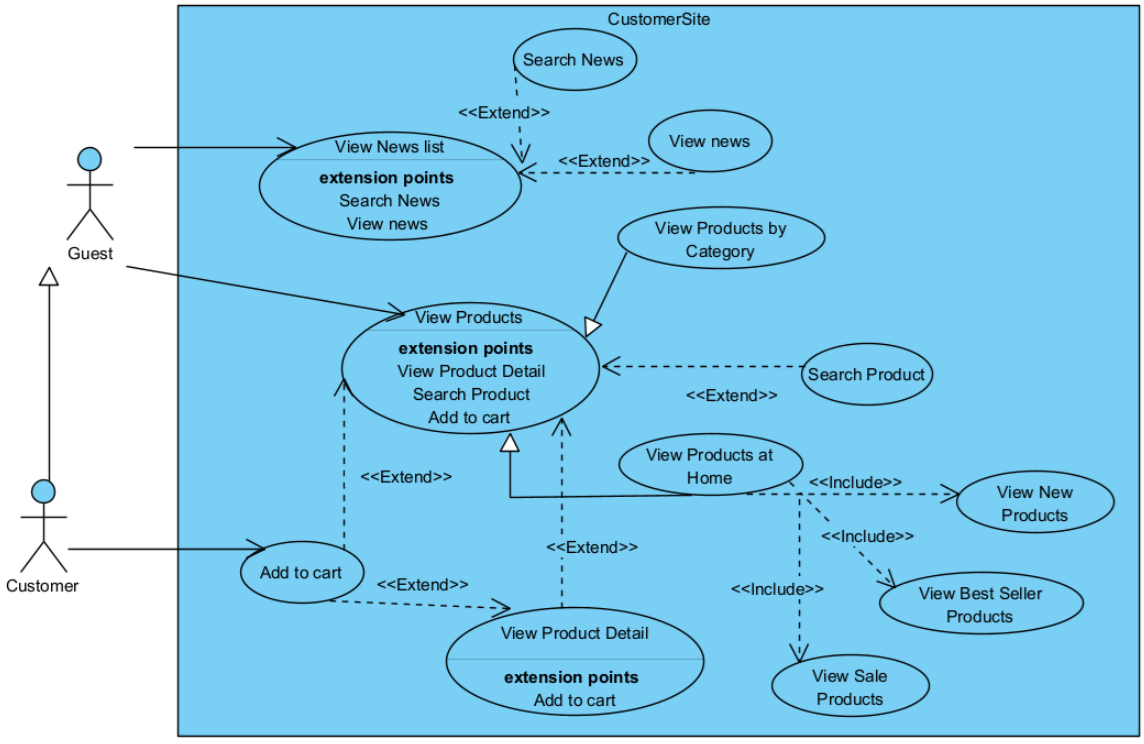

Figure I-7 Detailed Use Case Diagram for View News and View Products

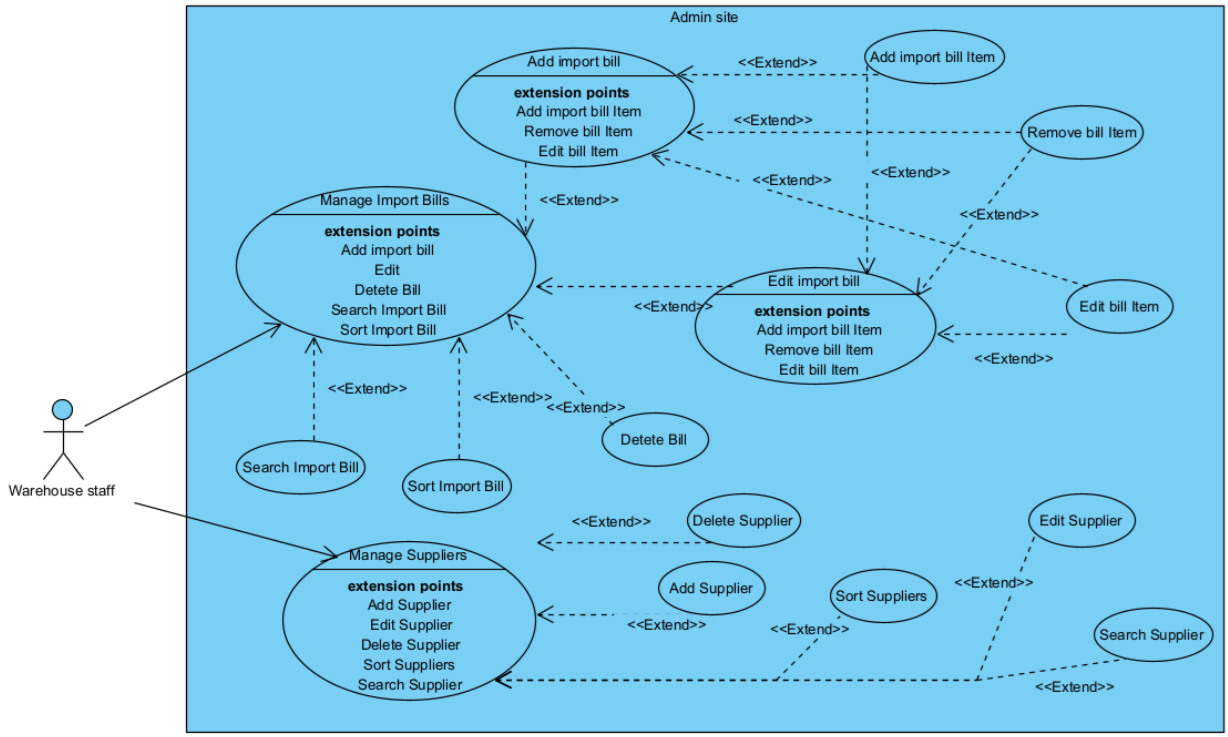

Figure I-8 Detailed Use Case Diagram for Managing Import Bills and Suppliers

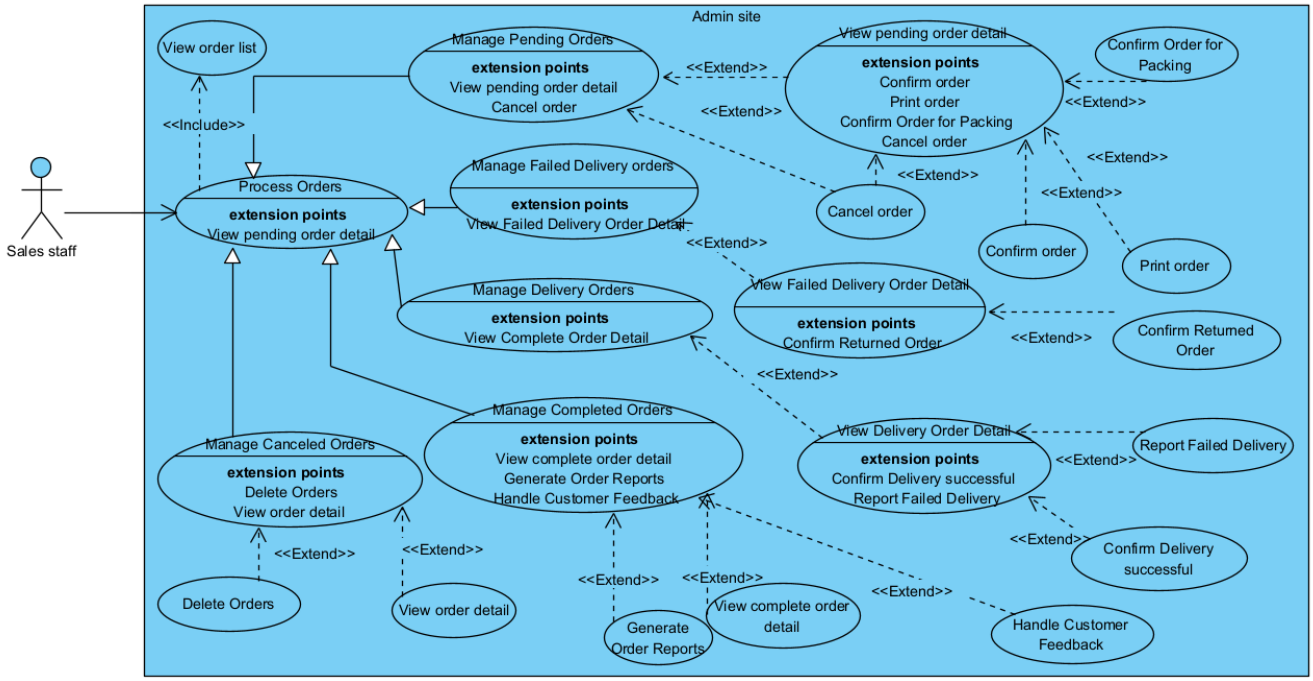

Figure I-9 Detailed Use Case Diagram for Process Orders

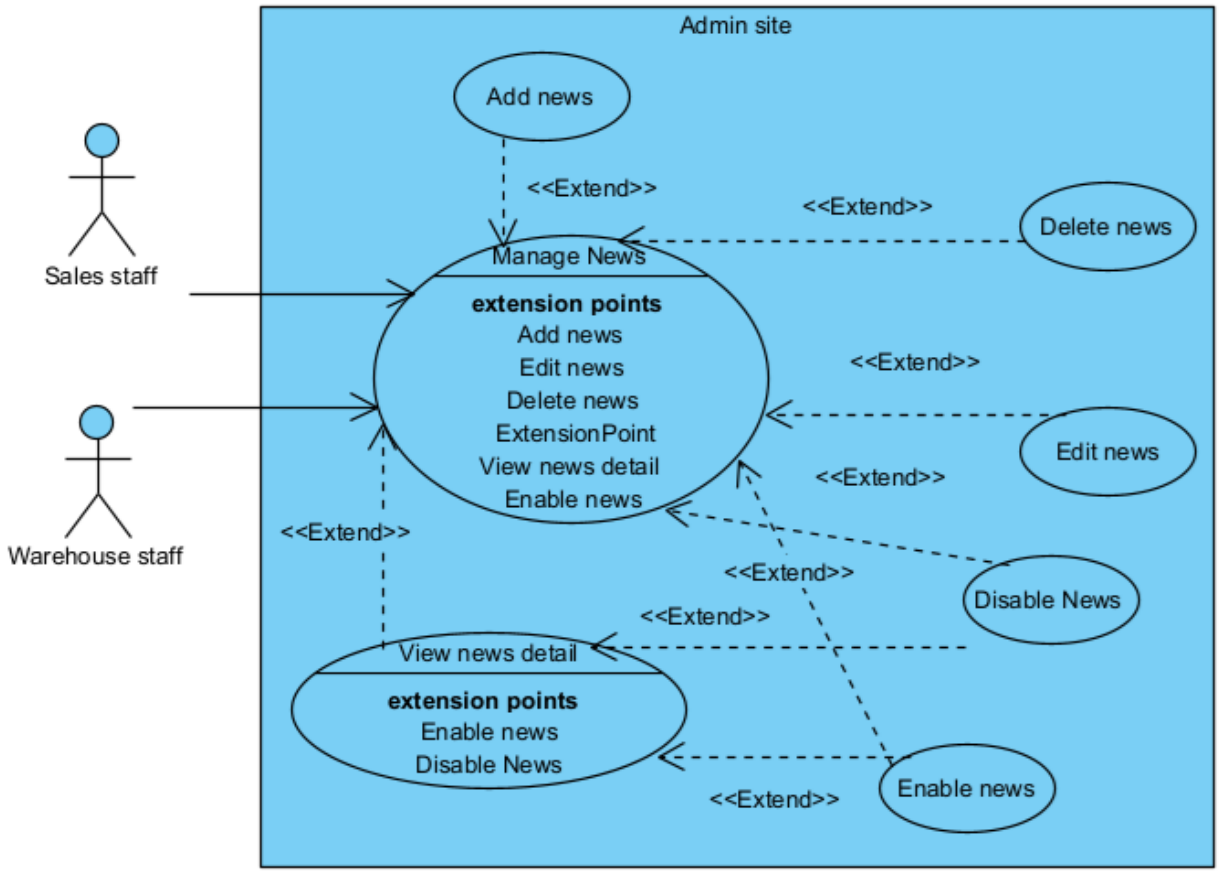

Figure I-10 Detailed Use Case Diagram for Manage News

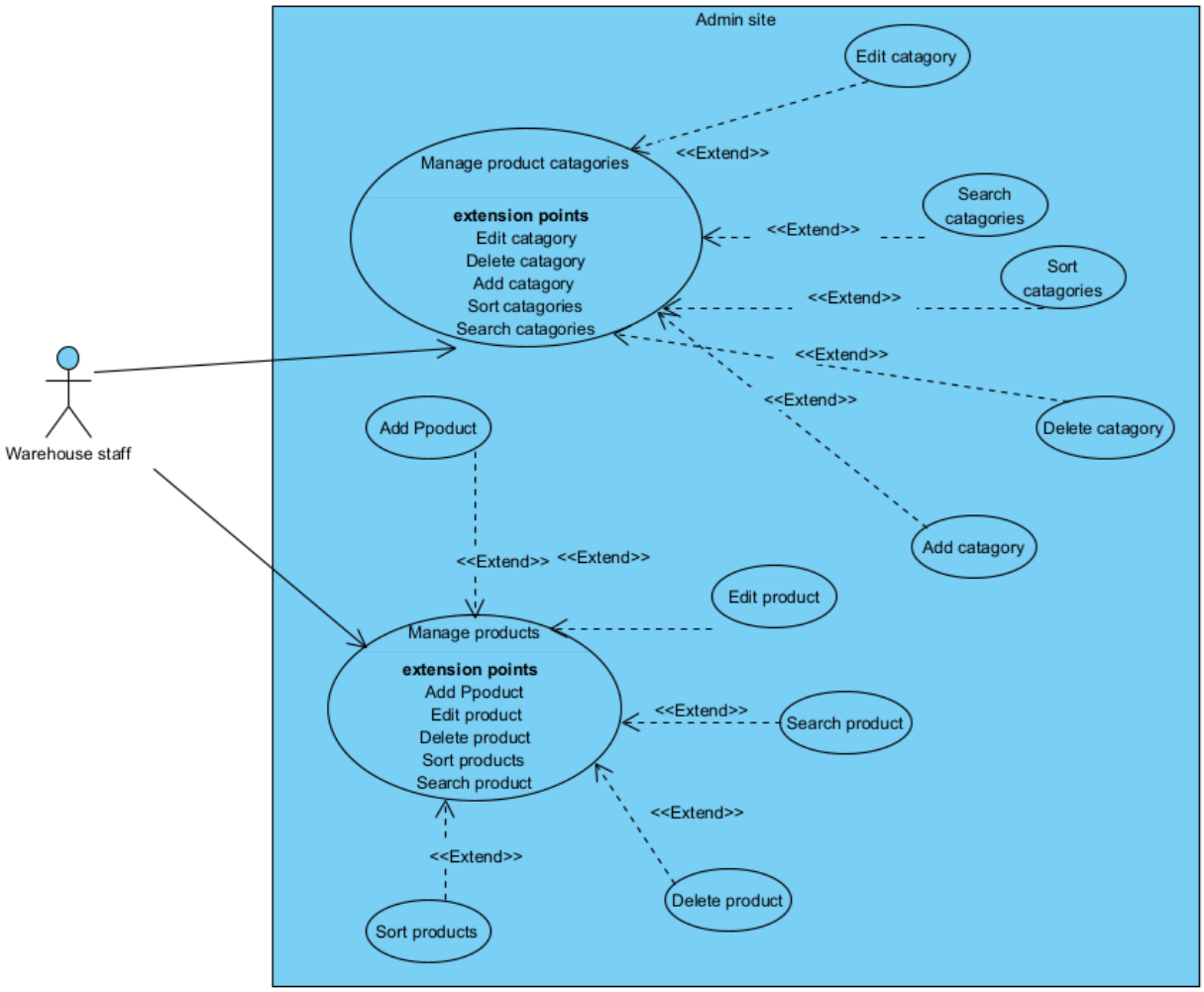

Figure I-11 Detailed Use Case Diagram for Manage products and product categories

I.5.2 Use case descriptions

Tables 1-1 to 1-6 describe the use cases: Manage Products, Add Product, Edit Product, Delete Product, Disable News, and Enable News. The event flows of these use cases can be modeled using activity diagrams. For example, Figures 1-12 and 1-13 are activity diagrams modeling the event flows of the Manage Products and Add Product use cases.

