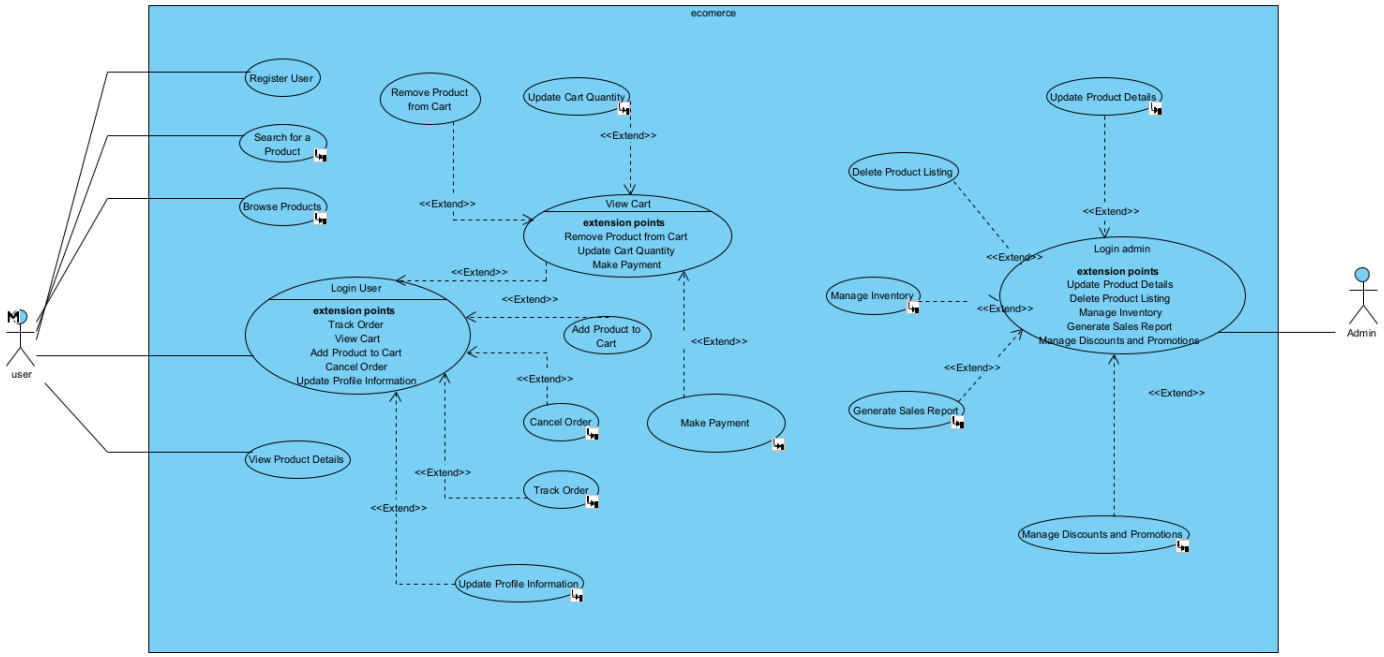
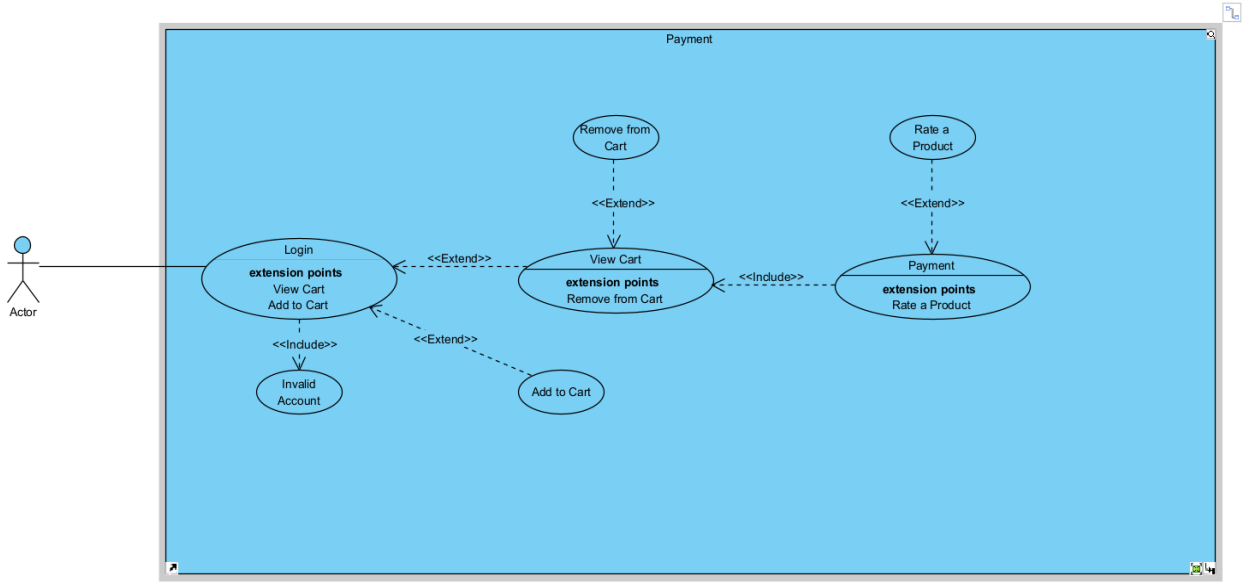
Lương Mạnh Hòa – B21DCCN378

1. Determine requirements
    - Define use cases and draw use case diagram in two levels: General level (>=20 use cases) and Level 1 (detail)
        - General level (>=20 use cases)



        - Level 1 (detail)



- Write 30 scenarios and 30 user stories + acceptance criteria
  - 30 scenarios

Use Case ID:	UC-1		
Use Case Name:	Register User		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer registers for an account to make purchases.
Preconditions:	Customer has no account.
Postconditions:	Account is created, and the customer is logged in.
Priority:	High
Frequency of Use:	Common
Normal Course of Events:	1. Customer navigates to the "Register" page. 2. Customer enters personal details and creates a password. 3. System creates the account and logs the customer in.
Alternative Courses:	If the email is already registered, prompt the customer to log in.
Exceptions:	None
Includes:	None

Special Requirements:	None
Assumptions:	Customer provides valid information.
Notes and Issues:	None

Use Case ID:	UC-2		
Use Case Name:	Login		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer logs into their account to make purchases.		
Preconditions:	Customer has an existing account.		
Postconditions:	Customer is logged in and can make purchases.		
Priority:	High		
Frequency of Use:	Common		
Normal Course of Events:	1. Customer navigates to the login page. 2. Customer enters email and password. 3. System authenticates the credentials and logs the customer in.		

Alternative Courses:	If the credentials are incorrect, the system prompts the user to try again.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Customer knows their credentials.
Notes and Issues:	None

Use Case ID:	UC-3		
Use Case Name:	Browse Products		
Created By:	Quan		
Date Created:	16/11/2024		

Actor:	Customer
Description:	The customer browses through the available products.
Preconditions:	Customer is logged in.
Postconditions:	Customer views a list of products available for purchase.
Priority:	High
Frequency of Use:	Common

Normal Course of Events:	1. Customer selects a category (e.g., electronics, books). 2. System displays a list of products in the chosen category.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Product listings are up-to-date.
Notes and Issues:	None

Use Case ID:	UC-4		
Use Case Name:	Search for a Product		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer searches for a specific product by name or category.		
Preconditions:	Customer is logged in.		
Postconditions:	Customer sees a list of products that match the search query.		

Priority:	Medium
Frequency of Use:	Common
Normal Course of Events:	3. Customer enters a keyword or category in the search bar. 4. System displays relevant search results.
Alternative Courses:	If no results are found, the system suggests related products.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Search terms are relevant and spelled correctly.
Notes and Issues:	None

Use Case ID:	UC-5		
Use Case Name:	Add Product to Cart		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer adds a product to their shopping cart.		

Preconditions:	Customer has selected a product.
Postconditions:	Product is added to the shopping cart.
Priority:	High
Frequency of Use:	Common
Normal Course of Events:	1. Customer selects a product to view its details. 2. Customer clicks "Add to Cart." 3. System adds the product to the cart.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Product is available in stock.
Notes and Issues:	None

Use Case ID:	UC-6		
Use Case Name:	View Cart		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer views the products in their shopping cart.
Preconditions:	Customer has added products to the cart.
Postconditions:	Customer sees the list of products in the cart.
Priority:	High
Frequency of Use:	Common
Normal Course of Events:	1. Customer clicks on the "Cart" icon. 2. System displays the contents of the cart with product details.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	The cart is accurately updated.
Notes and Issues:	None

Use Case ID:	UC-7
Use Case Name:	Remove Product from Cart

Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer removes a product from their shopping cart.
Preconditions:	Customer has added products to the cart.
Postconditions:	Product is removed from the cart.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer clicks the "Remove" button next to the product. 2. System removes the product from the cart.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	The customer wishes to change their cart contents.
Notes and Issues:	None

Use Case ID:	UC-8		
Use Case Name:	Update Cart Quantity		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer changes the quantity of an item in their cart.
Preconditions:	Customer has added products to the cart.
Postconditions:	The quantity of the product in the cart is updated.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer clicks the quantity box next to the product. 2. Customer updates the quantity. 3. System updates the cart with the new quantity.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None

Assumptions:	Product availability is sufficient for the requested quantity.
Notes and Issues:	None

Use Case ID:	UC-9		
Use Case Name:	Checkout		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer proceeds to checkout to complete their purchase.		
Preconditions:	Customer has items in their cart.		
Postconditions:	Customer enters payment details, and an order is created.		
Priority:	High		
Frequency of Use:	Common		
Normal Course of Events:	1. Customer clicks "Proceed to Checkout." 2. System prompts the customer to enter shipping and payment details. 3. Customer submits the order. 4. System processes the payment and confirms the order.		

Alternative Courses:	If payment fails, the system prompts the customer to try again.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Payment information is valid and complete.
Notes and Issues:	None

Use Case ID:	UC-10		
Use Case Name:	Make Payment		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer enters payment details to complete the purchase.		
Preconditions:	Customer has reached the checkout stage.		
Postconditions:	Payment is processed, and the order is confirmed.		
Priority:	High		

Frequency of Use:	Common
Normal Course of Events:	1. Customer selects a payment method (credit card, PayPal, etc.). 2. Customer enters payment details. 3. System processes the payment and confirms the transaction.
Alternative Courses:	If the payment is declined, the customer is asked to retry or choose another method.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Payment system is functioning properly.
Notes and Issues:	None

Use Case ID:	UC-11		
Use Case Name:	Track Order		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer tracks their order status after it has been shipped.		

Preconditions:	Order has been shipped.
Postconditions:	Customer is provided with the current status of their order.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to the "Order History" page. 2. Customer selects the order they wish to track. 3. System provides the shipping status and tracking details.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Shipping service updates tracking information.
Notes and Issues:	None

Use Case ID:	UC-12		
Use Case Name:	Cancel Order		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer cancels an order before it has been shipped.
Preconditions:	Order has been placed but not yet shipped.
Postconditions:	Order is canceled, and any payments are refunded.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to the order details and selects "Cancel Order." 2. System confirms the cancellation and processes a refund if applicable.
Alternative Courses:	If the order has already been shipped, cancellation is not possible.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Cancellation policy is clear and enforced.
Notes and Issues:	None

Use Case ID:	UC-13

Use Case Name:	Return a Product		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer returns a product they have received.
Preconditions:	Product return policy is still valid.
Postconditions:	Product return is processed, and refund or replacement is initiated.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to "Order History" and selects "Return" for the product. 2. System provides return instructions and updates the return status.
Alternative Courses:	If the return policy has expired, return is denied.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Return policy is clear and followed.

Notes and Issues:	None

Use Case ID:	UC-14		
Use Case Name:	Leave a Product Review		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer leaves a review and rating for a purchased product.		
Preconditions:	Customer has purchased the product.		
Postconditions:	Review is submitted and visible on the product page.		
Priority:	Medium		
Frequency of Use:	Occasionally		
Normal Course of Events:	1. Customer navigates to the product page. 2. Customer selects "Write a Review" and provides a rating and comments. 3. System submits the review and displays it on the product page.		
Alternative Courses:	None		
Exceptions:	None		

Includes:	None
Special Requirements:	None
Assumptions:	Review system is functioning properly.
Notes and Issues:	None

Use Case ID:	UC-15		
Use Case Name:	View Product Details		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer views the detailed information about a product.		
Preconditions:	Customer has selected a product.		
Postconditions:	Customer views detailed product information.		
Priority:	High		
Frequency of Use:	Common		
Normal Course of Events:	1. Customer clicks on a product. 2. System displays product details such as description, price, images, and specifications.		

Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Product details are accurate and complete.
Notes and Issues:	None

Use Case ID:	UC-16		
Use Case Name:	Cancel Order		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer cancels an order before it has been shipped.		
Preconditions:	Order has been placed but not yet shipped.		
Postconditions:	Order is canceled, and any payments are refunded.		
Priority:	Medium		

Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to the order details and selects "Cancel Order." 2. System confirms the cancellation and processes a refund if applicable.
Alternative Courses:	If the order has already been shipped, cancellation is not possible.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Cancellation policy is clear and enforced.
Notes and Issues:	None

Use Case ID:	UC-17		
Use Case Name:	Return a Product		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer returns a product they have received.		

Preconditions:	Product return policy is still valid.
Postconditions:	Product return is processed, and refund or replacement is initiated.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to "Order History" and selects "Return" for the product. 2. System provides return instructions and updates the return status.
Alternative Courses:	If the return policy has expired, return is denied.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Return policy is clear and followed.
Notes and Issues:	None

Use Case ID:	UC-18		
Use Case Name:	Apply Coupon Code		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer applies a discount coupon during checkout.
Preconditions:	Customer has a valid coupon.
Postconditions:	Discount is applied to the total price.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer enters the coupon code at checkout. 2. System validates the coupon and applies the discount.
Alternative Courses:	If the coupon is invalid, the system displays an error message.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Customer has a valid coupon code.
Notes and Issues:	None

Use Case ID:	UC-19
Use Case Name:	Track Order

Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer tracks the shipping status of their order.
Preconditions:	Order has been shipped.
Postconditions:	Customer receives updated tracking information.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to the "Order History" page. 2. Customer clicks "Track Order." 3. System provides the current shipping status and tracking number.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Shipping service updates tracking details.
Notes and Issues:	None

Use Case ID:	UC-20		
Use Case Name:	Update Profile Information		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer
Description:	The customer updates their profile information, such as email or address.
Preconditions:	Customer is logged in.
Postconditions:	Customer's profile information is updated.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Customer navigates to "Profile" page. 2. Customer updates the desired fields (email, phone number, address). 3. System saves the updated information.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None

Assumptions:	Customer provides valid information.
Notes and Issues:	None

Use Case ID:	UC-21		
Use Case Name:	View Order History		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Customer		
Description:	The customer views their previous orders.		
Preconditions:	Customer has made at least one order.		
Postconditions:	Customer can view the details of all their past orders.		
Priority:	Medium		
Frequency of Use:	Occasionally		
Normal Course of Events:	1. Customer navigates to "Order History" page. 2. System displays a list of past orders, with details such as dates and items.		
Alternative Courses:	None		
Exceptions:	None		

Includes:	None
Special Requirements:	None
Assumptions:	Order history is correctly maintained.
Notes and Issues:	None

Use Case ID:	UC-22		
Use Case Name:	Create New Product Listing		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Staff		
Description:	The staff adds a new product to the site.		
Preconditions:	Staff is logged in with appropriate privileges.		
Postconditions:	The new product is listed on the website.		
Priority:	High		
Frequency of Use:	Occasionally		

Normal Course of Events:	1. Staff selects "Create New Product" from the admin dashboard. 2. Staff enters product details (name, price, description, images). 3. System saves the product and displays it on the site.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Product information is accurate and complete.
Notes and Issues:	None

Use Case ID:	UC-23		
Use Case Name:	Update Product Details		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Staff		
Description:	The staff updates the details of an existing product.		
Preconditions:	Staff is logged in with appropriate privileges.		
Postconditions:	Product details are updated on the site.		

Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Staff selects an existing product to update. 2. Staff edits product details (e.g., price, description). 3. System updates the product information.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Product details are valid and complete.
Notes and Issues:	None

Use Case ID:	UC-24		
Use Case Name:	Delete Product Listing		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Staff		
Description:	The staff deletes an existing product listing from the site.		

Preconditions:	Staff is logged in with appropriate privileges.
Postconditions:	The product is removed from the website.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Staff selects the product to delete. 2. Staff confirms the deletion. 3. System removes the product from the website.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Staff has the necessary permissions.
Notes and Issues:	None

Use Case ID:	UC-25		
Use Case Name:	Manage Inventory		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Staff
Description:	The staff manages product inventory levels, updating stock quantities.
Preconditions:	Staff is logged in with appropriate privileges.
Postconditions:	Product inventory is updated.
Priority:	High
Frequency of Use:	Regularly
Normal Course of Events:	1. Staff selects the product whose inventory needs to be updated. 2. Staff adjusts the quantity in stock. 3. System updates the inventory count.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Staff has accurate inventory data.
Notes and Issues:	None

Use Case ID:	UC-26

Use Case Name:	Process Payment		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Staff
Description:	The staff processes payment for an order, confirming payment receipt.
Preconditions:	Order is confirmed and payment details are valid.
Postconditions:	Payment is confirmed, and the order is ready for shipment.
Priority:	High
Frequency of Use:	Regularly
Normal Course of Events:	1. Staff reviews the order for payment confirmation. 2. Staff processes the payment through the system. 3. System confirms the payment and updates the order status to "Paid."
Alternative Courses:	None
Exceptions:	If the payment is declined, the order status remains "Pending."
Includes:	None
Special Requirements:	Payment gateway must be functional.
Assumptions:	Payment system is integrated and secure.

Notes and Issues:	None

Use Case ID:	UC-27		
Use Case Name:	Update Order Status		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Staff		
Description:	The staff updates the order status, such as "Shipped," "Delivered," or "Returned."		
Preconditions:	Order is in the system and processing.		
Postconditions:	The order status is updated accordingly.		
Priority:	High		
Frequency of Use:	Regularly		
Normal Course of Events:	1. Staff selects an order from the dashboard. 2. Staff updates the order status (e.g., "Shipped," "Delivered"). 3. System updates the order status and notifies the customer.		
Alternative Courses:	None		
Exceptions:	None		

Includes:	None
Special Requirements:	None
Assumptions:	Order status tracking is available.
Notes and Issues:	None

Use Case ID:	UC-28		
Use Case Name:	Generate Sales Report		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Manager		
Description:	The manager generates a sales report to review total sales and performance.		
Preconditions:	Sales data is available in the system.		
Postconditions:	Sales report is generated and available for review.		
Priority:	High		
Frequency of Use:	Monthly		

Normal Course of Events:	1. Manager navigates to the "Reports" section. 2. Manager selects the "Generate Sales Report" option. 3. System generates the report and displays the total sales figures.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Sales data is correctly recorded.
Notes and Issues:	None

Use Case ID:	UC-29		
Use Case Name:	Manage Discounts and Promotions		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024
Actor:	Manager		
Description:	The manager adds, updates, or removes discounts and promotional offers.		
Preconditions:	Manager is logged in and has appropriate privileges.		

Postconditions:	Discount or promotion is successfully added, updated, or removed.
Priority:	Medium
Frequency of Use:	Occasionally
Normal Course of Events:	1. Manager navigates to the "Promotions" section. 2. Manager adds, updates, or removes a discount or promotion. 3. System applies the promotion to relevant products or categories.
Alternative Courses:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Promotions are valid for the selected products.
Notes and Issues:	None

Use Case ID:	UC-30		
Use Case Name:	User Login		
Created By:	Quan	Last Updated By:	Quan
Date Created:	16/11/2024	Date Last Updated:	16/11/2024

Actor:	Customer, Staff, Manager
Description:	The user logs into the system using their credentials.
Preconditions:	User has an account and valid login credentials.
Postconditions:	User is authenticated and redirected to the appropriate dashboard.
Priority:	High
Frequency of Use:	Common
Normal Course of Events:	1. User navigates to the login page. 2. User enters their username and password. 3. System authenticates the credentials. 4. User is redirected to their respective dashboard (Customer, Staff, or Manager).
Alternative Courses:	If login fails, user is prompted to re-enter credentials.
Exceptions:	If the account is locked, the user is notified and redirected to reset password.
Includes:	None
Special Requirements:	None
Assumptions:	User has a registered account with valid credentials.
Notes and Issues:	None

- 30 user stories + acceptance criteria

User story	**Acceptance Criteria**

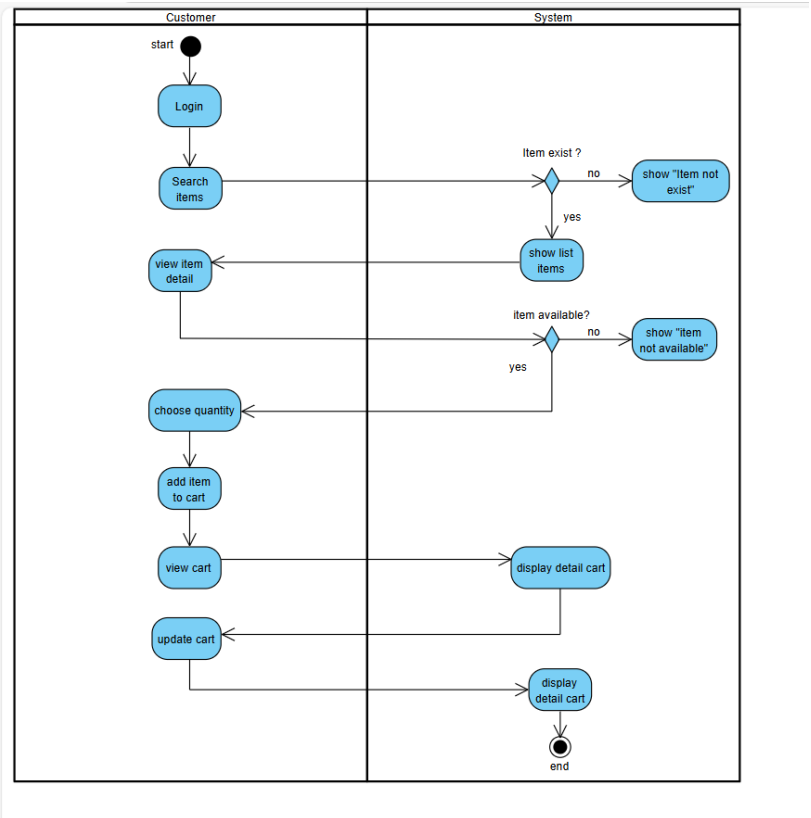
As a shopper, I want to view a list of products so I can select some to purchase.	- See a thumbnail image of each product - Click to view detail for a product - Add to cart from detail page - Search for a product - View product by category
As a shopper, I want to review my cart so I can make adjustments before checking out.	- View quantities and items in the cart - See total cost before tax and shipping - Remove items - Adjust quantity of items - Click to navigate to a product detail page
As a shopper, I want to check out so I can get my products shipped to me.	- Trigger checkout from any page if there are items in the cart - Enter a shipping address - Enter a billing address - Enter a credit card number
As a shopper, I want to view detailed product information so I can make an informed decision.	- See product description, price, and images - See customer reviews and ratings - See available sizes, colors, or models
As a shopper, I want to filter products by category so I can easily find what I’m looking for.	- Select product categories - Apply filters by price, brand, rating, etc. - See the filtered product list immediately
As a shopper, I want to view related products so I can find similar items of interest.	- See related products below the product details

