PEOPLE'S COMMITTEE OF HO CHI MINH CITY

SAIGON UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

FINAL ASSIGNMENT REPORT

Course: Software Testing

Topic: **Development and Testing of a Web Application Using CI/CD Process and Agile Methodology**

Supervisor: **Dr. Đỗ Như Tài**

Group 7:

No.	Name	Student ID
1	**Trang Gia Huy**	**3122411068**
2	**Nguyễn Lê Quỳnh Hương**	**3122411078**

HO CHI MINH CITY, DECEMBER 2025

TABLE OF CONTENTS

TASK ASSIGNMENT TABLE

No.	Name	Student ID	Work Content
1	Trang Gia Huy	3122411068	- Prepare project report - Prepare project plan - System analysis and design - Prepare test plan (support) - Develop Backend and deploy the project
2	Nguyễn Lê Quỳnh Hương	3122411078	- Prepare project overview - Draw Use Case and Screen diagrams - Prepare test plan (main) - Develop Frontend and deploy the project

Chapter 1: Project Overview

1.1. Introduction:

The testing project titled “Electro E-commerce Website” is conducted to evaluate the system’s compliance with functional requirements and to identify defects arising during the development process. The testing activities focus on verifying data accuracy, operational stability, and the correctness of the website’s core business functionalities.

The testing process includes requirements analysis, test case design and execution, and the performance of unit testing, integration testing, system testing, and acceptance testing to ensure that the system operates in accordance with the specifications before deployment.

1.2. Business Context:

The Electro E-Commerce website comprises the following core business modules: User Authentication & Authorization, Product Catalog, Shopping Cart & Ordering & Payment, and Inventory Management. These modules simulate the fundamental business processes of a modern e-commerce system, meeting the needs of both customers and system administrators.

With the **Authentication & Acess control** functionality, customers can register a new account using an email address, password, and other personal information, and then log in to access the shopping features of the system. After being authenticated, users can view and update their personal profiles as well as log out when needed. System administrators can also log in to access the admin dashboard, where they can view the list of users, inspect detailed account information, and update user statuses (activate or deactivate accounts). The system’s authentication mechanism is based on Spring Security integrated with JWT (JSON Web Token) to ensure secure access control and proper role-based authorization between regular users and administrators.

With the **Product Catalog** functionality, customers can browse the list of products displayed in the store and apply filters by category, price range, brand, or search keywords to easily find desired items. On the product detail page, users can view the product name, description, price, images, availability status, and stock quantity. Administrators have full authority to manage products, product categories, and related information, including creating new products, updating existing ones, and removing discontinued or obsolete items. They can also classify products into categories to optimize display and search. The system supports product image uploads to ensure that items are always presented with intuitive and appealing visuals to customers.

With the **Shopping Cart, Checkout & Payment** functionality, customers can add products to the cart from either the product listing page or the product detail page. The cart displays the selected items, quantities, unit prices, and the subtotal. Whenever users change quantities or remove items, the cart and summary will be updated automatically. When customers proceed to checkout, the system allows placing orders using the Cash on Delivery (COD) payment method. After an order is placed, a new order record is created and stored in the system with the initial status of “Pending Confirmation”. Users can track their order statuses (Pending Confirmation, In Delivery, Completed, or Cancelled). Administrators can view all orders, inspect detailed order information, update order statuses during processing, or cancel orders with clearly recorded reasons for internal management purposes.

With the **Inventory Management** functionality, administrators can monitor the stock levels of each product, perform stock-in operations when receiving goods from suppliers, and stock-out operations when products are shipped to customers. The system allows administrators to view detailed inventory lists, including product codes, product names, current quantities, and in–out transaction histories. In addition, administrators can create purchase orders to replenish inventory when stock levels fall below a threshold, ensuring uninterrupted business operations. Administrators can also view and update stock-in/out slips and purchase orders to maintain clear and accurate records of inventory transactions.

Initial Data (One-off Tasks): When the system is first initialized, a set of sample data is preloaded to support testing and functional demonstrations, including: one default Admin account for accessing the administration panel; two Client accounts for testing the shopping workflow; a sample product dataset with multiple categories and images; and several sample orders in different statuses (pending confirmation, in delivery, completed, cancelled). These initial datasets enable the system to be tested and showcased with full functionality immediately after startup.

1.2.1. Theoretical Background

1.2.1.1. What is an Actor?

An actor is an external entity that interacts with a system. An actor can be a human user, another system, or a device.

In a business context, an actor refers to the role that a person or an entity assumes while interacting with business processes. The following types of business users may be considered business actors: customers, suppliers, partners, colleagues involved in business processes that are not explicitly modeled, and so on. Therefore, an actor often corresponds to a system user. However, in certain situations, an information system itself can also play the role of an actor.

For example, a bank may handle most online transactions through its information system; in such a case, the use cases of the system interact with the bank, and the bank is regarded as an actor. This means that the actor, in this scenario, is an information system rather than a human user.

1.2.1.2. How to Identify Business-Level Actors

To identify actors at the business level, the focus is on human or organizational roles in business operations, rather than on specific software components. The identification process includes the following steps:

Identify stakeholders: Who are the parties involved in the main business processes (e.g., customers, staff, managers)?

Identify interaction goals: What do they aim to achieve when interacting with the system? (e.g., placing orders, managing inventory, confirming orders, etc.)

Differentiate roles: A single person may perform multiple roles → each role should be represented as a separate actor.

Select only external actors: Actors are not part of the software system itself; they are external entities that interact with it.

1.2.2. Actors (Business Level)

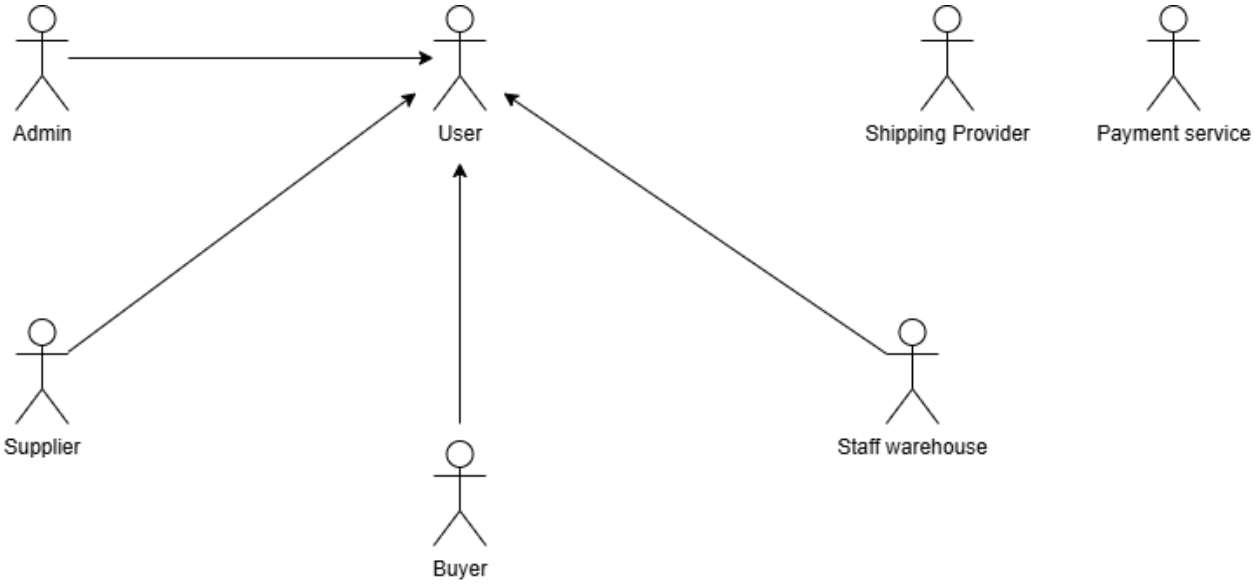
Actor Name	Role Description	Business Goal

Buyer (Customer)	A person who purchases goods via the website or store and represents the source of shopping demand.	Select products, make payments, and receive goods.
Administrator (Admin)	The person responsible for managing and operating the system's business activities.	Manage products, users, orders, and inventory, and handle operational issues.
Supplier	The entity that provides products or goods to the system.	Supply products in the correct quantity, quality, and within the required timeframe for stock replenishment.
Warehouse Staff	The person responsible for inventory checking and stock in/out operations in the warehouse.	Update inventory status; verify product quantity and quality during stock-in and stock-out processes.
Payment System (Payment Gateway / COD)	A module or API that processes COD or VNPAY payments.	Record and validate customers' payment transactions.
Shipping Provider	A third-party entity that supports goods delivery.	Receive delivery requests and perform shipping on behalf of the system.

1.2.3. Actors (System Level)

Tên Actor	**Role Description**	**System Goal**
Buyer (Customer)	A registered user in the system who performs shopping activities through the web interface.	Interact with the system via APIs to log in, browse products, add items to the cart, and place orders.
Administrator (Admin)	A privileged account with full authority to manage system data.	Manage products, categories, orders, users, and inventory.

1.2.4. Business Actor Diagram



Customer: System user who registers, logs in, manages the cart, places orders, makes payments, and tracks order status.

System Administrator (SysAdmin): Highest-level administrator with full access to manage users, products, and inventory.

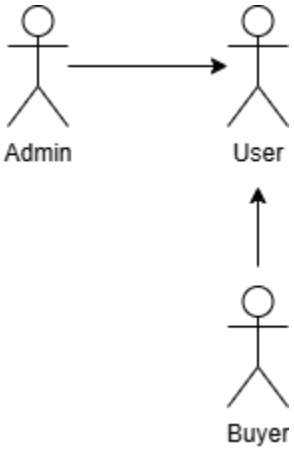
Warehouse Staff: Manages warehouse goods, processes purchase orders, and creates stock in/out vouchers.

Supplier: Provides products to store

Shipping Provider: Delivers goods from the warehouse to customers and updates delivery status in the system.

Payment Service: Handles payment transactions, confirms payment methods, and notifies the system of results.

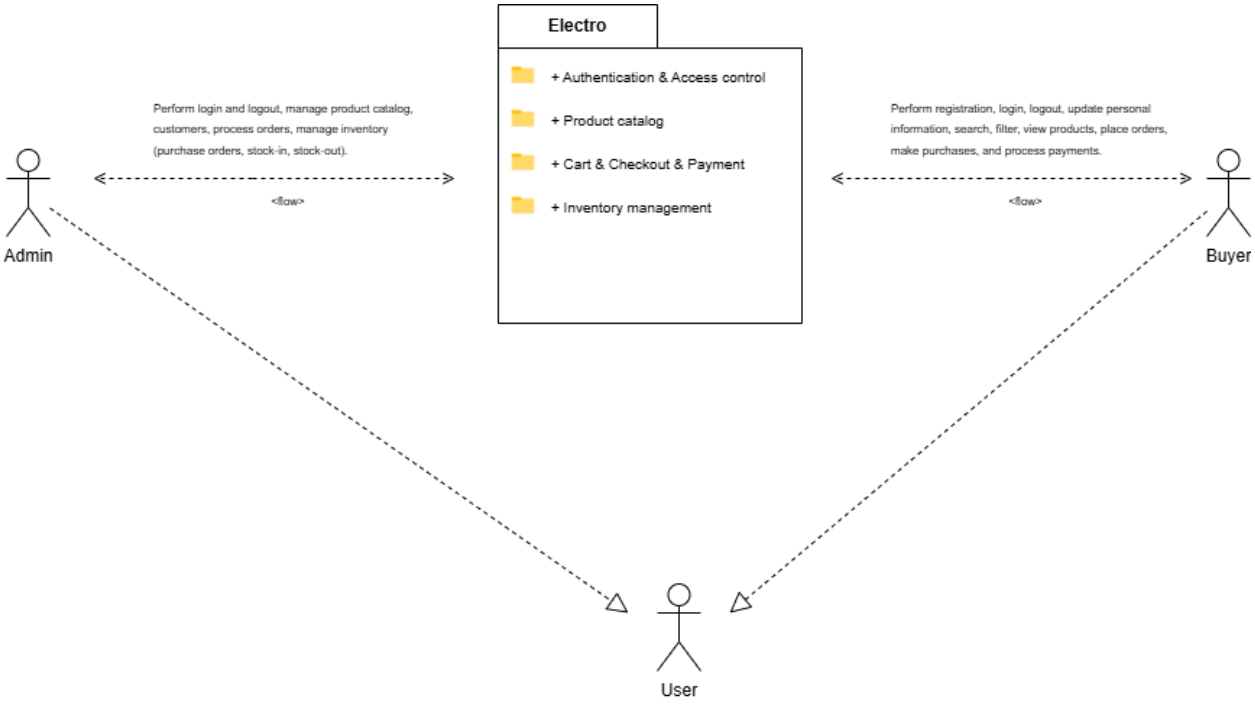
1.2.5. System Actor Diagram



Customer: System user who registers, logs in, manages the cart, places orders, makes payments, and tracks order status.

System Administrator (SysAdmin): Highest-level administrator with full access to manage users, products, and inventory.

1.2.6. System Context Diagram



Admin performs login and logout, manage product catalog, customers, process orders, manage inventory (purchase orders, stock-in, stock-out).

Buyer performs registration, login, logout, update personal information, search, filter, view products, place orders, make purchases, and process payments

1.3. Business Processes

In an enterprise, any activity that accepts inputs and transforms them into outputs can be considered a process.

A process is a documented set of repeatable steps or tasks performed to achieve a specific goal while minimizing errors.

A business process is a collection of tasks executed in a defined sequence to transform inputs into desired outputs.