Table I-1 Use case description for managing products

ID and Name:	**UC-01 Manage products**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff (WS)	Secondary Actors:	
Description:	This use case allows the Store Manager to manage the products in the system, including adding, editing, and deleting, Searching products.		
Trigger:	The Warehouse staff selects the Manage Product option from the system menu.		
Preconditions:	PRE-1. Warehouse staff is logged into FSO. PRE-2. Warehouse staff is authorized to manage products.		
Postconditions:	POST-1. Products are correctly managed, with updates reflected in the system		
Normal Flow:	1. The WS selects the Manage Product option from the system menu. 2. The system retrieves the list of products from the database 3. The system displays the Manage Product interface with the list of products. 4. The WS views the list of products. 5. The WS can select actions: - Add Product - Edit Product - Delete - Search Product		
Alternative Flows:			
Exceptions:			
Priority:	High		
Business Rules:			

Table I-2 Use case description for Add product

ID and Name:	**UC-02 Add product**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff (WS)	Secondary Actors:	
Description:	This use case allows the Warehouse staff to add new products to the system.		
Trigger:	The Warehouse staff selects the Add Product option from the Manage Products section		
Preconditions:	PRE-1. Warehouse staff is logged into FSO. PRE-2. Warehouse staff is authorized to add products.		
Postconditions:	**POST-1. Successful Addition**: The new product is added to the system and the updated product list at the product manage screen. **POST-2. Unsuccessful Addition:** The system provides feedback to the Store Manager, and no changes are made to the product list.		
Normal Flow:	1. WS selects Add Product from the Manage Product section. 2. The system displays an add new product form. 3. The system retrieves the list of product categories and populates the dropdown in the add a new product form. 4. WS fills in the product details and selects a category from the dropdown. (See A4.1) 5. WS submits "add product". 6. The system validates the input data. (See E6.1) 7. The system adds the new product to the product list. (See E7.1) 8. The system displays a confirmation message that the product was added successfully.		
Alternative Flows:	**A4.1 Select Close (Step 4)** 1. The WS selects the Cancel button. 2. The system closes the Add Product screen and returns to the Manage Product screen.		
Exceptions:	**E6.1 Invalid Data (Step 6)**: If the input data is invalid 1. The system displays an error message and prompts the WS to correct the data. **E7.1 System Error (Step 7):** If there is a system error while adding the product		

	1. The system logs the error and displays an error message to the WS.
Priority:	High
Business Rules:	

Table I-3 Use case description for Edit product

ID and Name:	**UC-03 Edit product**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff (WS)	Secondary Actors:	
Description:	This use case allows the Warehouse staff to edit existing products in the system.		
Trigger:	The Warehouse staff selects the Edit Product option from the Manage Product section.		
Preconditions:	PRE-1. Warehouse staff is logged into FSO. PRE-2. Warehouse staff is authorized to edit products. PRE-3. The product to be edited exists in the system		
Postconditions:	**POST-1. Successful Edit**: The product details are updated in the system. **POST-2. Unsuccessful Edit**: The system provides feedback to the WS, and no changes are made to the product details.		
Normal Flow:	1. The WS selects Edit Product from the Manage Product section. 2. The system displays an edit product form. 3. The system retrieves the list of product categories and the details of the selected product from the database and populates the form. 4. The WS edits the product details as needed (See A4.1). 5. The WS submits the form 6. The system validates the input data (See E6.1). 7. The system updates the product details in the database.		

	8. The system displays a confirmation message that the product was successfully updated. 9. The system closes this form and returns the Manage Product screen
Alternative Flows:	**A4.1 Select Close (Step 4)** 1. The WS selects the Close button. 2. The system closes the Edit Product screen and returns to the Manage Product screen.
Exceptions:	**E6.1. Invalid Data (Step 6)** 1. The system displays an error message and prompts the Store Manager to correct the data. **E7.1 System Error (Step 7)**: If there is a system error while updating the product 1. The system logs the error and displays an error message to the WS.
Priority:	High
Business Rules:	

Table I-4 Use case description for Delete product

ID and Name:	**UC-04 Delete product**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff (WS)	Secondary Actors:	
Description:	This use case allows the Warehouse staff to delete existing products in the system.		
Trigger:	The Warehouse staff selects the Delete Product option from the Manage Product section.		
Preconditions:	PRE-1. Warehouse staff is logged into FSO. PRE-2. Warehouse staff is authorized to Delete products. PRE-3. The product to be deleted exists in the system		
Postconditions:	**POST-1. Successful Deletion**: The product is removed from the system. **POST-2. Unsuccessful Deletion**: The system records the attempt and provides feedback to the WS, and no changes are made to the product list.		
Normal Flow:	1. The WS selects a product to delete.		

	2. The system checks if the selected product has been received in any other tables (See A2.1) 3. If the product has never been received, the system prompts the WS to confirm the deletion. 4. The WS confirms the deletion. 5. The system deletes the product from the database (See E5.1). 6. The system displays a confirmation message that the product was deleted successfully.
Alternative Flows:	**A2.1 Product Received in ImportBillProduct (Step 2)** 1. If the product has been received in any other tables, the system displays a message indicating that the product cannot be deleted and returns to the product list.
Exceptions:	**E5.1 System Error (Step 5):** If there is a system error while deleting the product 1. The system logs the error and displays an error message to the Store Manager.
Priority:	High
Business Rules:	

Table I-5 Use case description for Disable News

ID and Name:	**UC-04 Disable News**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff, Sales staff (staff)	Secondary Actors:	
Description:	This use case allows a staff to deactivate a news article on the website. Once disabled, the news article will no longer be visible to users on the customer site.		
Trigger:	The staff selects Disable News from the Manage News section or View News Detail.		
Preconditions:	PRE-1. The staff must be logged in and have the appropriate permissions to disable news articles.		

	PRE-2. The staff member is currently on the Manage News page, or view new detail. PRE-3. The news article must already exist and be active
Postconditions:	**POST-1.** The selected news article is deactivated and no longer visible on the customer site. **POST-2.** The system logs the action, recording which user disabled the news and when it was done.
Normal Flow:	1. The staff initiates the action to disable the news article. 2. The system displays a confirmation dialog, asking the actor to confirm whether they want to disable the news article (See A2.1) 3. The staff confirms the action in the dialog. 4. The system updates the status of the news article to "disabled" 5. The system logs the action for audit purposes.
Alternative Flows:	**A2.1: Staff Cancels the Disable Action:** 1. The staff selects the "Cancel" option in the confirmation dialog. 2. The system closes the confirmation dialog without making any changes to the news article.
Exceptions:	
Priority:	High
Business Rules:	

Table I-6 Use case description for Enable news

ID and Name:	**UC-05 Enable News**		
Created By:	Huectm	Date Created:	20/07/2024
Primary Actor:	Warehouse staff, Sales staff (staff)	Secondary Actors:	
Description:	This use case allows a staff to reactivate a previously disabled news article on the website. Once enabled, the news article will be visible to users on the customer site again.		
Trigger:	The staff selects Enable News from the Manage News section or View News Detail.		

Preconditions:	PRE-1. The staff must be logged in and have the appropriate permissions to enable news articles PRE-2. The staff is currently on the Manage News page, or view new detail. PRE-3. The news article must already exist and be in a disabled state
Postconditions:	**POST-1.** The selected news article is enabled and becomes visible on the customer site. **POST-2.** The system logs the action, recording which user enabled the news and when it was done.
Normal Flow:	1. The actor initiates the action to enable the news article. 2. The system displays a confirmation dialog, asking the actor to confirm whether they want to enable the news article (See A2.1). 3. The actor confirms the action in the dialog. 4. The system updates the status of the news article to "enabled" and makes it visible on the customer site. 5. The system logs the action for audit purposes.
Alternative Flows:	**A2.1: Staff Cancels the Enable Action:** 1. The staff selects the "Cancel" option in the confirmation dialog. 2. The system closes the confirmation dialog without making any changes to the news article.
Exceptions:	
Priority:	High
Business Rules:	

I.5.3 Activity diagrams

Figures 1-12 and 1-13 are activity diagrams modeling event flows of the use case Manage products and Add product.