	- Click to view a related product's details
As a shopper, I want to receive email confirmation after placing an order so I know it was successful.	- Receive an email confirmation within minutes of order placement - Email includes order summary, order number, and estimated delivery date
As a shopper, I want to track my order status so I know when it will arrive.	- View order status in the account order history - Receive email updates as the order status changes (shipped, out for delivery, etc.)
As a shopper, I want to save products to a wishlist so I can purchase them later.	- Click a button to add products to the wishlist - View wishlist items in the user account
As a shopper, I want to remove products from my wishlist so I can keep it updated.	- Click to remove an item from the wishlist - Confirm removal action
As a shopper, I want to update my account information so my details are current.	- Edit name, email, password, and address information - Confirm changes with password if necessary
As a shopper, I want to change my password so I can keep my account secure.	- Enter the current password to confirm identity - Enter a new password twice to confirm
As a shopper, I want to reset my password if I forget it so I can regain access to my account.	- Receive a password reset link via email - Enter a new password after following the link
As a shopper, I want to read product reviews so I can gauge the quality of an item.	- View a list of customer reviews on the product detail page - Sort reviews by rating or date
As a shopper, I want to leave a review for a purchased product so I can share my experience.	- Submit a review with a rating, title, and comment - Review is visible to other customers after approval

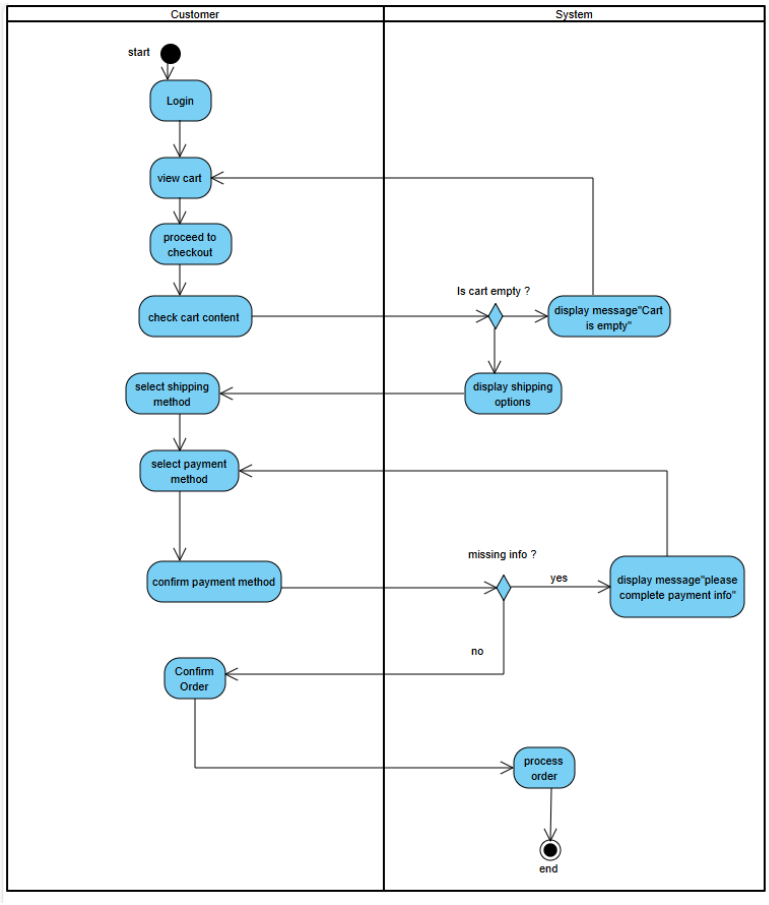
As a shopper, I want to see personalized product recommendations based on my purchase history.	- View recommended products on the homepage and product detail pages - Recommendations are updated based on recent purchases
As a shopper, I want to apply discount codes so I can save money on purchases.	- Enter a valid discount code at checkout - See discount applied to the total
As a shopper, I want to select a preferred shipping method so I can control delivery speed.	- Choose from standard, express, or same-day shipping options - See updated shipping cost based on the selected option
As a shopper, I want to choose a payment method so I can pay in the way I prefer.	- Select from credit card, PayPal, or other available payment methods - Enter relevant payment information securely
As a shopper, I want to see estimated delivery dates so I know when to expect my order.	- See estimated delivery date at checkout based on shipping method - Delivery date updates if the shipping method is changed
As a shopper, I want to cancel my order if it hasn’t been shipped yet.	- Option to cancel is available in order history if the order has not shipped - Receive email confirmation of the cancellation and refund
As a shopper, I want to return a product if I’m not satisfied.	- Initiate a return request within the return policy period - Receive instructions on how to return the product
As a shopper, I want to chat with customer service if I have an issue with my order.	- Access customer support chat from any page - Receive a response within a reasonable time
As a shopper, I want to receive notifications of sales and promotions so I can save on future purchases.	- Subscribe to promotional emails or SMS alerts - Receive notifications about upcoming sales
As a manager, I want to add new products to the site so customers have a variety to choose from.	- Enter product name, description, price, and images - Save product to make it visible on the site

As a manager, I want to remove products from the site if they are no longer available.	- Select product to delete from the system - Confirm the deletion action
As a manager, I want to adjust inventory levels so stock availability is accurate.	- Update stock quantity for each product - See updated stock levels immediately
As a manager, I want to view sales reports so I can assess business performance.	- View total sales, revenue, and best-selling products - Filter reports by date, category, or other criteria
As a manager, I want to view customer feedback so I can understand their satisfaction.	- Access product reviews and ratings - Sort feedback by rating or date
As a staff member, I want to review and approve product reviews before they are visible on the site.	- Access pending reviews - Approve or reject each review based on guidelines

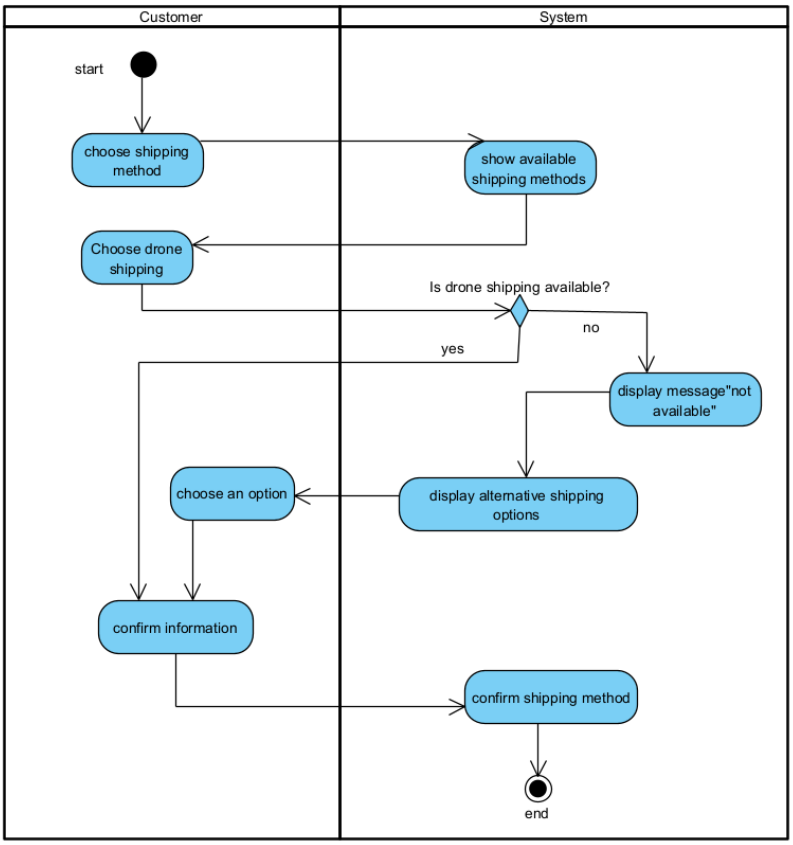
- Draw 5 activity/swimlane diagrams + 5 sequence diagrams for the processes executed by customer and staff:
  - 5 activity/swimlane diagrams
    - Create cart



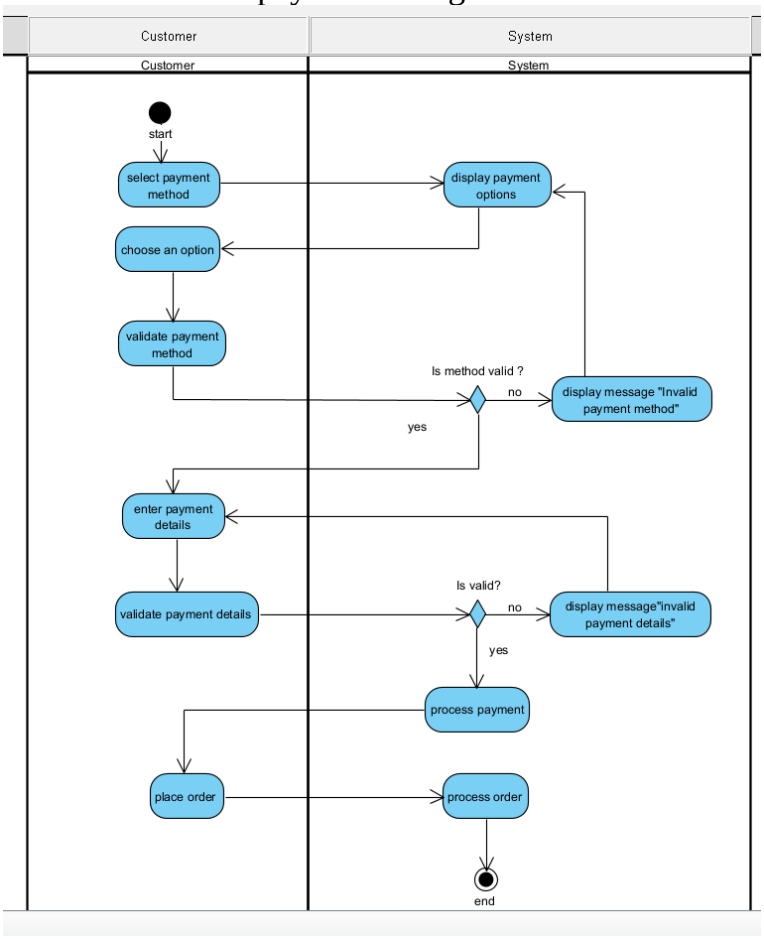
- Checkout



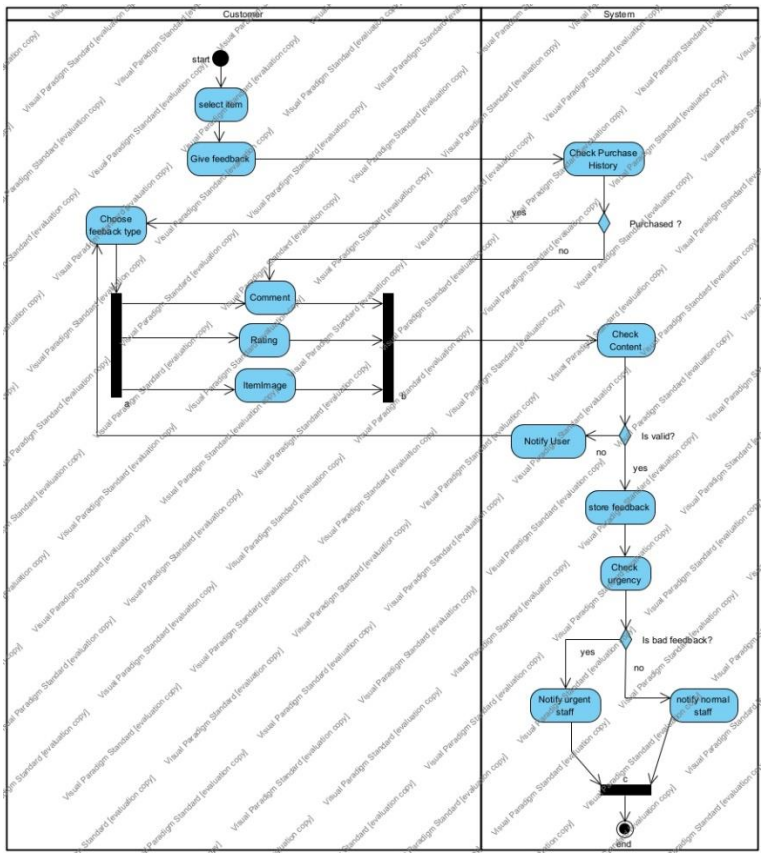
- Select ship (drone)



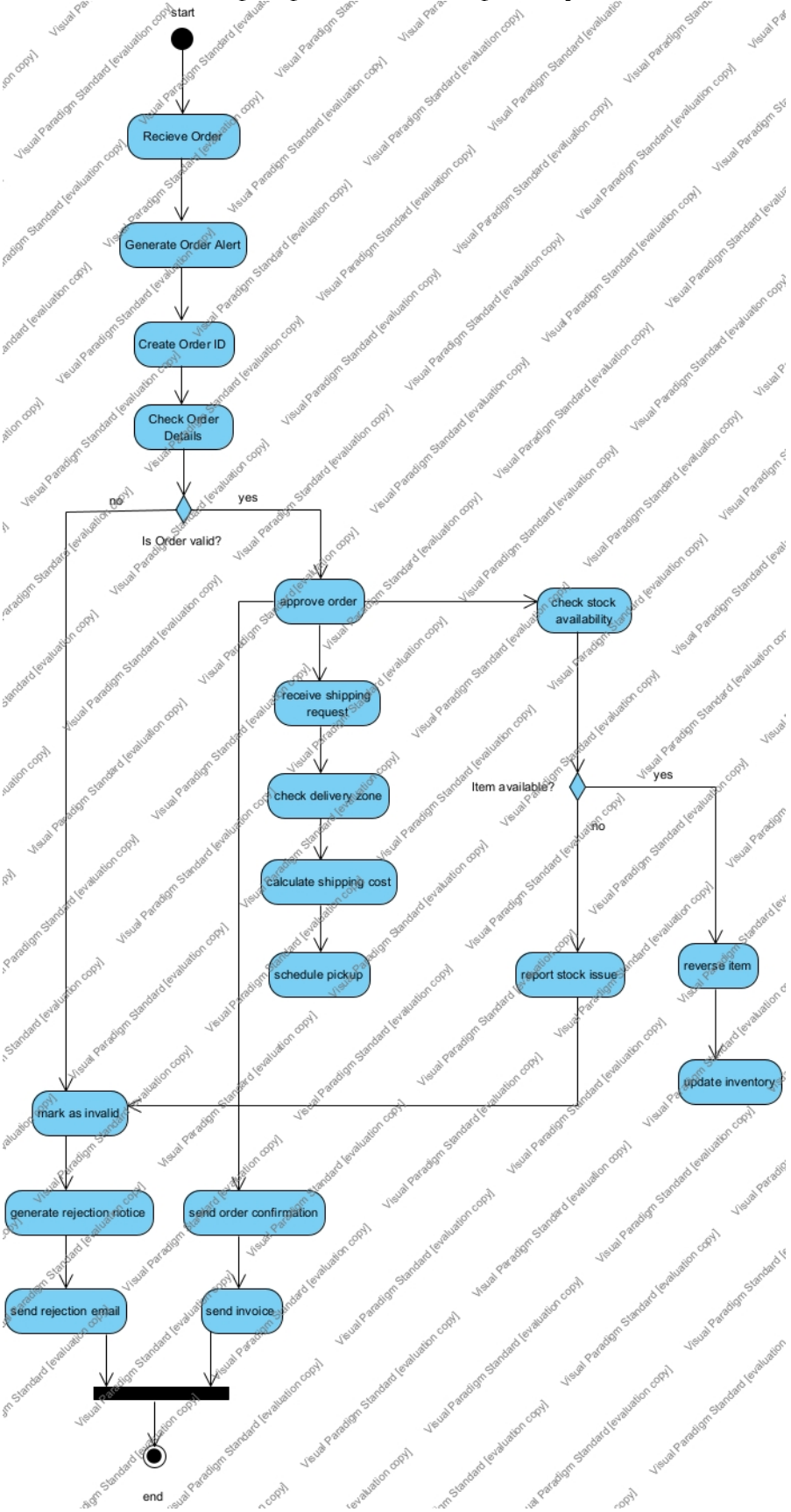
- Select pay and ordergive



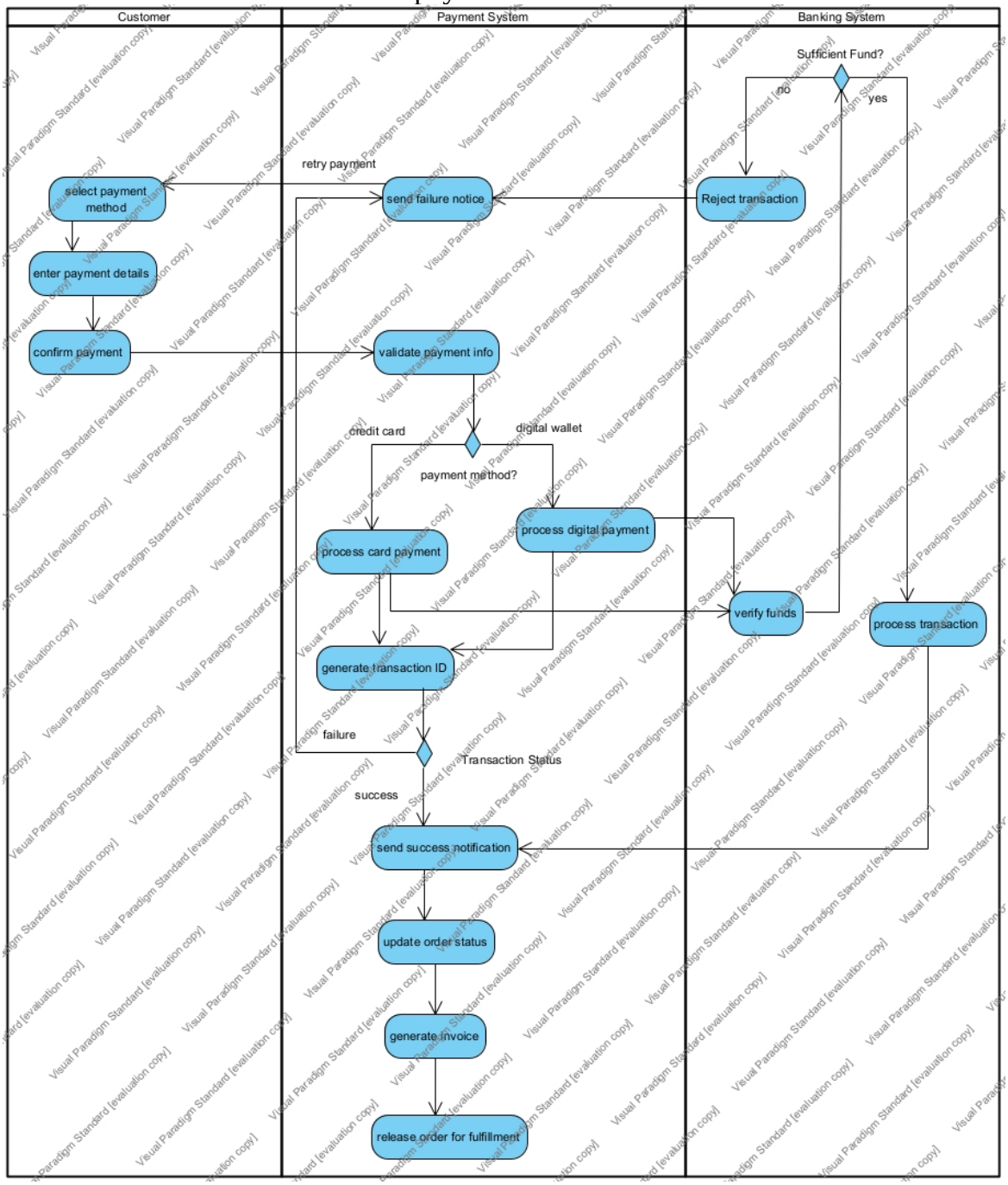
- Comments & feed back of items



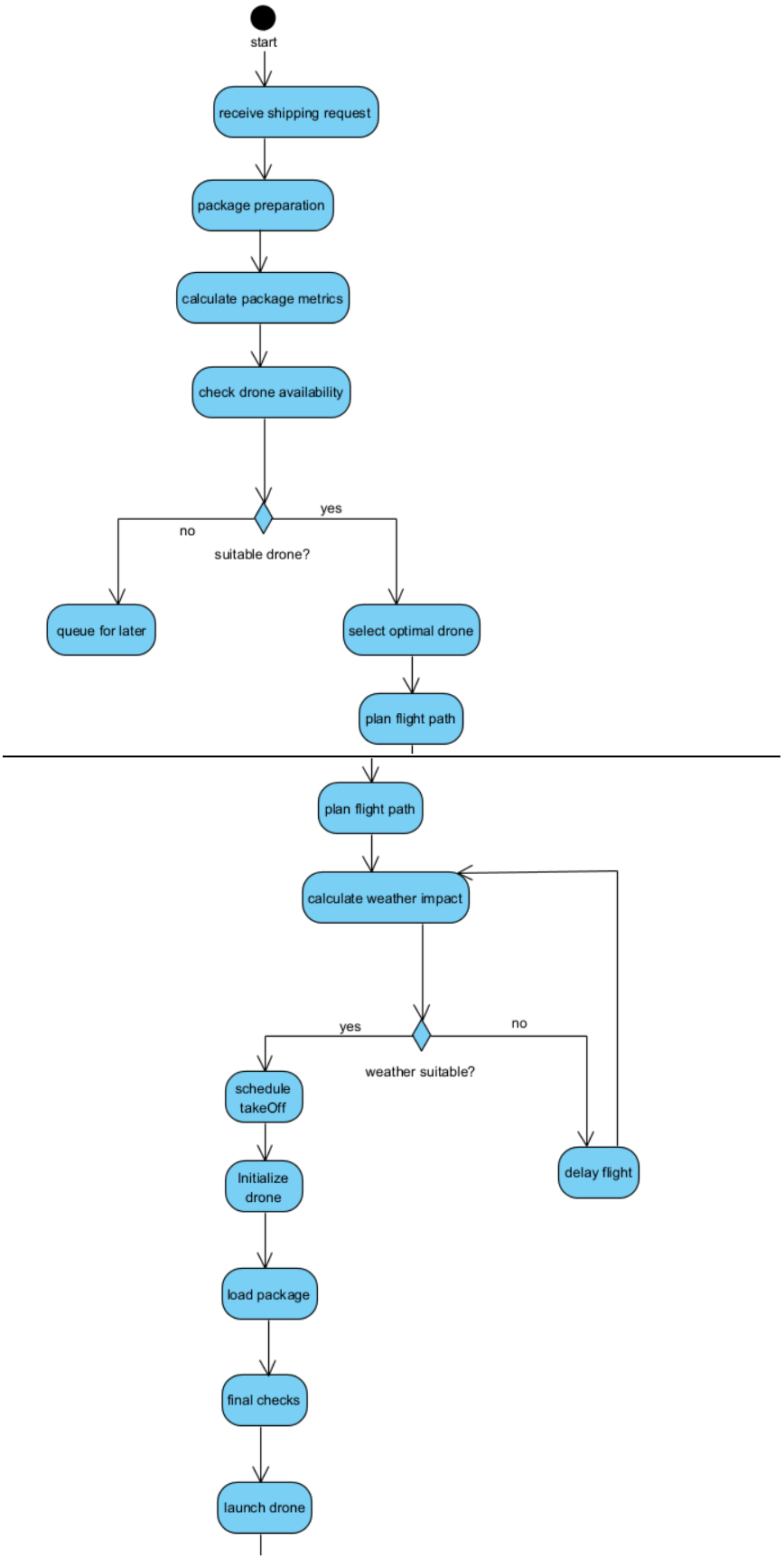
- Process order: alert order (automatically), check order, send invoice/feedback to customer, send requirements to ship department and repository