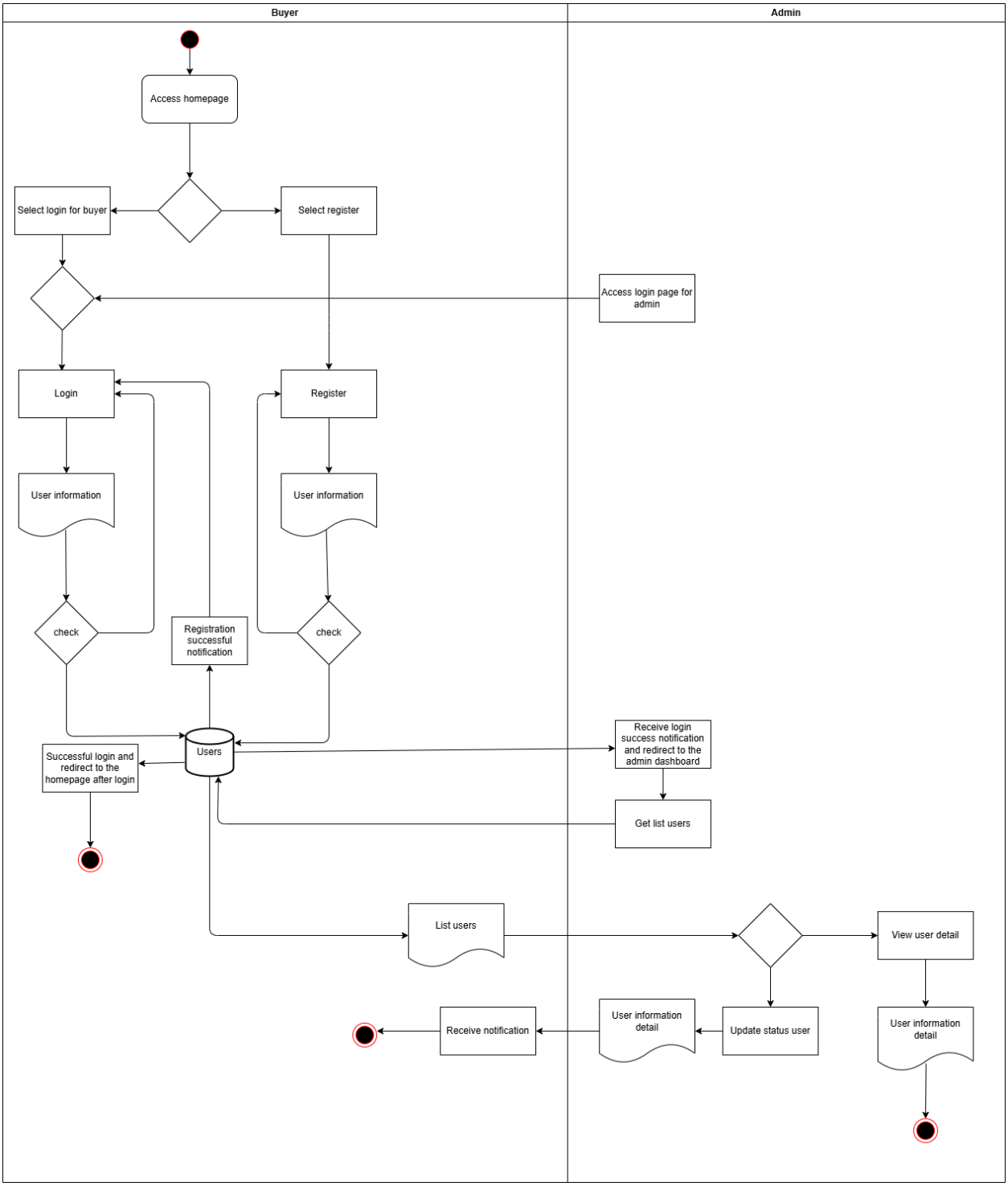
BPM (Business Process Management) refers to the discipline of managing and continuously improving an organization's processes so that outcomes are repeatable and errors can be avoided.

a. Authentication and Access control Process

The authentication and authorization process is carried out as follows:

When customers access the website for shopping, they are required to register for an account if they do not already have one, and then log in to the system. After successfully logging in, customers can view their personal information and proceed with shopping activities. Administrators are able to manage the user list by viewing all users, inspecting detailed user information, and updating user account statuses.

Participating entities/departments: Customer, Administrator.

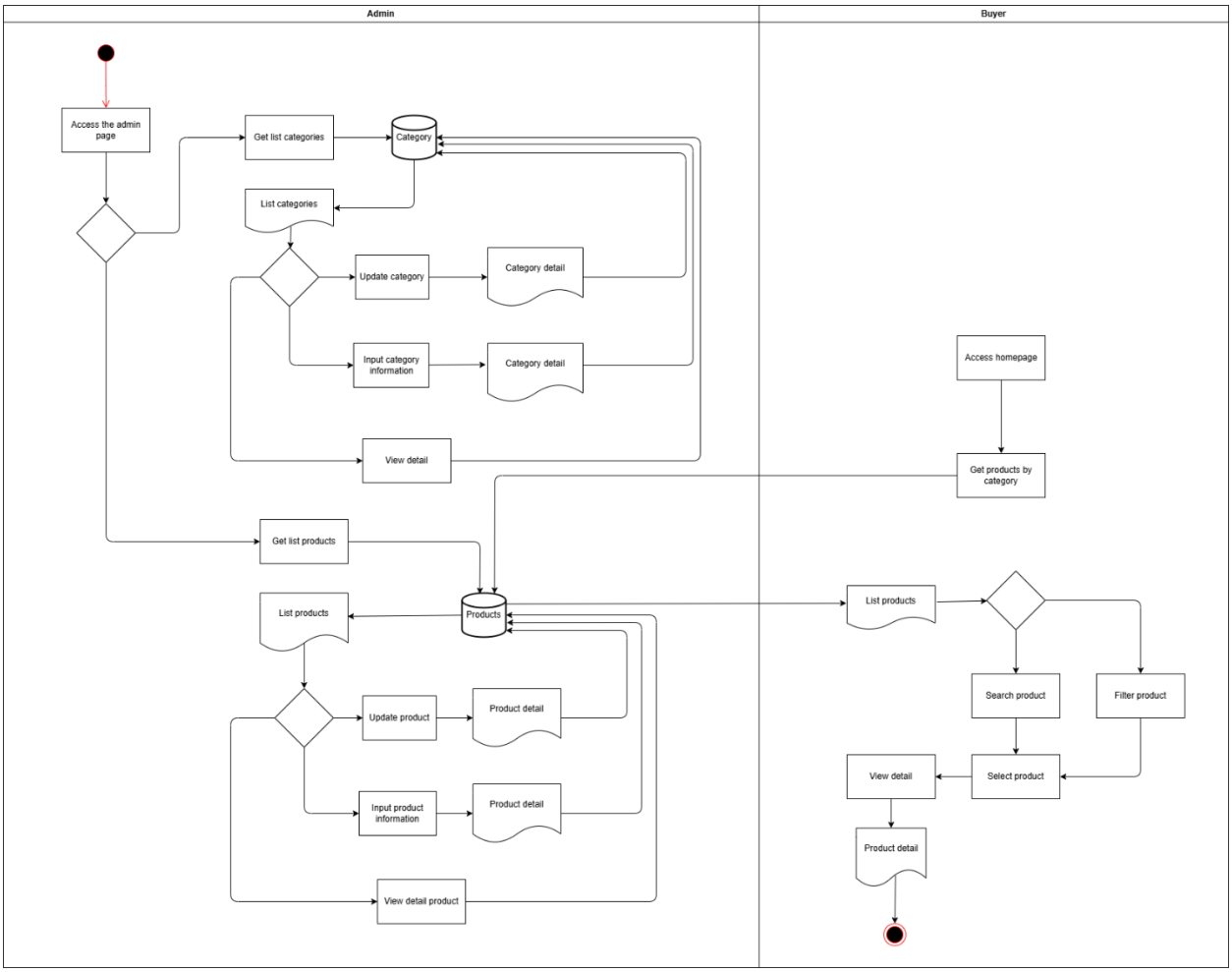


b. Product Catalog Management Process

The product catalog management process is carried out as follows:

Administrators can manage product catalogs by creating, updating, and deleting products and product categories. Customers can browse the product list and view product details; in addition, they can search and apply filters based on various criteria to quickly find desired products.

Participating entities/departments: Administrator, Customer.

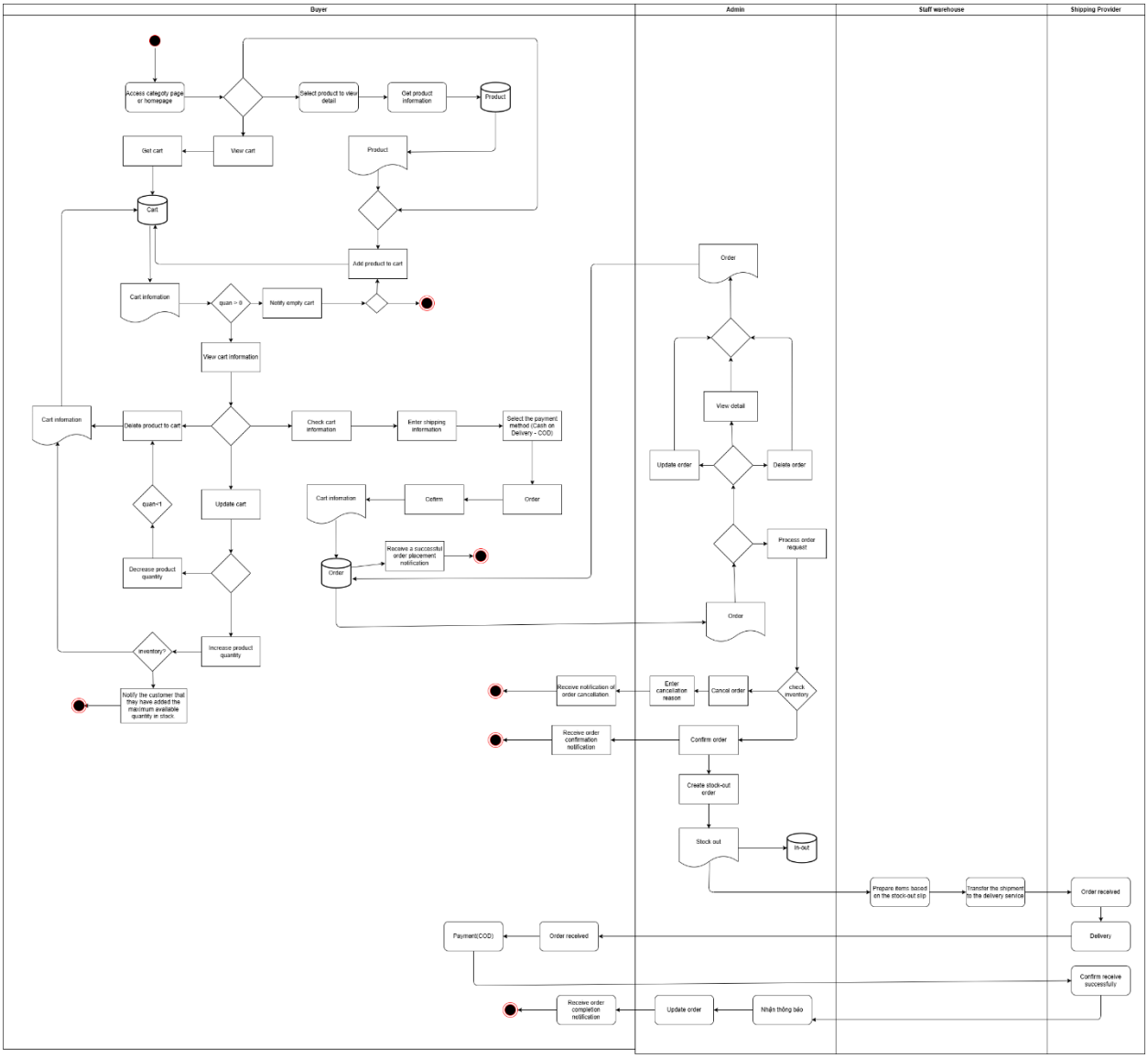


c. Shopping Cart, Checkout, and Payment Process

The shopping cart, ordering, and payment process is carried out as follows:

After logging into the system, customers can begin the shopping process. They can add products to the shopping cart from the home page, product category pages, or product detail pages. Once items are in the cart, customers may adjust quantities or remove products if they no longer wish to purchase them. After selecting the desired items, customers proceed to review order information, including product costs, shipping details, and the selected payment method. Once the information is confirmed, they place the order. After an order is successfully placed, administrators are notified of the new order. The administrator then checks product availability in the warehouse, confirms the order, and creates a stock-out note for warehouse staff to pack the products and hand over the order to the shipping provider. Once the goods are successfully delivered to the customer, the administrator receives a delivery confirmation and updates the order status accordingly for the customer.

Participating entities/departments: Administrator, Warehouse Staff, Shipping Provider.