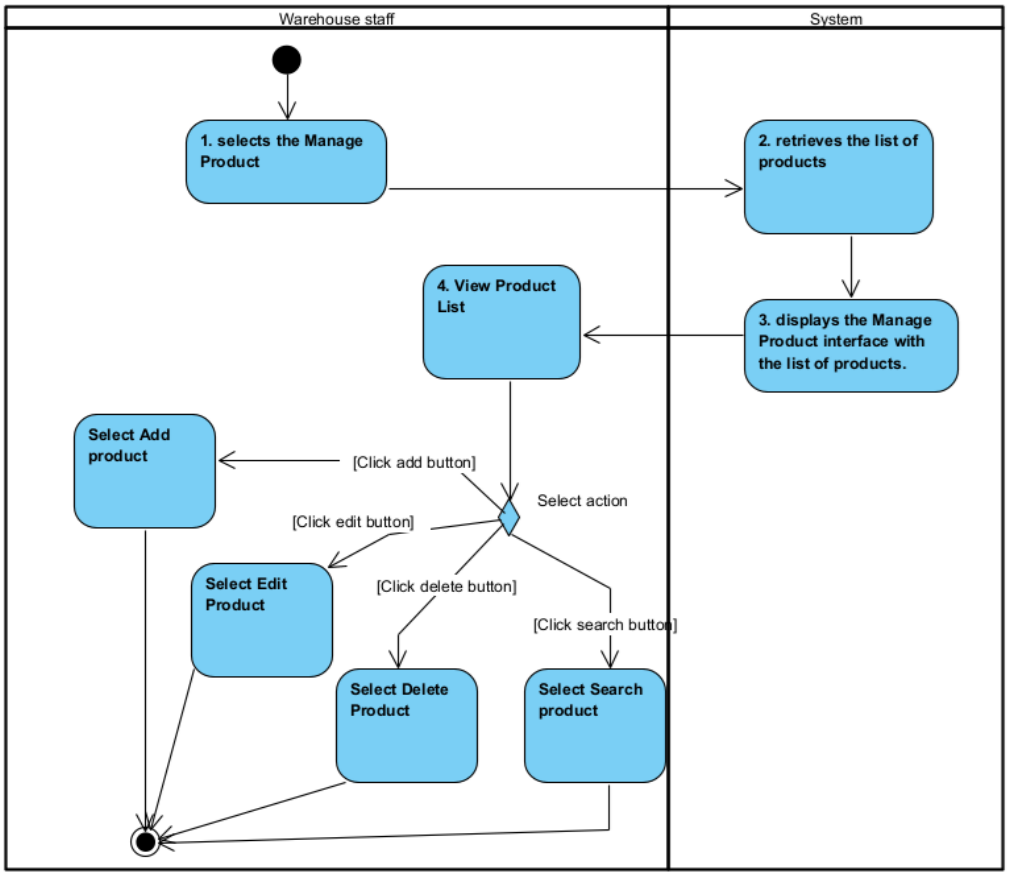

Figure I-12 Activity diagram modeling event flows of use case Manage product

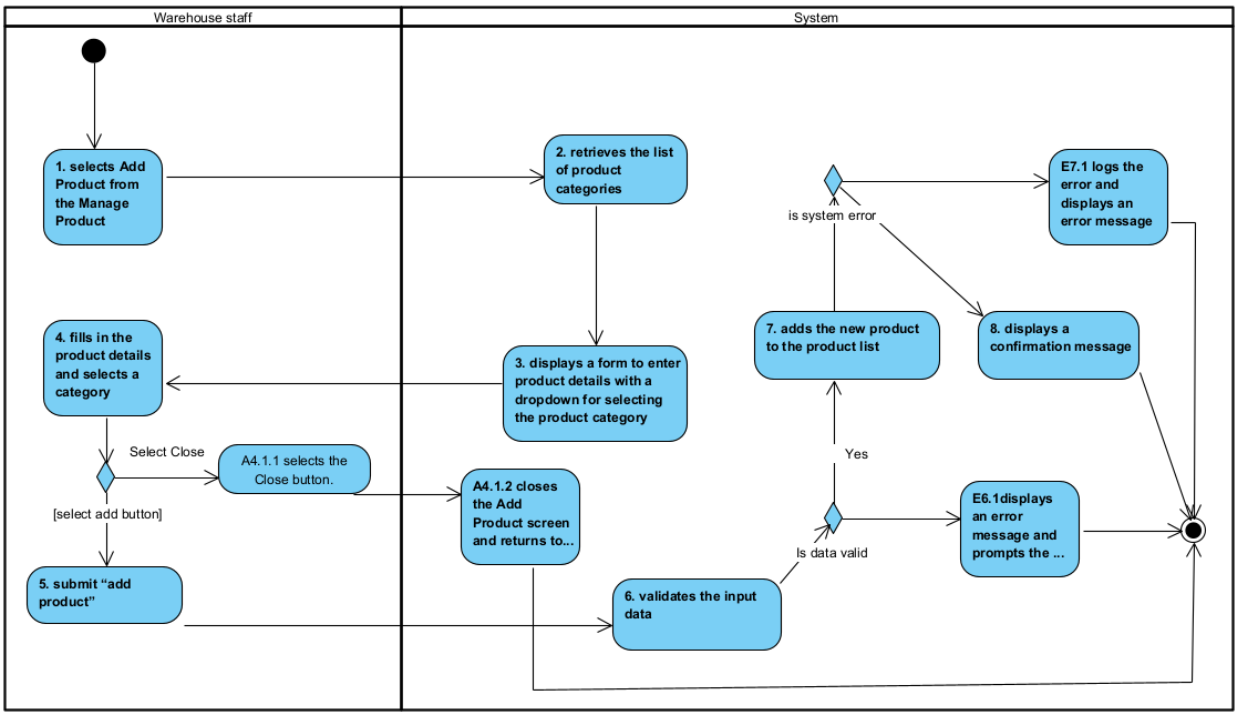

Figure I-13 Activity diagram modeling event flows of use case Add product

I.6 Data Requirements

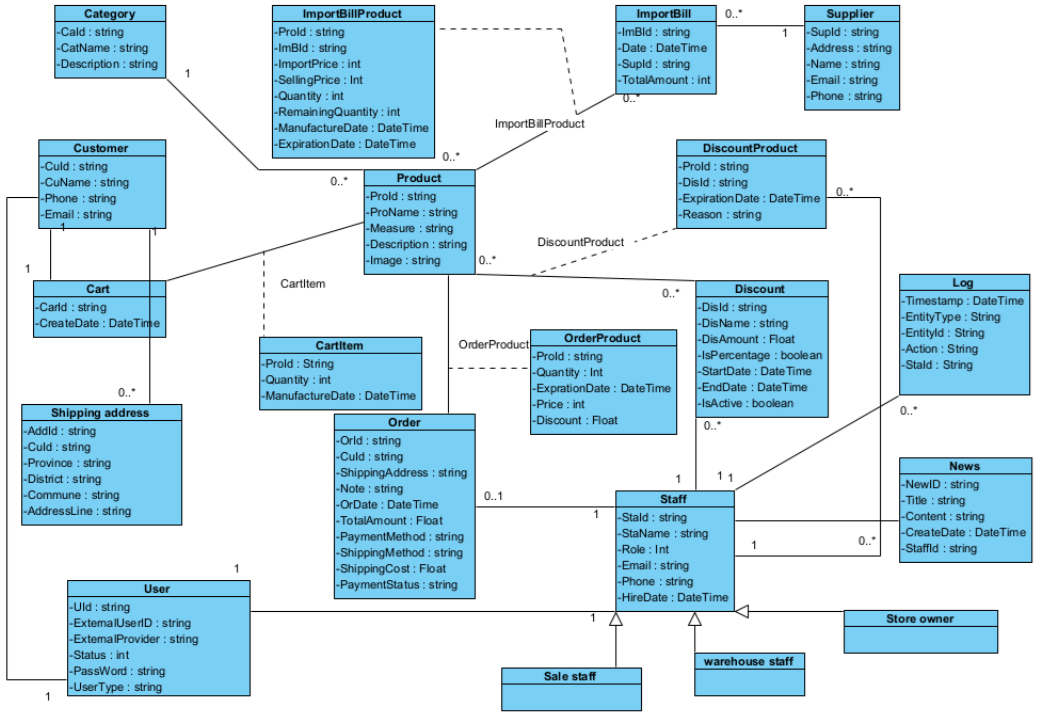

Figure I-14 Entity class diagram modeling data requirements

The class diagram in Figure 1.14 specifies the entity classes and the relationships between them, clearly representing the system's data requirements. Each entity class in the diagram represents a domain concept, relationships between entities are associations, aggregations, composition, or inheritances. To provide detailed information about these entity classes, they are all thoroughly described in Table 1-7, which is known as the data dictionary. This data dictionary serves as an explanatory tool, offering detailed descriptions of each class's attributes, data types, and relationships, helping the reader better understand the data structure and system requirements.

Table I-7 Data Description (Data dictionary)

Name	**Data Type**	**Length**	**Description**
Category			**Product category**
CatId	String	20	Product Category Identifier
CatName	String	50	Product Category Name
Description	String	500	Description of product category
Product			
ProId	String	20	Product Identify
ProName	String	50	Product Name
Measure	String	50	The unit of measurement used to quantify the product.
Description	String	Max	Description of product
Image	String	50	The visual representation of the product. It stores information about an image file that shows what the product looks like.
Supplier			
SupId	String	20	Supplier identifier
Name	String	50	Supplier Name
Address	String	500	Supplier Address
Phone	String	15	The contact telephone number for the supplier
Customer			
CusId	String	20	Customer Identifier

CusName	String	50	Customer Name
Phone	String	15	The contact telephone number for the customer
Email	String	50	The email address of the customer
Address	String	500	The physical address of the customer
Discount			
DisId	string	20	Discount Identifier
DisName	String	250	The name or description of a discount applied to orders or products.
DisAmount	Float		The monetary value or percentage of the discount applied to an order or product
IsPercentage	Boolean		It indicates whether the discount amount is represented as a percentage or a fixed monetary value
StartDate	DateTime		The date and time when a particular record or action becomes effective
EndDate	DateTime		The date and time when a particular record or action expires or is no longer effective
IsActive	Boolean		It indicates whether a particular discount is currently active and can be applied or not.
ShippingAddress			
AddId	String	20	Shipping address Identifier
CuId	String	20	Customer Identifier
Province	String	50	It is a part of the address and represents a specific administrative division within a country, such as a city, province, or region
District	String	50	It is a part of the address and represents a specific administrative division within provinces and cities
Commune	String	50	It is an address represents a smaller administrative division within a district
AddressLine	String	100	It typically includes street names, building numbers, apartment or suite numbers, and other address details.
Order			

OrdId	String	20	Order Identifier
CusId	String	50	Customer Identifier
ShippingAddress	String	255	Shipping address
OrDate	String	20	The date and time when the order was placed or created
Status	String	50	The current state or progress of the order through its lifecycle.
Note	String	50	The note provided by the customer when placing an order
TotalAmount	Int		The total cost of the order, including all items, taxes, discounts, and any other charges that may apply
PaymentMethod	String	30	The method or mode used by the customer to pay for the order
ShippingCost	Int		The total cost associated with shipping and delivering the order to the customer.
PaymentStatus	String	20	The current state of the payment for the order
OrderProduct			The OrderProduct entity is created from the many-to-many relationship between Order and Product.
ProId	String	20	Product identifier
Quantity	Int		Quantity of products that customers order
Price	int		Price of a product within each order
Discount	Float		discount of a product within each order
ImportBill			**Import Bill**
ImpId	String	20	Import bill Identifier
ImpDate	String	20	The date when the import of products into the store occurs
SupId	String	20	Supplier Identifier
StaffId	String	20	Identifier of the staff who processed or handled the import bill.
TotalAmount	int		The total monetary value of the import transaction
ImportBillProduct			The OrderProduct entity is created from the many-to-many relationship between ImportBill and Product.
ProId	String	20	Product Identifier

ImBId	String	20	Import Bill Identifier
ImportPrice	int		The price of each product when the store imports it
SellingPrice	int		The price at which the store intends to sell each product after it has been imported.
Quantity	Int		The quantity of each product when the store imports it
RemainingQuantity	int		The number of units of a specific product that are still available in the store's inventory from a particular import batch
ExpirationDate	String	50	The date after which a specific product is no longer considered safe or suitable for sale or consumption
Staff			It represents employees within the store, who can play different roles such as sales staff, warehouse staff, and store owner.
StaId	String	20	Staff identifier
StaName	String	50	Staff Name
Role	String	20	Role of staff in the system
Phone	String	15	Phone number of staff
HireDate	DateTime		The date when an employee is officially hired or begins working at the organization.
User			**Users of the system**
Uid	String	20	User identifier
ExternalUserID	String	50	Unique identifier for users from an external system or service that is integrated with your system. This could be used to track users who log in through third-party authentication services like Google, Facebook, or other external identity providers.
ExternalProvider	String		The name of the external authentication service or provider that is used to authenticate or identify the user.
Password	String	50	The password of the user if they have
UserType	String	50	It categorizes users based on their roles or access levels within the system.

News			It typically represents articles that are communicated to customers.
NewId	String	20	New identifier
Title	String	255	Title of news
Content	String	max	Content of news
CreateDate	String	20	Time creates the news
StaffId	String	20	Staff identifier of who writes news
Log			Log is an entity used to record events, actions, or changes within a system
TimeStamp	DateTime		The date and time when the log entry was created.
StaId	String	20	The identifier of the user who performed the action
EntityID	String	20	The identifier of the entity related to the action (e.g., order ID, product ID). This attribute specifies which particular instance of an entity was involved.
EntityType	String	50	The type of entity associated with the action (e.g., order, product). It indicates what kind of data was affected.
Action	String	20	The type of action or event recorded (e.g., create, update, delete). It describes the nature of the activity.
Description	String	255	A detailed description of the event or action
Cart			
CarId	String	20	A unique identifier for each cart
CreateDate	DateTime		The date and time when the cart was created
Status	String		The status of the cart (e.g., "active," "abandoned," "completed")
CartIterm			It represents an individual item within a customer's shopping cart. It captures details about each product the customer intends to purchase
ProId	String		A unique identifier for each product in the cart.
Quantity	Int		The number of units of the product added to the cart.

DateAdded	Date		The date and time when the item was added to the cart
Price	int		The price per unit of the product at the time it was added to the cart

II. Analysis models

II.1 Interaction diagrams

In this section, we will analyze objects and their interactions to realize use cases based on the MVC model's analysis pattern. Specifically, we will examine the following use cases: Manage Products, Add Product, Edit Product, Delete Product, Disable News, and Enable News. The interaction diagrams, illustrated in Figures 2-1 through 2-12 include sequence diagrams and communication diagrams that model the interactions (messages) of objects to realize these use cases.