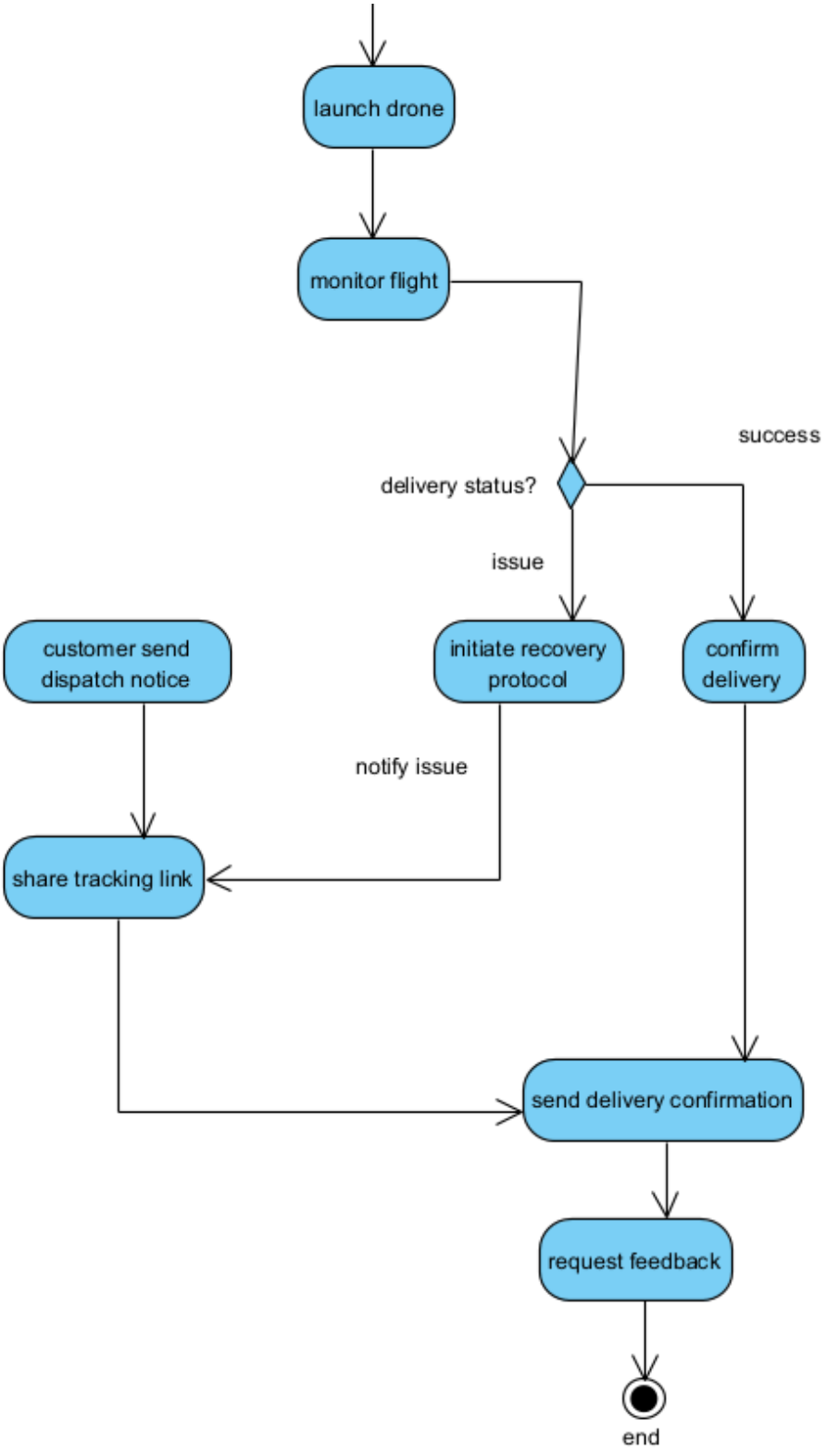


- Process payment

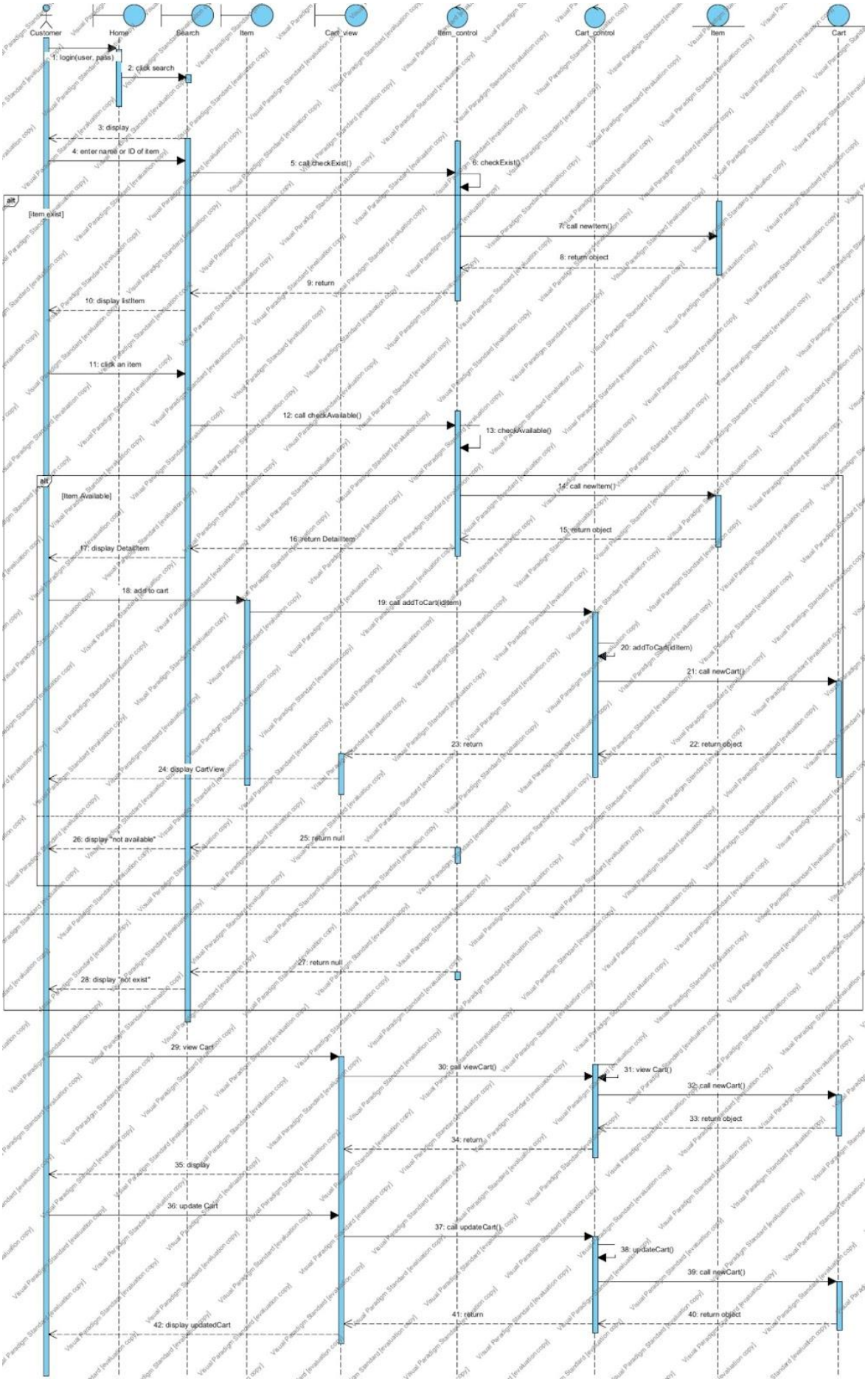


- Process shipment (using drone)

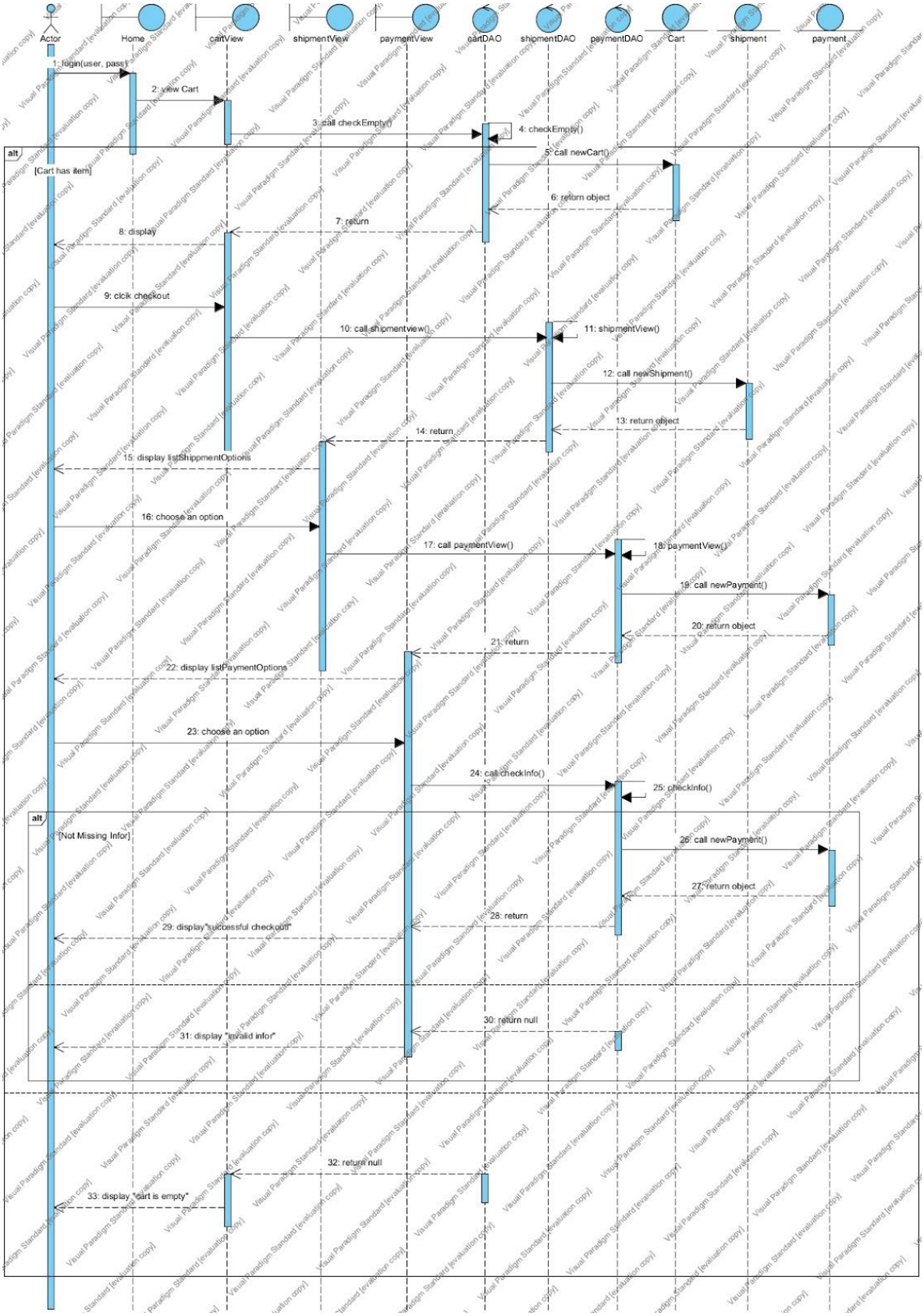

start
receive shipping request
package preparation
calculate package metrics
check drone availability
yes
no
suitable drone?
queue for later
select optimal drone
plan flight path
plan flight path
calculate weather impact
yes
no
weather suitable?
schedule takeOff
delay flight
Initialize drone
load package
final checks
launch drone



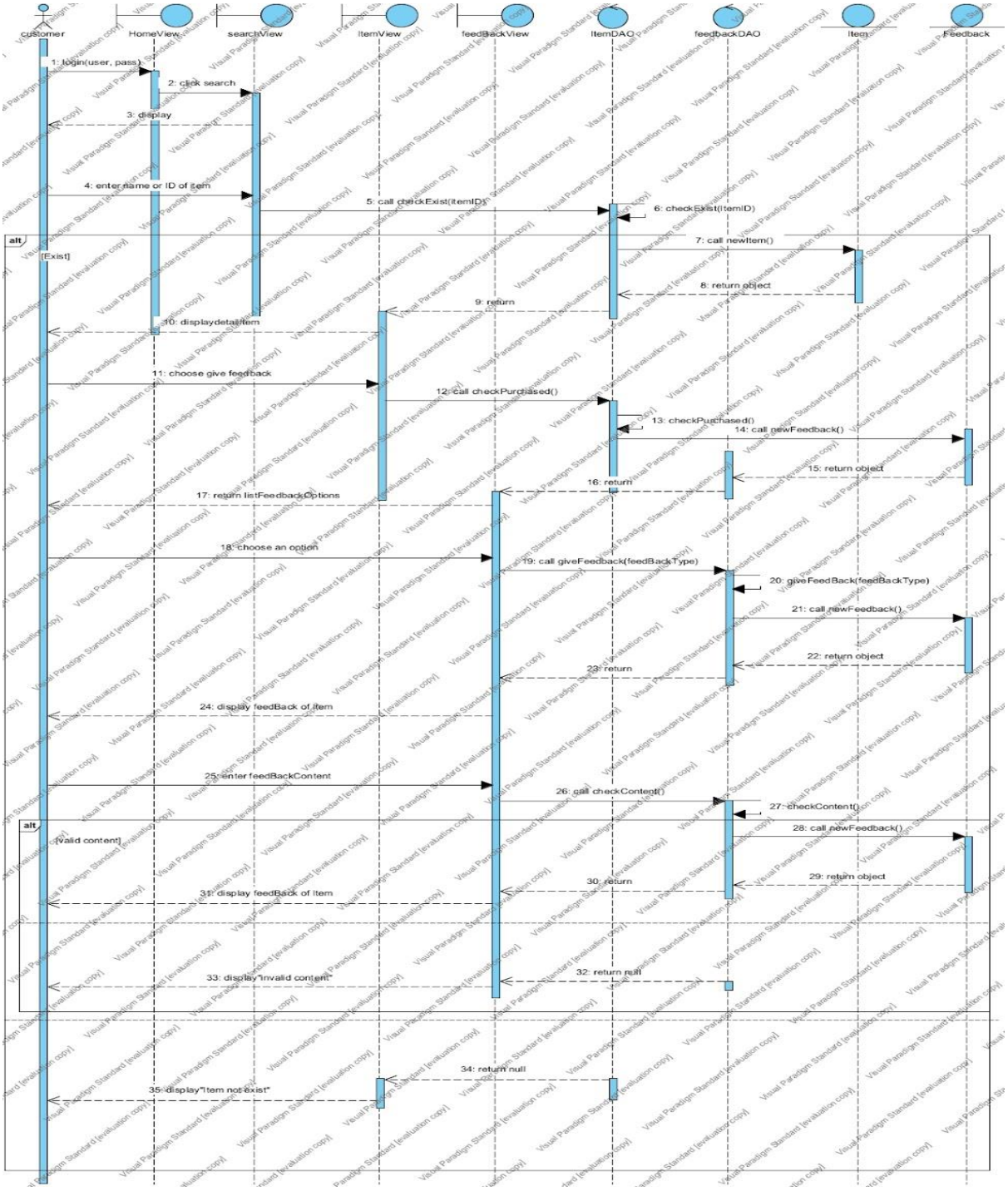
- 5 sequence diagrams
  - Create cart


Customer
Home
Search
Item
Cart_view
Item_control
Cart_control
Item
Cart
1: login(user, pass)
2: click search
3: display
4: enter name or ID of item
5: call checkExist()
6: checkExist()
alt
[item exist]
7: call newItem()
8: return object
9: return
10: display listItem
11: click an item
12: call checkAvailable()
13: checkAvailable()
alt
[Item Available]
14: call newItem()
15: return object
16: return DetailItem
17: display DetailItem
18: add to cart
19: call addToCart(idItem)
20: addToCart(idItem)
21: call newCart()
22: return object
23: return
24: display CartView
26: display "not available"
25: return null
27: return null
28: display "not exist"
29: view Cart
30: call viewCart()
31: view Cart()
32: call newCart()
33: return object
34: return
35: display
36: update Cart
37: call updateCart()
38: updateCart()
39: call newCart()
40: return object
41: return
42: display updatedCart

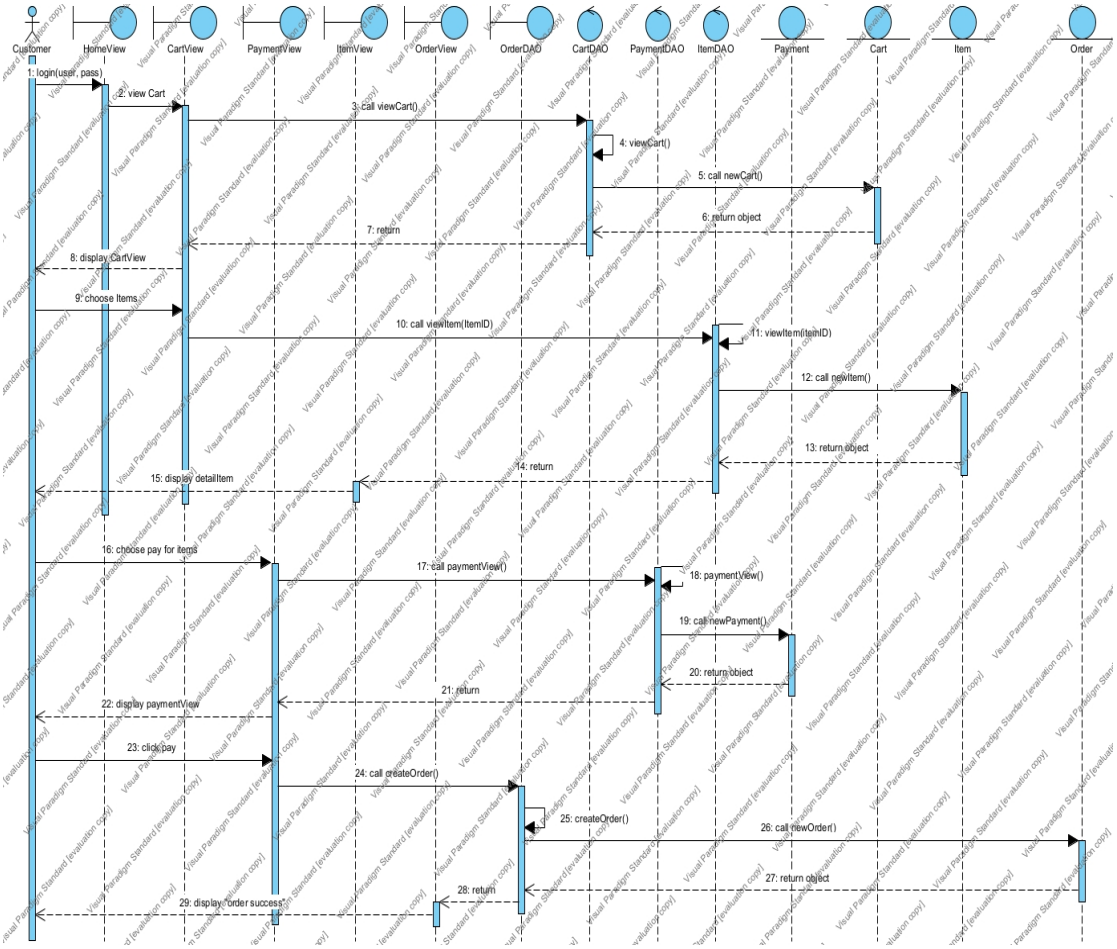
- Checkout


Actor
Home
cartView
shipmentView
paymentView
cartDAO
shipmentDAO
paymentDAO
Cart
shipment
payment
1: login(user, pass)
2: view Cart
3: call checkEmpty()
4: checkEmpty()
alt
[Cart has item]
5: call newCart()
6: return object
7: return
8: display
9: clcik checkout
10: call shipmentview()
11: shipmentView()
12: call newShipment()
13: return object
14: return
15: display listShippmentOptions
16: choose an option
17: call paymentView()
18: paymentView()
19: call newPayment()
20: return object
21: return
22: display listPaymentOptions
23: choose an option
24: call checkInfo()
25: checkInfo()
alt
[Not Missing Infor]
26: call newPayment()
27: return object
28: return
29: display "successful checkout"
30: return null
31: display "invalid infor"
32: return null
33: display "cart is empty"

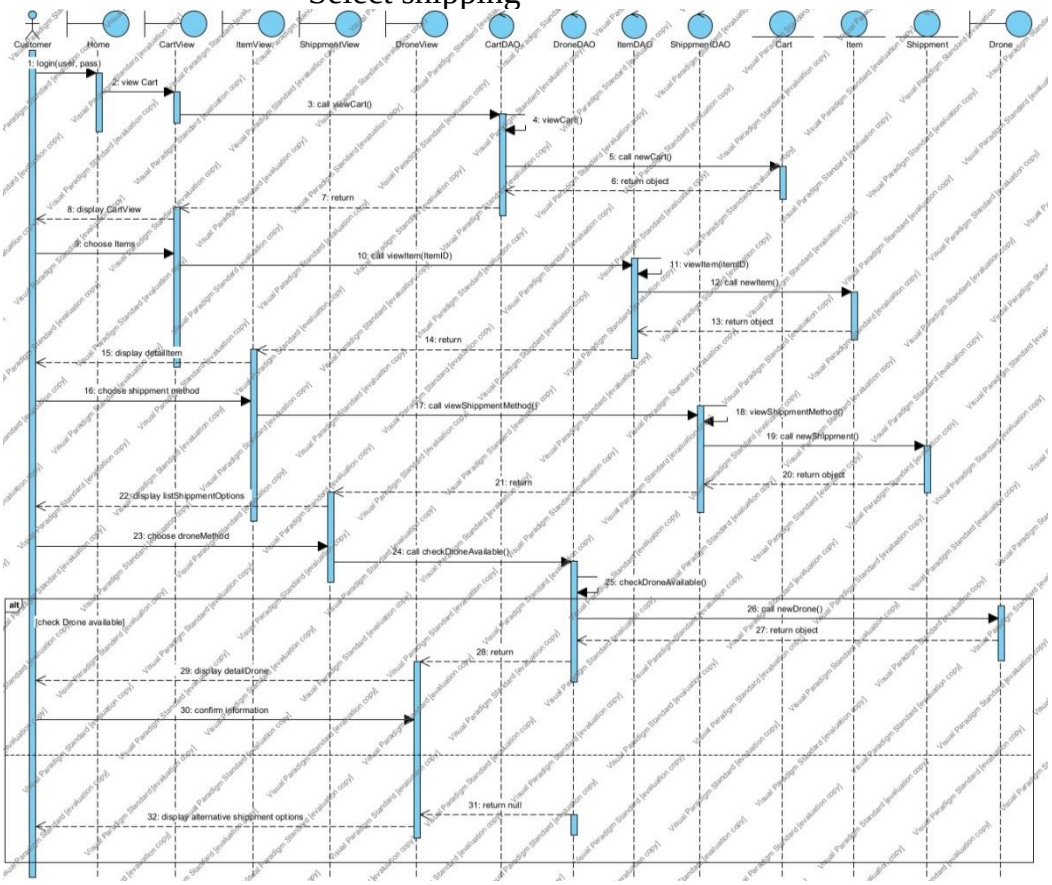
- Give feedback



- Select pay and order



- Select shipping



1. Requirement Analysis
    - Write 50 CRC Cards
        - User

User
Super Classes: None
Sub Classes: Customer, Staff
Description: Represents the basic functionality and attributes of any user interacting with the e-commerce system.