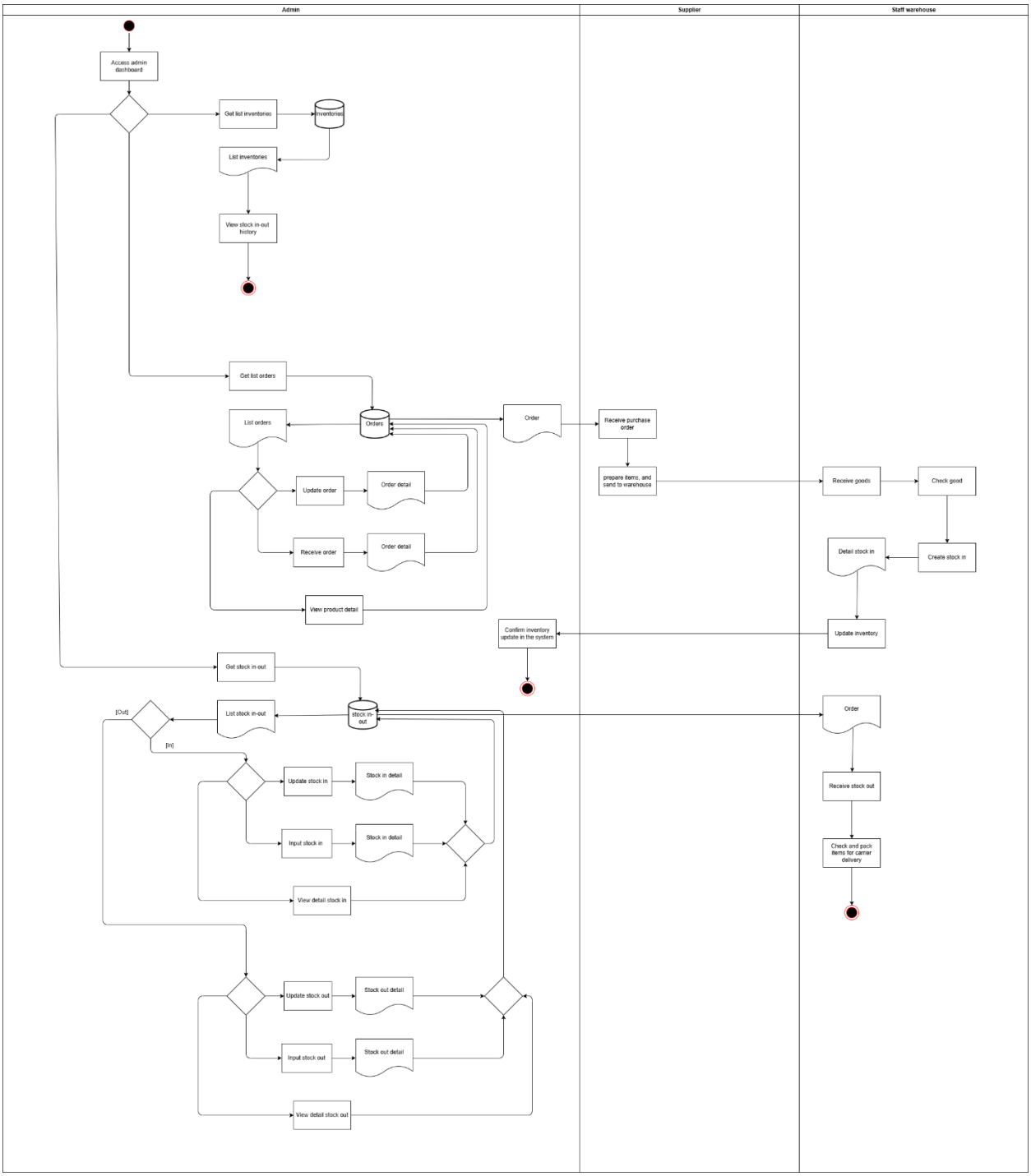


d. Inventory Management Process

The inventory management process is carried out as follows:

Administrators monitor inventory levels and create purchase orders when stock in the store falls below the required thresholds. They generate stock-in receipts when goods are received from suppliers, and create stock-out slips when products are sold. The stock-out slips are then handed over to warehouse staff to pack the goods and deliver them to the shipping provider.

Participating entities/departments: Administrators, Suppliers, Warehouse Staff.


Admin
Supplier
Staff warehouse
Access admin dashboard
Get list inventories
Inventories
List inventories
View stock in-out history
Get list orders
List orders
Orders
Order
Receive purchase order
prepare items, and send to warehouse
Receive goods
Check good
Create stock in
Detail stock in
Update inventory
Confirm inventory update in the system
Update order
Order detail
Receive order
Order detail
View product detail
Get stock in-out
List stock in-out
stock in-out
[Out]
[In]
Order
Receive stock out
Check and pack items for carrier delivery
Update stock in
Stock in detail
Input stock in
Stock in detail
View detail stock in
Update stock out
Stock out detail
Input stock out
Stock out detail
View detail stock out

1.4. Requirements Analysis

1.4.1. Functional Requirements

- Functional requirements are detailed descriptions of the functions or behaviors that the system must perform in order to achieve business objectives.

- Identification approach:

Analyze use cases or user stories to identify the primary actions performed by users.

Conduct interviews or surveys with users to understand business goals and expectations.

Derive required system functions from business process descriptions.

Identify who (actors) interacts with the system and for what purposes.

#	Function Name	Requirement Description
BR1	**Authentication & Acess control**	
BR1.1	User Registration	**Buyers** can register a new account using an email address, password, and personal information (full name, address, phone number) to shop online.
BR1.2	Login	**Buyers and administrators** can log in with valid credentials to access authorized functions.
BR1.3	Update Profile Information	**Buyers and administrators** can update their personal information after logging in.
BR1.4	View User List	**Administrators** can view the list of users to monitor active accounts in the system.
BR1.5	View User Details	**Administrators** can view detailed information of each user for management and handling purposes.
BR1.6	Update Account Status	**Administrators** can update account status (activate/deactivate) to ensure security and proper access control.
BR1.7	Logout	**Buyers and administrators** can log out to secure their accounts.
BR2	**Product Catalog**	

BR2.1	View Latest Products	**Buyers** can view the list of the latest products.
BR2.2	Filter Products	**Buyers** can filter products by category, price range, and brand to find items more quickly.
BR2.3	Search Products by Keyword	**Buyers** can search for products using keywords.
BR2.4	View Product Details	**Buyers** can view detailed product information (name, description, price, images, availability).
BR2.5	Check Stock Quantity	**Buyers** can view the available stock quantity for each product.
BR2.6	Add New Product	**Administrators** can add new products to the system.
BR2.7.	View Product Details	**Administrators** can view detailed information of products in the system
BR2.7	Update Product Information	**Administrators** can update product information.
BR2.8	Delete Product	**Administrators** can delete discontinued or obsolete products.
BR2.9	Manage Product Categories	**Administrators can manage product categories (view, update, delete).**
BR3	**Shopping Cart – Checkout – Payment**	
BR3.1	Add to Cart	**Buyers** can add products to the shopping cart.
BR3.2	View Cart	**Buyers** can view the list of products in the shopping cart.
BR3.3	Update Cart	**Buyers** can change quantities or remove products from the cart.
BR3.4	View Order Summary	Buyers can view subtotal, shipping fee, and total order amount.

BR3.5	Cash on Delivery (COD) Payment	**Buyers** can complete purchases using the COD payment method.
BR3.6	Track Order Status	**Buyers** can track order status (Pending, In Delivery, Completed, Cancelled).
BR3.7	View Order List	**Administrators** can view the list of all customer orders.
BR3.8	View Order Details	**Administrators** can view detailed information of each order.
BR3.9	Update Order Status	**Administrators** can update order status (Preparing, Shipping, Completed).
BR3.10	Cancel Order	**Administrators** can cancel orders and record cancellation reasons.
BR4	**Inventory Management**	
BR4.1	View Inventory List	**Administrators** can view the detailed inventory list.
BR4.2	Create Stock-in Receipt	**Administrators** can create stock-in receipts when goods are purchased from suppliers.
BR4.3	View Stock-in Details	**Administrators** can view details of stock-in receipts.
BR4.4	Update Stock-in Receipt	**Administrators** can update stock-in receipts.
BR4.5	Create Stock-out Slip	**Administrators** can create stock-out slips when orders are shipped to customers to accurately record inventory changes.
BR4.6	View Stock-out Details	**Administrators** can view details of stock-out slips.
BR4.7	Update Stock-out Slip	**Administrators** can update stock-out slips.
BR4.8	View In–Out History	**Administrators** can view inventory in–out history.

BR4.9	Create Purchase Order	**Administrators** can create new purchase orders when inventory is running low.
BR4.10	View Purchase Order Details	**Administrators** can view detailed purchase order information.
BR4.11	Update Purchase Order	**Administrators can update purchase orders.**

1.4.2. Non-functional Requirements

- Non-functional requirements include all requirements that are not functional requirements. They specify criteria for evaluating how the system performs rather than what the system does.

- Identification approach:

Identify technical, design, and security constraints that the system must comply with.

Consider usability, performance, compatibility, security, scalability, and other quality attributes.

BR5	**Usability**	
BR5.1		The system shall support user session management to maintain login state throughout usage. The system shall provide intuitive navigation, allowing users to update the shopping cart and perform other actions in real time.
BR6	**Compatibility**	
BR6.1		The system shall operate stably and correctly on common web browsers, including: Google Chrome (latest version, Windows 11) Mozilla Firefox (latest version, Windows 11) Microsoft Edge (latest version, Windows 11).
BR7	**Interfaces**	
BR7.1		The user interface shall display appropriate information and functions according to each user role (Buyer, Administrator). The interface shall be simple and easy to use.

1.6. User Stories:

1.6.1. User story

- A user story is a brief description of a requirement or system feature written from the perspective of the end user.

- How to identify user stories:

- Identify user roles: Who will use the system? (e.g., buyer, administrator)
- Identify their goals: What do they want to achieve when using the system?
- Convert goals into user stories: Write using the template "As a …, I want …, so that …".
- Group user stories by functional modules: Cluster related stories (Authentication, Products, Cart, Orders, Inventory, etc.).

- **User stories for the Website ShopEase E-Commerce:**

Authentication & Access control

- As a buyer, I want to register an account so that I can shop online.
- As a buyer, I want to log in with my credentials so that I can access my personal account.
- As a buyer, I want to update my personal information so that my data is always accurate and up to date.
- As a buyer, I want to log out of the system so that my account remains secure after shopping.
- As an administrator, I want to log in with an admin account so that I can access the system management portal.
- As an administrator, I want to view the list of users so that I can monitor active accounts in the system.
- As an administrator, I want to view detailed information of each user so that I can manage and handle issues when necessary.
- As an administrator, I want to update user account statuses (activate/deactivate) so that security and proper access control are ensured.
- As an administrator, I want to log out of the system so that my admin account remains secure.

Product Catalog

- As a buyer, I want to view the list of the latest products so that I can easily find items to purchase.
- As a buyer, I want to filter products by category, price range, and brand so that I can find products faster.
- As a buyer, I want to search for products by keywords so that I can quickly locate desired items.

- As a buyer, I want to view detailed product information (name, description, price, images, availability) so that I can decide whether to purchase.
- As a buyer, I want to see the stock quantity of each product so that I can ensure the product is in stock.
- As an administrator, I want to view product details so that I clearly understand product information.
- As an administrator, I want to add new products to the system so that I can expand the product catalog.
- As an administrator, I want to update product information so that displayed data is always accurate.
- As an administrator, I want to delete discontinued or obsolete products so that the catalog remains clean and valid.
- As an administrator, I want to manage product categories (view, update, delete) so that products are organized logically and easy for buyers to find.

Shopping Cart, Checkout & Payment

- As a buyer, I want to add products to my cart so that I can prepare for checkout later.
- As a buyer, I want to view the list of items in my cart so that I can review my selected products.
- As a buyer, I want to change quantities or remove items from my cart so that I can adjust my order before checkout.
- As a buyer, I want to view my shipping address, delivery method, subtotal, shipping fee, and total order amount so that I know the total cost before purchasing.
- As a buyer, I want to pay using the Cash on Delivery (COD) method so that I can complete my purchase.
- As a buyer, I want to track my order status so that I know the progress of my order.
- As an administrator, I want to view all customer orders so that I can manage the sales process effectively.
- As an administrator, I want to view order details so that I can verify order information and handle issues if needed.
- As an administrator, I want to update order statuses so that they reflect the correct processing progress.
- As an administrator, I want to cancel orders and record cancellation reasons so that they can be tracked and managed later.

Inventory Management

- As an administrator, I want to view the detailed inventory list so that I know current stock levels.

- As an administrator, I want to create stock-in receipts when purchasing from suppliers so that missing inventory can be replenished.
- As an administrator, I want to view stock-in receipt details so that I can control and verify stock-in information.
- As an administrator, I want to update stock-in receipt information so that stock-in records are accurate.
- As an administrator, I want to create stock-out slips when orders are shipped to customers so that inventory changes are recorded accurately.
- As an administrator, I want to view stock-out slip details so that I can control and verify stock-out information.
- As an administrator, I want to update stock-out slip information so that stock-out records are accurate.
- As an administrator, I want to view inventory in–out history so that I can track previous warehouse transactions.
- As an administrator, I want to create new purchase orders when stock is running low so that business operations are not interrupted.
- As an administrator, I want to view purchase order details so that I can control purchase order information.
- As an administrator, I want to update purchase order information so that purchase order records are accurate.

1.6.2. Use case:

- A use case is a detailed description of how users interact with a system to achieve a specific goal.