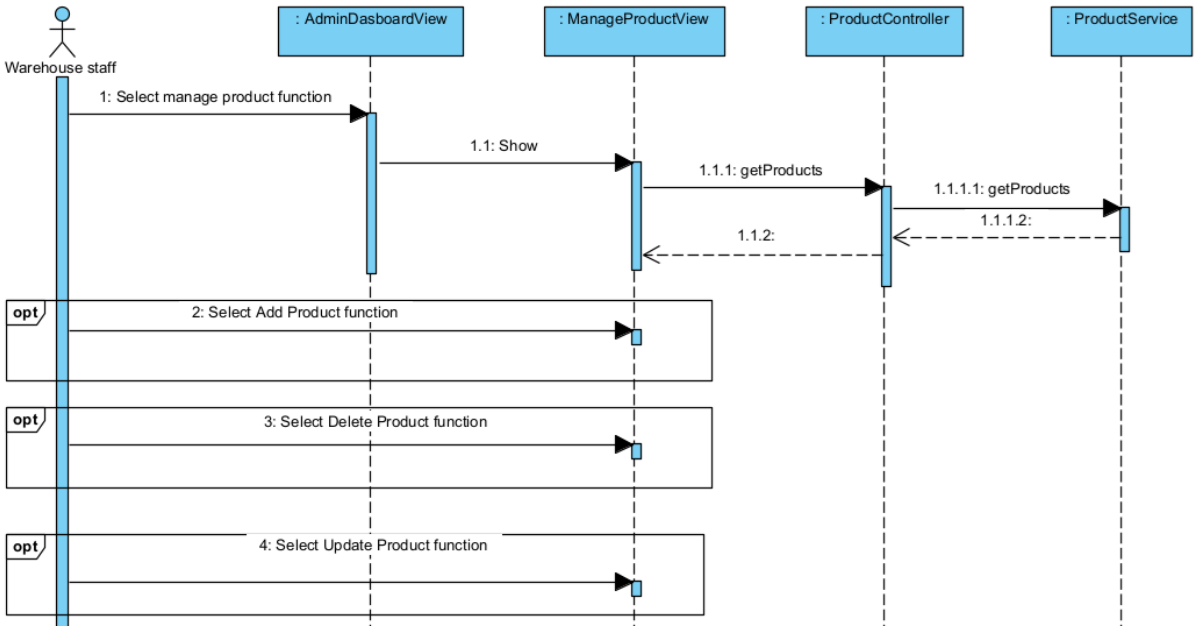

Figure II-1 Sequence Diagram for Manage Products Use Case

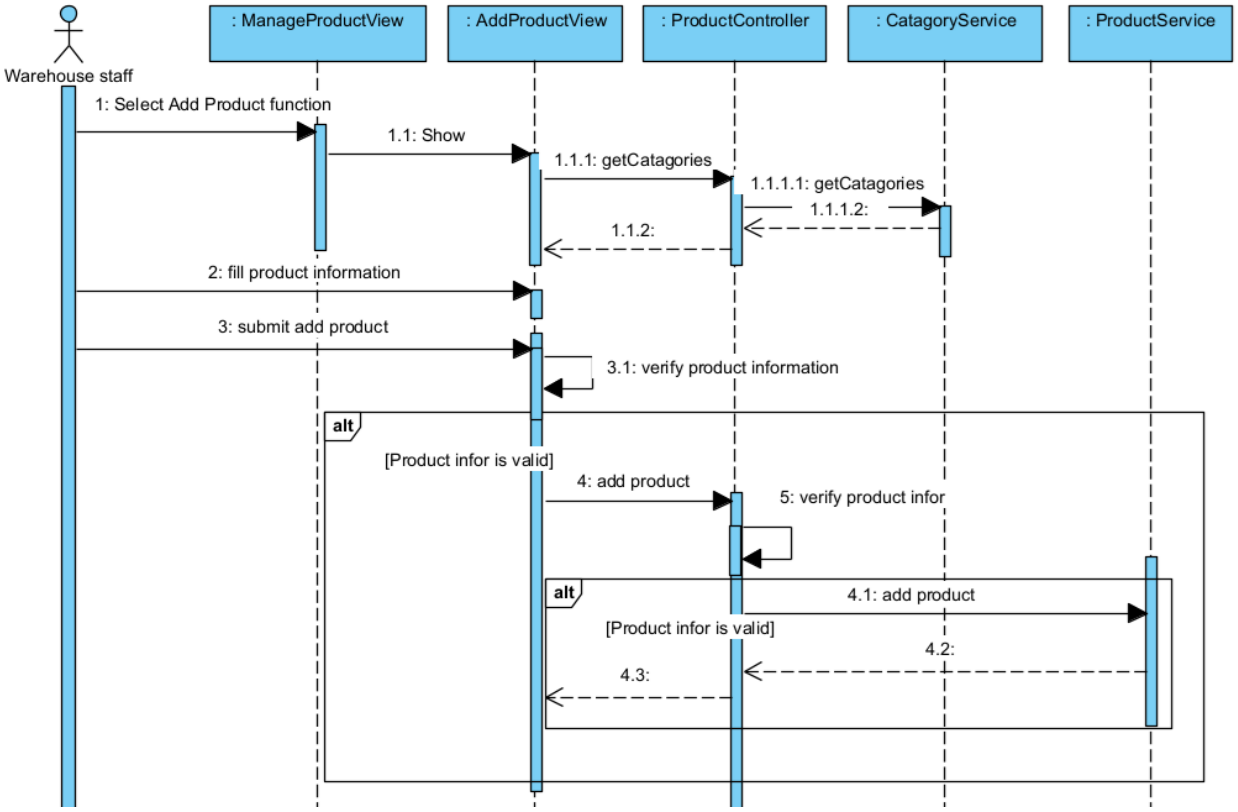

Figure II-2 Sequence Diagram for Add Product Use Case

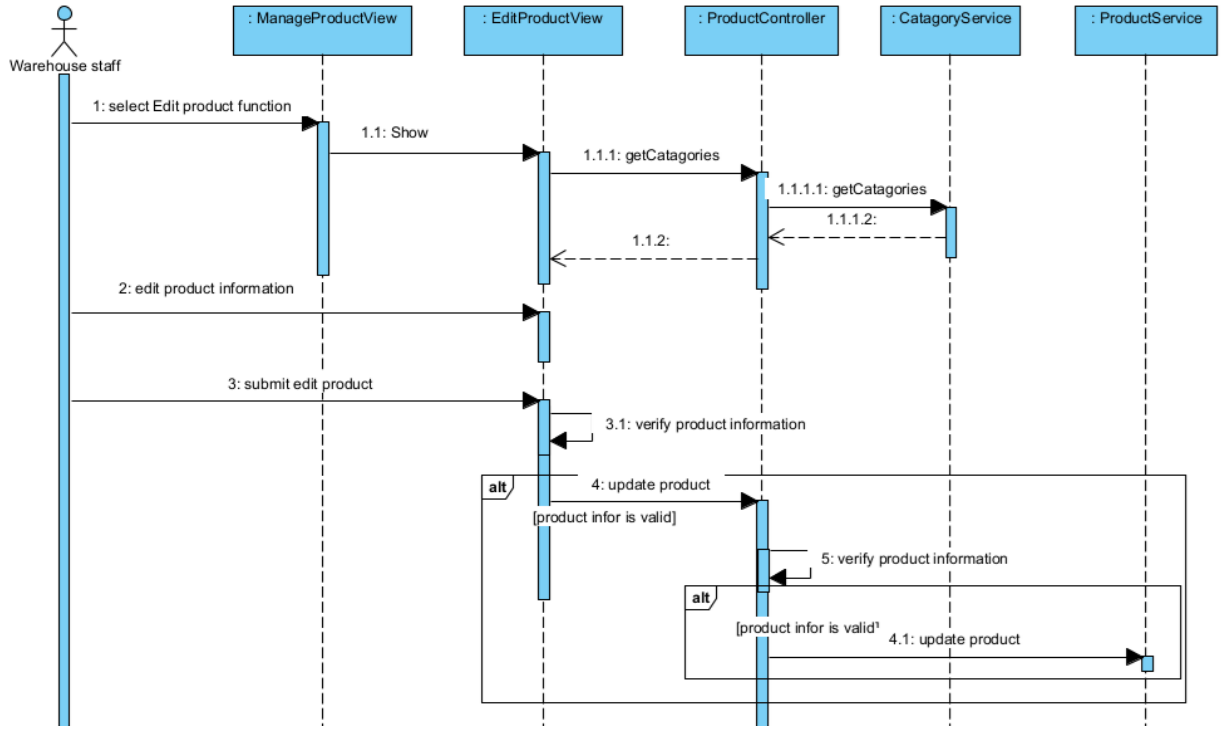

Figure II-3 Sequence Diagram for Edit Product Use Case

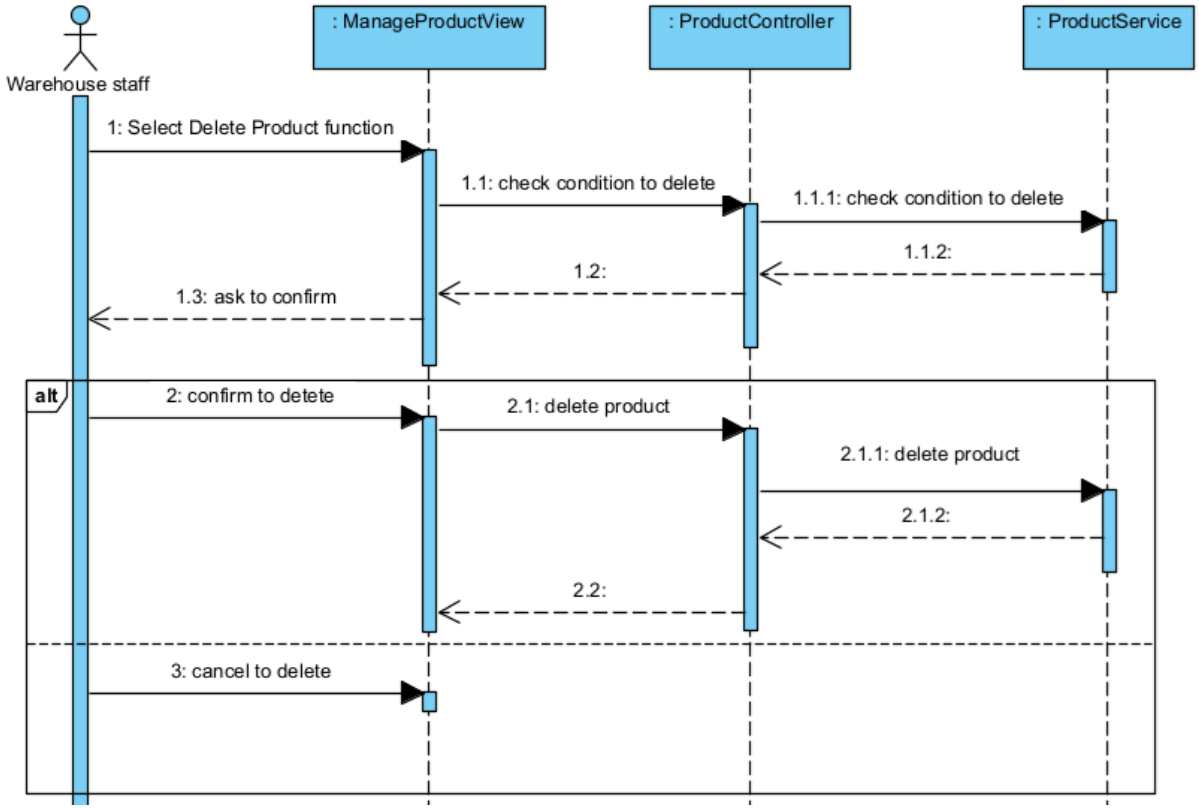

Figure II-4 Sequence Diagram for Delete Product Use Case

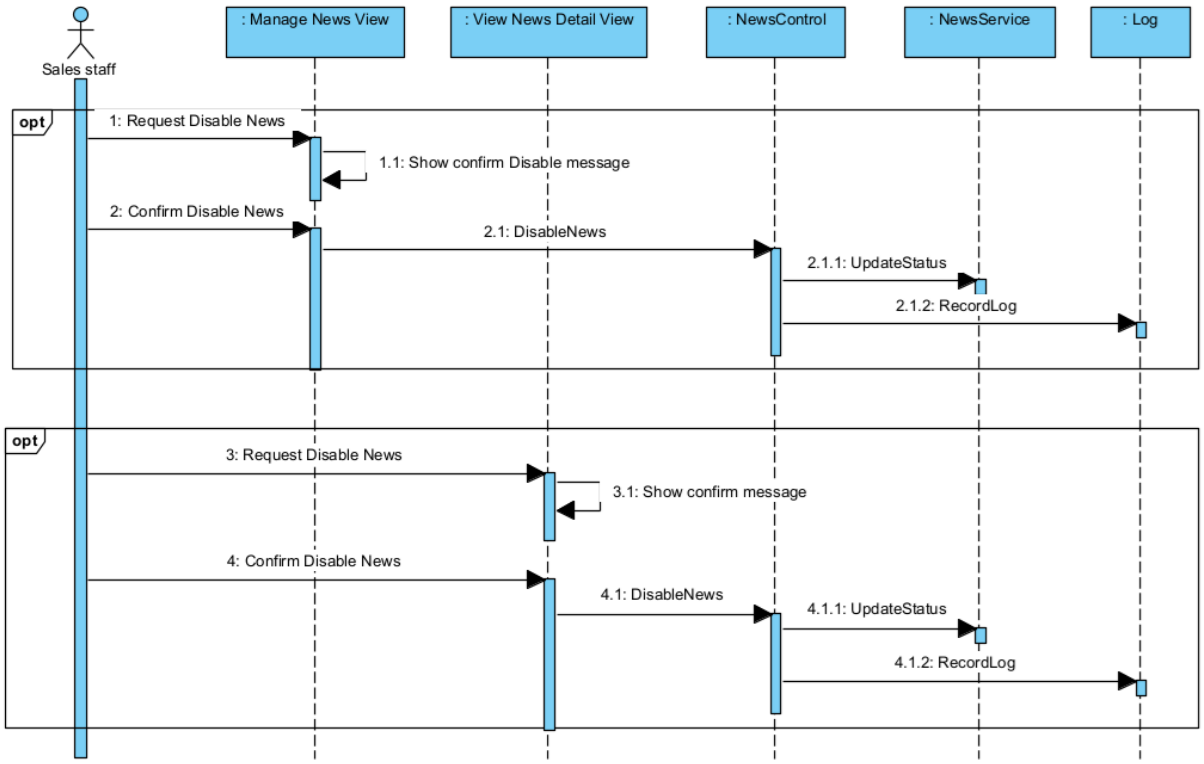

Figure II-5 Sequence Diagram for Disable News Use Case

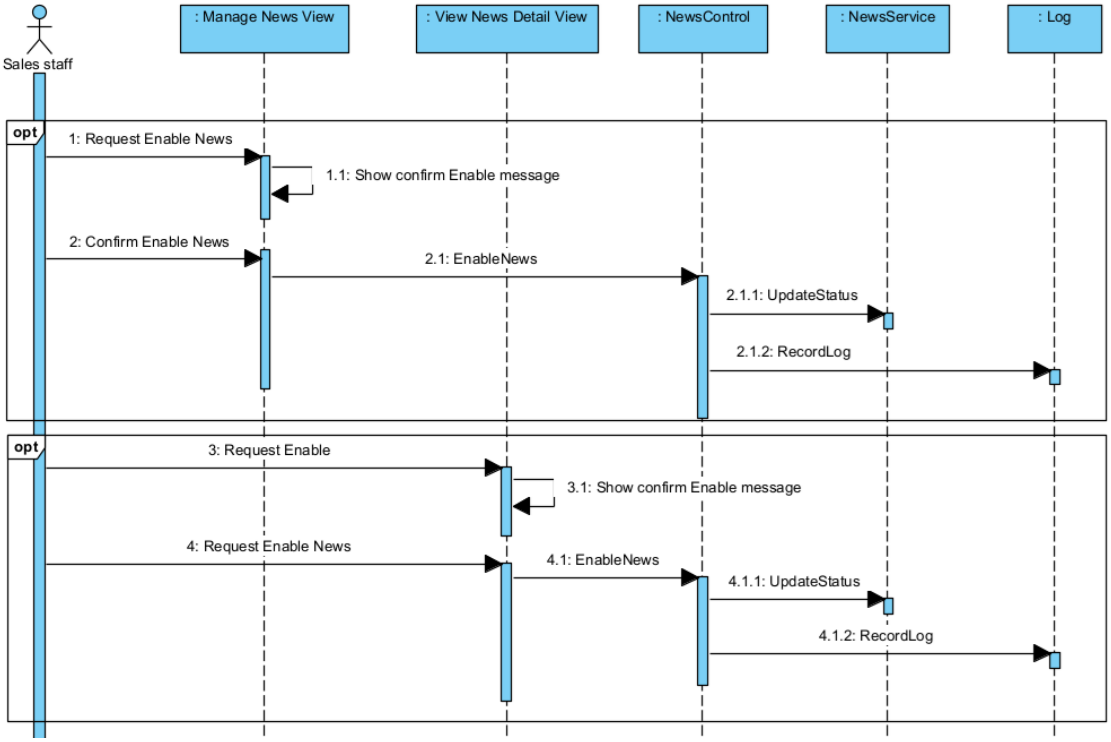

Figure II-6 Sequence Diagram for Enable News Use Case

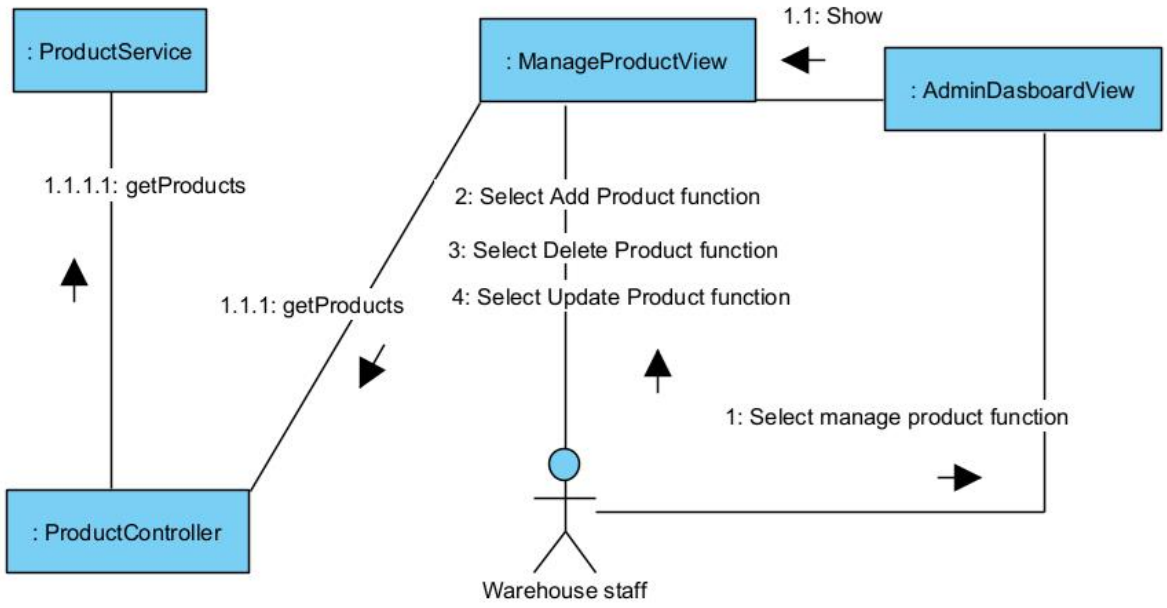

Figure II-7 Communication Diagram for Manage Products Use Case

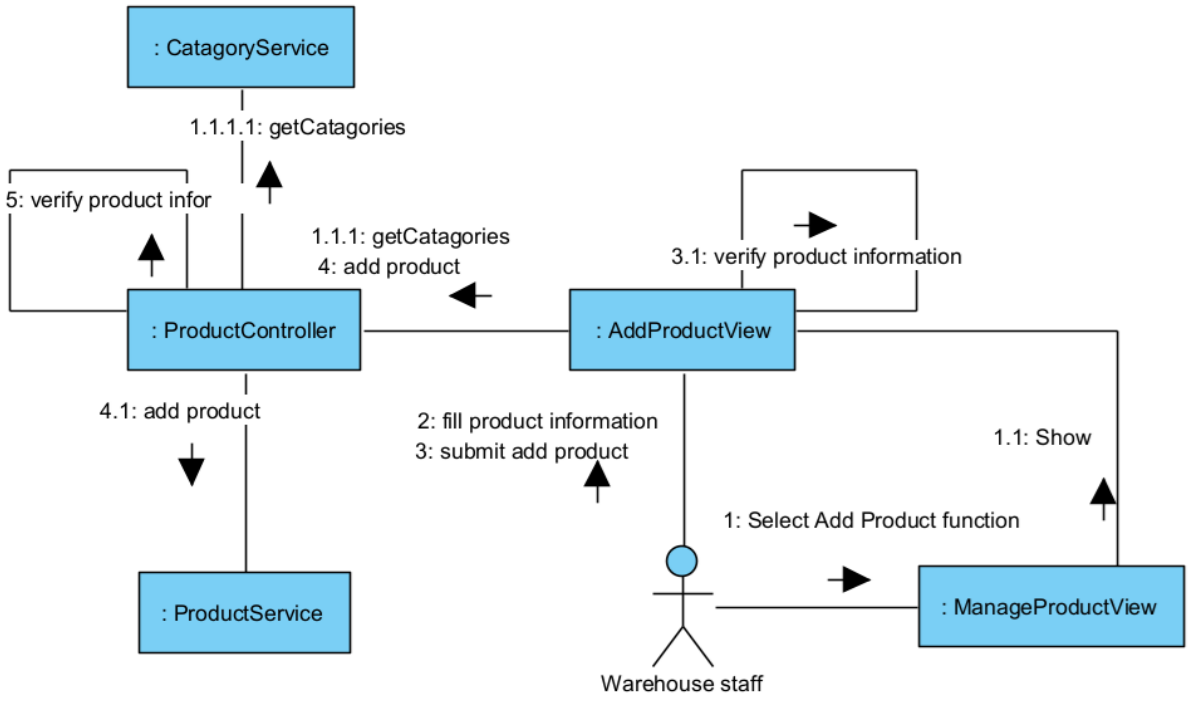

Figure II-8 Communication Diagram for Add Product Use Case

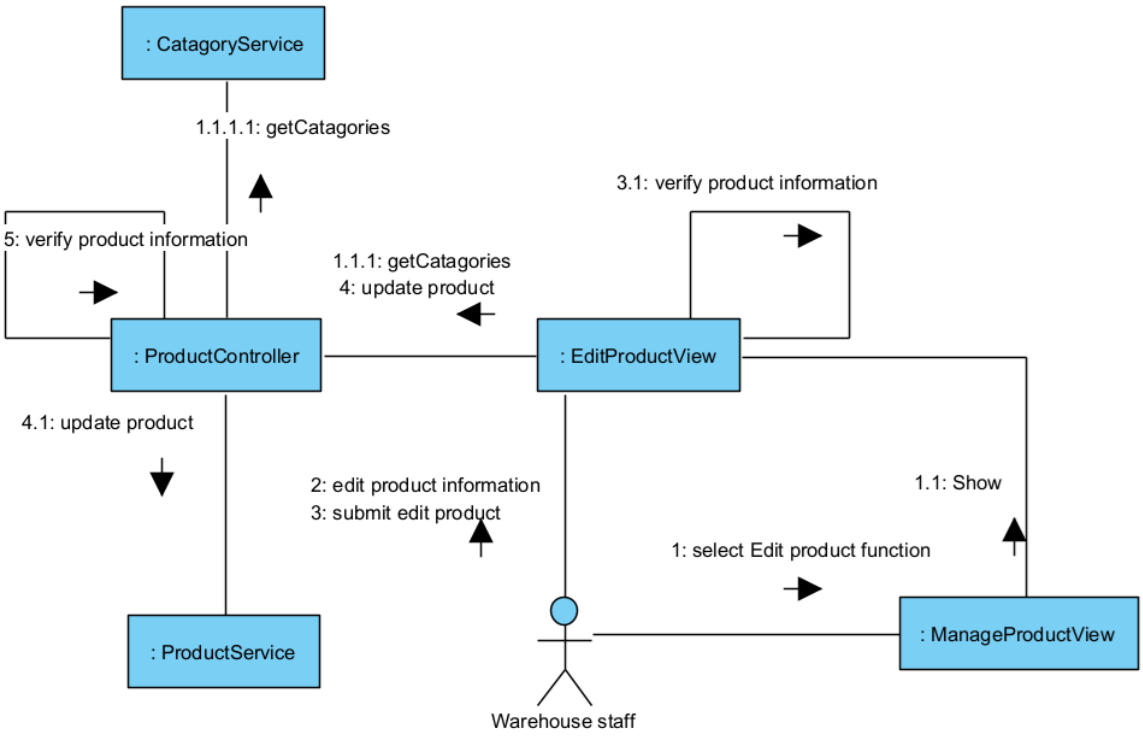

Figure II-9 Communication Diagram for Edit Product Use Case

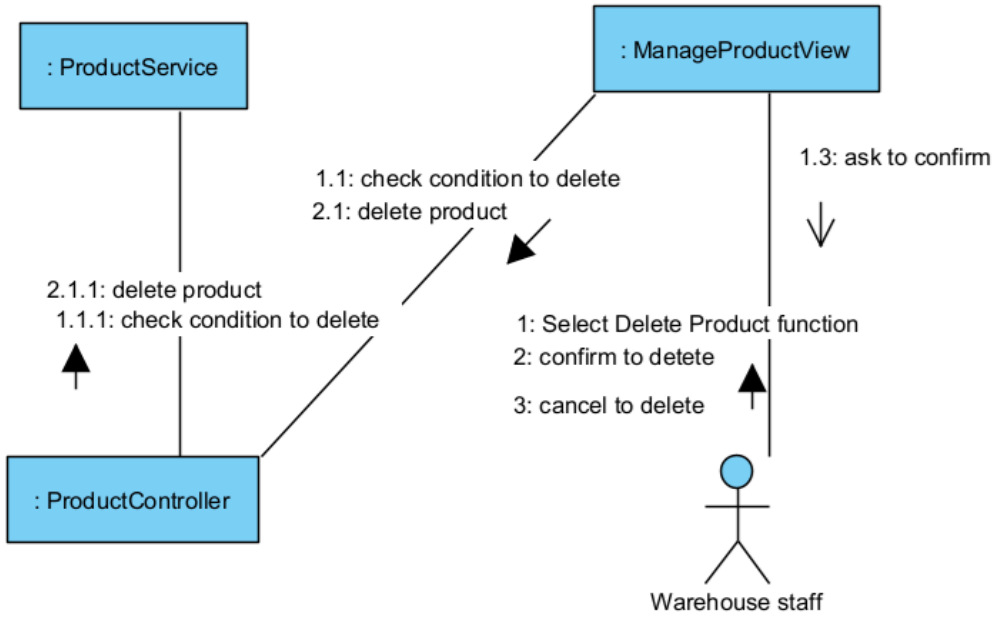

Figure II-10 Communication Diagram for Delete Product Use Case

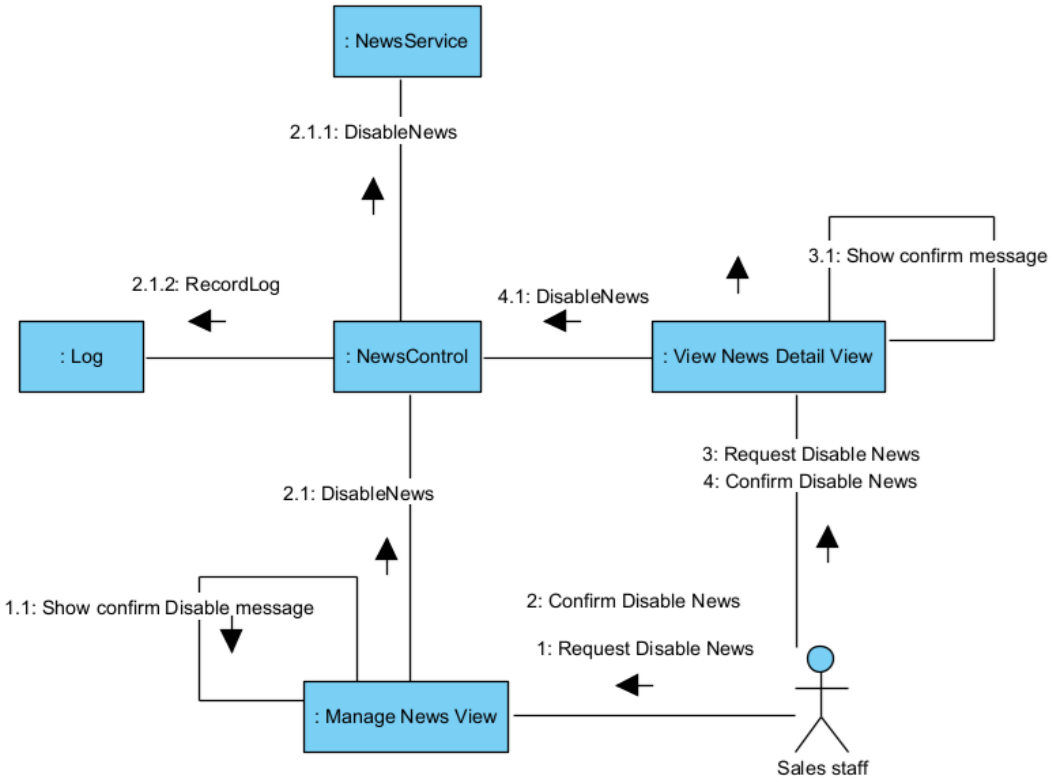

Figure II-11 Communication Diagram for Disable News Use Case

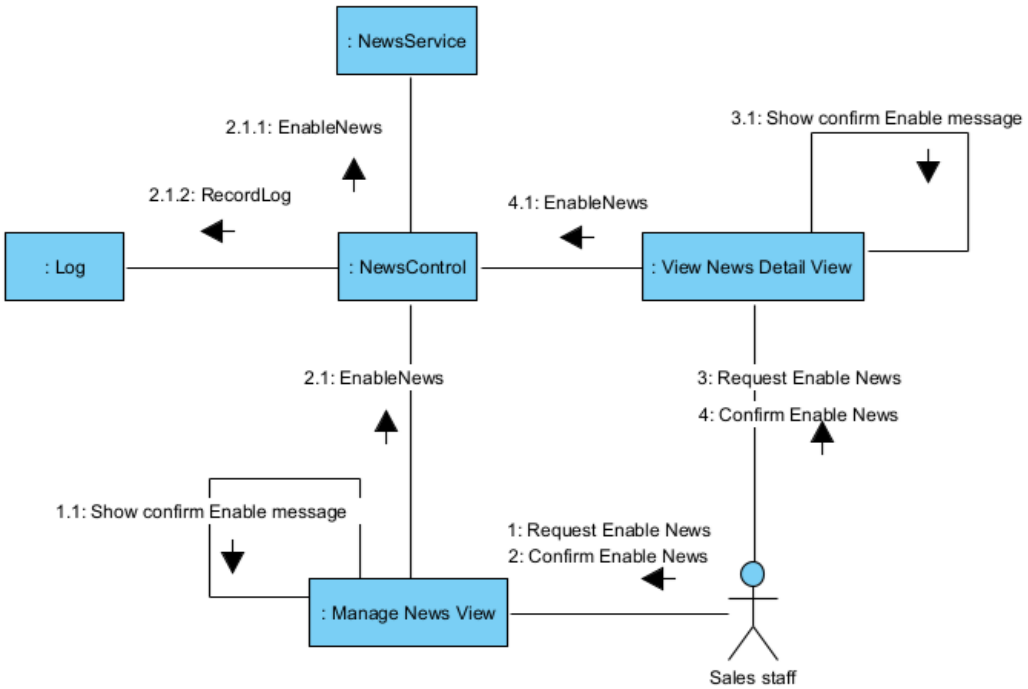

Figure II-12 Communication Diagram for EnableNews Use Case

II.2 State diagram

In object-oriented analysis and design, a finite state machine can model the states and state transitions of a system or an object. Figure 2-13 illustrates a state diagram depicting the states and state transitions of a News object. The events DisableNews and EnableNews cause state transitions to share the same names as two messages arriving at the state-dependence object in the sequence diagrams (Figures 2-5, 2-6) or the communication diagrams (Figures 2-11, 2-12).