Attributes:

Name	Description
id	Unique identifier for the user.
name	Full name of the user.
email	User's email address.
phone	Contact phone number.
password	User's account password.
User's account password.	Physical address of the user.
role	Role assigned to the user.

Responsibilities:

Name	Collaborator
Register user	Database
Login user	Authentication system
Update profile	Database

        - Customer

Customer
Super Classes: User
Sub Classes: None
Description: Extends the User class for customers with additional functionality.

Attributes:

Name	Description
shippingAddress	The shipping address of the customer.
balance	The customer's account balance.

Responsibilities:

Name	Collaborator
displayInfo()	System display framework

Right click: add Attribute;

- Staff

Staff	
Super Classes: User	
Sub Classes: Manager	
Description: Extends the User class to represent staff members with specific roles.	
Attributes:	
Name	Description
department	Department to which the staff belongs.
staffLevel	Staff level within the organization.
Responsibilities:	
Name	Collaborator
manageStaff()	Task assignment system
promoteStaff()	HR management system
assignTask()	Task management system
evaluatePerformance()	Performance tracking system

- Manager

Manager	
Super Classes: Staff	
Sub Classes: Admin	
Description: Represents managers with team-specific responsibilities and oversight roles.	
Attributes:	
Name	Description
team	Team managed by the manager.
subordinates	Staff members reporting to the manager.
Responsibilities:	
Name	Collaborator
approveRequests()	Approval system
manageProjects()	Project management system
delegateTasks()	Task assignment system
handleBudget()	Budget tracking system

- Admin

Admin	
Super Classes: Manager	
Sub Classes: None	
Description: Extends the Manager class with high-level administrative privileges.	
Attributes:	
Name	Description
privileges	Privileges assigned to the admin.
accessLevel	Access level for the admin (e.g., Full).
Responsibilities:	
Name	Collaborator
accessAdminPanel( )	Administration system
assignRoles( )	Role management system
removeUser( )	User database system
viewReports( )	Reporting system
Double click: edit Class Name;	

- Product

Product	
Super Classes: None	
Sub Classes: Book, Clothing, ElectronicDevice, Shoe,..	
Description: Represents a product in the system with attributes like name, price, description, category, and stock quantity.	
Attributes:	
Name	Description
id	Unique identifier for the product.
name	Name of the product.
price	Price of the product.
description	Description of the product.
category	Category to which the product belongs.
stockQuantity	Quantity of the product available in stock.
Responsibilities:	
Name	Collaborator
Manage product details	Category
Add, update, and remove products	Database or storage system

- Category

Category
Super Classes: None
Sub Classes: None
Description: Represents a product category with its attributes like name and description.

Attributes:

Name	Description
id	Unique identifier for the category.
name	Name of the category.
description	Description of the category.

Responsibilities:

Name	Collaborator
Manage category details	Product
Add and update categories	Database or storage system

- Book

Book
Super Classes: Product
Sub Classes: None
Description: Represents a book product, inheriting general product properties and adding book-specific details like author, publisher, and ISBN.

Attributes:

Name	Description
author	Author of the book.
publisher	Publisher of the book.
isbn	ISBN number of the book.

Responsibilities:

Name	Collaborator
Manage book-specific details	Product,Author and publisher
Inherit general product properties	Database or storage system

- Clothes

Clothes	
Super Classes: Product	
Sub Classes: None	
Description: Represents a clothing product, inheriting general product properties and adding clothing-specific details like size, color, and material.	
Attributes:	
Name	Description
size	Size of the clothing item.
color	Color of the clothing item.
material	Material of the clothing item.
Responsibilities:	
Name	Collaborator
Manage clothing-specific details	Product
Inherit general product properties	Database or storage system

- Electronic Device

ElectronicDevice	
Super Classes: Product	
Sub Classes: Laptop, MobilePhone	
Description: Represents an electronic device product, inheriting general product properties and adding device-specific details like brand and warranty period.	
Attributes:	
Name	Description
brand	Brand of the electronic device.
warrantyPeriod	Warranty period of the electronic device.
Responsibilities:	
Name	Collaborator
Manage electronic device-specific details	Product
Inherit general product properties	Database or storage system

- Laptop

Laptop	
Super Classes: ElectronicDevice	
Sub Classes: None	
Description: Represents a laptop product, inheriting general electronic device properties and adding laptop-specific details like processor, RAM, and storage.	
Attributes:	
Name	Description
processor	Processor type of the laptop.
ram	RAM size of the laptop.
storage	Storage capacity of the laptop.
Responsibilities:	
Name	Collaborator
Manage laptop-specific details	ElectronicDevice
Inherit general electronic device properties	Database or storage system

- MobilePhone

MobilePhone	
Super Classes: ElectronicDevice	
Sub Classes: None	
Description: Represents a mobile phone product, inheriting general electronic device properties and adding phone-specific details like screen size and battery capacity.	
Attributes:	
Name	Description
screenSize	Screen size of the mobile phone.
batteryCapacity	Battery capacity of the mobile phone.
Responsibilities:	
Name	Collaborator
Manage mobile phone-specific details	Customer, Staff
Inherit general electronic device properties	Database or storage system

- Shoes

shoes	
Super Classes: Product	
Sub Classes: None	
Description: Represents a shoe product, inheriting general product properties and adding shoe-specific details like size and gender.	
Attributes:	
Name	Description
size	Size of the shoe.
gender	Gender for which the shoe is designed (e.g., male, female).
Responsibilities:	
Name	Collaborator
Manage shoe-specific details	Product
Inherit general product properties	Database or storage system

- Cart

Cart	
Super Classes: None	
Sub Classes: None	
Description: Represents a shopping cart, containing a list of products (cart items) and methods for managing the cart's contents and calculating the total value.	
Attributes:	
Name	Description
id	Unique identifier for the cart.
customerId	Unique identifier for the customer who owns the cart.
products	List of CartItem objects that represent the products in the cart.
Responsibilities:	
Name	Collaborator
Manage cart details	CartItem
Store the list of products	Product

- CartItem

CartItem	
Super Classes: None	
Sub Classes: None	
Description: Represents a single item in the cart, including the product ID, quantity, and price.	
Attributes:	
Name	Description
productId	Unique identifier for the product in the cart.
quantity	Quantity of the product in the cart.
price	Price of the product in the cart.
Responsibilities:	
Name	Collaborator
Represent a product in the cart	Product,Cart
Provide methods for setting and getting the details of the cart item.	CartItem,Database or storage system

- Order

Order	
Super Classes: None	
Sub Classes: None	
Description: Represents an order made by a customer, with details such as order date, status, and total amount. It contains methods for placing, canceling, and calculating the total amount of the order.	
Attributes:	
Name	Description
id	Unique identifier for the order.
customerId	ID of the customer who placed the order.
orderDate	Date when the order was placed.
status	Current status of the order (e.g., "Pending", "Placed", "Cancelled").
totalAmount	Total cost of the order.
orderItems	List of OrderItem objects that represent the items in the order.
Responsibilities:	
Name	Collaborator
Manage order details	OrderItem,Invoice,Payment
Store the list of OrderItem	Database or storage system

- OrderItem

OrderItem	
Super Classes: None	
Sub Classes: None	
Description: Represents a product in the order, including the product ID, quantity, and price.	
Attributes:	
Name	Description
orderId	ID of the order that this item belongs to.
productId	Unique identifier of the product.
quantity	Quantity of the product in the order.
price	Price of the product in the order.
Responsibilities:	
Name	Collaborator
Represent a product in an order	Order,Product

- Invoice

Invoice	
Super Classes: None	
Sub Classes: None	
Description: Represents an invoice for an order, including the invoice date and total amount.	
Attributes:	
Name	Description
id	Unique identifier for the invoice.
orderId	ID of the associated order.
invoiceDate	Date when the invoice was issued.
totalAmount	Total amount for the order on the invoice.
Responsibilities:	
Name	Collaborator
Represent the invoice for a given order.	Order

- OrderStatus

OrderStatus	
Super Classes: None	
Sub Classes: None	
Description: Represents the status of an order, with a name and unique identifier.	
Attributes:	
Name	Description
id	Unique identifier for the order status.
statusName	Name of the status (e.g., "Pending", "Placed", "Cancelled").
Responsibilities:	
Name	Collaborator
Represent the status of an order.	Order

- Payment

Payment	
Super Classes: None	
Sub Classes: None	
Description: Represents a payment for an order, including the payment method, date, and amount.	
Attributes:	
Name	Description
id	Unique identifier for the payment.
orderId	ID of the order being paid for.
paymentMethod	Method used for payment (e.g., CreditCard, PayPal).
paymentDate	Date the payment was made.
amount	Total amount paid.
Responsibilities:	
Name	Collaborator
Represent a payment for an order.	Order
Process the payment using the appropriate payment method.	PaymentMethod

- PaymentMethod

PaymentMethod	
Super Classes: None	
Sub Classes: CreditCard, PayPal	
Description: Represents a generic payment method, with a unique identifier and a method for processing the payment.	
Attributes:	
Name	Description
id	Unique identifier for the payment method.
methodName	Name of the payment method (e.g., "CreditCard", "PayPal").
Responsibilities:	
Name	Collaborator
Provide an abstract method to process payments.	Payment

- CreditCard

CreditCard	
Super Classes: PaymentMethod	
Sub Classes: None	
Description: Represents a credit card as a payment method, with attributes like card number, expiry date, and cardholder name.	
Attributes:	
Name	Description
cardNumber	Credit card number.
expiryDate	Expiry date of the card.
cardHolderName	Name of the cardholder.
Responsibilities:	
Name	Collaborator
Represent the details of a credit card.	Payment,PaymentMethod
Process payments	Payment,PaymentMethod

- PayPal

PayPal	
Super Classes: PaymentMethod	
Sub Classes: None	
Description: Represents PayPal as a payment method, with an associated account email.	
Attributes:	
Name	Description
accountEmail	PayPal account email associated with the payment.
Responsibilities:	
Name	Collaborator
Represent the details of a PayPal account.	Payment,PaymentMethod
Process payments using the PayPal account.	Payment,PaymentMethod

- Shipment

Shipment	
Super Classes: None	
Sub Classes: None	
Description: Represents a shipment for an order, with tracking information and delivery details.	
Attributes:	
Name	Description
id	Unique identifier for the shipment.
orderId	ID of the order associated with this shipment.
shipmentDate	The date the shipment was dispatched.
deliveryDate	The expected or actual delivery date of the shipment.
status	The current status of the shipment (e.g., Pending, Delivered).
trackingId	The tracking number for the shipment.
Responsibilities:	
Name	Collaborator
Represent the details	Order,Drone
Update shipment status	Order,Drone

- Drone

Drone	
Super Classes: None	
Sub Classes: None	
Description: Represents a drone used for delivering shipments, with capacity and status.	
Attributes:	
Name	Description
id	Unique identifier for the drone.
model	Model name or type of the drone.
capacity	The maximum weight capacity the drone can carry.
status	The current status of the drone (e.g., Idle, In Use).
Responsibilities:	
Name	Collaborator
Represent the details	Shipment
Update drone status when it changes	ShipmentStatus

- ShipmentStatus

ShipmentStatus	
Super Classes: None	
Sub Classes: None	
Description: Represents the different possible statuses of a shipment (e.g., Pending, Delivered).	
Attributes:	
Name	Description
id	Unique identifier for the shipment status.
statusName	The name of the shipment status (e.g., Pending, Delivered).
Responsibilities:	
Name	Collaborator
LoginProvide a way to identify and manage different shipment statuses.	Shipment

- Feedback

Feedback	
Super Classes: None	
Sub Classes: None	
Description: Represents a customer's feedback for a product, including a rating and comments.	
Attributes:	
Name	Description
id	Unique identifier for the feedback.
customerId	ID of the customer providing the feedback.
productId	ID of the product being reviewed.
rating	Rating given by the customer (from 1 to 5).
comments	Additional comments provided by the customer.
Responsibilities:	
Name	Collaborator
Store the rating and comments for a specific product from a customer.	Product,Customer

- Review

Review	
Super Classes: None	
Sub Classes: None	
Description: Represents a customer's review of a product, including a rating and detailed review text.	
Attributes:	
Name	Description
id	Unique identifier for the review.
customerId	ID of the customer providing the review.
productId	ID of the product being reviewed.
reviewText	Detailed review text provided by the customer.
rating	Rating given by the customer (from 1 to 5).
Responsibilities:	
Name	Collaborator
DisplayPolicyStore and display the review text and rating for a specific product.	Product,Customer

- Inventory

Inventory	
Super Classes: None	
Sub Classes: None	
Description: Represents the inventory for a product, including stock quantity and reorder level.	
Attributes:	
Name	Description
id	Unique identifier for the inventory entry.
productId	ID of the product associated with the inventory.
stockQuantity	The current quantity of the product in stock.
reorderLevel	The stock level at which new inventory should be ordered.
Responsibilities:	
Name	Collaborator
Track the stock quantity of a product.	Product
Monitor when to reorder based on the reorder level.	Supplier,Warehouse

- Warehouse

Warehouse	
Super Classes: None	
Sub Classes: None	
Description: Represents a warehouse that holds products in stock.	
Attributes:	
Name	Description
id	Unique identifier for the warehouse.
location	The physical location of the warehouse.
capacity	The maximum capacity (e.g., number of products it can store).
Responsibilities:	
Name	Collaborator
Track the capacity to ensure that the warehouse does not exceed its storage limit.	Inventory,Supplier

- Supplier

Supplier	
Super Classes: None	
Sub Classes: None	
Description: Represents a supplier that provides products for inventory.	
Attributes:	
Name	Description
id	Unique identifier for the supplier.
name	Name of the supplier.
contact	Contact information for the supplier.
Responsibilities:	
Name	Collaborator
Provide contact information for purchasing products.	Inventory,Warehouse

- Report

Report	
Super Classes: None	
Sub Classes: SalesReport, CustomerReport	
Description: Represents a generic report with an ID, type, generation date, and associated data.	
Attributes:	
Name	Description
id	Unique identifier for the report.
type	Type of the report (e.g., "Sales Report", "Customer Report").
generatedDate	Date when the report was generated.
data	Data associated with the report.
Responsibilities:	
Name	Collaborator
Provide common functionality for generating reports.	SalesReport,CustomerReport
Hold general attributes for reports	SalesReport,CustomerReport

- SalesReport

SalesReport	
Super Classes: Report	
Sub Classes: None	
Description: Represents a sales report, including total sales and top-selling products.	
Attributes:	
Name	Description
totalSales	Total sales amount in the report.
topProducts	List or description of top-selling products.
Responsibilities:	
Name	Collaborator
Generate a sales report by providing details such as total sales and top-selling products.	Report,Product

- CustomerReport

CustomerReport	
Super Classes: Report	
Sub Classes: None	
Description: Represents a customer-specific report with customer data.	
Attributes:	
Name	Description
customerData	Data related to customers (e.g., demographics, purchase history).
Responsibilities:	
Name	Collaborator
Generate a customer report with customer-specific data.	Report,Customer

- Analystic

Analytics	
Super Classes: None	
Sub Classes: None	
Description: Represents an analytics process, including a type and associated data for analysis.	
Attributes:	
Name	Description
id	Unique identifier for the analytics instance.
type	Type of analysis being performed (e.g., sales, customer trends).
data	Data on which the analysis is performed.
Responsibilities:	
Name	Collaborator
Perform analytics on provided data based on the type of analysis.	Report,Product,Customer

- Notification

Notification	
Super Classes: None	
Sub Classes: None	
Description: Represents a notification message that is sent to a recipient.	
Attributes:	
Name	Description
id	Unique identifier for the notification.
recipientId	ID of the recipient who will receive the notification.
message	Content of the notification message.
sentDate	Date when the notification was sent.
Responsibilities:	
Name	Collaborator
Represent and send notifications to recipients.	Customer,Product

- Alert

Alert	
Super Classes: None	
Sub Classes: None	
Description: Represents an alert message, typically for system issues or important notices.	
Attributes:	
Name	Description
id	Unique identifier for the alert.
type	Type of the alert (e.g., "System Alert", "Warning").
message	Content of the alert message.
Responsibilities:	
Name	Collaborator
Represent and display alerts for system issues or other important information.	System,User

- SearchModule

SearchModule	
Super Classes: None	
Sub Classes: None	
Description: Represents a module for searching based on a given query and returning search results.	
Attributes:	
Name	Description
id	Unique identifier for the search module.
query	The search query to be processed.
results	The list of search results returned after the search is completed.
Responsibilities:	
Name	Collaborator
Perform search operations based on the query.	Product, RecommendationModule
Print the search results once the search is completed.	Product, RecommendationModule

- RecommendationModule

RecommendationModule	
Super Classes: None	
Sub Classes: None	
Description: Represents a module for generating product recommendations based on predefined data.	
Attributes:	
Name	Description
id	Unique identifier for the recommendation module.
recommendations	A list of products that are recommended.
Responsibilities:	
Name	Collaborator
Generate product recommendations.	Product
Use machine learning or deep learning algorithms to generate recommendations.	SearchModule

- OpinionAnalysisModule

OpinionAnalysisModule	
Super Classes: None	
Sub Classes: None	
Description: Represents a module for analyzing user opinions, extracting sentiment information from comments.	
Attributes:	
Name	Description
id	Unique identifier for the opinion analysis module.
comments	List of user comments that will be analyzed for sentiment.
sentiments	List of sentiments (e.g., positive, negative, neutral) derived from the comments.
Responsibilities:	
Name	Collaborator
Analyze user comments for sentiment.	SearchModule,RecommendationModule
Process sentiment analysis using NLP or sentiment algorithms.	SearchModule,RecommendationModule

- SystemConfig

SystemConfig	
Super Classes: None	
Sub Classes: None	
Description: Represents a system configuration setting, which can be updated dynamically.	
Attributes:	
Name	Description
id	Unique identifier for the configuration.
key	The key identifying the configuration setting.
value	The value of the configuration setting.
Responsibilities:	
Name	Collaborator
Provide a method to update the configuration value.	SearchModule,RecommendationM

- ErrorLog

ErrorLog	
Super Classes: None	
Sub Classes: None	
Description: Represents an error log entry that records errors occurring in the system.	
Attributes:	
Name	Description
id	Unique identifier for the error log.
errorDate	The date when the error occurred.
errorMessage	The message or details of the error.
Responsibilities:	
Name	Collaborator
Record and log errors in the system.	SystemConfig,AuditLog
UpgradePlaPrint out error	SystemConfig,AuditLog

- AuditLog

AuditLog	
Super Classes: None	
Sub Classes: None	
Description: Represents an audit log entry that records user actions with timestamps.	
Attributes:	
Name	Description
id	Unique identifier for the audit log.
action	The action that the user performed (e.g., "Login", "Update").
timestamp	The timestamp of when the action occurred.
userId	The ID of the user who performed the action.
Responsibilities:	
Name	Collaborator
Provide a method to log actions performed by users.	ErrorLog,SystemConfig

- Address

Address	
Super Classes: None	
Sub Classes: None	
Description: Represents an address with street, city, state, and zip code information.	
Attributes:	
Name	Description
id	Unique identifier for the address.
street	The street of the address.
city	The city of the address.
state	The state of the address.
zip	The zip code of the address.
Responsibilities:	
Name	Collaborator
Provide a method to return the full address as a string.	None

- FileUpload

FileUpload	
Super Classes: None	
Sub Classes: None	
Description: Represents a file upload with an ID, file name, and file path.	
Attributes:	
Name	Description
id	Unique identifier for the file upload.
fileName	The name of the file being uploaded.
filePath	The path where the file is being uploaded.
Responsibilities:	
Name	Collaborator
DisplaySuggestionsStore file upload information (ID, file name, and file path).	None
Provide a method to upload the file and print upload details.	None

- Session

Session	
Super Classes: None	
Sub Classes: None	
Description: Represents a user session with start and end times.	
Attributes:	
Name	Description
id	Unique identifier for the session.
userId	The ID of the user associated with the session.
startTime	The start time of the session.
endTime	The end time of the session.
Responsibilities:	
Name	Collaborator
Store session details	None
Provide a method to calculate the session duration.	None

- Discount

Discount	
Super Classes: LoyaltyProgram	
Sub Classes: Voucher	
Description: Represents a discount with a percentage and validity period.	
Attributes:	
Name	Description
id	Unique identifier for the discount.
percentage	The discount percentage.
validityPeriod	The date until which the discount is valid.
Responsibilities:	
Name	Collaborator
Store discount details (ID, percentage, validity period).	None
Provide a method to check if the discount is valid.	None

- Voucher

Voucher	
Super Classes: Discount	
Sub Classes: None	
Description: Represents a voucher, which is a type of discount, including a code.	
Attributes:	
Name	Description
code	The unique voucher code.
Responsibilities:	
Name	Collaborator
Store voucher details (code, ID, percentage, validity period).	Discount
Provide a method to return the voucher as a string.	Discount

- Role

Role
Super Classes: None
Sub Classes: None
Description: Represents a user role with a name.

Attributes:	
Name	Description
id	Unique identifier for the role.
roleName	The name of the role (e.g., "Admin", "User").

Responsibilities:	
Name	Collaborator
Store role details (ID and role name).	None
Provide a method to display role information.	None

- Permission

Permission
Super Classes: None
Sub Classes: None
Description: Represents a user permission with a name.

Attributes:	
Name	Description
id	Unique identifier for the permission.
permissionName	The name of the permission (e.g., "Read", "Write").