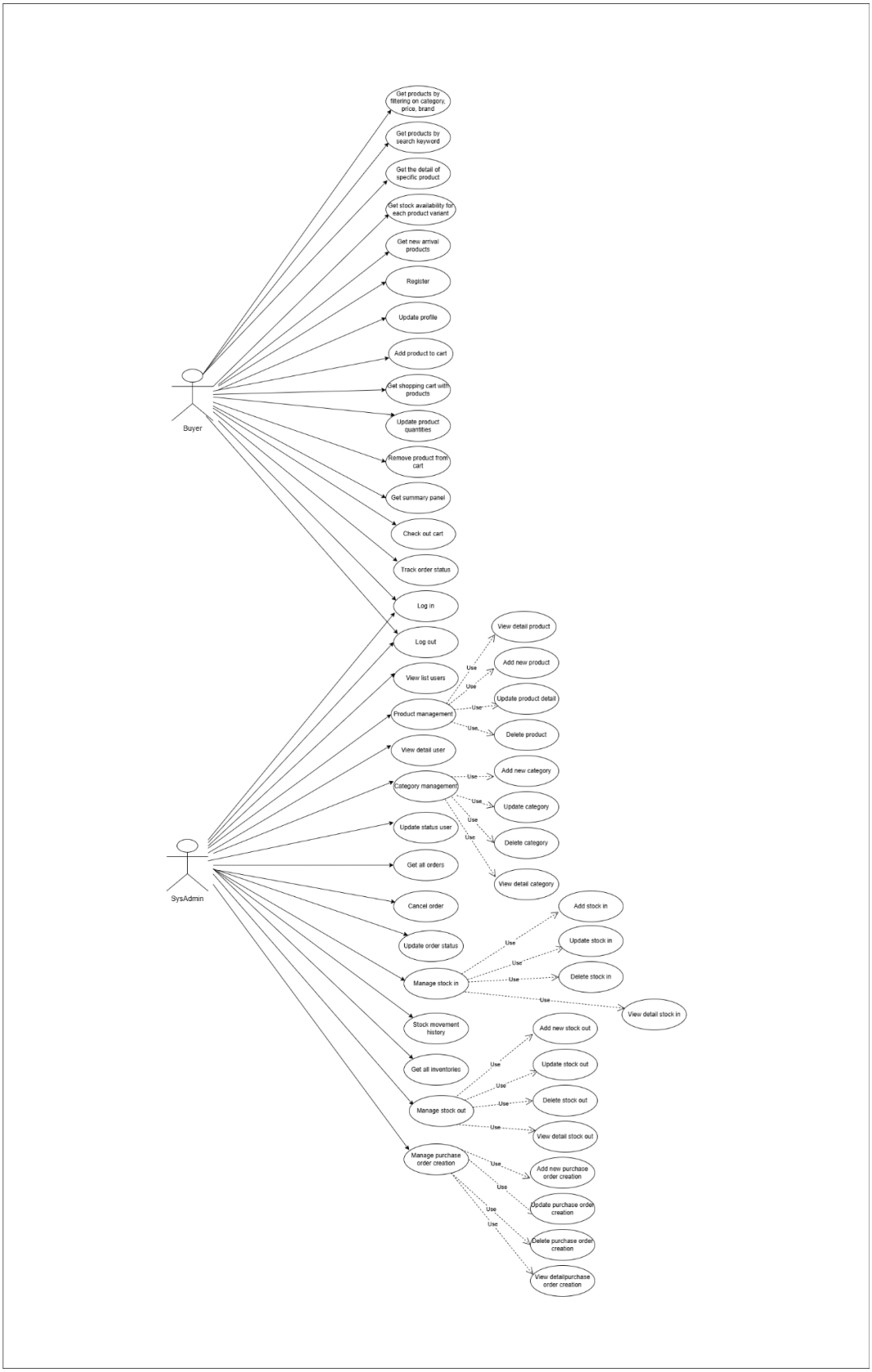
- How to identify use cases:

Identify actors: people or external systems that interact with the system.

Identify actor goals: what each actor wants to achieve.

Define the system scope.

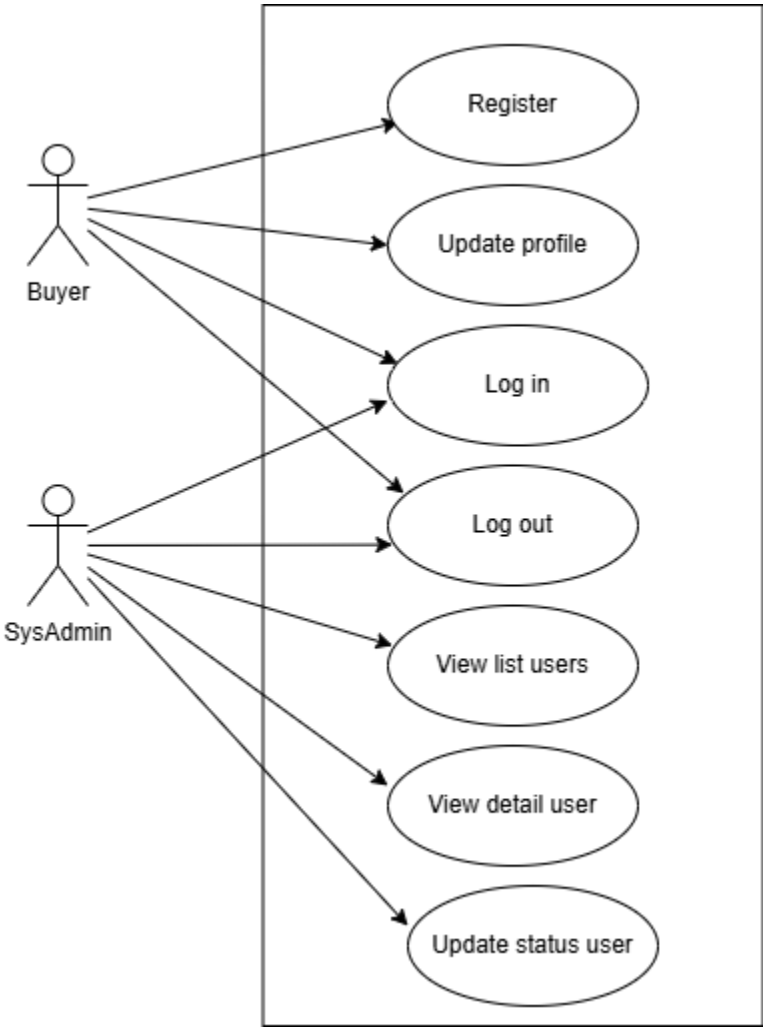
Use Case for the Website ShopEase E-Commerce:



Customers perform the following functions: authentication and authorization, product catalog, cart & order placement & payment.

System administrators perform the following functions: authentication and authorization, product catalog management, inventory management

Authentication & Access Control



UC1: Authentication & Access Control	
Description	Allows buyers to register, log in, log out, and update personal profiles, and allows administrators to log in, log out, and manage users.
Pre-condition	Actors: Buyer, Administrator. Conditions: The buyer does not yet have an account when registering. The administrator account has been pre-created. Valid username and password are required for login.

Trigger	Buyer or administrator selects “Login”. Buyer selects “Register” or “Update Profile” from the user interface. Buyer or administrator selects “Logout” to end the session. Administrator accesses the user management page to manage users.

Basic flow

The use case starts when an actor accesses the website.

The system displays the corresponding forms and available functions, including:

Register: allows buyers to create an account for shopping.

Login: allows actors to log in to the system.

Update Profile: allows buyers to edit their own personal information.

User Management: allows administrators to view the user list and update user account statuses.

a. Registration

The registration flow starts when the buyer selects “Register.”

The system displays a form requesting the following information:

- Username
- Password
- Email
- Phone number
- Gender
- Province/City
- District
- Ward
- Address

The buyer enters the information.

Selects Register.

The system validates the input data.

(Alternative: if validation fails)

The system sends an OTP to the buyer’s email address.

The system displays a form for OTP entry.

The buyer enters the OTP.

The system validates the OTP.

(Alternative: if validation fails.)

b. Login

The login flow starts when the buyer or administrator selects “Login.”

The system requests:

- Username

- Password

The system validates the credentials.

(Alternative: if validation fails.)

The system issues an authentication token and redirects the user:

- Buyer → Shopping page.

- Administrator → Admin dashboard.

c. Update profile

The update profile flow starts when a logged-in buyer selects “Profile.”

The system displays the current information.

The user edits the following fields:

- Phone number

- Gender

- Province/City

- District

- Ward

- Address

The system validates the updated information.

(Alternative: if validation fails.)

The system updates the database and displays a success message.

d. User Management

The user management flow starts when the administrator successfully logs in to the admin dashboard.

The system displays the user list.

The administrator may choose:

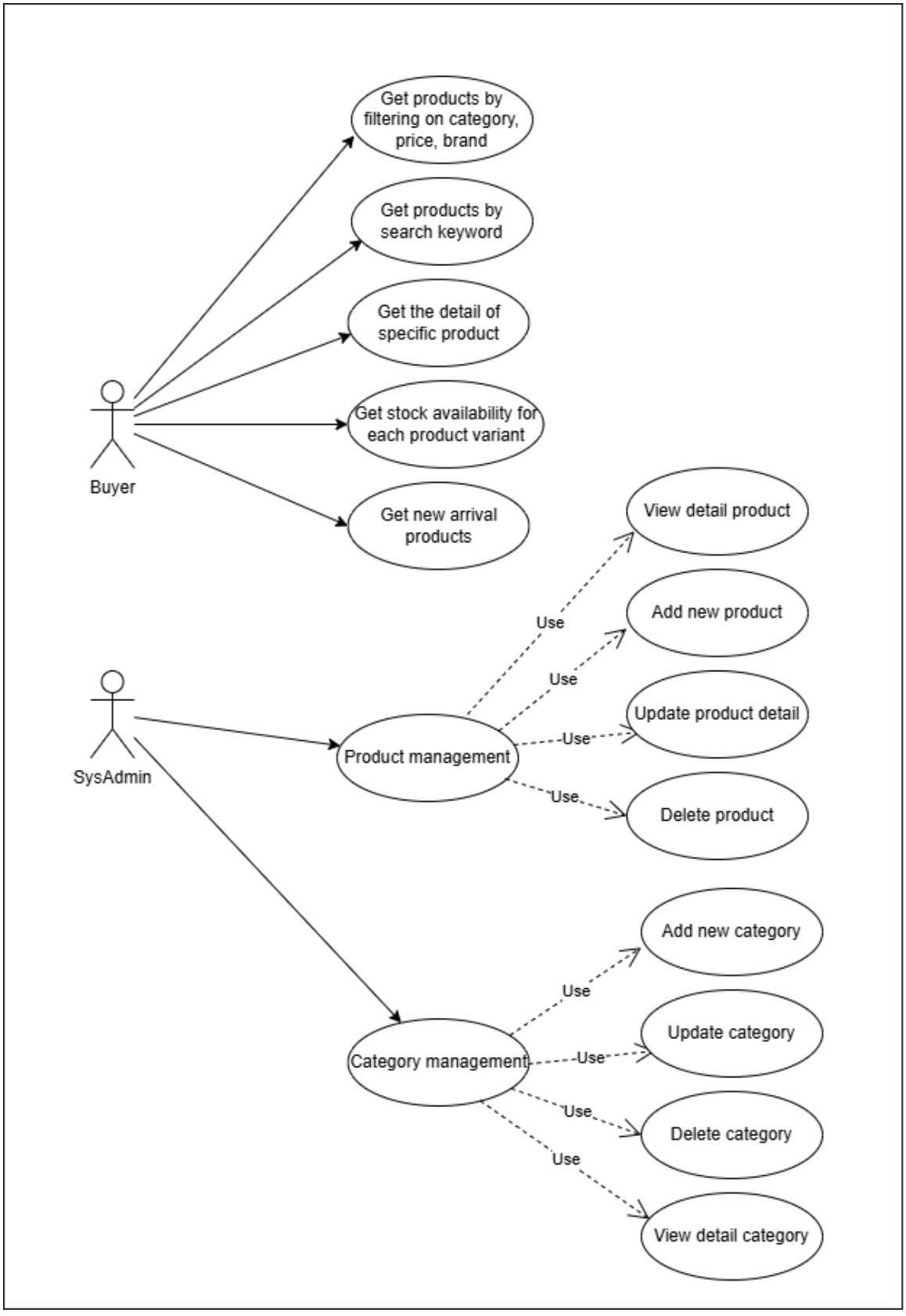
- View user details:

The view user details flow starts when the administrator selects “View”

The system displays detailed user information.

- Update account status: The update account status flow starts when the administrator selects “Update” The administrator selects “Activated” or “Deactivated” to change the user’s account status. The administrator selects Update. The system updates the data and notifies that the update is successful. **e. Logout** The logout flow starts when the user selects “Logout.” The system removes the refresh token and terminates the session. The system displays a logout success message.
Alternative Flow
Invalid validation: The system displays an error message and allows the user to re-enter the information.

Product Catalog



UC2: Product catalog	
Description	Allows buyers to browse, search, filter, and view product details, and allows administrators to manage products and product categories.
Pre-condition	Actors: Buyer, Administrator. Condition: The actor has already logged in to the system.
Trigger	Buyer selects “Product Catalog”, “Search”, “Filter”, or “View Product Details.” Administrator selects “Manage Products” or “Manage Categories” from the admin dashboard.

Basic flow

The use case starts when the actor accesses the product catalog page.

The system displays the list of available products and provides the following functions: Search & Filter Products: allows buyers to search by keyword and filter by category, price, or brand.

View Product Details: allows buyers to view detailed information of a selected product.

Manage Products / Categories: allows administrators to view, add, edit, and delete products and categories.

a. Search & Filter Products

The flow starts when the buyer enters a search keyword or selects filters by category, price range, or brand.