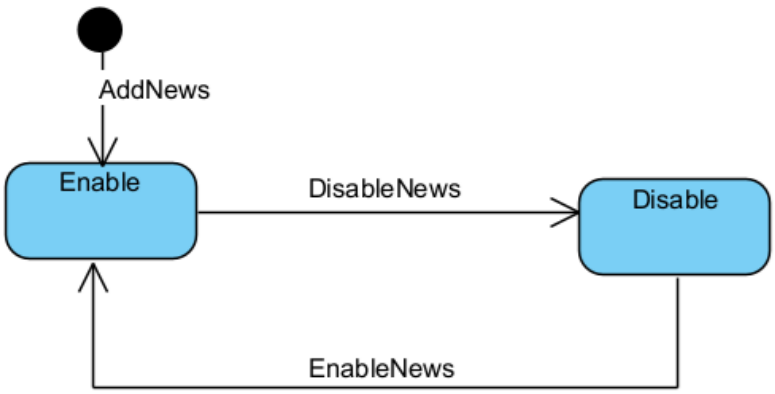

Figure II-13 State diagram model states of a News object

III. Design specification

III.1 Integrated Communication Diagrams

To transition from analysis to design and to identify the subsystems, it is necessary to synthesize an initial software design based on the analysis conducted so far. This involves integrating the use case based interaction diagrams developed as part of the dynamic model [1].

The integrated collaboration diagrams will then incorporate all interactions within the system. For large and complex systems, this integration may be performed for each subsystem individually. In this example, we will integrate the collaboration diagrams for the four use cases we have developed: Manage Products, Add Product, Edit Product, and Delete Product.

As an alternative to integrating interactive diagrams, design often involves incorporating class diagrams. This integration helps identify all the necessary classes and methods required to implement use cases. For example, Figure 3-7 shows a design-level class diagram that integrates the use cases: Manage Products, Add Product, Edit Product, and Delete Product.

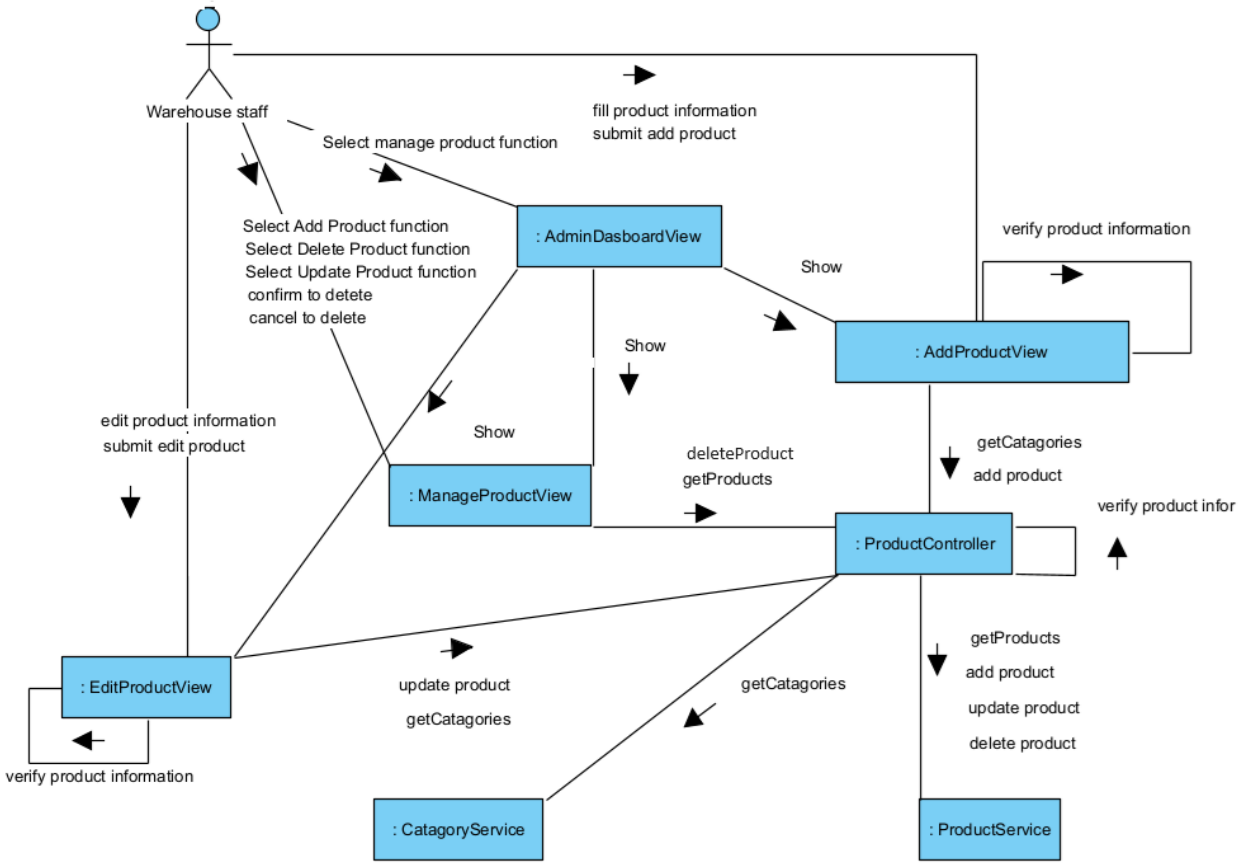

Figure III-1 Integrated communication diagram of Manage products, Add product, Edit product, Delete product.