Responsibilities:	
Name	Collaborator
Store permission details (ID and permission name).	None
Provide a method to display permission information.	None

- Draw a class diagram (analysis) with 50 classes

- Construct data dictionary
  - Customer

Field Name	Description	Data Type	Size	Format	Valid Values	Default	Notes
customer_id	Unique identifier for each customer	Integer	10	-	-	Auto-increment	Primary key
name	Full name of the customer	Varchar	100	-	-	-	-
email	Customer's email address	Varchar	255	Email	-	-	Must be unique and valid

phone_number	Customer's phone number	Varchar	10	-	-	-	-
address	Customer's shipping address	Varchar	255	-	-	-	-

- Product

Field Name	Description	Data Type	Size	Format	Valid Values	Default	Notes
product_id	Unique identifier for each product	Integer	10	-	-	Auto-increment	Primary key
name	Name of the product	Varchar	255	-	-	-	-
description	Detailed description of the product	Text	-	-	-	-	-
category	Product category	Varchar	50	-	"book", "clothing", "mobile", etc.	"book"	-
price	Price of the product	Decimal	10.2	-	-	0.0	-
stock_quantity	Number of items in stock	Integer	10	-	>=0	0	-

- Staff

Field Name	Description	Data Type	Size	Format	Valid Values	Default	Notes
staff_id	Unique identifier for each staff member	Integer	10	-	-	Auto-increment	Primary key
name	Full name of the staff	Varchar	100	-	-	-	-

email	Staff member's email address	Varchar	255	Email	-	-	Must be unique and valid
role	Role of the staff (manager, sales, admin)	Varchar	50	-	"manager", "sales", "admin"	"sales"	Defines staff access level
phone_number	Staff member's phone number	Varchar	15	-	-	-	-
address	Staff member's address	Varchar	255	-	-	-	-

- Order

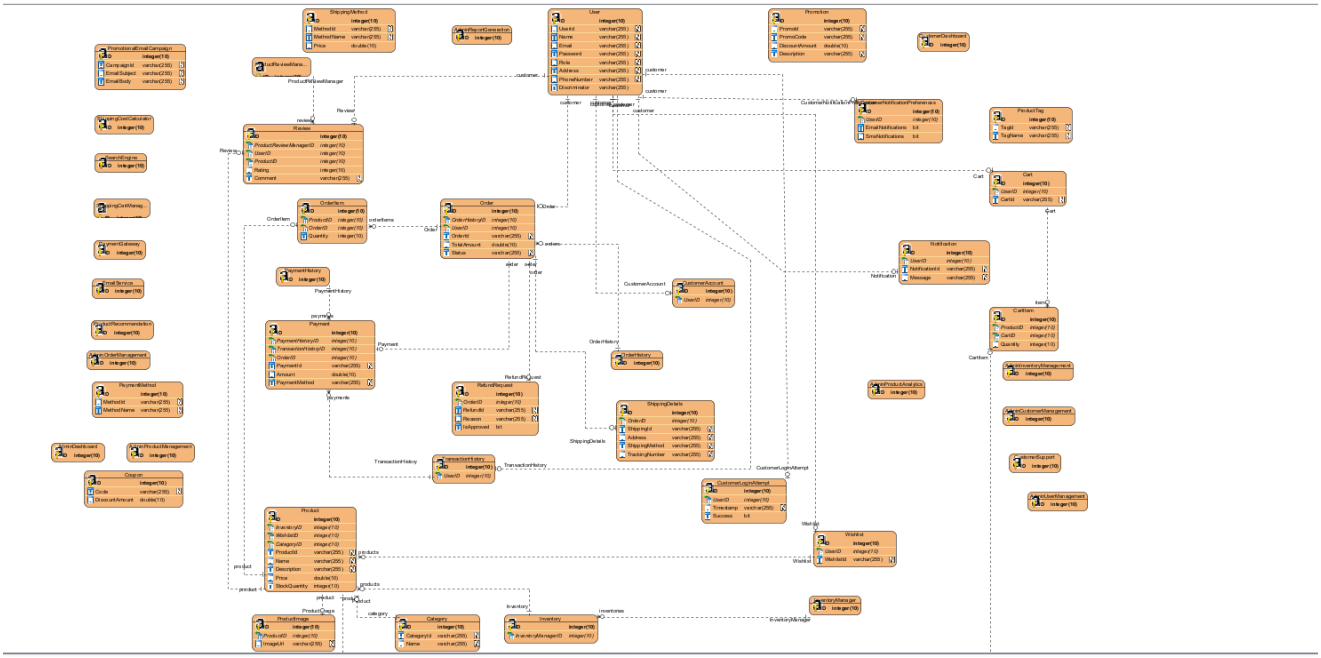
Field Name	Description	Data Type	Size	Format	Valid Values	Default	Notes
order_id	Unique identifier for each order member	Integer	10	-	-	Auto-increment	Primary key
customer_id	ID of the customer placing the order	Integer	10	-	-	-	Foreign key linking to Customer
status	Status of the order	Varchar	20	-	"pending", "shipped", "delivered", "canceled"	"pending"	Current order status
total_amount	Total cost of the order	Decimal	10,2	-	>=0	0.0	-
shipping_address	Shipping address for the order	Varchar	255	-	-	-	-

- Payment

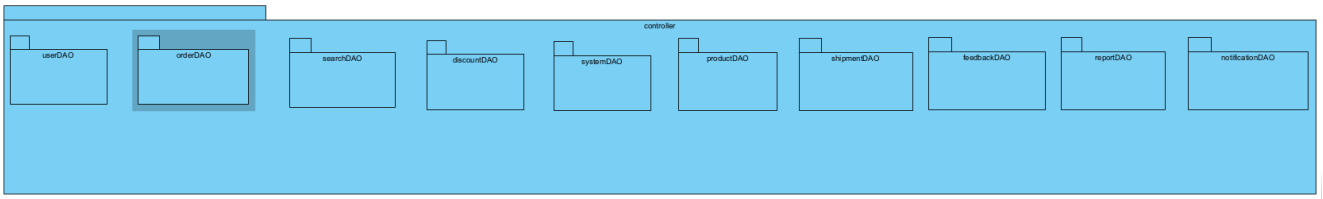
Field Name	Description	Data Type	Size	Format	Valid Values	Default	Notes

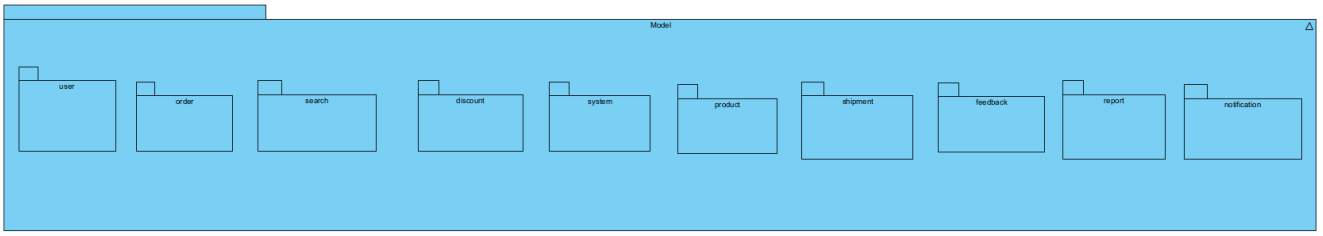
payment_id	Unique identifier for each payment	Integer	10	-	-	Auto-increment	Primary key
order_id	Associated order ID	Integer	10	-	-	-	Foreign key linking to Order
payment_date	Date of payment	Date	-	YYYY-MM-DD	-	-	-
amount	Amount paid	Decimal	10,2	-	>=0	0.0	-
payment_method	Method of payment	Varchar	50	-	"credit card", "PayPal", "bank transfer", etc.	"credit card"	-
status	Status of the payment	Varchar	20	-	"completed", "pending", "failed"	"pending"	Current payment status