The system queries the database and displays the matching product list.

b. View Product Details

The flow starts when the buyer selects a product.

The system displays detailed information, including:

- Product name
- Price
- Description
- Images
- Quantity
- Category
- Brand
- Version

c. View Latest Products

The flow starts when the buyer accesses the home page.

The system displays the list of the latest products available in the store.

d. Product Management

The flow starts when the administrator selects “Manage Products” on the admin dashboard.

The system displays the current product list with available actions:

- Add new product:

The flow starts when the administrator selects “Add New.”

The system displays a form to enter product information:

- Product name
- SKU
- Slug
- Short description
- Full description
- Image
- Specifications
- Attributes
- Versions
- Additional information

The administrator submits the form.

The system validates the input.

(Alternative: if validation fails.)

The system saves the product to the database and displays a success message

- View Product Details

The flow starts when the administrator selects “View.”

The system displays a modal with product details, including:

- ID
- Created date
- Updated date
- Product name
- SKU
- Slug
- Short description
- Full description

- Image
- Specifications
- Attributes
- Weight
- Warranty name

- Edit Product:

The flow starts when the administrator selects “Update.”

The system displays the update form.

The administrator modifies the required fields and submits.

The system validates the input.

(Alternative: if validation fails.)

The system updates the product in the database and displays a success message.

- Delete Product:

The flow starts when the administrator selects “Delete.”

The system displays a confirmation dialog.

(Alternative: Cancel deletion.)

If confirmed, the system deletes the product and displays a success message.

e. Category Management

The flow starts when the administrator selects “Manage Categories” from the admin dashboard.

The system displays the list of existing categories with available actions:

- Add New Category:

The flow starts when the administrator selects “Add New.”

The system displays a form to enter category information:

- Category name
- Slug
- Description
- Thumbnail iamge
- Parent category
- Status

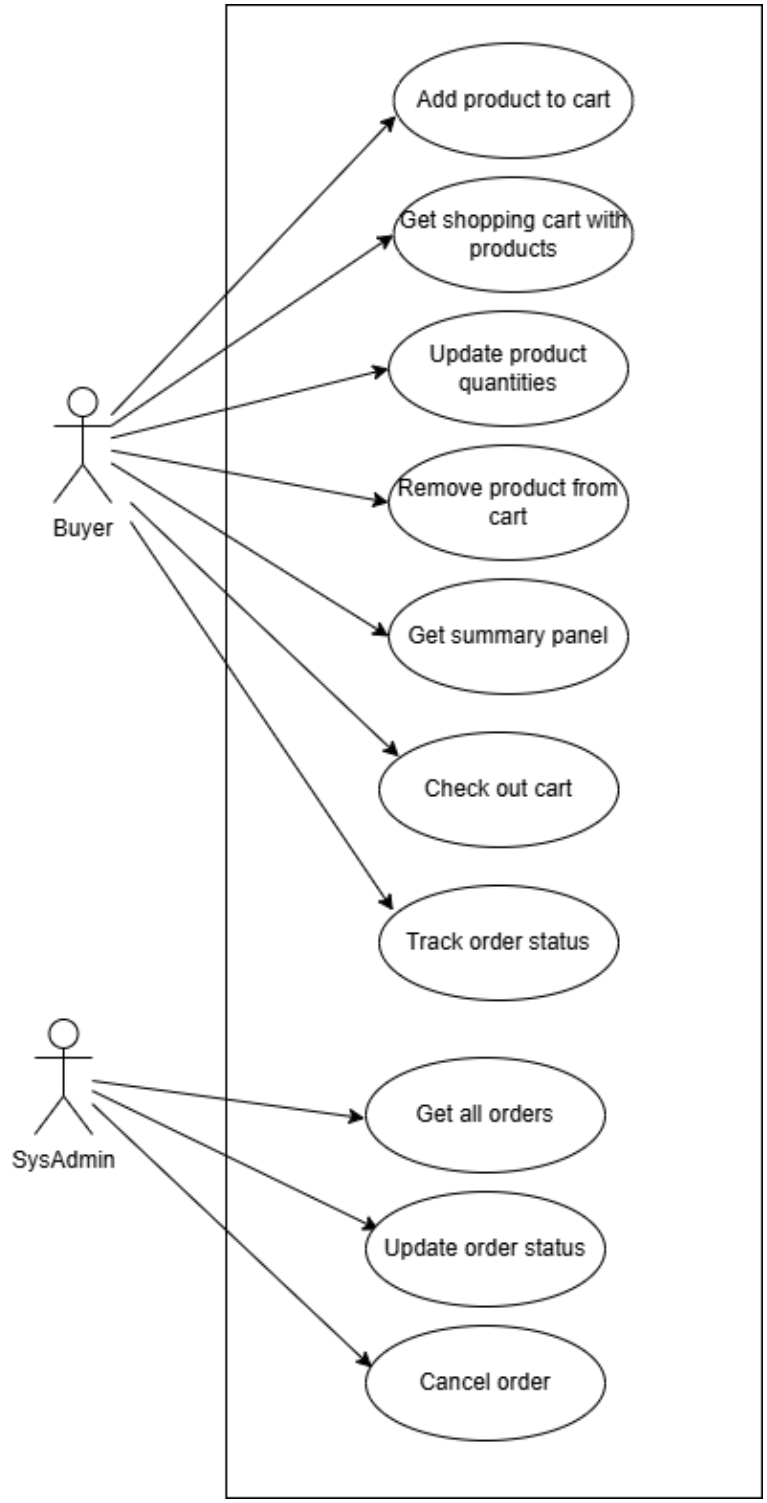
The administrator submits the form.

The system validates the input. (Alternative: if validation fails.)

The system saves the category to the database and displays a success message.

- View Category Details: The flow starts when the administrator selects “View.” The system displays a modal with category details: - ID - Created date - Updated date - Category name - Slug - Description - Thumbnail iamge - Parent category - Status **- Edit Category:** The flow starts when the administrator selects “Update.” The system displays the update form. The administrator modifies the required fields and submits. The system validates the input. (Alternative: if validation fails.) The system updates the category in the database and displays a success message. **- Delete Category:** The flow starts when the administrator selects “Delete.” The system displays a confirmation dialog. (Alternative: Cancel deletion.) If confirmed, the system deletes the category and displays a success message.
Alternative Flows
Invalid validation: The system displays error messages and allows the actor to re-enter information. Cancel deletion: The system closes the confirmation dialog and does not delete the product or category.

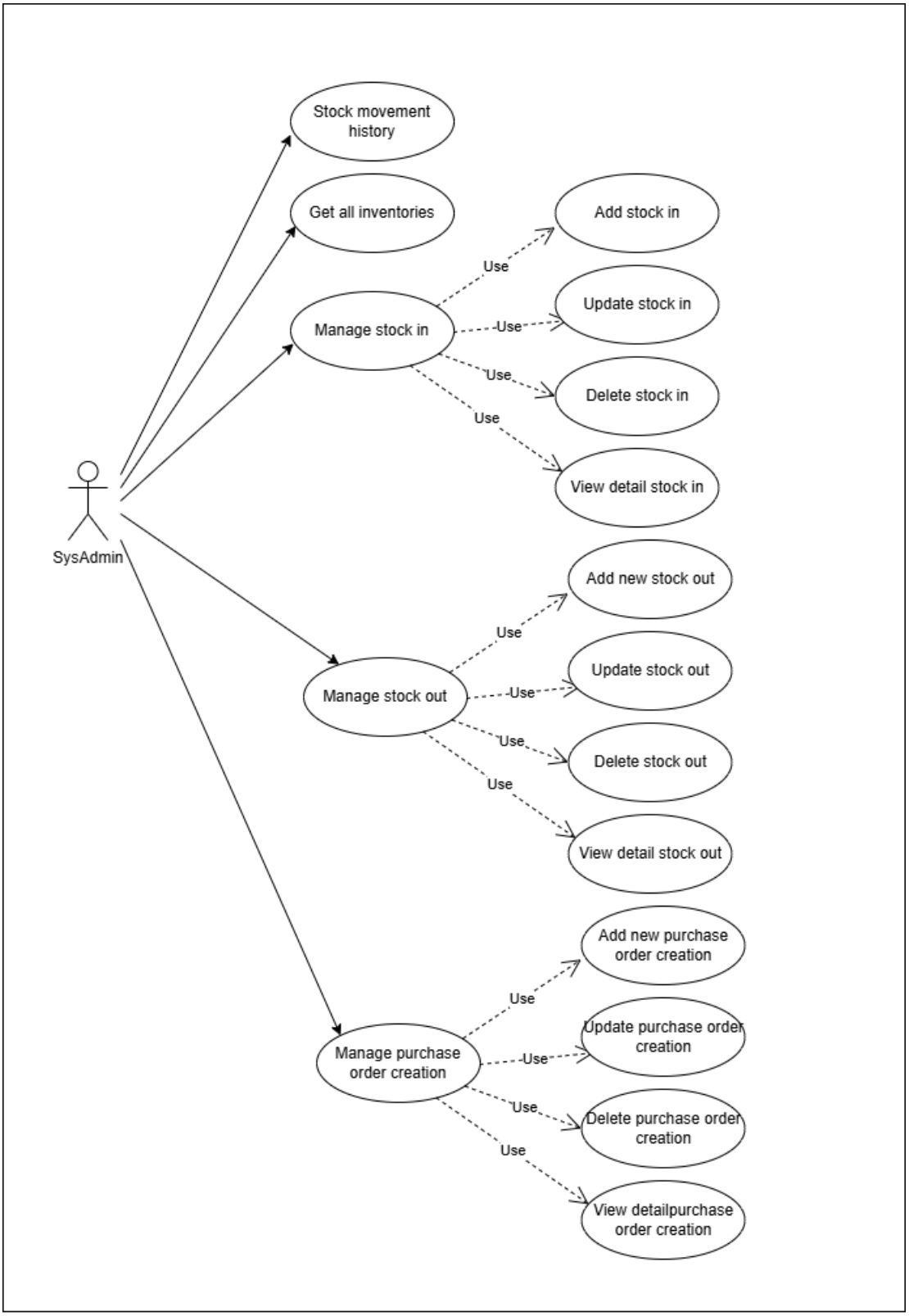
Shopping Cart, Checkout & Payment



Customers can manage the shopping cart, place orders, make payments, and view and track order status.

Administrators can manage orders.

Inventory Management



Administrators can manage inventory, including orders and stock in/out slips

1.7. Project Implementation Plan

1.7.1. Theoretical Background

1.7.1.1. Agile Software Development Model

Agile is a flexible software development methodology that emphasizes rapid response to change, incremental product delivery, and close collaboration among team members and stakeholders. Instead of defining a detailed plan for the entire project upfront, Agile divides the project into multiple iterations, in which the product is continuously designed, implemented, tested, and reviewed.

Core principles of the Agile Manifesto:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a fixed plan.

1.7.1.2. Scrum:

Scrum is an Agile framework that focuses on product development through a series of Sprints, each typically lasting from one to four weeks.

Scrum enables teams to effectively monitor progress, respond quickly to changes, and continuously improve product quality.

Key roles in Scrum:

- **Product Owner:** Responsible for defining requirements and prioritizing features in the Product Backlog.
- **Scrum Master:** Ensures that Scrum practices are followed and helps remove impediments that hinder the team.
- **Development Team:** A cross-functional team (Backend, Frontend, QA, UI/UX, etc.) responsible for delivering the product increment.

Scrum Events:

Event	Purpose	Outcome
Sprint	Product development cycle (1–4 weeks)	A potentially shippable product increment
Sprint Planning	Plan the Sprint and select user stories to be implemented	Sprint Backlog
Daily Scrum	Daily 15-minute stand-up meeting	Progress synchronization
Sprint Review	Present the increment to the Product Owner and stakeholders	Feedback and evaluation

Sprint Retrospective	Reflect on the Sprint and identify improvements	Process improvement actions

Scrum documents:

Document	Description
Product Backlog	An ordered list of all project requirements in the form of user stories
Sprint Backlog	A list of tasks selected for implementation in the current Sprint
Burndown Chart	A chart that visualizes the remaining work over time to track Sprint progress

1.7.1.3. Rationale for Choosing the Scrum Model

The project consists of many small and tightly related features that need to be developed and tested continuously, such as: viewing product lists; managing product information, lists, and statuses; placing orders and processing payments; tracking orders; handling stock in/out operations; monitoring inventory status, etc.

Therefore, the team chose Scrum because:

- It is well-suited for projects with an open scope and frequently changing requirements.
- It enables rapid incremental releases of system components for demonstrations and early feedback.
- It enhances collaboration among team members (backend, frontend, testers).
- It allows easy measurement of progress and productivity through Sprints.

1.7.1.4. Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration (CI): Whenever developers commit code, the system automatically builds and runs tests to detect defects early.

Continuous Delivery (CD): After successful build and testing, the system automatically deploys the new version to the staging or production environment.

Benefits:

- Reduces errors caused by manual operations.
- Enables fast and frequent release of updates.
- Ensures the product is always in a deployable state.

1.7.2. Implementation Plan

1.7.2.1. Scope and Objectives

The Electro system is divided into four main subsystems:

Subsystem	Brief Description
1. Product Catalog	Displays product lists and details, and supports product search and filtering for users. Admins can add products and update product information and status.
2. Authentication & Acess control	Users can register, log in, view personal profiles, order information, and order history. Admins can view the user list, view user details, and update user information and account status.
3. Cart / Checkout / Payment	Users can add products to the cart, proceed to checkout, and track orders. Admins can view order lists and details, update order status, and handle order cancellations.
4.Inventory Management	Admins can view inventory lists, create purchase orders, and manage stock in/out operations.