III.2 System High-Level Design

The software architecture separates the overall structure of the system. The design of the software architecture can be depicted from different perspectives, referred to as different views. The static view of the software architecture is depicted on class diagrams. The dynamic view of the software architecture is depicted on communication diagrams. The deployment view of the software architecture is depicted deployment diagram. Another view of software architecture is the Component-Based view [1].

Based on the functional and non-functional requirements, the FSO system is designed using a Multiple Client/Single Service architecture. In this design, the FSO system is divided into two subsystems: one subsystem for customers and another subsystem for store staff. The software high-level architecture of FSO is depicted by different views as in Figures 3-2, 3-3, and 3-4.

Note: To meet the non-functional requirements NF-02 and NF-03, we need to design the interface to ensure ease of use, enable the website to run on multiple browsers, and support Responsive capabilities. Additionally, to meet the non-functional requirements NF-01 regarding

maintainability and NF-04 regarding performance, we need to design the architecture to fulfill these characteristics.

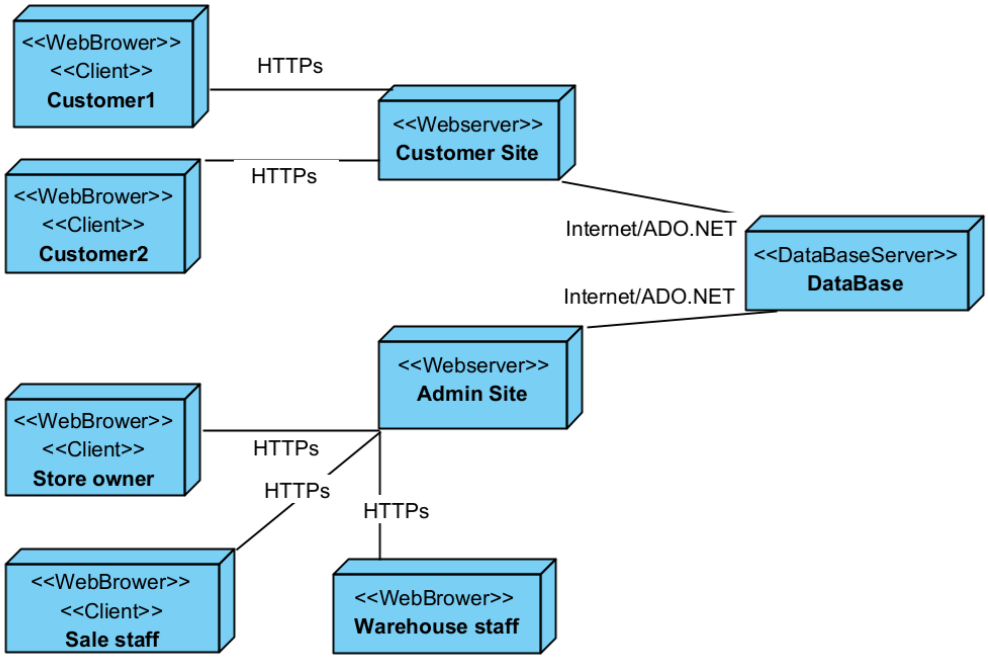

Figure III-2 Deployment diagram for the FSO system architecture

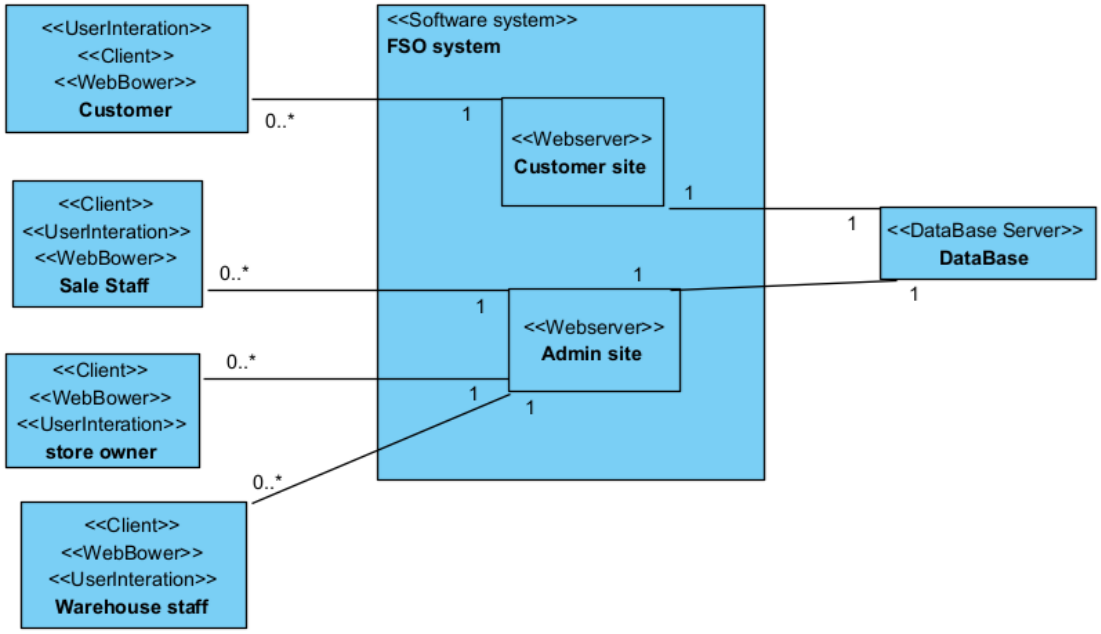

Figure III-3 Class diagram for the FSO system architecture

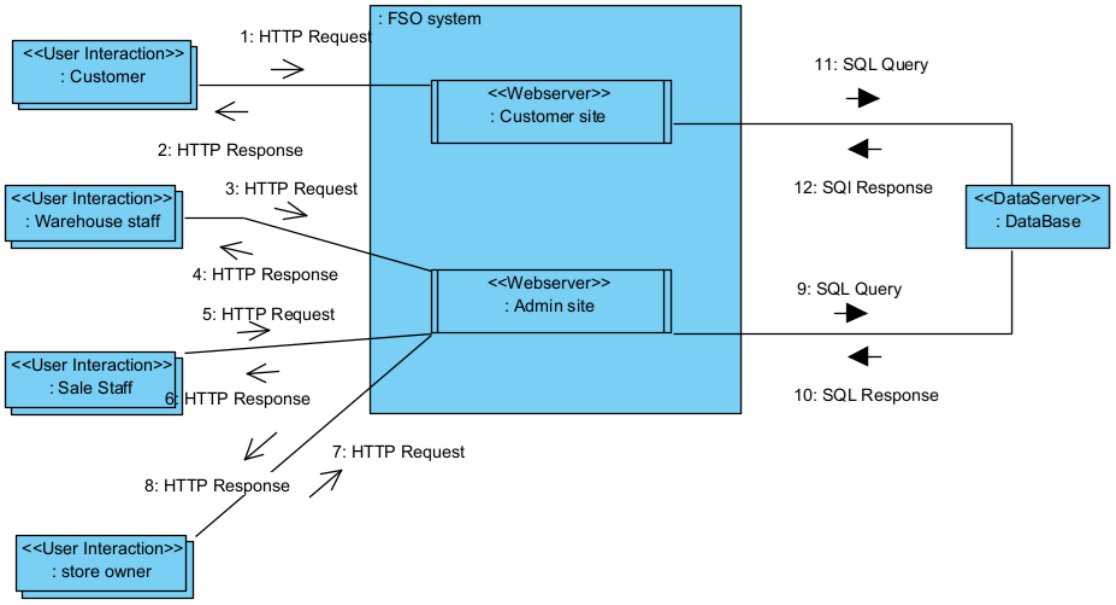

Figure III-4 Component diagram for the FSO system architecture

III.3 Component and Package Diagram

After designing the system into subsystems, the organization of classes within the subsystems into packages and components can also be determined. This organization is modeled in the corresponding package and component diagrams in UML. Figure 3-5 is the package diagram of the FSO system. This diagram shows packages and the relationships between packages. Figure 3-6 is a component diagram of the FSO system. This diagram shows components and the relationships between components.

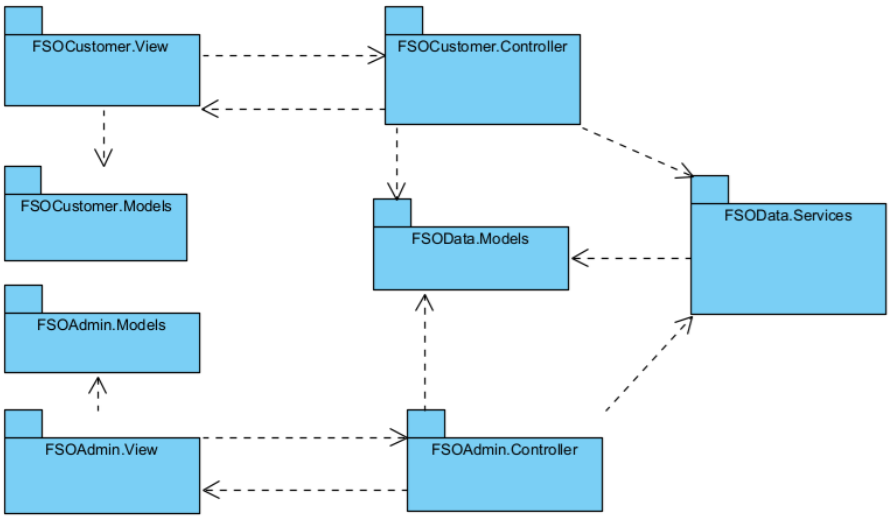

Figure III-5 Package diagram for the FSO system

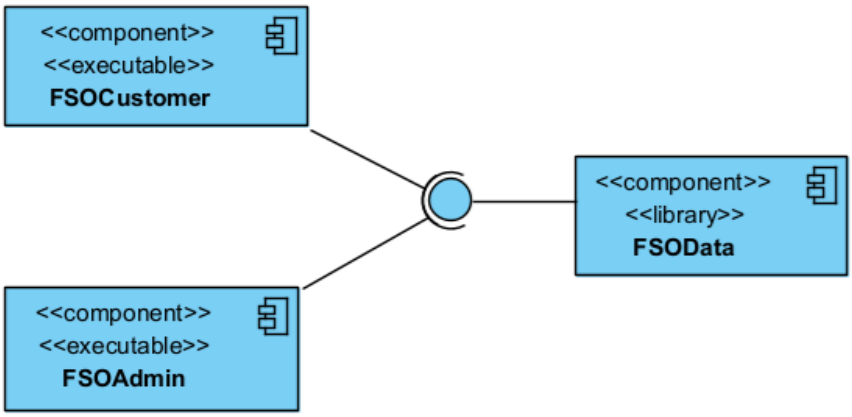

Figure III-6 Component diagram for the FSO system

III.4 Detail Design

After high-level architectural design, the classes in the analysis model are mapped to design classes. The operations of a class can be determined from either the static model and the dynamic model. Information hiding classes are categorized by stereotype: entity classes, boundary classes, control classes, and application logic classes. In design modeling, each entity class is usually split into two separate classes. The first is a database wrapper class, which hides how data is accessed, especially when using a relational database. The second is a data abstraction class, which encapsulates the data structures [1].

Figure 3-7 shows the detailed design classes for the use cases: Manage Products, Add Product, Edit Product, and Delete Product. These classes are identified based on the architecture of the FSO system, as designed in Section 3.2. The operations of these classes are defined based on the messages in the sequence diagrams for these use cases, as shown in Figures 2-1 through 2-4.

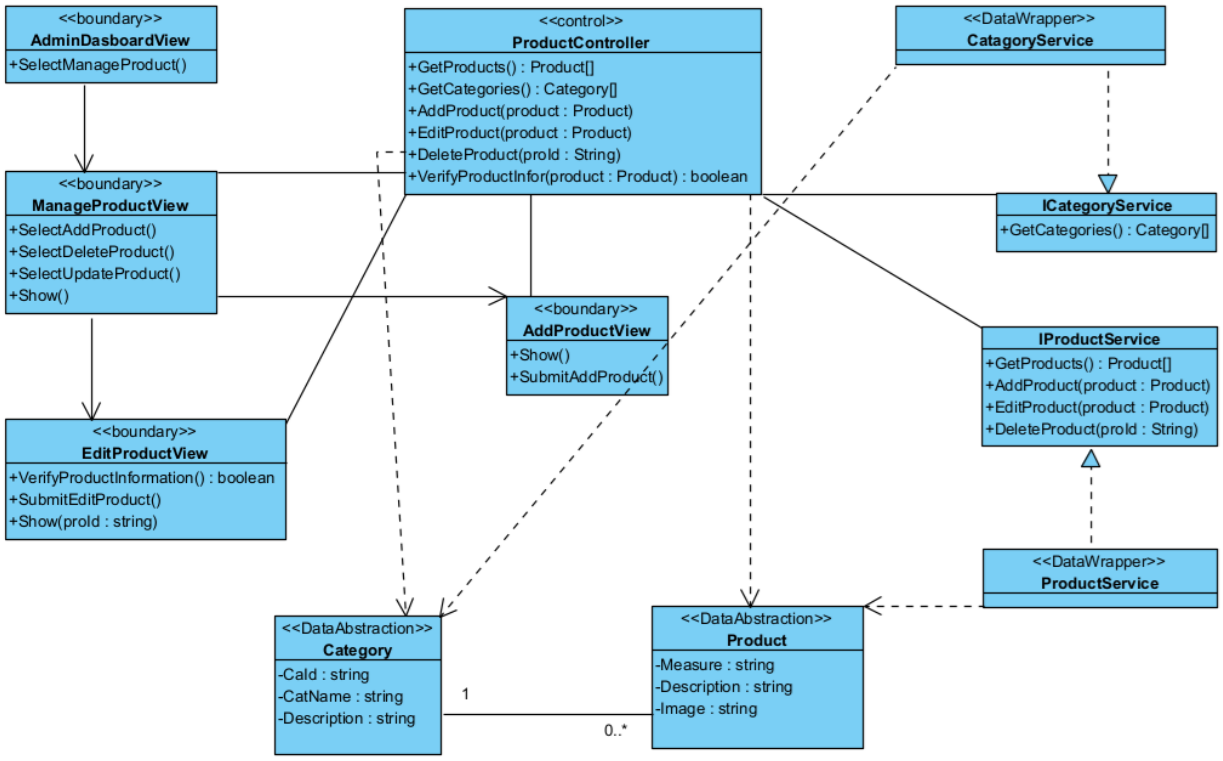

Figure III-7 Detailed Design Classes for Use Cases: Manage Products, Add Product, Edit Product, Delete Product

III.5 Database Design

The data contained in the entity classes of the static model are mapped to a database (in this system, we will design a relational database). The entity class diagram of the FSO system, shown in Figure 1-4, which models the data requirements, will be mapped to a relational database design for the system.