- Create data model

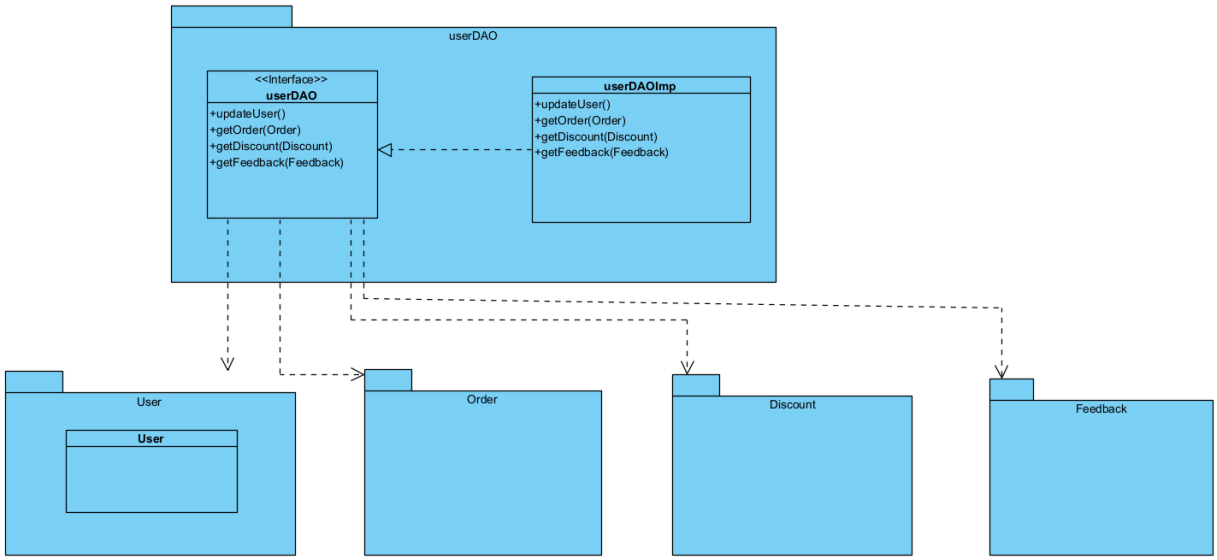


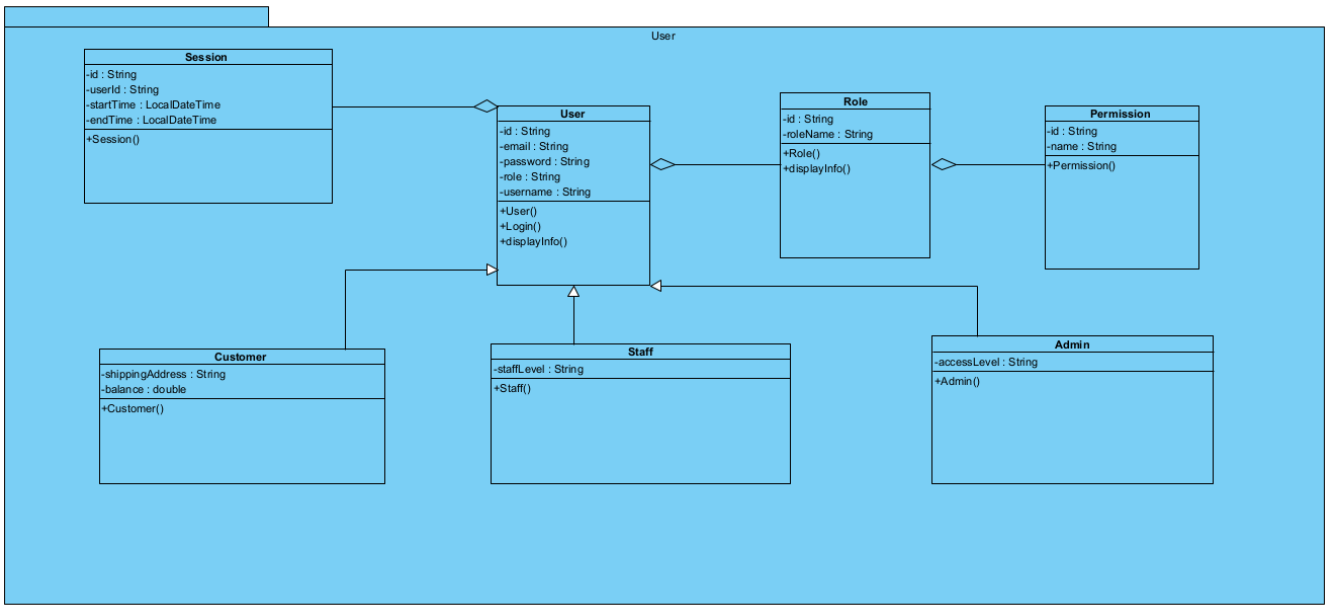
2. Design
    - Overall design



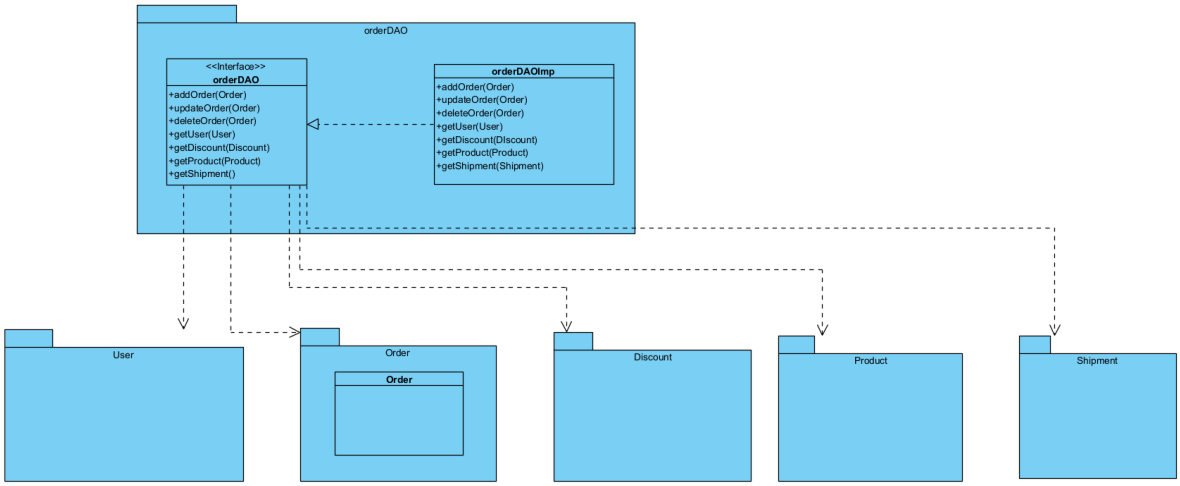


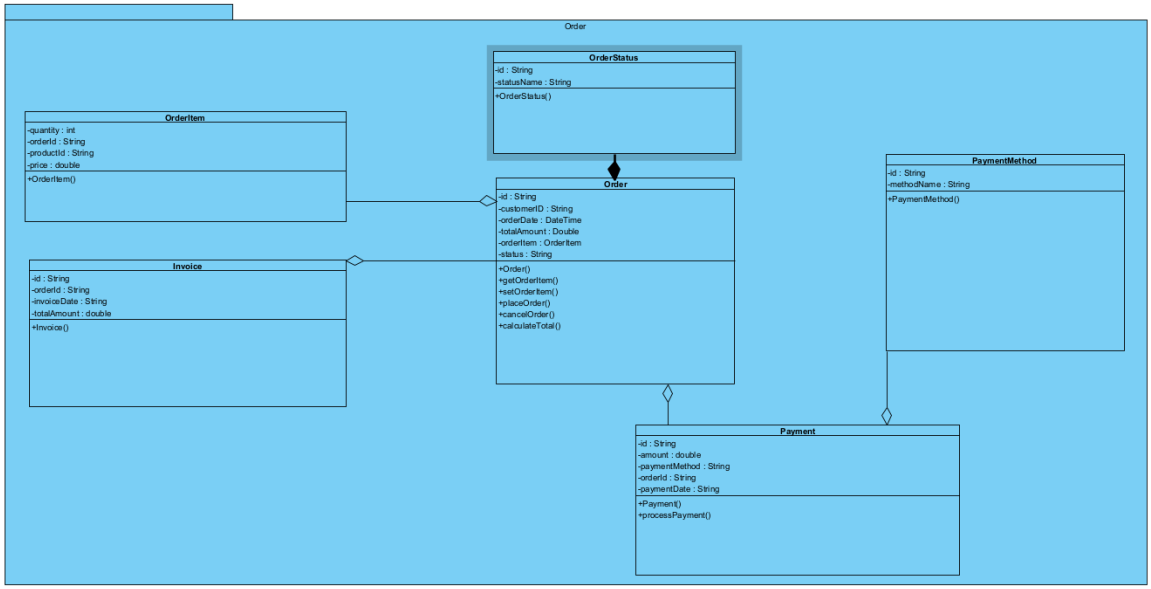
- Detailed design
  - User



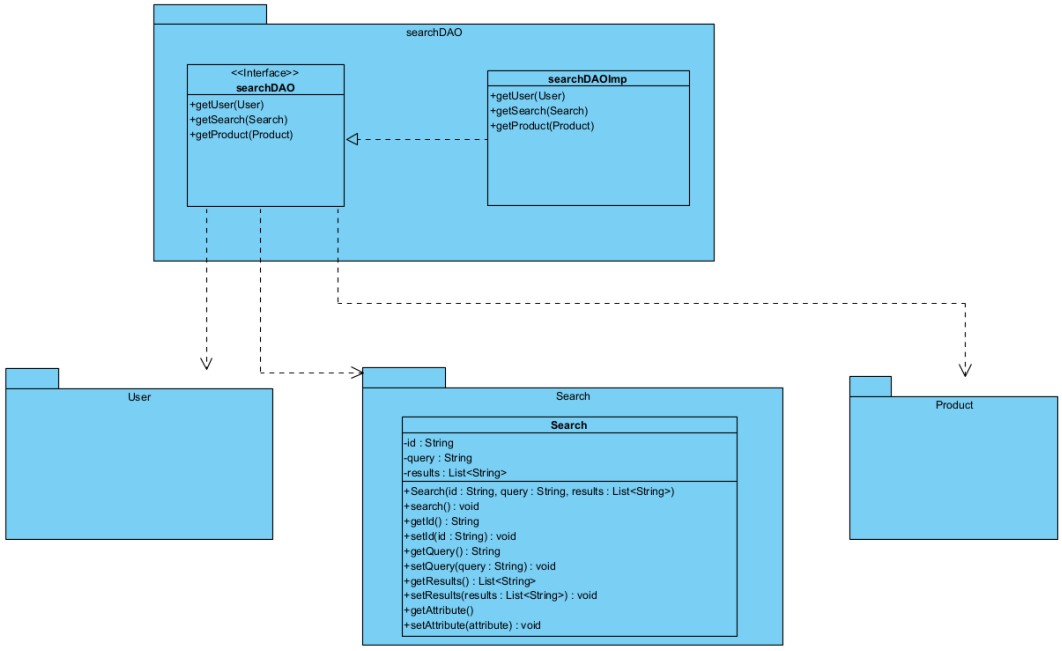


  - Order

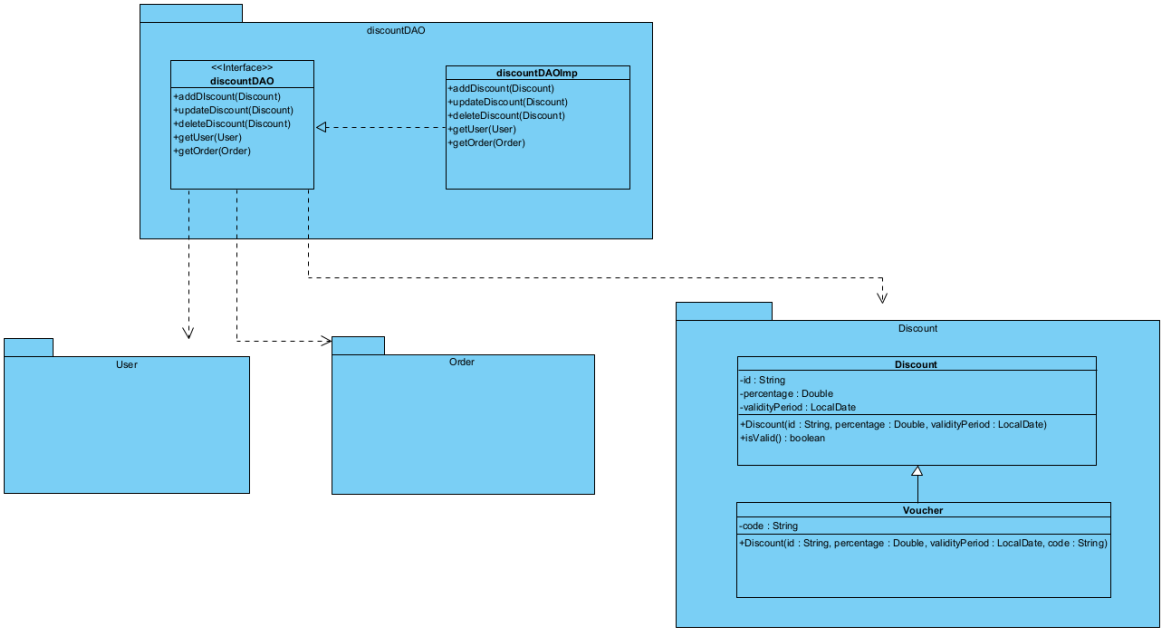




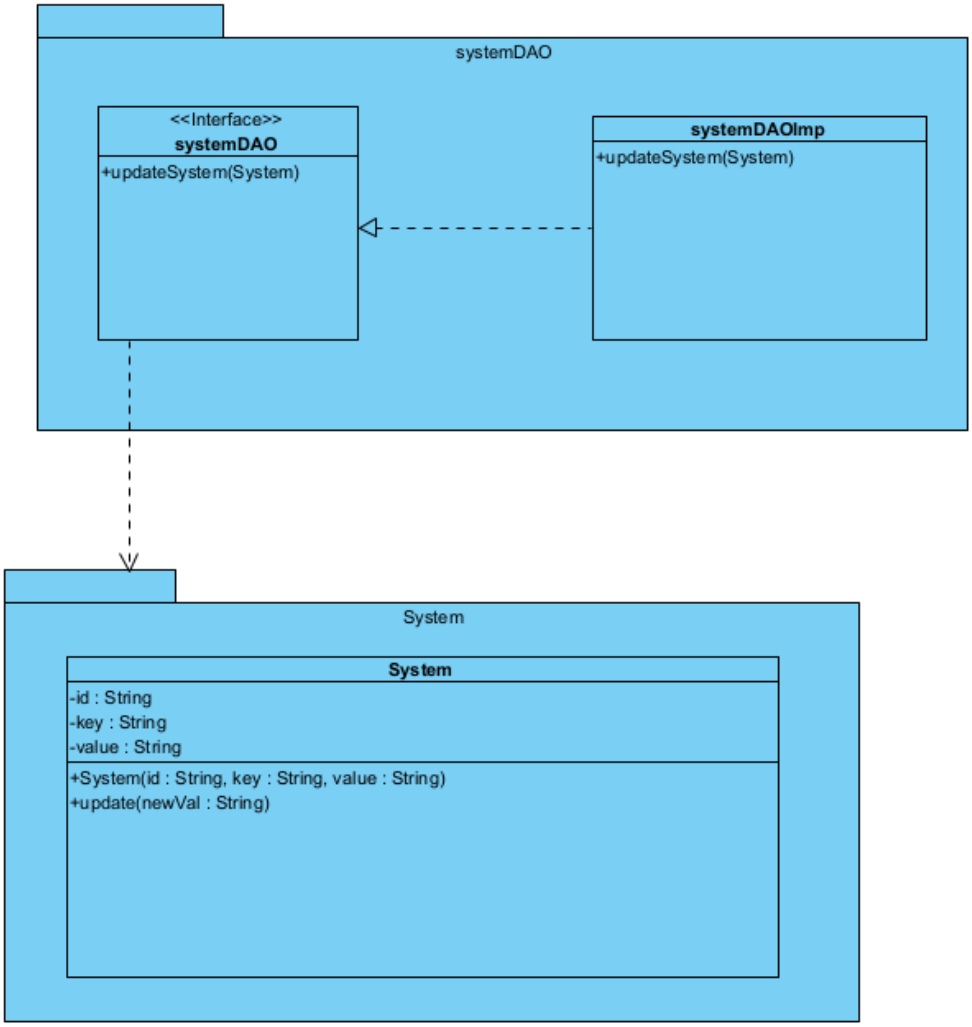
- Search



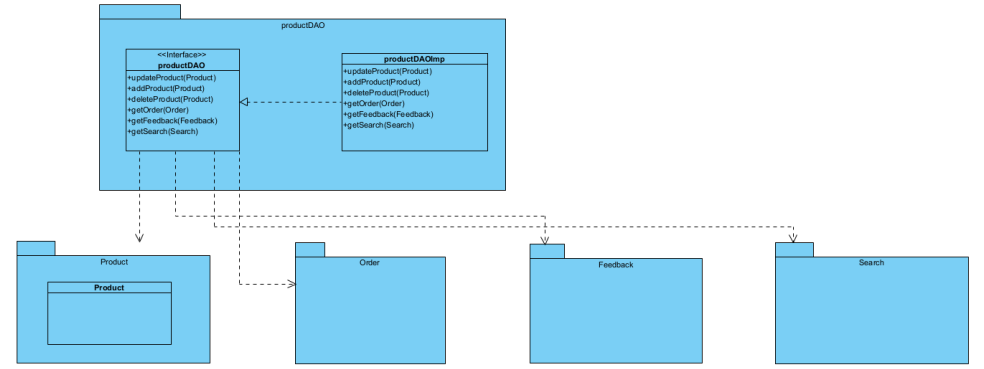
- Discount

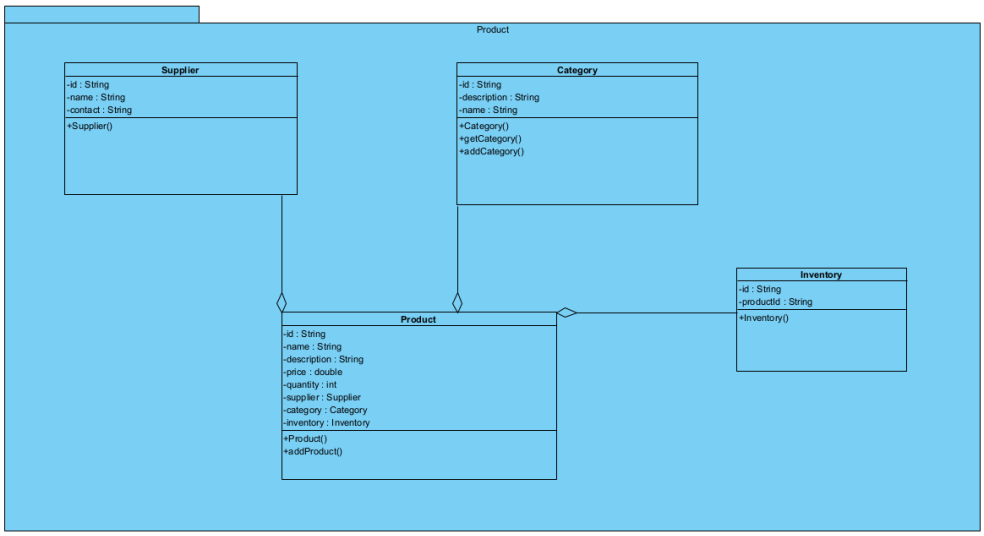


- System

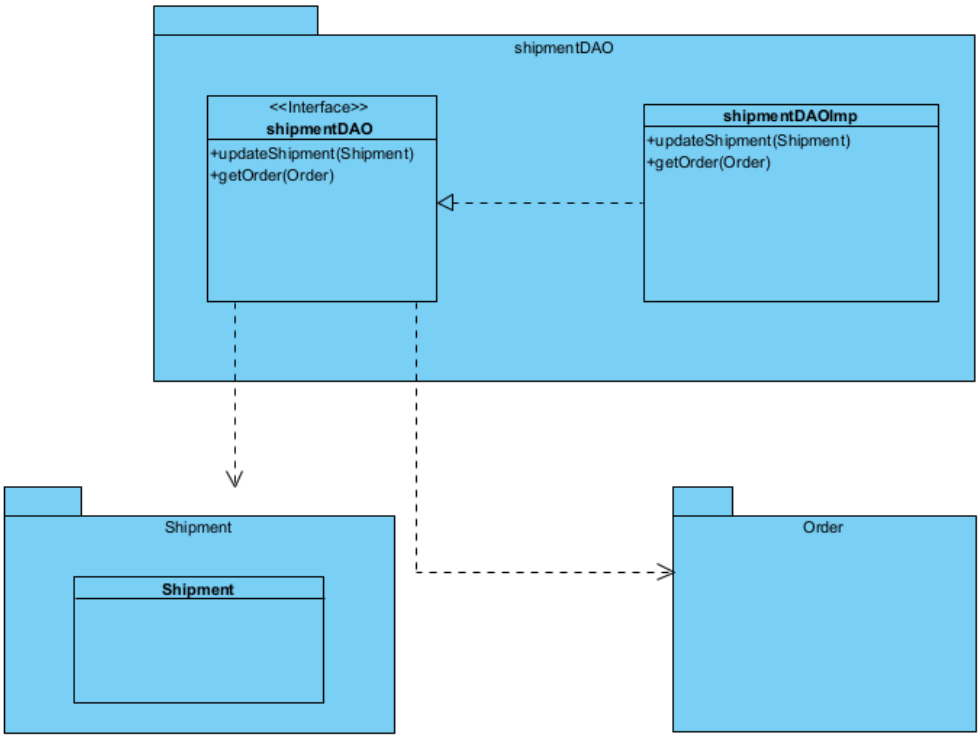


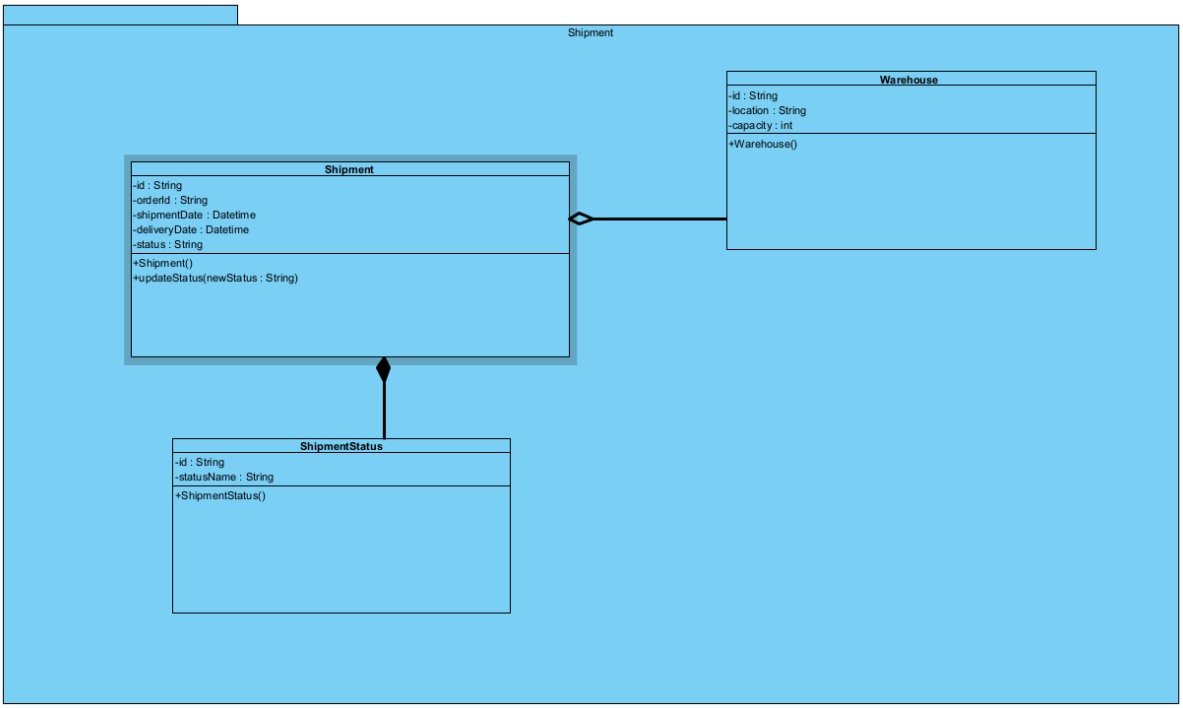
- Product


productDAO
<<Interface>>
productDAO
+updateProduct(Product)
+addProduct(Product)
+deleteProduct(Product)
+getOrder(Order)
+getFeedback(Feedback)
+getSearch(Search)
productDAOImp
+updateProduct(Product)
+addProduct(Product)
+deleteProduct(Product)
+getOrder(Order)
+getFeedback(Feedback)
+getSearch(Search)
Product
Product
Order
Feedback
Search

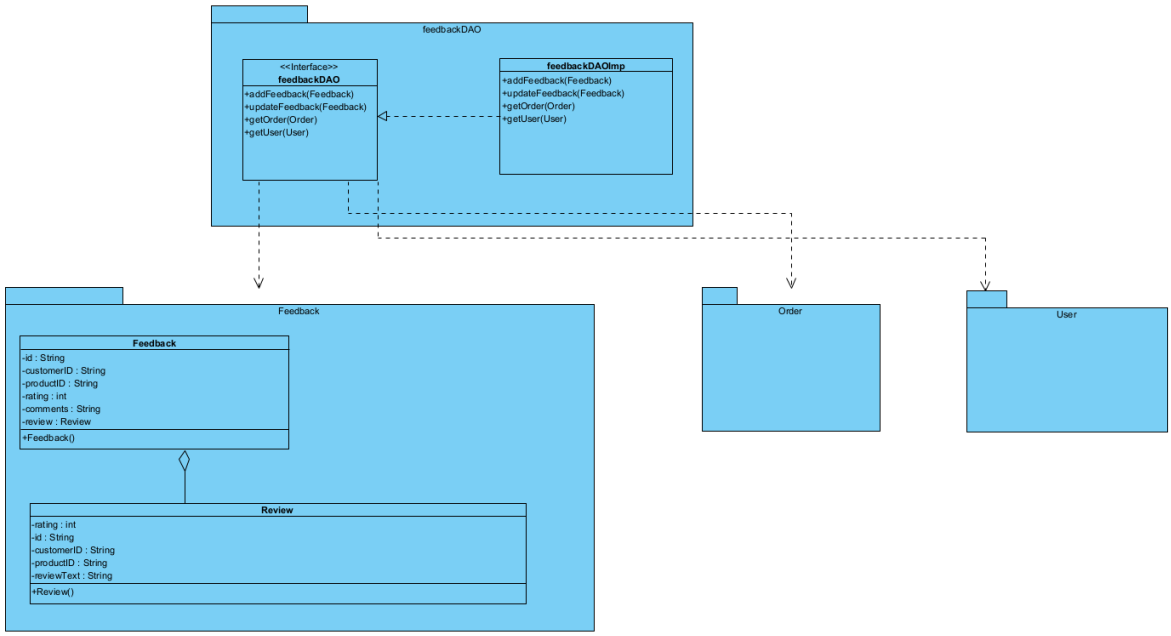

Product
Supplier
-id : String
-name : String
-contact : String
+Supplier()
Category
-id : String
-description : String
-name : String
+Category()
+getCategory()
+addCategory()
Inventory
-id : String
-productId : String
+Inventory()
Product
-id : String
-name : String
-description : String
-price : double
-quantity : int
-supplier : Supplier
-category : Category
-inventory : Inventory
+Product()
+addProduct()

- Shipment

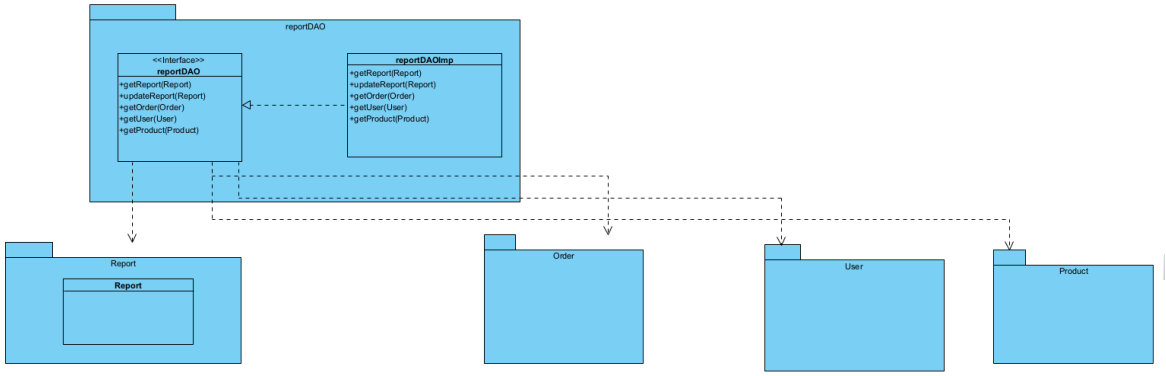

shipmentDAO
<<Interface>>
shipmentDAO
+updateShipment(Shipment)
+getOrder(Order)
shipmentDAOImp
+updateShipment(Shipment)
+getOrder(Order)
Shipment
Shipment
Order



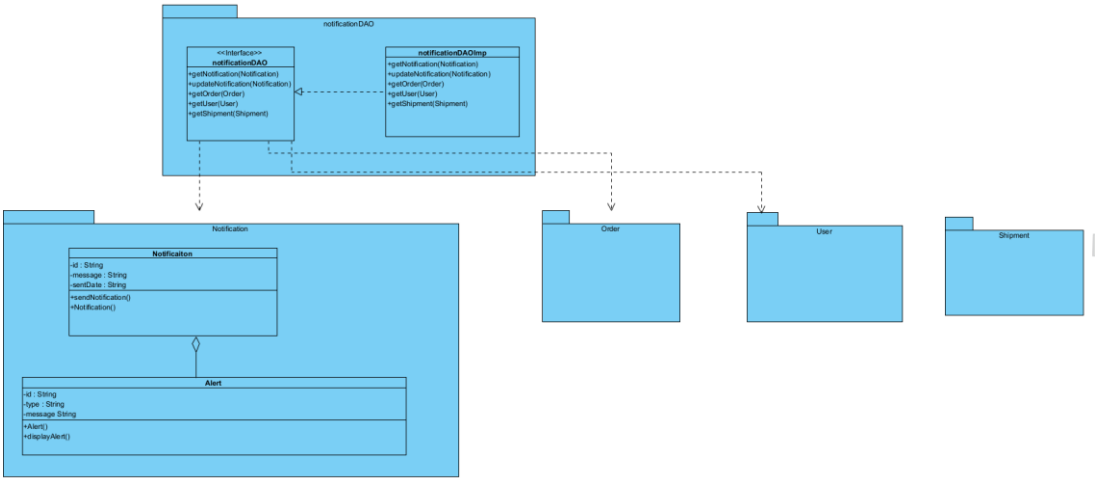
- FeedBack



- Report



- Notification



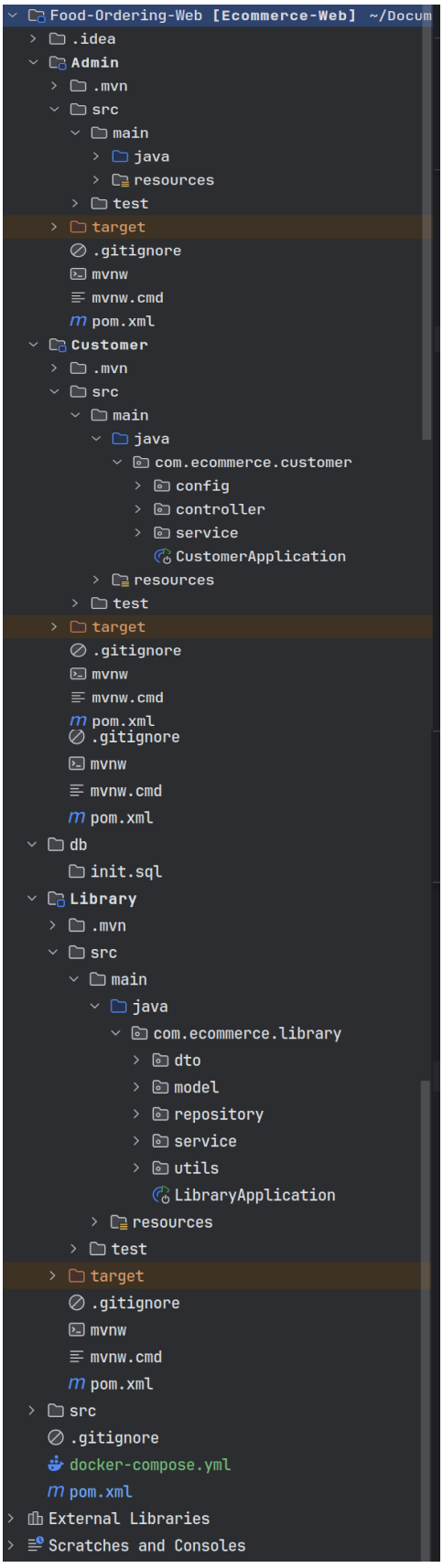
- Design forms such as bill, report....
  - Bill

Hóa đơn				
Khách hàng	Nguyễn Văn A			
SDT				
Địa chỉ				
STT	Tên sản phẩm	Số lượng	Đơn giá	Thành tiền
1	Laptop	1	1000	1000
2	Áo khoác	3	200	600
			Tổng tiền	1600

  - FeedBack

Feedback	
Khách hàng	Nguyễn Văn A
SDT	
Địa chỉ	
Rating	*****
Comment	
Rất hài lòng về dịch vụ!!	
	Gửi

3. Programming and Integration
    - Project structure



- Code

```
package com.ecommerce.admin.config;


import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;


@Configuration
@EnableWebSecurity
public class AdminConfiguration {


  @Bean
  public UserDetailsService userDetailsService(){
     return new AdminServiceConfig();
  }


  @Bean
  public BCryptPasswordEncoder passwordEncoder(){
     return new BCryptPasswordEncoder();
  }



  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
     AuthenticationManagerBuilder authenticationManagerBuilder
           = http.getSharedObject(AuthenticationManagerBuilder.class);


     authenticationManagerBuilder
           .userDetailsService(userDetailsService())
           .passwordEncoder(passwordEncoder());


     AuthenticationManager authenticationManager = authenticationManagerBuilder.build();


     http
           .csrf(AbstractHttpConfigurer::disable)
           .authorizeHttpRequests( author ->
                 author.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
                       .requestMatchers("/admin/**").hasAuthority("ADMIN")
                       .requestMatchers("/forgot-password", "/register", "/register-new").permitAll()
                       .anyRequest().authenticated()


           )
           .formLogin(login ->
                 login.loginPage("/login")
                       .loginProcessingUrl("/do-login")
                       .defaultSuccessUrl("/index", true)
                       .permitAll()
           )
           .logout(logout ->
                 logout.invalidateHttpSession(true)
                       .clearAuthentication(true)
                       .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                       .logoutSuccessUrl("/login?logout")
                       .permitAll()
           )
           .authenticationManager(authenticationManager)
```

```
            .sessionManagement(session ->
                    session.sessionCreationPolicy(SessionCreationPolicy.ALWAYS)
            )
        ;
        return http.build();
    }

}
```

```
package com.ecommerce.admin.config;

import com.ecommerce.library.model.Admin;
import com.ecommerce.library.model.Role;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class AdminDetails implements UserDetails {
    private Admin admin;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        List<SimpleGrantedAuthority> authorities = new ArrayList<>();
        for (Role role : admin.getRoles()){
            authorities.add(new SimpleGrantedAuthority(role.getName()));
        }
        return authorities;
    }

    @Override
    public String getPassword() {
        return admin.getPassword();
    }

    @Override
    public String getUsername() {
        return admin.getUserName();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

```
package com.ecommerce.admin.config;

import com.ecommerce.library.model.Admin;
```

```java
import com.ecommerce.library.repository.AdminRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;


import java.util.stream.Collectors;
@Service
public class AdminServiceConfig implements UserDetailsService {
  @Autowired
  private AdminRepository adminRepository;


  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
     Admin admin = adminRepository.findByUserName(username);
     if (admin == null) {
        throw new UsernameNotFoundException("Could not find username");
     }
     return new User(
           admin.getUserName(),
           admin.getPassword(),
           admin.getRoles()
                 .stream()
                 .map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList()));
  }
}
package com.ecommerce.admin.controller;


import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;


@Controller
public class AdminController {
//    @GetMapping("/categories")
//    public String categories(Model model){
//        model.addAttribute("title", "Category");
//        return "categories";
//    }
}
package com.ecommerce.admin.controller;


import com.ecommerce.library.model.Category;
import com.ecommerce.library.service.CategoryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;


import java.security.Principal;
import java.util.List;


@Controller
public class CategoryController {
  @Autowired
  private CategoryService categoryService;
  @GetMapping("/categories")
  public String categories(Model model, Principal principal){
     if(principal == null){
        return "redirect:/login";
     }
     List<Category> categories = categoryService.findAll();
     model.addAttribute("categories", categories);
```

```
        model.addAttribute("size", categories.size());
        model.addAttribute("title", "Category");
        model.addAttribute("categoryNew", new Category());
        return "categories";
    }
    @PostMapping("/add-category")
    public String add(@ModelAttribute("categoryNew") Category category, RedirectAttributes attributes){
        try{
            categoryService.save(category);
            attributes.addFlashAttribute("success", "Added successfully!");
        }catch(DataIntegrityViolationException e){
            e.printStackTrace();
            attributes.addFlashAttribute("failed", "Failed to add because duplicate name");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("failed", "Error server");
        }
        return "redirect:/categories";
    }


    @RequestMapping(value = "/findById/{id}", method = RequestMethod.GET)
    @ResponseBody
    public Category findById(@PathVariable Long id){
        return categoryService.findById(id);
    }



    @GetMapping("/update-category")
    public String update(Category category, RedirectAttributes attributes){
        try{
            categoryService.update(category);
            attributes.addFlashAttribute("success", "Updated successfully");
        }catch(DataIntegrityViolationException e){
            e.printStackTrace();
            attributes.addFlashAttribute("failed", "Failed to update because duplicate name");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("failed","Error server");
        }
        return "redirect:/categories";
    }


    @RequestMapping(value = "/delete-category/{id}", method = {RequestMethod.GET, RequestMethod.PUT})
    public String delete(@PathVariable("id") Long id, RedirectAttributes attributes){
        try{
            categoryService.deleteById(id);
            attributes.addFlashAttribute("success", "Deleted successfully");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("failed", "Failed to deleted" );
        }
        return "redirect:/categories";
    }


    @RequestMapping(value = "/enable-category/{id}", method = {RequestMethod.PUT, RequestMethod.GET})
    public String enable(@PathVariable("id") Long id, RedirectAttributes attributes){
        try{
            categoryService.enableById(id);
            attributes.addFlashAttribute("success","Enabled successfully");
        }catch(Exception e) {
            e.printStackTrace();
            attributes.addFlashAttribute("failed", "Failed to enabled");
        }
        return "redirect:/categories";
    }


}
package com.ecommerce.admin.controller;
```

```java
import com.ecommerce.library.dto.AdminDto;
import com.ecommerce.library.model.Admin;
import com.ecommerce.library.service.AdminService;
import com.ecommerce.library.service.impl.AdminServiceImpl;
import jakarta.servlet.http.HttpSession;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.kafka.KafkaProperties;
import org.springframework.boot.autoconfigure.neo4j.Neo4jProperties;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;


@Controller
public class LoginController {
  @Autowired
  private AdminServiceImpl adminService;


  @Autowired
  private BCryptPasswordEncoder passwordEncoder;


  @GetMapping("/login")
  public String loginForm(Model model) {
     model.addAttribute("title","Login Admin");
     return "login";
  }


  @RequestMapping("/index")
  public String  home(Model model){
     model.addAttribute("title", "Home Page");
     Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
     if(authentication == null || authentication instanceof AnonymousAuthenticationToken){
        return "redirect:/login";
     }
     return"index";
  }


  @GetMapping("/register")
  public String register(Model model){
     model.addAttribute("title","Register");
     model.addAttribute("adminDto", new AdminDto());
     return "register";
  }


  @GetMapping("/forgot-password")
  public String forgotPassword(Model model){
     model.addAttribute("title","Forgot pasword");
     return "forgot-password";
  }


  @PostMapping("/register-new")
  public String addNewAdmin(@Valid @ModelAttribute("adminDto") AdminDto adminDto,
                            BindingResult result,
                            Model model){
     try {
        if(result.hasErrors()){
           model.addAttribute("adminDto", adminDto);
           result.toString();
```

```
            return "register";
        }
        String username = adminDto.getUsername();
        Admin admin = adminService.findByUsername(username);
        if(admin != null){
            model.addAttribute("adminDto", adminDto);
            System.out.println("admin not null");
            model.addAttribute("emailError","Your email has been registered!");
            return "register";
        }
        if(adminDto.getPassword().equals(adminDto.getRepeatPassword())){
            adminDto.setPassword(passwordEncoder.encode(adminDto.getPassword()));
            adminService.save(adminDto);
            System.out.println("success");
            model.addAttribute("success","Register successfully!");
            model.addAttribute("adminDto", adminDto);
        }
        else{
            model.addAttribute("adminDto", adminDto);
            model.addAttribute("passwordError","Your password maybe wrong! Check again!");
            System.out.println("password not same");
            return "register";
        }