1.7.2.2. Infrastructure and Development Environment

Task / Tool	Purpose
PostgreSQL	Database management system.
Javascript & Java	Programming languages used for system development.
VSCode & Intellij IDEA	Integrated Development Environments (IDEs) for coding.
Docker & Docker Hub	Containerization and image repository for deployment.
Github & Github Actions	Source code management and CI/CD automation.
Microsoft Word	Documentation authoring tool.
Draw.io & PlanUML	Diagramming and modeling tools.

1.7.2.3. Project Schedule

Phase	Key Activities	Start	End	Duration (days)	Assignees
Project Initiation	Requirements: • Identify overall project requirements, objectives, and scope. • Identify business processes and customer business constraints. • Identify necessary user stories and use cases for the system. Deliverables: • **Business Requirements Document** (BRD).	01/09/2025	15/09/2025	15	Nguyễn Lê Quỳnh Hương

Analysis	Requirements: • The **Business Requirements Document (BRD)** is reviewed, approved, and stabilized. • Analyze customer requirements into use case diagrams • Specify detailed use cases • Describe UI mockups for each use case • Identify high-level tasks and major milestones for the project. Deliverables: • Software Requirements Specification (SRS), including functional & non-functional requirements, use cases, focused use cases, UI, constraints, and assumptions. • Excel summary of use cases. • Excel summary of GUIs and UI usage guidelines. • Excel Project Plan: schedule, sprint backlog, product backlog, burndown chart	16/09/2025	05/10/2025	20	Trang Gia Huy Nguyễn Lê Quỳnh Hương
Design	Requirements: • SRS is reviewed, approved, and stabilized. • Define system architecture. • Define communication diagrams. • Define deployment diagrams. • Define the C4 architectural model. • Design the database schema. • Define the test plan. Deliverables: • **System Analysis and Design Document** (SADD). • **Test Plan** document.	06/10/2025	19/10/2025	14	Trang Gia Huy Nguyễn Lê Quỳnh Hương

Sprint 1	Scope:
	• Users can view product lists and details, and search/filter products.
	• Admins can add products, edit product details, and update product status.
	• Basic CI/CD pipeline is set up using GitHub Actions:
	  o Automatically build and test backend on commits to the develop branch.
	  o Push new images to Docker Hub after tests pass.
	  o Deploy new images from Docker Hub to the staging
	• Set up a staging environment for feature demos.
	Deliverables:
	• Automated CI/CD pipeline in operation.
	• Users can access and test the Product Catalog module on staging.
		20/10/2025	26/10/2025	7	Trang Gia Huy
					Nguyễn Lê Quỳnh Hương

Sprint 2	Scope:
	• Users can register, log in, view profiles, order info, and order history.
	• Admins can view user lists and details, and update user info and account status.
	• Continue using the established CI/CD pipeline:
	  o Extend pipeline to run automated tests for the Authentication module.
	  o When the Auth module is merged into the main branch, automatically build and deploy to staging.
	Deliverables:
	• Stable CI/CD pipeline.
	• Users can access Product Catalog and Authentication modules on staging.
		27/10/2025	02/11/2025	7	Trang Gia Huy
					Nguyễn Lê Quỳnh Hương

Sprint 3	Scope:
	• Users can add products to cart, checkout, and track orders.
	• Admins can view order lists and details, update order status, and handle cancellations.
	• Continue using the CI/CD pipeline:
	  o Extend pipeline to run automated tests for Cart, Checkout & Payment modules.
	  o When these modules are merged into main, automatically build and deploy to staging.
	Deliverables:
	• Stable CI/CD pipeline.
	• Users can access Product Catalog, Authentication, Cart, Checkout & Payment modules on staging.
		03/11/2025	09/11/2025	7	Trang Gia Huy
					Nguyễn Lê Quỳnh Hương

Sprint 4	Scope:
	• Admins can view inventory, create purchase orders, and manage stock in/out.
	• Continue using the CI/CD pipeline:
	  o Extend pipeline to run automated tests for the Inventory module.
	  o When the Inventory module is merged into main, automatically build and deploy to staging.
	  o Deploy to the production environment. The pipeline supports both staging and production.
	Deliverables:
	• CI/CD pipeline fully operational with production deployment.
	• System is ready for acceptance testing and go-live.
		10/11/2025	16/11/2025	7	Trang Gia Huy
					Nguyễn Lê Quỳnh Hương

Chapter 2: System Analysis and Design

2.1. System Architecture Overview

2.1.1. Methodological Foundation

2.1.1.1. Monolithic Architecture

Monolithic Architecture is a design model in which all components of a system are integrated and deployed as a single application.

All parts, from the User Interface (UI), Business Logic, to Data Access, are contained within a single codebase and run in the same process.

In this model, all components are packaged, compiled, and deployed together.

Layers/components directly invoke each other without using APIs or message queues.

2.1.1.2. Three-Tier Architecture (N-Tier Architecture):

Three-Tier Architecture (also known as N-Tier Architecture) is a monolithic software architecture model in which the system is divided into three main functional layers:

- Presentation Layer
- Application / Business Layer
- Data Layer

Each layer has a distinct responsibility, helping to separate the user interface, business logic, and data storage.

Layer	Role	Selected Technology
Presentation Layer	Displays the user interface, sends and receives requests to/from the application layer	React
Application Layer	Handles business logic, authentication, and data flow coordination	Spring Boot
Data Layer	Stores and retrieves persistent data	MySQL

The Three-Tier Architecture (N-Tier Architecture) is chosen for this project due to the following advantages:

- Easy to scale and maintain.
- Clear separation between business logic and user interface.
- Easy integration of different technologies at each layer.

2.1.1.3. Single Page Application (SPA) Model

SPA is a web application model that loads a single HTML page and dynamically updates content when users interact, without reloading the entire page.

Operating mechanism:

- When users access the application, the browser loads all JavaScript code.
- JavaScript handles routing, renders corresponding content, and calls backend APIs to retrieve data in JSON format.
- The user interface is dynamically updated without a full page reload.

The SPA model is selected for this project because of the following advantages:

- Fast response time and smooth user experience.
- Reduced server load as UI processing is handled on the client side.
- Easy separation between frontend and backend (client–server model).

2.1.1.4. MVC Model (Model–View–Controller)

MVC is a popular source code organization model in web applications, dividing the backend system into three distinct parts:

- Model: Manages data and business logic.
- View: Displays information to users.
- Controller: Receives requests, processes them, and coordinates data between Model and View.

Components are organized into technical layers, including:

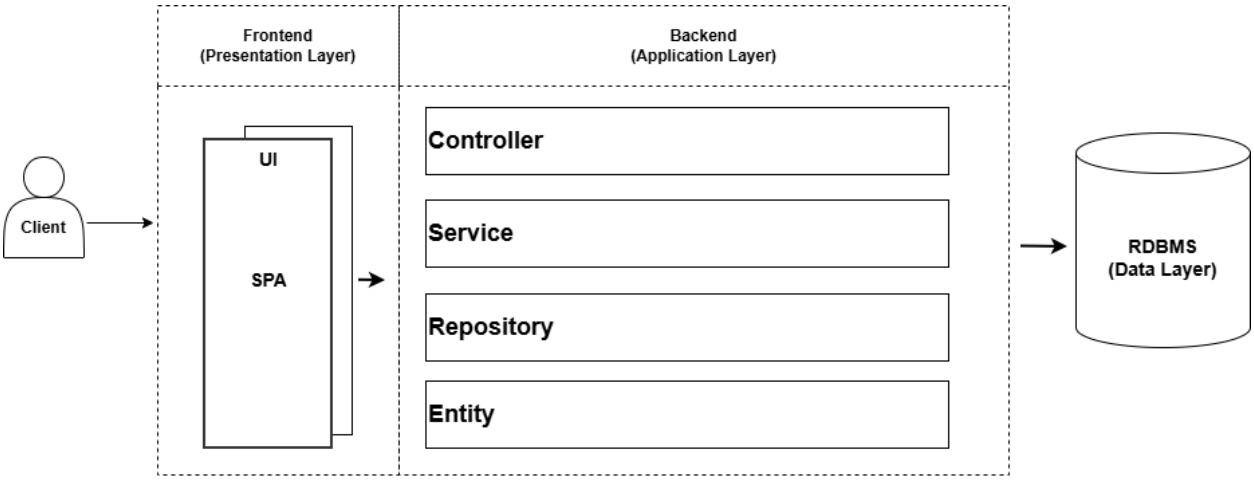
- Controller: Receives and routes requests from the client.
- Service: Handles business logic.
- Repository: Accesses data.
- Entity: Represents database tables.

Classes belonging to different business domains (product, order, user, etc.) are managed together within the same layer and are not separated into independent modules. Therefore, the system follows a traditional monolithic architecture rather than a Modular Monolith.

Implementation in Spring Boot:

Component	Role in the Project	System Entities
Model	Handles business logic and data persistence	Entity, Repository, Service
View	Displays results to users	JSON responses returned to SPA
Controller	Routes requests and invokes services	Controller classes

2.1.2. Block Diagram Design:



The system is designed based on Monolithic Architecture combined with Three-Tier Architecture (N-Tier Architecture).

The Presentation Layer uses the Single Page Application (SPA) model, while the Application Layer adopts the MVC (Model–View–Controller) model to ensure clear layering, scalability, and maintainability.

1. Presentation Layer

- Developed using ReactJS and operates under the SPA model, meaning the entire user interface is loaded once, and content is dynamically updated through API calls (RESTful APIs) to the backend without reloading the whole page.
- Responsible for displaying data, handling user interactions, and sending HTTP requests to the application layer.

2. Application Layer

- Built using Spring Boot (Java) and follows the MVC model with four main components:
- Controller: Receives and processes requests from the frontend and routes them to corresponding services.
- Service: Executes core business operations and handles the main system logic.
- Repository: Communicates with the database via JPA/Hibernate to perform CRUD operations.
- Entity (Model): Represents database tables.

Main functionalities implemented in the application layer include:

Functional Group	Brief Description
1. Product Catalog	Displays product lists and details, supports searching and filtering for users. Administrators can add products, update information, and manage product status.

2. Authentication & Authorization	Users can register, log in, view personal information, view orders, and order history. Administrators can view user lists, user details, and update user information and status.
3. Cart / Checkout / Payment	Users can add products to the cart, proceed to checkout, and track orders. Administrators can view order lists and details, update order status, and handle order cancellations.
4. Inventory Management	Administrators can view inventory lists, create purchase orders, manage stock-in and stock-out operations.

Additionally, this layer includes supporting components such as:

- DTO, Mapper, Constants, Exception Handlers, Configuration, which help standardize data, handle errors, and ensure system scalability.

3. Data Layer

Uses MySQL as the database management system to store all system data.

Communicates with the application layer through the Repository layer, ensuring data integrity and security.

Layer Interaction and Responsibility Separation

The relationship between layers follows the principle of Separation of Concerns:

- Frontend (React SPA): Responsible only for presentation and user interaction.
- Backend (Spring Boot MVC): Responsible for business logic processing and API provisioning.
- Database (MySQL): Responsible for data storage and retrieval.

System Benefits

Thanks to the combination of N-Tier Architecture, SPA, and MVC, the system achieves:

- High modularity and maintainability, as each layer has a distinct responsibility.
- High performance, with dynamic UI updates via APIs without page reloads.
- Good scalability, allowing individual layers to be extended or replaced without affecting the entire system.

2.2. Communication View

2.2.1. Methodological Foundation

The Communication View (also referred to as the Communication Diagram or Component Interaction View) describes how components within the system communicate with each other, including:

- System components (components / modules): such as frontend, backend, database, and third-party services.
- Communication flows (interactions): via REST APIs, HTTP, message queues, database queries, and other communication mechanisms.

The Communication View explains how different parts of the system interact with one another, which protocols are used, and whether the communication occurs at a logical or physical level.