Supplier(SupId: nvarchar(20), Name: nvarchar(50), Address: nvarrchar(500), Phone: nvarchar (15))

Customer(CusId: nvarchar(20), CusName:nvarchar(50), PhoneNumber: nvarchar(15), Email:nvarchar(50), Address: nvarchar(500))

Staff(StaId: nvarchar(20), StaName: nvarchar(50), Role: nvarchar(20), Phone: nvarchar(15), HireDate: DateTime)

Discount(DisId: nvarchar(20), DisName: nvarchar(250), DisAmount: float, IsPercentage, StartDate: DateTime, EndDate: DateTime, IsActive: bit)

Staff(StaId: nvarchar(20), StaName: nvarchar(50), Role: nvarchar(20), Phone: nvarchar(15), HireDate: DateTime)

User(Uid: nvarchar(20), *StaId:nvarchar(20), CusId: nvarchar(20),* ExternalUserID:nvarchar(50), ExternalProvider:nvarchar(50), Password:nvarchar(50), UserType: nvarchar(50))

News(NewId: nvarchar(20), *StaffId: nvarchar(20),* Title: nvarchar(255), Content: LongTex, CreateDate: DateTime)

Log(LogId: nvarchar(20), TimeStamp: DateTime, *StaId: nvarchar(20),* EntityID: nvarchar(20), EntityType: nvarchar(50), Action: nvarchar(20), Description: nvarchar(255))

Category(CatId: nvarchar(20), CatName: nvarchar(50),Description: nvarchar(500))

Product(ProId: nvarchar(20), *CatId: nvarchar(20),* ProName: nvarchar(50), Measure: nvarchar(50), Description: nvarchar(500), Image: nvarchar(255))

Cart(CusId: nvarchar(20), CreateDate: DateTime, Status: nvarchar(20))

ShippingAdress(AddId: nvarchar(20), *CuId: nvarchar(20),* Province: nvarchar(50), District: nvarchar(50), Commune: nvarchar(50), AddressLine: nvarchar(100))

Order(OrdId: nvarchar(20), *CusId: nvarchar(20), StaId: nvarchar(20),* ShippingAddress: nvarchar(255), OrDate, Status, Note, TotalAmount, PaymentMethod, ShippingCost, PaymentStatus)

ImportBill(ImpId: nvarchar(20), *SupId: nvarchar(20), StaffId: nvarchar(20),* ImpDate: DateTime, TotalAmount: int)

CartIterm(CaIId: nvarchar(20), *CusId: nvarchar(20),* ProId: nvarchar(20), Quantity: int, DateAdded: DateTime, Price: int)

ImportBillProduct(ProId: nvarchar(20), ImBId: nvarchar(20), ImportPrice: int, SellingPrice:int, Quantity: int, RemainingQuantity: int, ManufactureDate: DateTime, ExpirationDate: DateTime)

OrderProduct(OrdId: nvarchar(20), ProId: nvarchar(20), Quantity: int, Price: int, Discount:Float)

IV. Implementation

IV.1 Map architecture to the structure of the project

In the system architecture outlined in Section 3.2, the FSO (FreshFood Online) system is designed using a Multi-client/one-server model. It is divided into two subsystems: FSOAdmin and FSOCustomer, deployed in a web environment. To manage data operations effectively, these tasks are separated into a distinct component. As illustrated in Figure 3-6 (Component Diagram), the system consists of three main components: FSOAdmin (Executable - Web app), FSOCustomer (Executable - Web app), and FSOData (Library). These components are further organized into packages, as shown in Figure 3-5. Each package groups classes with similar roles and functionalities.

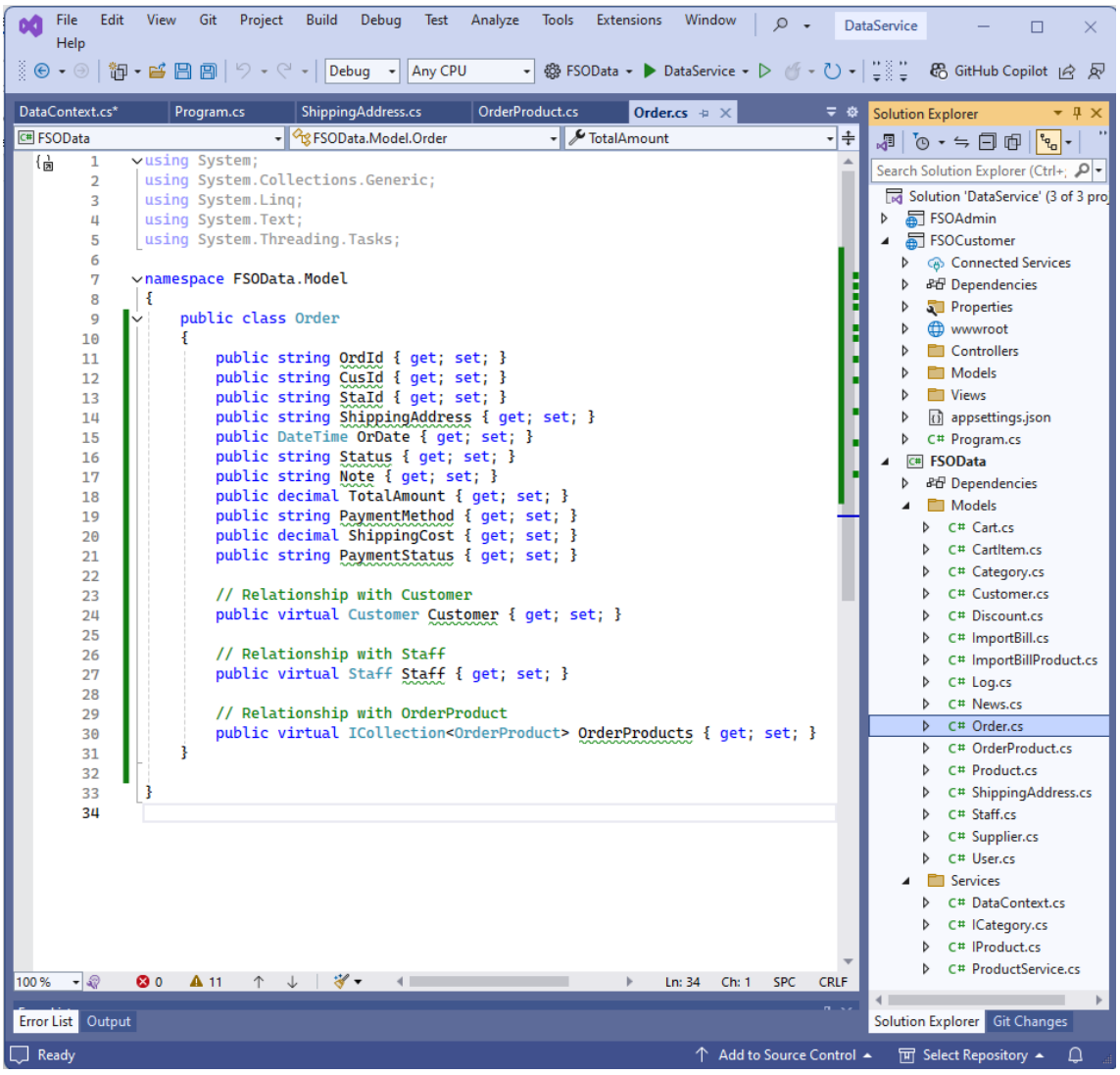

Figure IV-1 the structure of the project

IV.2 Map Class Diagram and Interaction Diagram to Code

For example, two classes (ProductService, ProductController) in the detailed design classes in Figure 3-7 are mapped into code c# as follows:

```
using FSOData.Models;
namespace FSOData.Services
{
        public interface IProductService
        {
           Task<IEnumerable<Product>> GetProducts ();
           Task AddProduct (Product product);
           Task EditProduct(Product product);
           Task DeleteProduct(string proId);
        }
}
```

```
using FSOData.Models;
namespace FSOData.Services
{
public class ProductService : IProductService
{
   private readonly DataContext _context;

   public ProductService(DataContext context)
   {
      _context = context;
   }

   // Method to get all products
```

```
    public async Task<IEnumerable<Product>> GetProducts ()
    {
        return await _context.Products.ToListAsync();
    }

    // Method to add a new product
    public async Task AddProduct (Product product)
    {
        _context.Products.Add(product);
        await _context.SaveChangesAsync();
    }

    // Method to update an existing product
    public async Task EditProductAsync(Product product)
    {
        _context.Products.Update(product);
        await _context.SaveChangesAsync();
    }
// Method to delete a product by ID
public async Task<bool> DeleteProduct (string proId)
{
    // Check if the product exists in any related tables (e.g., OrderProduct, ImportBillProduct)
    bool isReferenced = await _context.OrderProducts.AnyAsync(op => op.ProId == proId) ||
                        await _context.ImportBillProducts.AnyAsync(ibp => ibp.ProId == proId);
    if (isReferenced) return false;
    var product = await _context.Products.FindAsync(proId);
    if (product != null)
    {
        _context.Products.Remove(product);
```

```
        await _context.SaveChangesAsync();
        return true; // Deletion successful
    }
    return false; // Product not found
}
}
}

using FSOData.Models;
using FSOData.Services;
namespace FSOAdmin.Controller
{
public class ProductController: Controller
{
    private readonly IProductService _productService;
   private readonly ICategoryService _categoryService;

    // Constructor
    public ProductController(IProductService productService)
    {
        _productService = productService;
    }

    // GET: Product/Manage
    public async Task<IActionResult> Manage()
    {
        var products = await _productService.GetProducts();
        return View(products);
    }
```

```
// GET: Product/GetProduct/{id}
public async Task<IActionResult> GetProduct(string id)
{
    var product = await _productService.GetProduct(id);
    if (product == null)
    {
        return NotFound();
    }
    return PartialView("_ProductDetails", product);
}

// GET: Product/GetCategories
public async Task<IActionResult> GetCategories()
{
    var categories = await _ categoryService.GetCategories();
    return Json(categories);
}

// GET: Product/Add
public IActionResult Add()
{
    return PartialView("_AddProductModal");
}

// POST: Product/Add
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Add(Product product)
```

```
{
    if (ModelState.IsValid)
    {
        bool isVerified = _productService.VerifyProductInfor(product);
        if (!isVerified)
        {
            ModelState.AddModelError("", "Product information is not valid.");
            return PartialView("_AddProductModal", product);
        }

        await _productService.AddProduct(product);
        return RedirectToAction("Manage");
    }

    return PartialView("_AddProductModal", product);
}

// GET: Product/Edit/{id}
public async Task<IActionResult> Edit(string id)
{
    var product = await _productService.GetProduct(id);
    if (product == null)
    {
        return NotFound();
    }

    return PartialView("_EditProductModal", product);
}
```

```
// POST: Product/Edit
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Product product)
{
    if (ModelState.IsValid)
    {
        bool isVerified = _productService.VerifyProductInfor(product);
        if (!isVerified)
        {
            ModelState.AddModelError("", "Product information is not valid.");
            return PartialView("_EditProductModal", product);
        }

        await _productService.EditProduct(product);
        return RedirectToAction("Manage");
    }

    return PartialView("_EditProductModal", product);
}

// POST: Product/Delete/{id}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(string id)
{
    var product = await _productService.GetProduct(id);
    if (product == null)
    {
```

```
            return NotFound();
        }

        bool isDeleted = await _productService.DeleteProduct(id);
        if (isDeleted)
        {
            return Json(new { success = true, message = "Product deleted successfully." });
        }
        return Json(new { success = false, message = "Failed to delete the product. It may be referenced in other records." });
    }
}
}
```

V. Applying Alternative Architecture Patterns

Assuming that the functional requirements remain unchanged, the system is now being expanded to incorporate additional characteristics related to scalability and reusability. This enhancement allows the system to support the integration of new user-facing applications, including the potential development of a mobile app.

V.1 Applying the service-oriented architecture

NF-05 Reusability: The system should be built in a way that allows us to easily use the same code and features across different platforms. This means that the same functionality can be adapted for both the web and mobile apps, saving time and effort when adding new platforms or updating existing ones.

The current system architecture described in Section 3.2 does not fully meet the requirements of NF-05. To address this, the FSOData component must be restructured as a distributed component and designed using a service-oriented architecture (SOA). This adjustment will allow FSOData to function as a distributed service, thereby enhancing the system's ability to achieve Reusability, traceability, modularity, and effective maintenance. The software architecture is redesigned as shown below.