      } catch (Exception e){
        e.printStackTrace();
        model.addAttribute("errors","The server has been wrong!");
      }
      return "register";
  }
}
package com.ecommerce.admin.controller;

import com.ecommerce.library.dto.ProductDto;
import com.ecommerce.library.model.Category;
import com.ecommerce.library.model.Product;
import com.ecommerce.library.service.CategoryService;
import com.ecommerce.library.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.security.Principal;
import java.util.List;

@Controller
public class ProductController {
  @Autowired
  private ProductService productService;
  @Autowired
  private CategoryService categoryService;

  @GetMapping("/products")
  public String products(Model model, Principal principal){
      if(principal == null){
          return "redirect:/login";
      }
      List<ProductDto> productDtoList = productService.findAll();
      model.addAttribute("title", "Manager Product");
      model.addAttribute("product", productDtoList);
      model.addAttribute("size", productDtoList.size());
      return "products";
  }
```

```
  @GetMapping("/products/{pageNo}")
  public String productsPage(@PathVariable("pageNo") int pageNo, Model model, Principal principal){
      if(principal == null){
          return "redirect:/login";
      }
      Page<ProductDto> products = productService.pageProducts(pageNo);
      model.addAttribute("title","Manager Product");
      model.addAttribute("size", products.getSize());
      model.addAttribute("totalPages", products.getTotalPages());
      model.addAttribute("currentPage", pageNo);
      model.addAttribute("products", products);
      return "products";
  }


  @GetMapping("/search-result/{pageNo}")
  public String searchProducts(@PathVariable("pageNo") int pageNo,@RequestParam("keyword") String keyword, Model model,
Principal principal){
      if(principal == null){
          return "redirect:/login";
      }
      Page<ProductDto> products = productService.searchProducts(pageNo, keyword);
      model.addAttribute("title","Search Result");
      model.addAttribute("products", products);
      model.addAttribute("size", products.getSize());
      model.addAttribute("currentPage", pageNo);
      model.addAttribute("totalPages", products.getTotalPages());
      return "result-products";
  }


  @GetMapping("/add-product")
  public String addProductForm(Model model, Principal principal){
      if(principal == null){
          return "redirect:/login";
      }
      List<Category> categories = categoryService.findAllByActivated();
      model.addAttribute("categories", categories);
      model.addAttribute("product", new ProductDto());
      return "add-product";
  }


  @PostMapping("/save-product")
  public String saveProduct(@ModelAttribute("product")ProductDto productDto,
                            @RequestParam("imageProduct")MultipartFile imageProduct,
                            RedirectAttributes attributes){
      try{
          productService.save(imageProduct, productDto);
          attributes.addFlashAttribute("success", "Add successfully!");
      }catch(Exception e){
          e.printStackTrace();
          attributes.addFlashAttribute("error", "Add failed!");
      }
      return "redirect:/products/0";
  }


  @GetMapping("/update-product/{id}")
  public String updateProductForm(@PathVariable("id") Long id, Model model, Principal principal){
      if(principal == null){
          return "redirect:/login";
      }
      model.addAttribute("title", "Update products");
      List<Category> categories = categoryService.findAllByActivated();
      ProductDto productDto = productService.getById(id);
      model.addAttribute("categories", categories);
      model.addAttribute("productDto", productDto);
      return "update-product";
  }


  @PostMapping("/update-product/{id}")
  public String processUpdate(@PathVariable("id") Long id,
```

```
                        @ModelAttribute("productDto") ProductDto productDto,
                        @RequestParam("imageProduct") MultipartFile imageProduct,
                        RedirectAttributes attributes){
        try{
            productService.update(imageProduct, productDto);
            attributes.addFlashAttribute("success", "Update successfully!");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("error", "Update failed!");
        }
        return "redirect:/products/0";
    }

    @RequestMapping(value = "/enable-product/{id}", method = {RequestMethod.PUT, RequestMethod.GET})
    public String enabledProduct(@PathVariable("id") Long id, RedirectAttributes attributes){
        try{
            productService.enableById(id);
            attributes.addFlashAttribute("success", "Enabled successfully!");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("error", "Enabled failed!");
        }
        return "redirect:/products/0";
    }

    @RequestMapping(value = "/delete-product/{id}", method = {RequestMethod.PUT, RequestMethod.GET})
    public String deletedProduct(@PathVariable("id") Long id, RedirectAttributes attributes){
        try{
            productService.deleteById(id);
            attributes.addFlashAttribute("success", "Delete successfully!");
        }catch(Exception e){
            e.printStackTrace();
            attributes.addFlashAttribute("error", "Delete failed!");
        }
        return "redirect:/products/0";
    }
}
package com.ecommerce.admin;

import jakarta.persistence.Entity;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication(scanBasePackages = {"com.ecommerce.library.*", "com.ecommerce.admin.*"})
@EnableJpaRepositories(value = "com.ecommerce.library.repository")
@EntityScan(value = "com.ecommerce.library.model")
public class AdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(AdminApplication.class, args);
    }
}
package com.ecommerce.customer.config;

import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
```

```java
@Configuration
@EnableWebSecurity
public class CustomerConfiguration {

  @Bean
  public UserDetailsService userDetailsService() {
     return new CustomerServiceConfig();
  }

  @Bean
  public BCryptPasswordEncoder passwordEncoder() {
     return new BCryptPasswordEncoder();
  }



  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
     AuthenticationManagerBuilder authenticationManagerBuilder
           = http.getSharedObject(AuthenticationManagerBuilder.class);


     authenticationManagerBuilder
           .userDetailsService(userDetailsService())
           .passwordEncoder(passwordEncoder());


     AuthenticationManager authenticationManager = authenticationManagerBuilder.build();


     http
           .csrf(AbstractHttpConfigurer::disable)
           .authorizeHttpRequests( author ->
                 author.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
                       .requestMatchers("/*", "/product-detail/**").permitAll()
                       .requestMatchers("/shop/**", "/find-product/**").permitAll()
                       .requestMatchers("/*", "/products-in-category/**").permitAll()
                       .requestMatchers("/*", "/shop/cart").hasAuthority("CUSTOMER")
                       .requestMatchers("/do-login").permitAll()
           )
           .formLogin(login ->
                 login.loginPage("/login")
                       .loginProcessingUrl("/do-login")
                       .defaultSuccessUrl("/index", true)
                       .permitAll()
           )
           .logout(logout ->
                 logout.invalidateHttpSession(true)
                       .clearAuthentication(true)
                       .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                       .logoutSuccessUrl("/login?logout")
                       .permitAll()
           )
           .authenticationManager(authenticationManager)
           .sessionManagement(session ->
                 session.sessionCreationPolicy(SessionCreationPolicy.ALWAYS)
           )
     ;
     return http.build();
  }

//   @Bean
//   public WebSecurityCustomizer webSecurityCustomizer() {
//      return (web) ->
//            web.ignoring()
//                  .requestMatchers("/js/**", "/css/**");
//   }


}
package com.ecommerce.customer.config;
```

```java
import com.ecommerce.library.model.Customer;
import com.ecommerce.library.model.Role;
import org.springframework.boot.autoconfigure.task.TaskSchedulingProperties;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class CustomerDetails implements UserDetails {

   private Customer customer;
   @Override
   public Collection<? extends GrantedAuthority> getAuthorities() {
      List<SimpleGrantedAuthority> authorities = new ArrayList<>();
      for (Role role : customer.getRoles()) {
         authorities.add(new SimpleGrantedAuthority(role.getName()));
      }
      return authorities;
   }

   @Override
   public String getPassword() {
      return customer.getPassword();
   }

   @Override
   public String getUsername() {
      return customer.getUsername();
   }

   @Override
   public boolean isAccountNonExpired() {
      return true;
   }

   @Override
   public boolean isAccountNonLocked() {
      return true;
   }

   @Override
   public boolean isCredentialsNonExpired() {
      return true;
   }

   @Override
   public boolean isEnabled() {
      return true;
   }
}
package com.ecommerce.customer.config;

import com.ecommerce.library.model.Customer;
import com.ecommerce.library.repository.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
```

```
import java.util.stream.Collectors;


@Service
public class CustomerServiceConfig implements UserDetailsService {


   @Autowired
   private CustomerRepository customerRepository;
   @Override
   public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      Customer customer = customerRepository.findByUsername(username);
      if(customer == null) {
         throw new UsernameNotFoundException("Customer not found");
      }
      return new User(customer.getUsername(), customer.getPassword(), customer.getRoles().stream().map(role -> new
SimpleGrantedAuthority(role.getName())).collect(Collectors.toList()));
   }
}
package com.ecommerce.customer.config;


import java.util.HashMap;
import java.util.Map;


import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;


import com.paypal.api.payments.PaymentHistory;
import com.paypal.base.rest.APIContext;
import com.paypal.base.rest.OAuthTokenCredential;
import com.paypal.base.rest.PayPalRESTException;


@Configuration
public class PaypalConfiguration {


   @Value("${paypal.client.id}")
   private String clientId;
   @Value("${paypal.client.secret}")
   private String clientSecret;
   @Value("${paypal.mode}")
   private String mode;


   @Bean
   public Map<String, String> paypalSdkConfig() {
      Map<String, String> configMap = new HashMap<>();
      configMap.put("mode", mode);
      return configMap;
   }


   @Bean
   public OAuthTokenCredential oAuthTokenCredential() {
      return new OAuthTokenCredential(clientId, clientSecret, paypalSdkConfig());
   }


   @Bean
   public APIContext apiContext() throws PayPalRESTException {
      APIContext context = new APIContext(oAuthTokenCredential().getAccessToken());
      context.setConfigurationMap(paypalSdkConfig());
      return context;
   }


}
package com.ecommerce.customer.controller;



import com.ecommerce.library.model.City;
import com.ecommerce.library.model.Customer;
```

```java
import com.ecommerce.library.service.CityService;
import com.ecommerce.library.service.CustomerService;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.Banner;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;


import java.security.Principal;
import java.util.List;


@Controller
public class AccountController {


  @Autowired
  private CustomerService customerService;


  @GetMapping("/account")
  public String accountHome(Model model, Principal principal){
     if(principal == null){
        return "redirect:/login";
     }
     String username = principal.getName();
     Customer customer = customerService.findByUsername(username);
     model.addAttribute("customer", customer);


     return "account";
  }


  @RequestMapping(value = "/update-infor", method = {RequestMethod.GET, RequestMethod.PUT})
  public String updateCustomer(
        @ModelAttribute("customer") Customer customer,
        Model model,
        RedirectAttributes redirectAttributes,
        Principal principal){
     if(principal == null){
        return "redirect:/login";
     }
     Customer customerSaved = customerService.saveInfor(customer);
     redirectAttributes.addFlashAttribute("customer", customerSaved);
     return "redirect:/account";
  }
}
package com.ecommerce.customer.controller;



import com.ecommerce.library.dto.CustomerDto;
import com.ecommerce.library.model.Customer;
import com.ecommerce.library.service.CustomerService;
import com.ecommerce.library.service.impl.CustomerServiceImpl;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;


@Controller
public class AuthController {


  @Autowired
  private CustomerService customerService;
```

```
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login(){
        return "login";
    }

    @GetMapping("/register")
    public String register(Model model){
        model.addAttribute("customerDto", new CustomerDto());
        return "register";
    }

    @PostMapping("/do-register")
    public String processRegister(@Valid @ModelAttribute("customerDto")CustomerDto customerDto, BindingResult result, Model
model){
        try {
            if(result.hasErrors()){
                model.addAttribute("customerDto", customerDto);
                return "register";
            }
            Customer customer = customerService.findByUsername(customerDto.getUsername());
            if(customer != null){
                model.addAttribute("username", "Username have been registered!");
                model.addAttribute("customerDto", customerDto);
                return "register";
            }
            if (customerDto.getPassword().equals(customerDto.getRepeatPassword())){
                customerDto.setPassword(passwordEncoder.encode(customerDto.getPassword()));
                customerService.save(customerDto);
                model.addAttribute("success", "Successfully registered!");
                return "register";
            }
            else{
                model.addAttribute("password", "Password does not same!");
                model.addAttribute("customerDto", customerDto);
                return "register";
            }

        }catch (Exception e){
            model.addAttribute("error", "Server have ran some problems");
            model.addAttribute("customerDto", customerDto);
        }
        return "register";
    }
}
package com.ecommerce.customer.controller;

import com.ecommerce.library.model.Customer;
import com.ecommerce.library.model.Product;
import com.ecommerce.library.model.ShoppingCart;
import com.ecommerce.library.service.CustomerService;
import com.ecommerce.library.service.ProductService;
import com.ecommerce.library.service.ShoppingCartService;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.security.Principal;

@Controller
public class CartController {
```

```
@Autowired
private CustomerService customerService;

@Autowired
private ShoppingCartService cartService;

@Autowired
private ProductService productService;

@GetMapping("/cart")
public String cart(Model model, Principal principal, HttpSession session){
    if(principal == null){
        return "redirect:/login";
    }
    String username = principal.getName();
    Customer customer = customerService.findByUsername(username);
    ShoppingCart shoppingCart = customer.getCart();
    if(shoppingCart == null){
        model.addAttribute("check", "No item in your cart");

    }
    double subTotal = shoppingCart.getTotalPrice();
    session.setAttribute("subTotal", subTotal);

    session.setAttribute("totalItems", shoppingCart.getTotalItems());
    model.addAttribute("subTotal", subTotal);
    model.addAttribute("shoppingCart", shoppingCart);
    return "cart";
}

@PostMapping("/add-to-cart")
public String addItemToCart(
        @RequestParam("id") Long productId,
        @RequestParam(value = "quantity", required = false, defaultValue = "1") int quantity,
        Principal principal,
        HttpServletRequest request){

    if(principal == null){
        return "redirect:/login";
    }
    Product product = productService.getProductById(productId);
    String username = principal.getName();
    Customer customer = customerService.findByUsername(username);

    ShoppingCart cart = cartService.addItemToCart(product,quantity,customer);
    return "redirect:" + request.getHeader("Referer");
}

@RequestMapping(value = "/update-cart", method = RequestMethod.POST, params = "action=update")
public String updateCart(@RequestParam("quantity") int quantity,
                         @RequestParam("id") Long productId,
                         Model model,
                         Principal principal){
    if(principal == null){
        return "redirect:/login";
    }else {
        String username = principal.getName();
        Customer customer = customerService.findByUsername(username);
        Product product = productService.getProductById(productId);
        ShoppingCart cart = cartService.updateItemInCart(product,quantity,customer);
        model.addAttribute("shoppingCart", cart);
        return "redirect:/cart";



    }
```

```
    }

    @RequestMapping(value = "/update-cart", method = RequestMethod.POST, params = "action=delete")
    public String deleteItemFromCart(@RequestParam("id") Long productId,
                                     Model model,
                                     Principal principal){
        if(principal == null){
            return "redirect:/login";
        }else {
            String username = principal.getName();
            Customer customer = customerService.findByUsername(username);
            Product product = productService.getProductById(productId);
            ShoppingCart cart = cartService.deleteItemFromCart(product,customer);
            model.addAttribute("shoppingCart", cart);
            return "redirect:/cart";
        }
    }

}
package com.ecommerce.customer.controller;



import com.ecommerce.library.dto.ProductDto;
import com.ecommerce.library.model.Category;
import com.ecommerce.library.model.Customer;
import com.ecommerce.library.model.Product;
import com.ecommerce.library.model.ShoppingCart;
import com.ecommerce.library.service.CategoryService;
import com.ecommerce.library.service.CustomerService;
import com.ecommerce.library.service.ProductService;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.security.Principal;
import java.util.List;

@Controller
public class HomeController {

    @Autowired
    private ProductService productService;

    @Autowired
    private CategoryService categoryService;

    @Autowired
    private CustomerService customerService;

    @RequestMapping(value = {"/index","/"}, method = RequestMethod.GET)
    public String home(Model model, Principal principal, HttpSession session){
        if(principal != null){
            session.setAttribute("username", principal.getName());
            Customer customer = customerService.findByUsername(principal.getName());
            ShoppingCart cart = customer.getCart();
            session.setAttribute("totalItems", cart.getTotalItems());
        }
        else {
            session.removeAttribute("username");
        }
        return "home";
    }
    @GetMapping("/home")
```

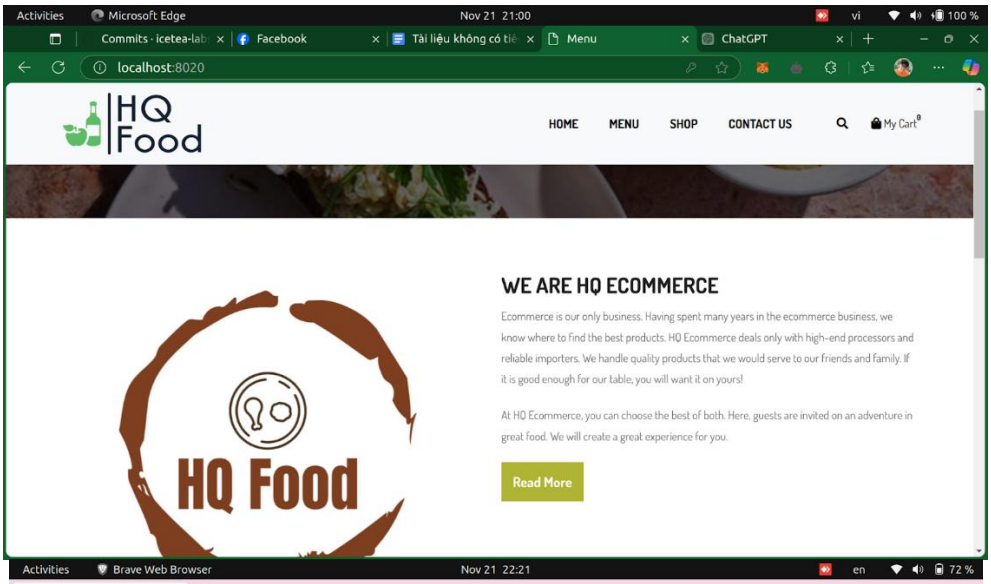
```
    public String index(Model model){
        List<Category> categories = categoryService.findAll();
        List<ProductDto> productDtos = productService.findAll();
        model.addAttribute("categories", categories);
        model.addAttribute("products", productDtos);
        return "index";
    }
}
package com.ecommerce.customer.controller;



import com.ecommerce.library.model.Customer;
import com.ecommerce.library.service.CustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;


import java.security.Principal;

@Controller
public class OrderController {


    @Autowired
    private CustomerService customerService;


    @GetMapping("/check-out")
    public String checkout(Model model, Principal principal) {
        if(principal == null) {
            return "redirect:/login";
        }
        String username = principal.getName();


        Customer customer = customerService.findByUsername(username);


        if(customer.getPhoneNumber().trim().isEmpty() || customer.getAddress().trim().isEmpty() ||
customer.getCity().trim().isEmpty() || customer.getCountry().trim().isEmpty()){
            model.addAttribute("customer", customer);
            model.addAttribute("error", "You must fill the information before checkout!");
            return "account";
        }


        return "checkout";
    }
}
package com.ecommerce.customer.controller;


import com.ecommerce.customer.service.PaypalService;
//import com.paypal.api.payments.Order;
import com.ecommerce.library.model.Order;
import com.ecommerce.library.model.Pay;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;


import com.paypal.api.payments.Links;
import com.paypal.api.payments.Payment;
import com.paypal.base.rest.PayPalRESTException;


@Controller
public class PaypalController {
```

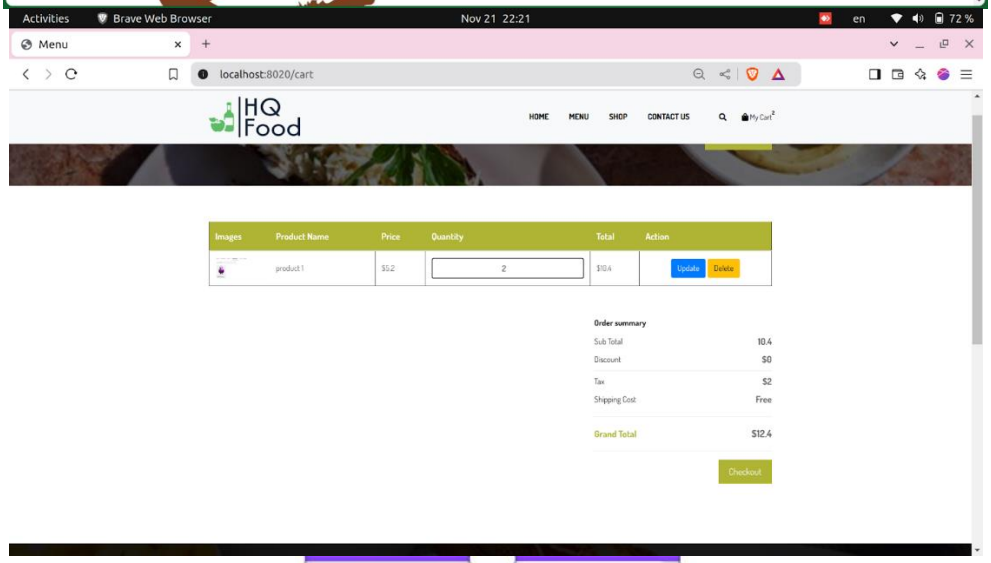
```
    @Autowired
    private PaypalService service;

    public static final String SUCCESS_URL = "payment/success";
    public static final String CANCEL_URL = "payment/cancel";

    @PostMapping("/pay")
    public String payment(Model model, HttpSession session, @ModelAttribute("order") Pay order) {
        try {

            double subTotal = (double) session.getAttribute("subTotal");
            model.addAttribute("subTotal", subTotal);

            Payment payment = service.createPayment(subTotal, order.getCurrency(), order.getMethod(),
                    order.getIntent(), order.getDescription(), "http://localhost:8020/" + CANCEL_URL,
                    "http://localhost:8020/" + SUCCESS_URL);
            for(Links link:payment.getLinks()) {
                if(link.getRel().equals("approval_url")) {
                    return "redirect:"+link.getHref();
                }
            }

        } catch (PayPalRESTException e) {

            e.printStackTrace();
        }
        return "redirect:/check-out";
    }

    @GetMapping(value = CANCEL_URL)
    public String cancelPay() {
        return "paymentCancel";
    }

    @GetMapping(value = SUCCESS_URL)
    public String successPay(@RequestParam("paymentId") String paymentId, @RequestParam("PayerID") String payerId) {
        try {
            Payment payment = service.executePayment(paymentId, payerId);
            System.out.println(payment.toJSON());
            if (payment.getState().equals("approved")) {
                return "paymentSuccess";
            }
        } catch (PayPalRESTException e) {
            System.out.println(e.getMessage());
        }
        return "redirect:/check-out";
    }

}
package com.ecommerce.customer.controller;

import com.ecommerce.library.dto.CategoryDto;
import com.ecommerce.library.model.Category;
import com.ecommerce.library.model.Product;
import com.ecommerce.library.service.CategoryService;
import com.ecommerce.library.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import java.util.List;