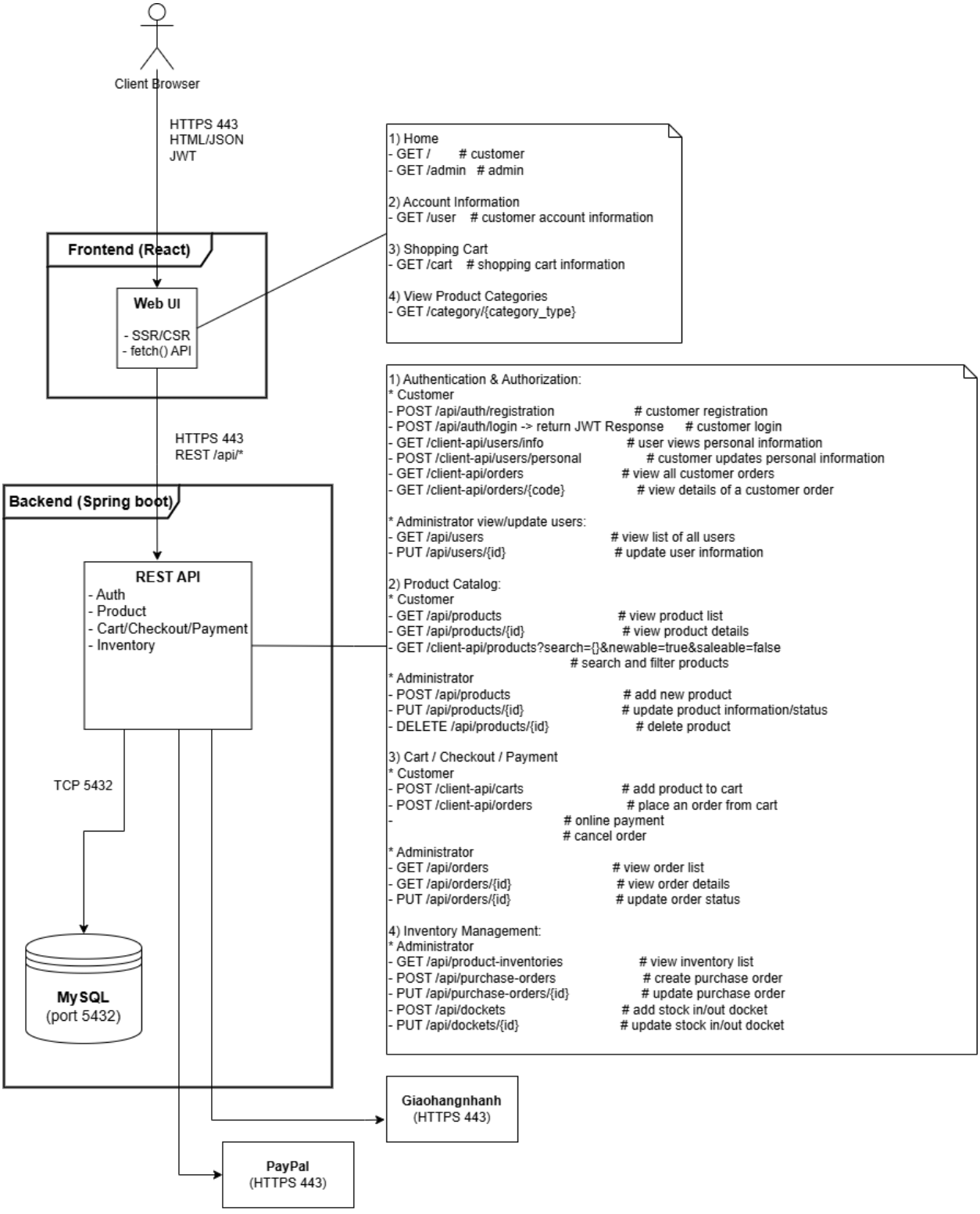
The Communication View is used to:

- Describe how data or requests flow between different parts of the system.
- Clarify dependencies between layers/components or between the system and external services.
- Help developers, testers, and system architects clearly understand key interaction flows, supporting system implementation and integration testing.

How to Define the Communication View

1. Identify the system components (frontend, backend, database, external services, etc.).
2. Identify the communication channels between them (HTTP, REST APIs, JDBC, gRPC, etc.).
3. Identify typical data flows or representative actions, for example:
4. A user sends a request from the web UI → the backend processes the request → reads/writes data to the database → returns the result to the frontend.
5. The backend calls third-party APIs (e.g., PayPal, GiaoHangNhanh, etc.).
6. Draw a diagram illustrating the components and the interaction arrows between them.

2.2.2. Diagram Design:



The Communication View of the system describes how the main components exchange data within a three-tier monolithic architecture:

- Frontend (React): User interface layer (Presentation Layer).
- Backend (Spring Boot): Business and data processing layer (Application Layer + Business Logic).
- Database (MySQL): Data storage layer (Data Layer).
- Third-party services: PayPal (payment processing) and GiaoHangNhanh (shipping and delivery).

Main Communication Flows

1. Frontend ↔ Backend

The frontend (React) sends HTTP requests (via the Fetch API) to the backend (Spring Boot REST APIs).

The main API endpoints are grouped as follows:

- /api/auth/... — authentication, login, and user management.
- /api/products/... — product listing, searching, adding, and deleting.
- /api/orders/... — order placement, payment processing, and order history viewing.
- /api/inventories/... — inventory management, stock-in, and stock-out operations.

This communication occurs over HTTP (port 8080) following the client–server model.

2. Backend ↔ Database (MySQL)

The backend communicates with the MySQL database via JDBC (port 5432).

Repositories (JPA/Hibernate) are responsible for executing data queries for entities such as User, Product, Order, and Inventory.

3. Backend ↔ Third-Party Services

Communication with the PayPal API (HTTPS, port 443) for online payment processing.

Communication with the GiaoHangNhanh API (HTTPS, port 443) for shipment creation and delivery status tracking.

2.3. Deployment View

2.3.1. Methodological Foundation

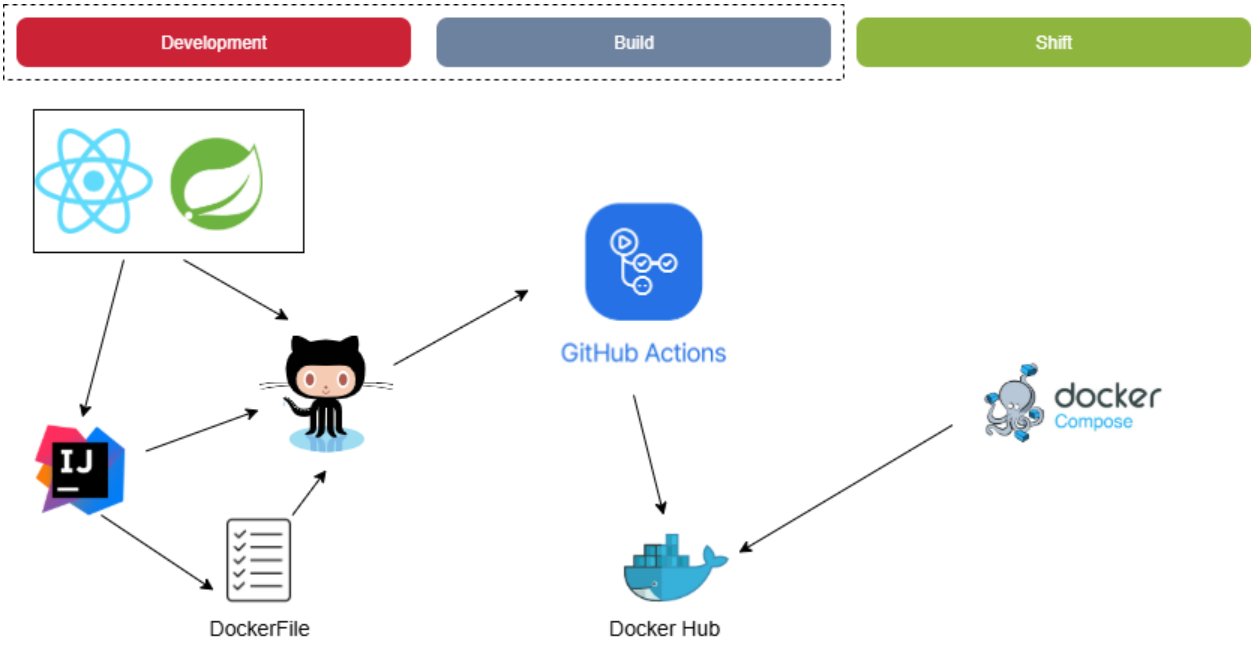
The Deployment View (also referred to as the Physical View in the 4+1 architectural model) describes how software is deployed on physical infrastructure or real operational environments.

The Deployment View explains where and how the system is deployed, and which environments the components run in.

The objectives of the Deployment View include:

- Describing physical nodes or deployment environments.
- Illustrating the relationship between software components and the infrastructure on which they are deployed.
- Showing the build, packaging, and deployment process from development to production.
- Serving as a foundation for deployment automation (CI/CD) and software lifecycle management.

2.3.2. Deployment Diagram Design



Phase	CI/CD Stage	Description
Development	Pre-Build	Coding and committing source code to GitHub. IntelliJ IDEA is used to develop both React and Spring Boot applications.
Build	Build + Test	GitHub Actions builds the application, runs tests, and creates Docker images.

Shift	Deploy	Docker images are pushed to Docker Hub and deployed using Docker Compose.

The Deployment View describes the system deployment process from development to actual operation, in the context of a project developed using the Scrum model combined with a CI/CD pipeline.

The entire process is automated through GitHub Actions and Docker, ensuring continuity, consistency, and reliability in deployment.

Phase 1: Development

During the development phase, developers work in parallel on two main components:

- Frontend: React application (SPA – Single Page Application).
- Backend: Spring Boot application (RESTful API).

Both components are developed and managed on GitHub, with IntelliJ IDEA as the primary development environment.

At this stage, Dockerfiles are predefined to describe how each component is built and how the runtime environment is configured.

Once user stories in a sprint are completed, developers commit and push source code to GitHub, triggering the automated CI/CD pipeline.

Phase 2: Build & Test (Continuous Integration)

After new changes are pushed to the repository, GitHub Actions automatically triggers the Continuous Integration (CI) process.

The main steps include:

- Build code: GitHub Actions automatically compiles and builds the backend (Spring Boot) and frontend (React) applications according to their Dockerfiles.
- Run tests: The system executes automated tests (unit tests and integration tests) to ensure code quality.
- Build Docker images: If both build and tests succeed, Docker images are created from the source code.
- Push images to Docker Hub: The images are stored on Docker Hub, which acts as a central registry for container images ready for deployment.

This phase ensures that every change in Git is automatically integrated, tested, and packaged, eliminating risks caused by manual operations and maintaining stability across environments.

Phase 3: Deploy (Continuous Deployment)

In this phase, Docker images are deployed to the runtime environment using Docker Compose.

Docker Compose is responsible for:

- Pulling the latest images from Docker Hub.
- Initializing and running containers for each service (React Frontend, Spring Boot Backend, MySQL Database).
- Configuring networking, environment variables, and port mappings between containers.

This process enables automated, consistent, and easily repeatable deployments.

Thanks to Docker, both development and operations (DevOps) teams can ensure that the application runs consistently across all environments, from local machines to production servers.

2.4. Decomposition View – C4 Model

The C4 Model is a visual model used to describe software architecture at multiple levels of detail from a high-level system overview down to concrete source code structures.

Proposed by Simon Brown, this model divides the architecture into four levels (C1–C4), with each level answering a different question:

Level	Name	Question	Target
C1	Context Diagram	What is the system's context?	Provides a "big picture" view — who interacts with the system and for what purpose.
C2	Container Diagram	How is the system divided into containers?	Shows the main deployable components such as web apps, databases, APIs, etc.
C3	Component Diagram	What functional components exist inside each container?	Helps developers understand the internal logical structure.
C4	Code Diagram	How is each component implemented?	Intended for developers — shows classes, modules, and package structures.

2.4.1. C1 Diagram – System Context

2.4.1.1. Theoretical Foundation

The C1 System Context diagram presents an overall picture of the system within its operating environment by describing the relationships between the system and external actors (users, other systems, integrated services).

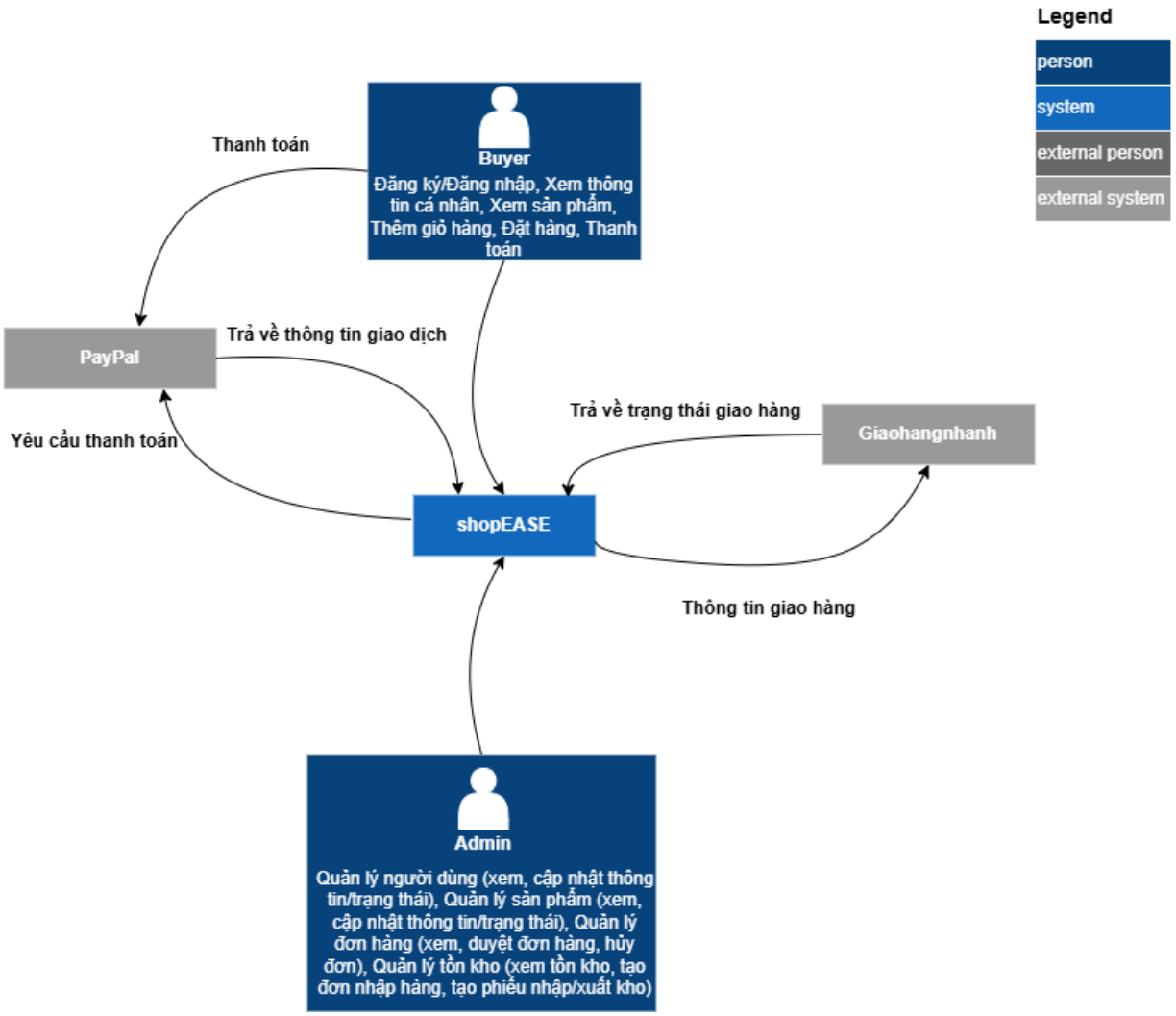
It helps readers (especially non-technical stakeholders) understand:

- What problem the system is solving;
- Who uses the system;
- Which other systems it interacts with.