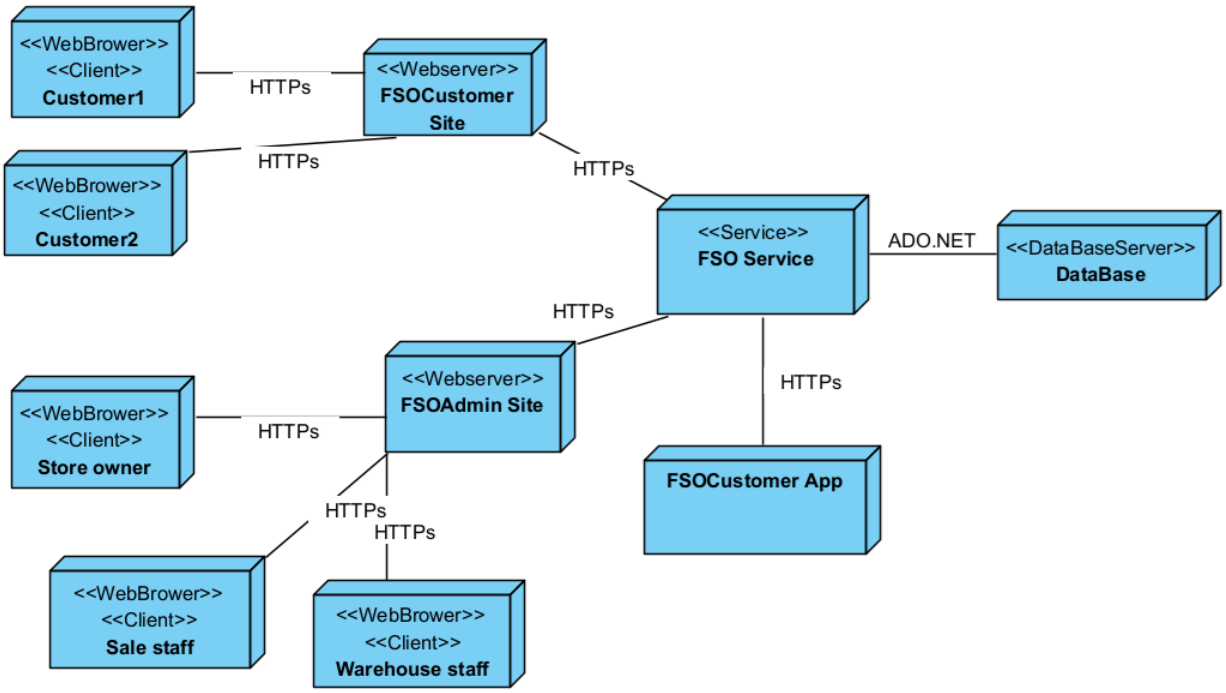

Figure V-1 Deployment Diagram for the FSO System Architecture Based on Service-Oriented Architecture

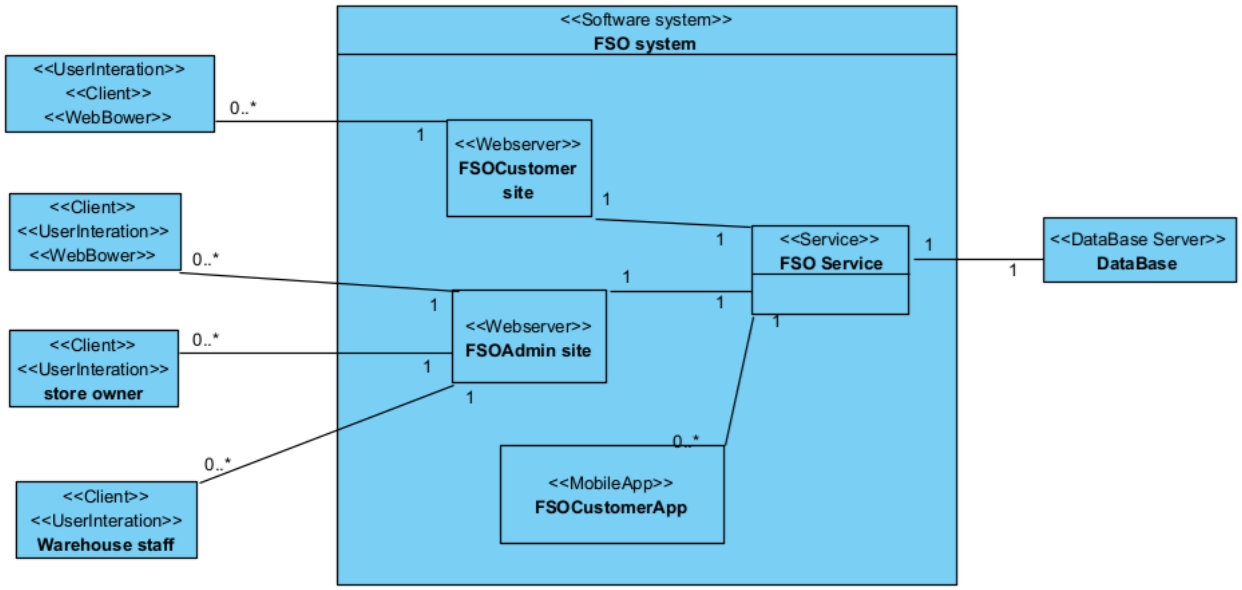

Figure V-2 Class Diagram for the FSO System Architecture Based on Service-Oriented Architecture

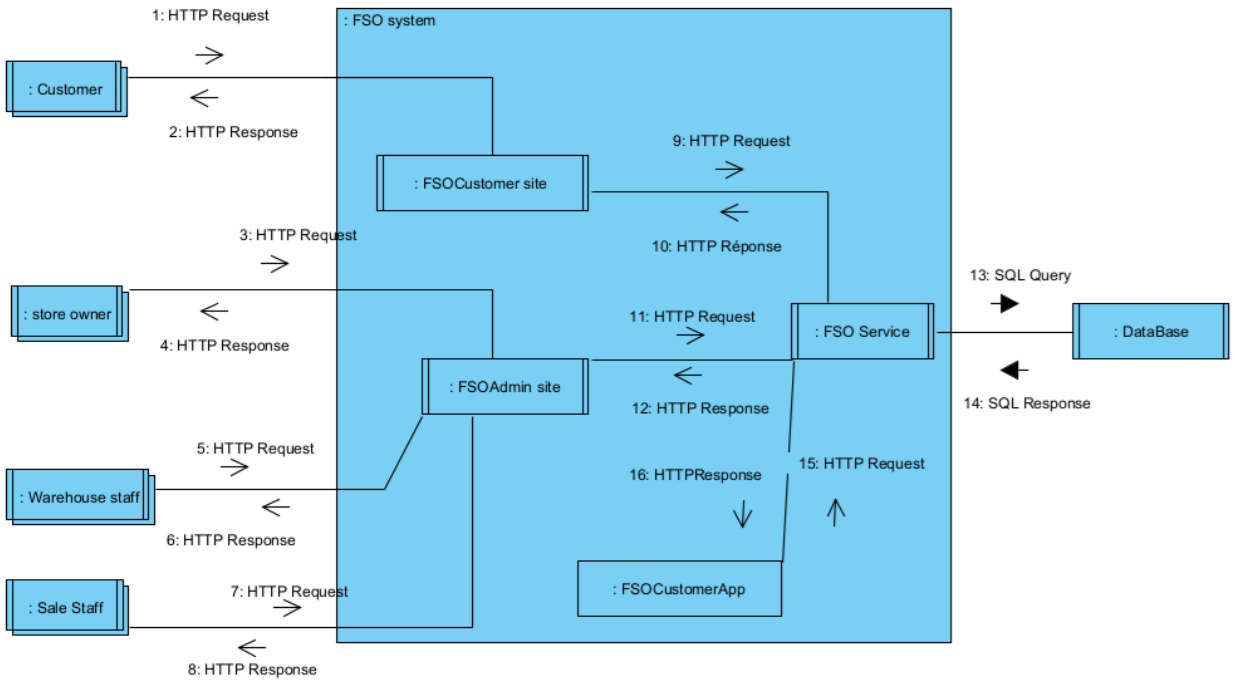

Figure V-3 Component Diagram for the FSO System Architecture Based on Service-Oriented Architecture

V.2 Applying Service Discovery Pattern in the service-oriented architecture

Assuming that user requirements change and the store's scale may later expand into a chain of stores with an increasing number of users, the system will need to address an additional non-functional requirement, NF-06, as follows:

NF-06: The system must be designed to accommodate an increasing number of stores and users while maintaining performance and reliability. This includes the ability to scale both horizontally and vertically. The architecture should support the addition of new stores and handle a growing volume of transactions and concurrent users without significant performance degradation. The system should implement load balancing to evenly distribute traffic and employ scalable database solutions to manage increasing amounts of data effectively.

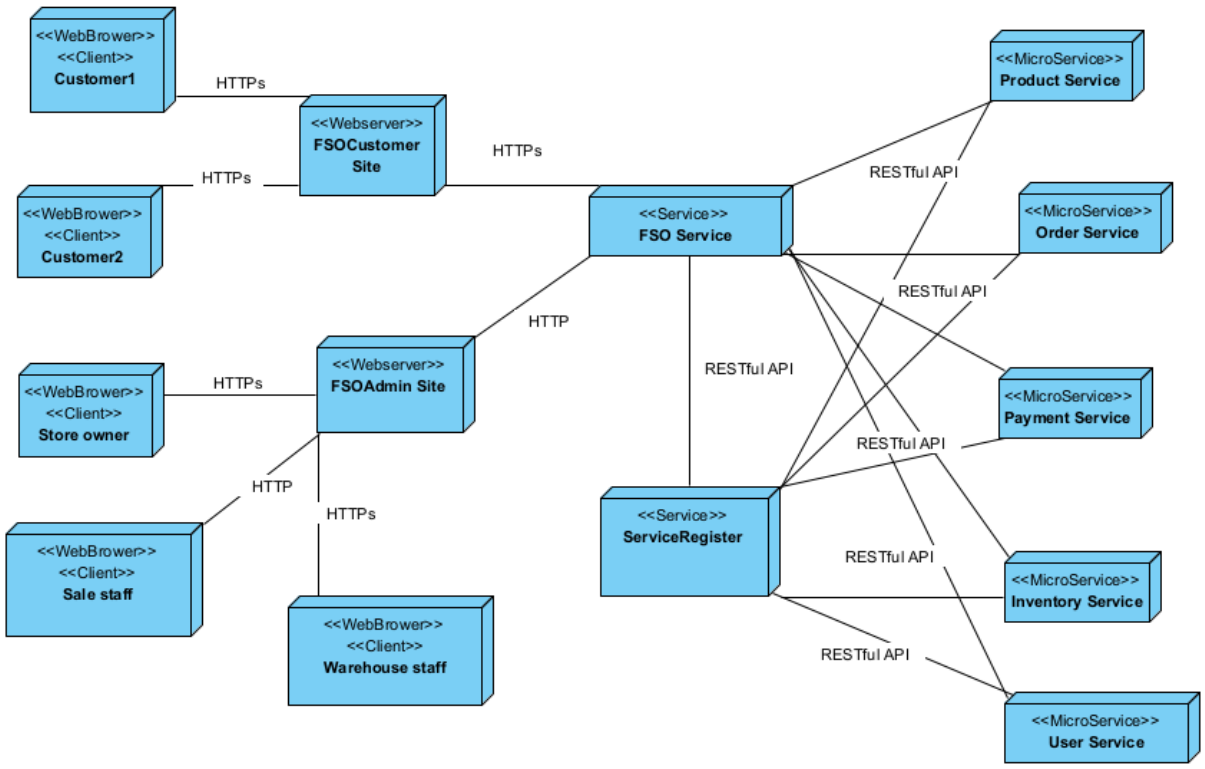

Figure V-4 Deployment Diagram for the FSO System Architecture Based on Microservice Architecture

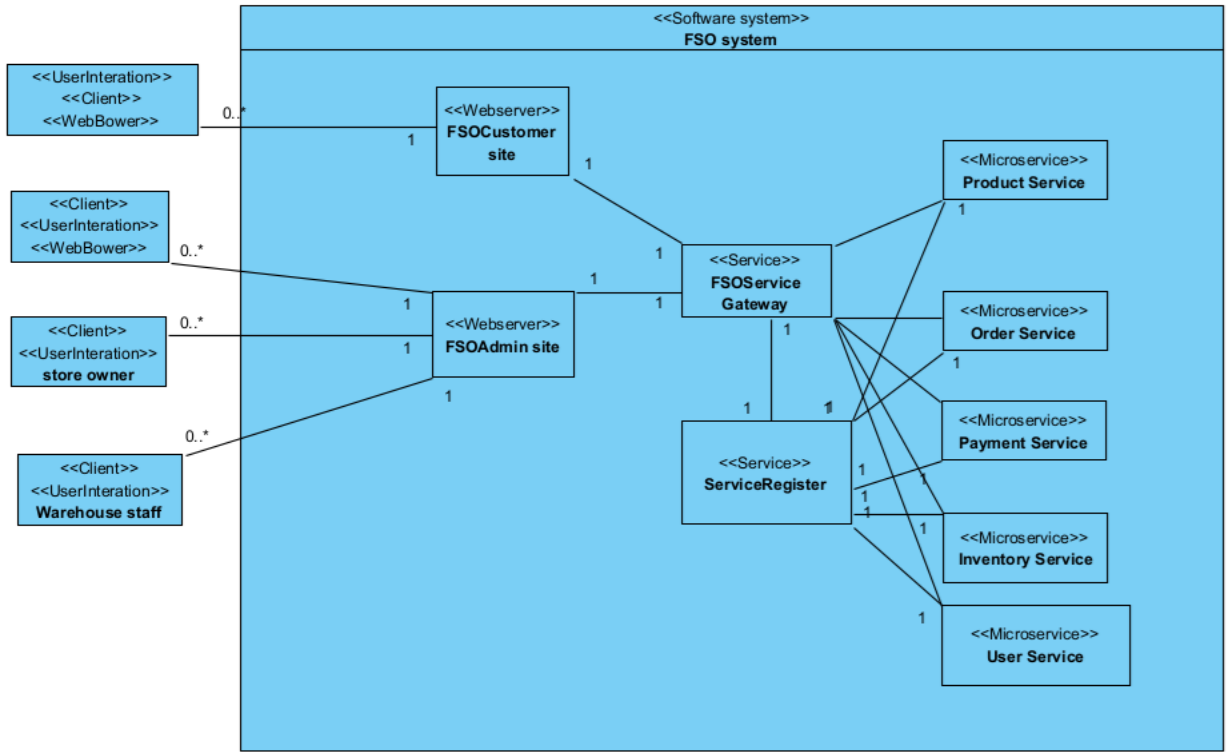

Figure V-5 Class Diagram for the FSO System Architecture Based on Microservice Architecture

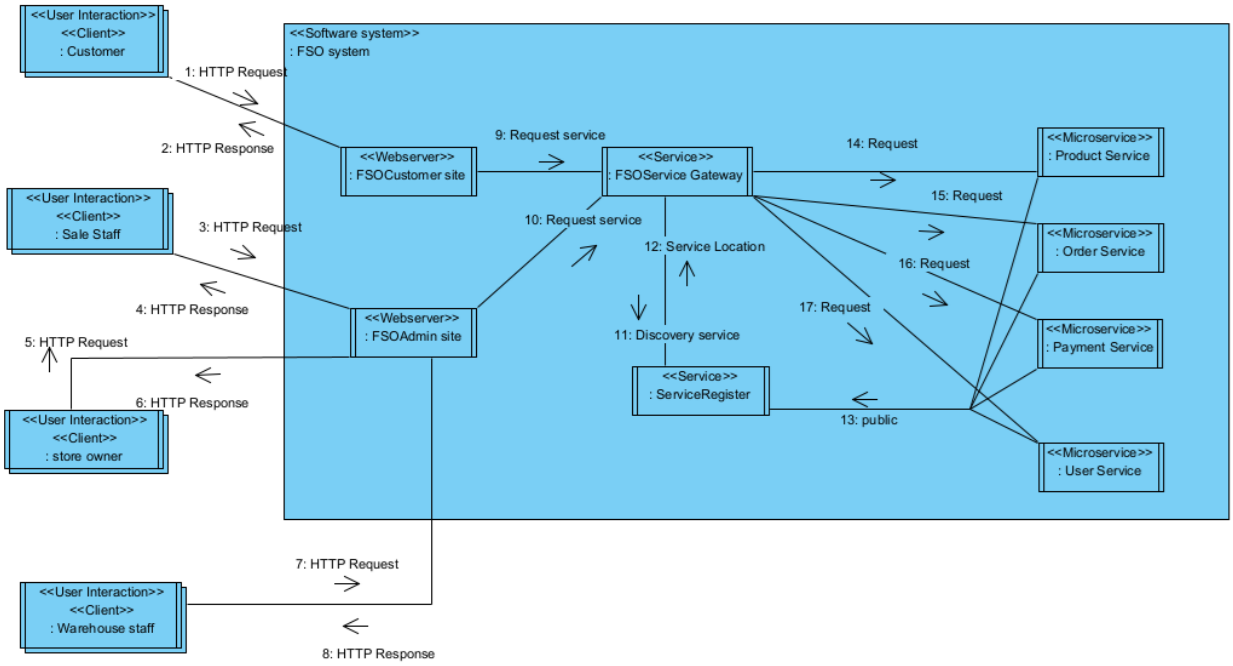

Figure V-6 Component Diagram for the FSO System Architecture Based on Microservice Architecture