@Controller
```

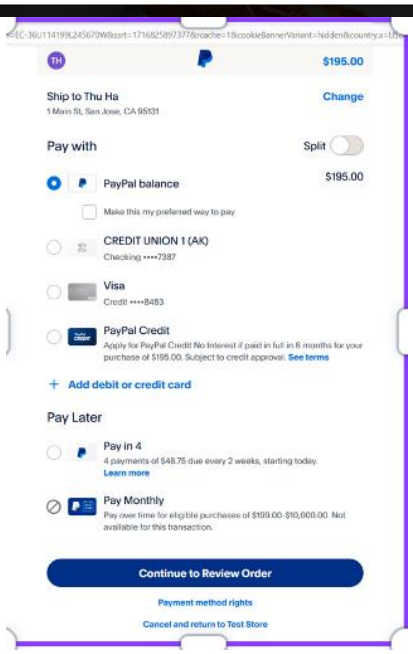
```
public class ProductController {

    @Autowired
    private ProductService productService;

    @Autowired
    private CategoryService categoryService;

    @GetMapping("/products")
    public String products(Model model){
        List<CategoryDto> categoryDtoList = categoryService.getCategoryAndProduct();
        List<Product> products = productService.getAllProducts();
        List<Product> listViewProducts = productService.listViewProducts();
        model.addAttribute("categories", categoryDtoList);
        model.addAttribute("viewProducts", listViewProducts);
        model.addAttribute("products", products);
        return "shop";
    }

    @GetMapping("/find-product/{id}")
    public String findProductById(@PathVariable("id") Long id, Model model){
        Product product = productService.getProductById(id);
        Long categoryId = product.getCategory().getId();
        List<Product> products = productService.getRelatedProducts(categoryId);
        model.addAttribute("product", product);
        model.addAttribute("products", products);
        return "product-detail";
    }

    @GetMapping("/products-in-category/{id}")
    public String getProductsInCategory(@PathVariable("id") Long categoryId, Model model){
        Category category = categoryService.findById(categoryId);
        List<CategoryDto> categories = categoryService.getCategoryAndProduct();
        List<Product> products = productService.getProductsInCategory(categoryId);
        model.addAttribute("categories", categories);
        model.addAttribute("products", products);
        model.addAttribute("category", category);
        return "products-in-category";
    }

    @GetMapping("/high-price")
    public String filterHighPrice(Model model){
        List<Category> categories = categoryService.findAllByActivated();
        List<CategoryDto> categoryDtoList = categoryService.getCategoryAndProduct();
        List<Product> products = productService.filterHighPrice();
        model.addAttribute("categoryDtoList", categoryDtoList);
        model.addAttribute("products", products);
        model.addAttribute("categories", categories);
        return "filter-high-price";
    }

    @GetMapping("/low-price")
    public String filterLowPrice(Model model){
        List<Category> categories = categoryService.findAllByActivated();
        List<CategoryDto> categoryDtoList = categoryService.getCategoryAndProduct();
        List<Product> products = productService.filterLowPrice();
        model.addAttribute("categoryDtoList", categoryDtoList);
        model.addAttribute("products", products);
        model.addAttribute("categories", categories);
        return "filter-low-price";
    }

}
package com.ecommerce.customer.service;

import java.math.BigDecimal;
import java.math.RoundingMode;
```

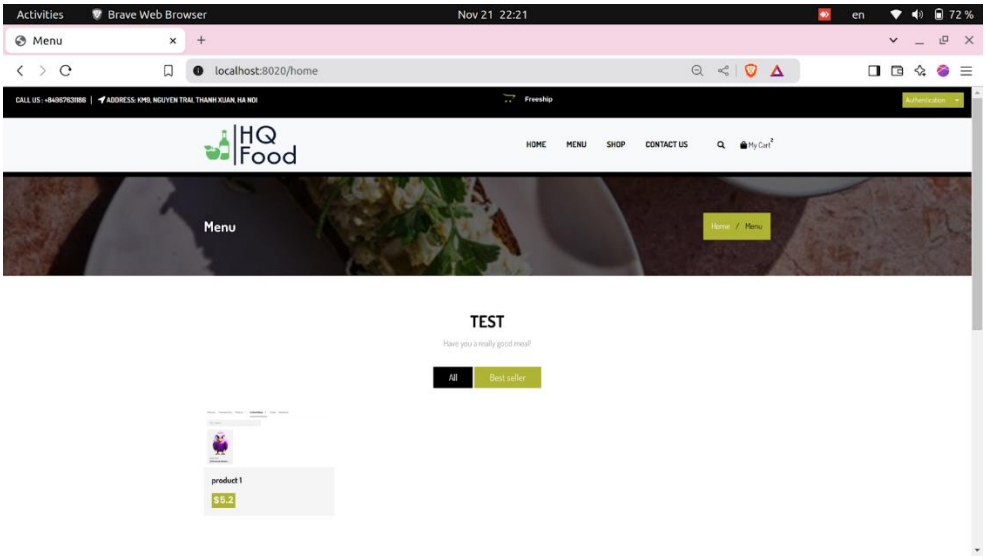
```java
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.paypal.api.payments.Amount;
import com.paypal.api.payments.Payer;
import com.paypal.api.payments.Payment;
import com.paypal.api.payments.PaymentExecution;
import com.paypal.api.payments.RedirectUrls;
import com.paypal.api.payments.Transaction;
import com.paypal.base.rest.APIContext;
import com.paypal.base.rest.PayPalRESTException;

@Service
public class PaypalService {

  @Autowired
  private APIContext apiContext;



  public Payment createPayment(
        Double total,
        String currency,
        String method,
        String intent,
        String description,
        String cancelUrl,
        String successUrl) throws PayPalRESTException{
    Amount amount = new Amount();
    amount.setCurrency(currency);
    total = new BigDecimal(total).setScale(2, RoundingMode.HALF_UP).doubleValue();
    amount.setTotal(String.format("%.2f", total));


    Transaction transaction = new Transaction();
    transaction.setDescription(description);
    transaction.setAmount(amount);


    List<Transaction> transactions = new ArrayList<>();
    transactions.add(transaction);


    Payer payer = new Payer();
    payer.setPaymentMethod(method.toString());


    Payment payment = new Payment();
    payment.setIntent(intent.toString());
    payment.setPayer(payer);
    payment.setTransactions(transactions);
    RedirectUrls redirectUrls = new RedirectUrls();
    redirectUrls.setCancelUrl(cancelUrl);
    redirectUrls.setReturnUrl(successUrl);
    payment.setRedirectUrls(redirectUrls);


    return payment.create(apiContext);
  }

  public Payment executePayment(String paymentId, String payerId) throws PayPalRESTException{
    Payment payment = new Payment();
    payment.setId(paymentId);
    PaymentExecution paymentExecute = new PaymentExecution();
    paymentExecute.setPayerId(payerId);
    return payment.execute(apiContext, paymentExecute);
  }

}
```

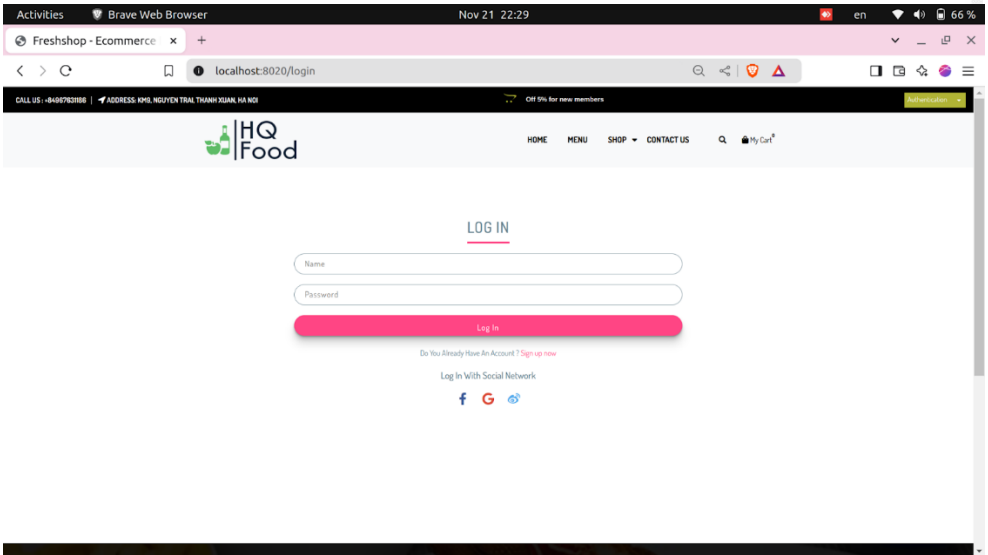
- Giao diện
  - Customer

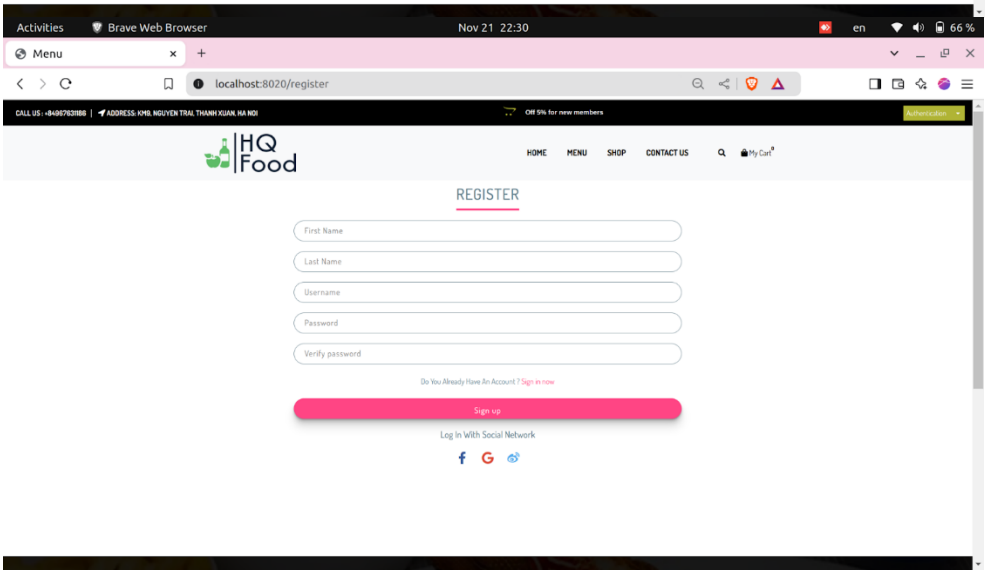


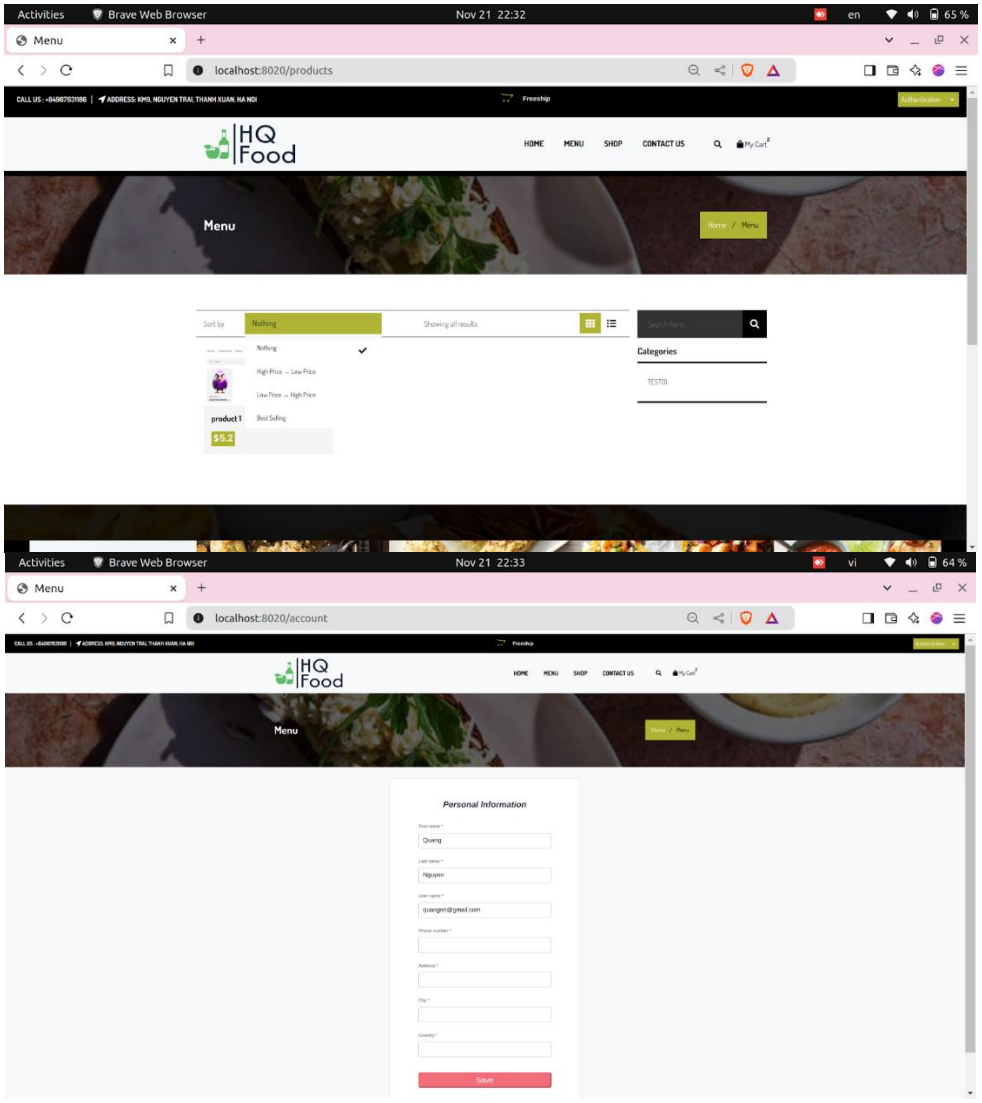




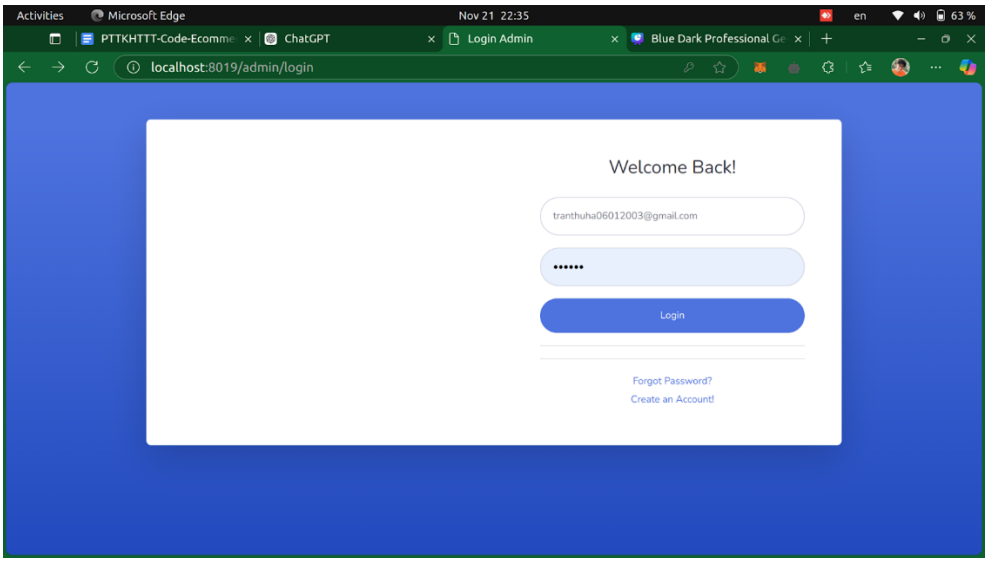

Activities
Brave Web Browser
Nov 21 22:21
en
72 %
Menu
localhost:8020/home
CALL US : +84987831188 | ADDRESS: KM9, NGUYEN TRAI, THANH XUAN, HA NOI
Freeship
Authentication
HQ Food
HOME
MENU
SHOP
CONTACT US
My Cart
Menu
Home / Menu
TEST
Have you a really good meal!
All
Best seller
product 1
$5.2

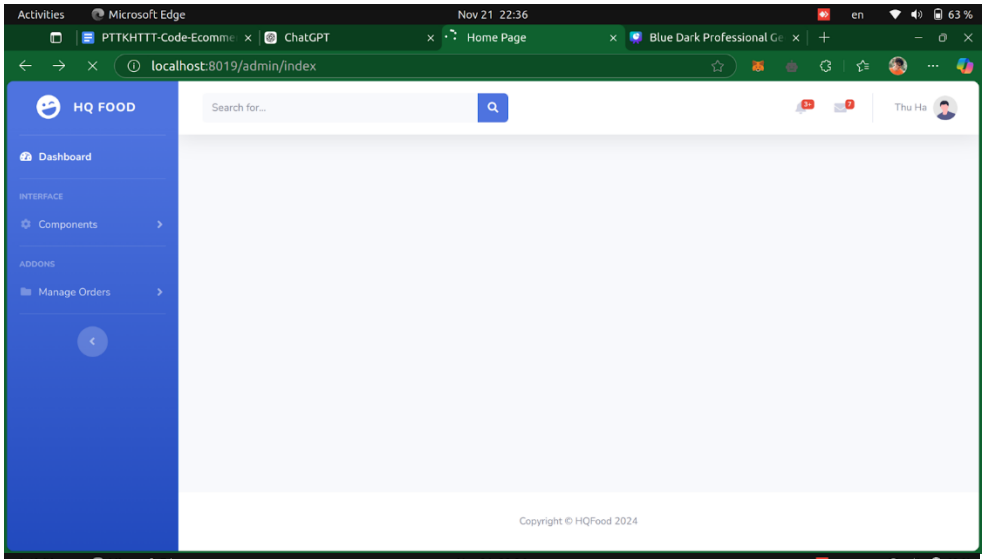

Activities
Brave Web Browser
Nov 21 22:29
en
66 %
Freshshop - Ecommerce
localhost:8020/login
CALL US : +84987831188 | ADDRESS: KM9, NGUYEN TRAI, THANH XUAN, HA NOI
Off 5% for new members
Authentication
HQ Food
HOME
MENU
SHOP
CONTACT US
My Cart
LOG IN
Name
Password
Log In
Do You Already Have An Account ? Sign up now
Log In With Social Network

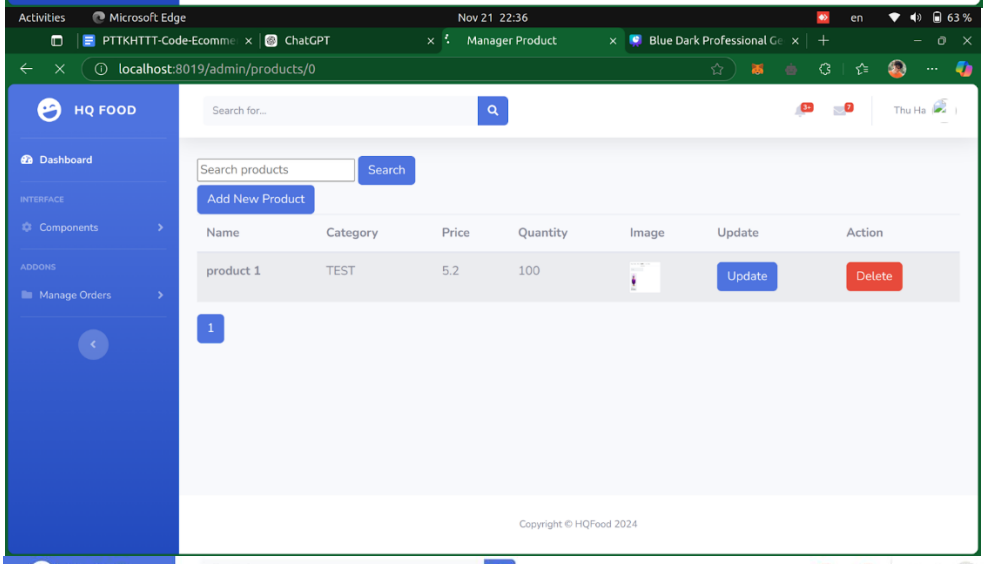

Activities
Brave Web Browser
Nov 21 22:30
en
66 %
Menu
localhost:8020/register
CALL US : +84987831188 | ADDRESS: KM9, NGUYEN TRAI, THANH XUAN, HA NOI
Off 5% for new members
Authentication
HQ Food
HOME
MENU
SHOP
CONTACT US
My Cart
REGISTER
First Name
Last Name
Username
Password
Verify password
Do You Already Have An Account ? Sign in now
Sign up
Log In With Social Network

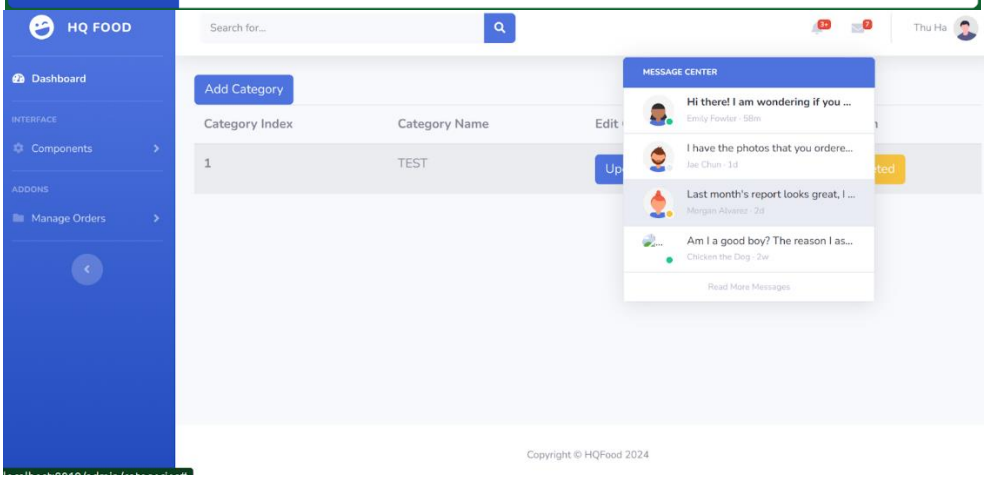

Activities
Brave Web Browser
Nov 21 22:32
Menu
localhost:8020/products
HQ Food
HOME
MENU
SHOP
CONTACT US
Menu
Categories
Activities
Brave Web Browser
Nov 21 22:33
Menu
localhost:8020/account
HQ Food
Menu
Personal Information
Save

- Admin


Activities
Microsoft Edge
Nov 21 22:35
PTTKHTTT-Code-Ecomme
ChatGPT
Login Admin
Blue Dark Professional
localhost:8019/admin/login
Welcome Back!
tranthuha06012003@gmail.com
Login
Forgot Password?
Create an Account!


Activities
Microsoft Edge
Nov 21 22:36
en
63 %
PTTKHTTT-Code-Ecomme
ChatGPT
Home Page
Blue Dark Professional Ge
localhost:8019/admin/index
HQ FOOD
Search for...
Thu Ha
Dashboard
INTERFACE
Components
ADDONS
Manage Orders
Copyright © HQFood 2024


Activities
Microsoft Edge
Nov 21 22:36
en
63 %
PTTKHTTT-Code-Ecomme
ChatGPT
Manager Product
Blue Dark Professional Ge
localhost:8019/admin/products/0
HQ FOOD
Search for...
Thu Ha
Dashboard
INTERFACE
Components
ADDONS
Manage Orders
Search products
Search
Add New Product
Name
Category
Price
Quantity
Image
Update
Action
product 1
TEST
5.2
100
Update
Delete
1
Copyright © HQFood 2024


HQ FOOD
Search for...
Thu Ha
Dashboard
INTERFACE
Components
ADDONS
Manage Orders
Add Category
Category Index
Category Name
Edit
1
TEST
MESSAGE CENTER
Hi there! I am wondering if you ...
I have the photos that you ordere...
Last month's report looks great, I ...
Am I a good boy? The reason I as...
Read More Messages
Copyright © HQFood 2024