How to define the C1 diagram:

1. Identify the main actors (users, administrators, partner systems).
2. Identify the central system (System Under Design).
3. Draw the relationships (data flows or interactions) between actors ↔ system ↔ external systems.

2.4.1.2. Diagram Design



Summary:

The C1 System Context Diagram illustrates the shopEASE system as a central e-commerce coordination platform connecting buyers, administrators, the PayPal payment gateway, and the GiaoHangNhanh logistics service.

This model clarifies the system scope, functional boundaries, and primary interaction flows, serving as the foundation for further decomposition at the C2 (Container) and C3 (Component) levels.

In the diagram, the system is placed at the center and interacts with external actors as follows:

1. Buyer

The buyer is the primary user of the system.

They perform the following functions:

- Register and log in;

- View and update personal information;
- Browse the product catalog;
- Add products to the shopping cart;
- Place orders and make payments.
- Payment requests are sent through the shopEASE system.
- Buyers receive feedback on delivery status and payment information from the system.

2. Admin

The administrator is responsible for operating and controlling system data.

Main responsibilities include:

- User management (viewing and updating user information);
- Product management (adding, editing, and updating product status);
- Order management (approving, canceling, and updating order status);
- Inventory management and creating stock-in/stock-out records.

Administrators interact directly with the shopEASE system to perform management operations.

3. PayPal (External Payment System)

PayPal is an external system integrated to handle online payments.

The shopEASE system sends payment requests to PayPal, which then returns transaction results (success or failure).

Communication is performed through secure APIs using the HTTPS protocol.

4. GiaoHangNhanh (Logistics System)

GiaoHangNhanh is an external system responsible for product delivery.

The shopEASE system sends shipment information (delivery address, order ID, product details, weight, etc.) to GiaoHangNhanh.

The system receives delivery status updates (received, in transit, completed, or canceled).

2.4.2. C2 Diagram – Container

2.4.2.1. Theoretical Foundation

The C2 Container Diagram shows how the system is divided into containers, where each container represents a deployable unit (such as a web application, mobile application, database, or API service).

It helps in understanding the logical deployment architecture and the main technologies used, including:

Independently running components (backend, frontend, database, external services).

Communication mechanisms between them (REST APIs, message queues, database connections).

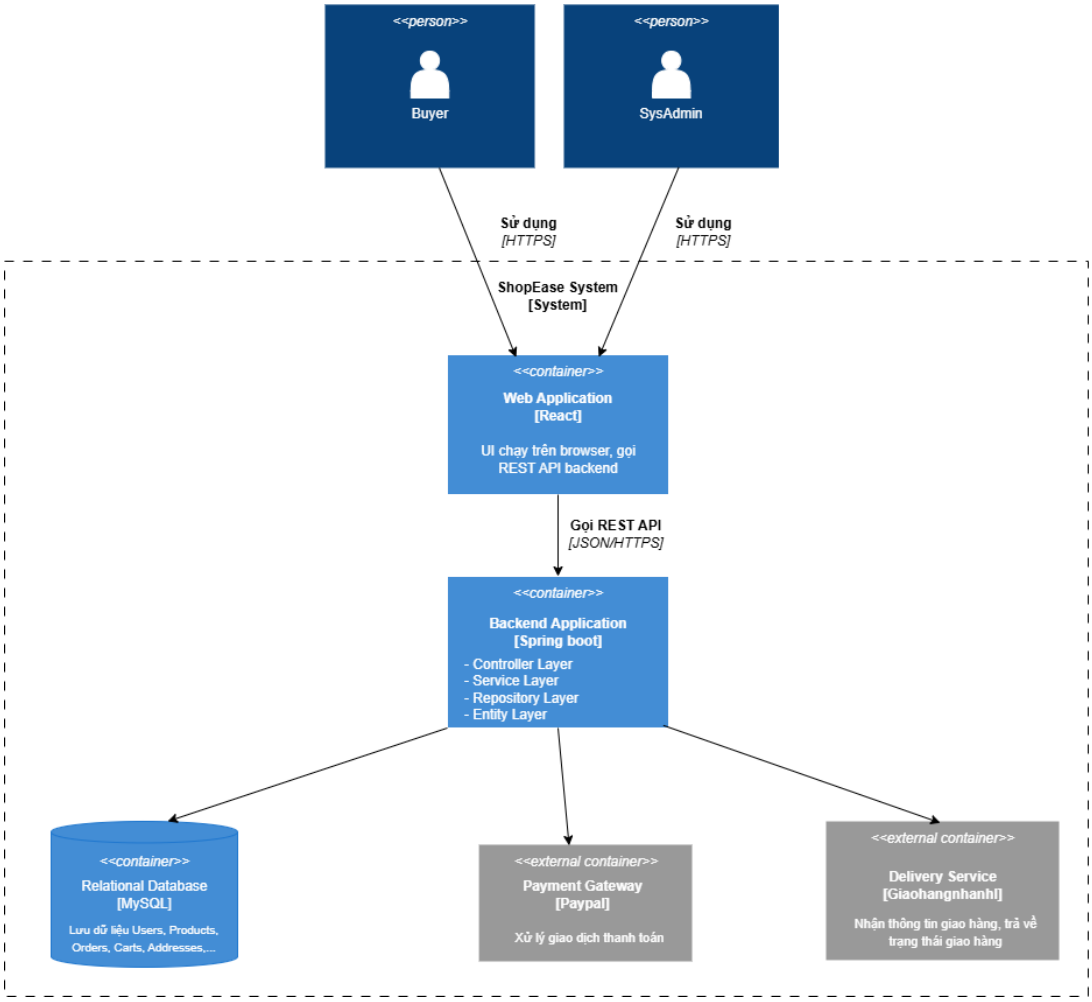
How to identify containers:

Decompose the system based on deployment constraints or functional responsibilities.

Each container may correspond to:

- A separately deployed application (e.g., Spring Boot application, React frontend),
- A background service or worker,
- A database.

2.4.2.2. Diagram Design



The C2 Container Diagram of shopEASE illustrates that the system is built using a Monolithic Architecture combined with Three-Tier Architecture (N-Tier Architecture) and the MVC model. The main containers include:

1. Web Application [React]

Role:

- Serves as the user interface (UI) of the system and runs in the web browser.

Functions:

- Allows Buyers and System Administrators to interact directly with the system through the web interface.
- Supports operations such as viewing product lists, managing the shopping cart, placing orders, and administering orders, users, and inventory.

Communication:

- Sends and receives data from the Backend Application via REST APIs using the HTTPS/JSON protocol.

- The entire user interface is built following the Single Page Application (SPA) model, providing a smooth user experience.

2. Backend Application [Spring Boot]

Role:

- Handles all business processes, system logic, and data access.

Architecture:

- Organized according to the MVC pattern with clear layer separation:
- Controller Layer: Receives requests from the frontend and maps API endpoints.
- Service Layer: Processes core business logic.
- Repository Layer: Interacts with the database via JPA/Hibernate.
- Entity Layer: Represents data objects.

Communication:

- Connects to the MySQL database for data persistence.
- Calls the PayPal API to process payments.
- Calls the GiaoHangNhanh API to manage shipment information and delivery status.

3. Relational Database [MySQL]

Role:

- Stores the core data of the system, including:

User information, products, orders, shopping carts, inventory, addresses, delivery status, and related data.

The backend connects to the database through JPA repositories using the MySQL Connector (JDBC).

4. External Container – Payment Gateway [PayPal]

PayPal is an external service used for online payment processing.

When a user places an order and selects a payment method, the backend:

- Sends a payment request to PayPal via secure HTTPS APIs.
- Receives transaction results (success or failure) and stores them in the system.

5. External Container – Delivery Service [GiaoHangNhanh]

GiaoHangNhanh is an external service used to manage order delivery.

The backend:

- Sends shipping information to GiaoHangNhanh.
- Receives delivery status updates (received, in transit, completed, or canceled).

2.4.3. C3 Diagram – Component

2.4.3.1. Theoretical Foundation

At the C3 level, the software components within each container are described.

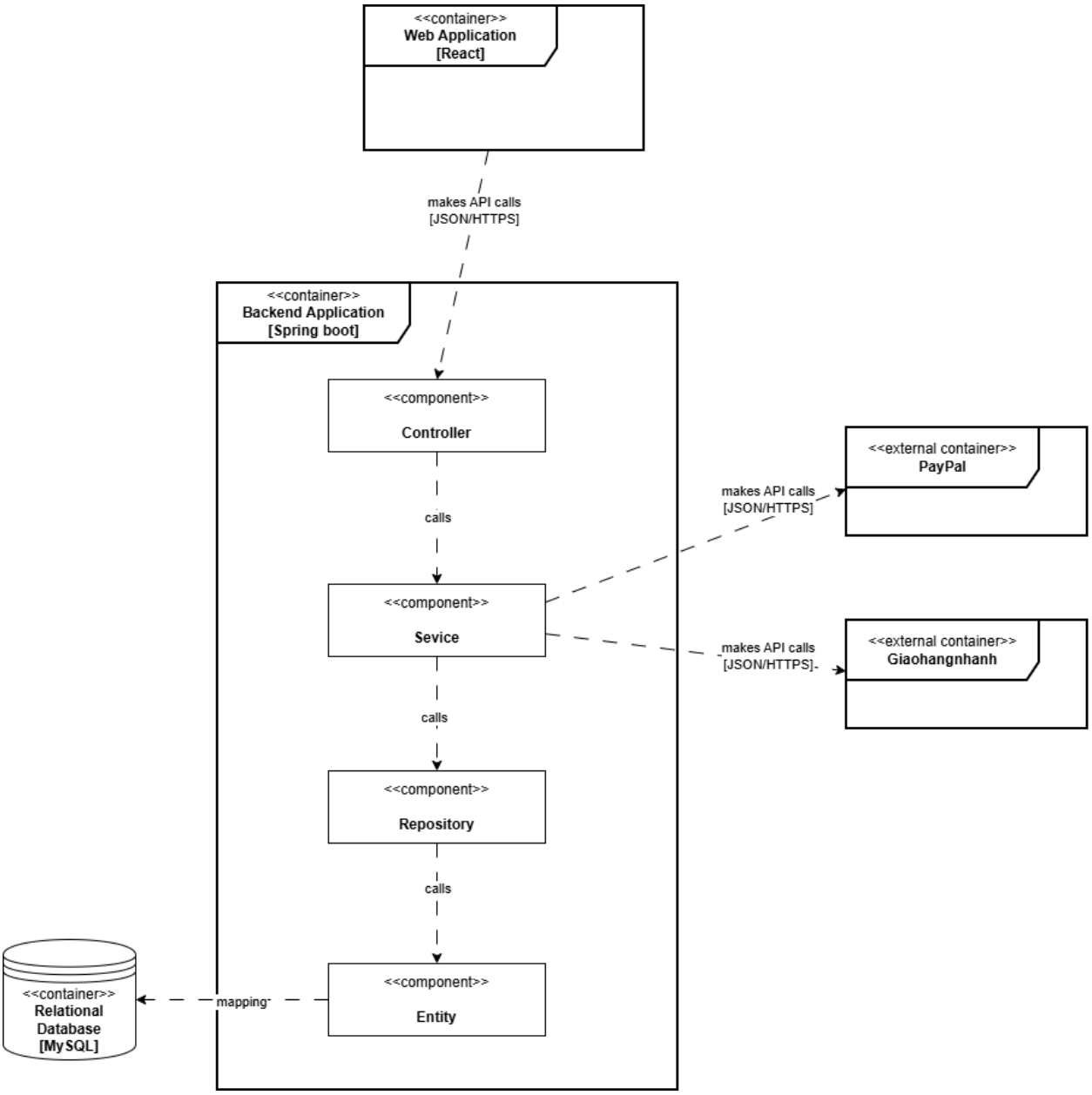
For the backend of this system, the main components correspond to the technical layers

Each layer is considered an independent component with a specific responsibility.

Component	Description
Controller Layer	Receives and processes HTTP requests from the client (React). Maps REST endpoints to the corresponding services.
Service Layer	Contains business logic and coordinates interactions between controllers and repositories.
Repository Layer	Communicates with the database (MySQL) through Spring Data JPA.
Entity Layer	Represents database tables.
External Integration	Invokes external APIs such as PayPal (payment processing) and GiaoHangNhanh (delivery tracking).

2.4.3.2. Diagram Design

High-Level



The C3 Component Diagram provides a detailed view of the components inside the Backend Application (Spring Boot) container.

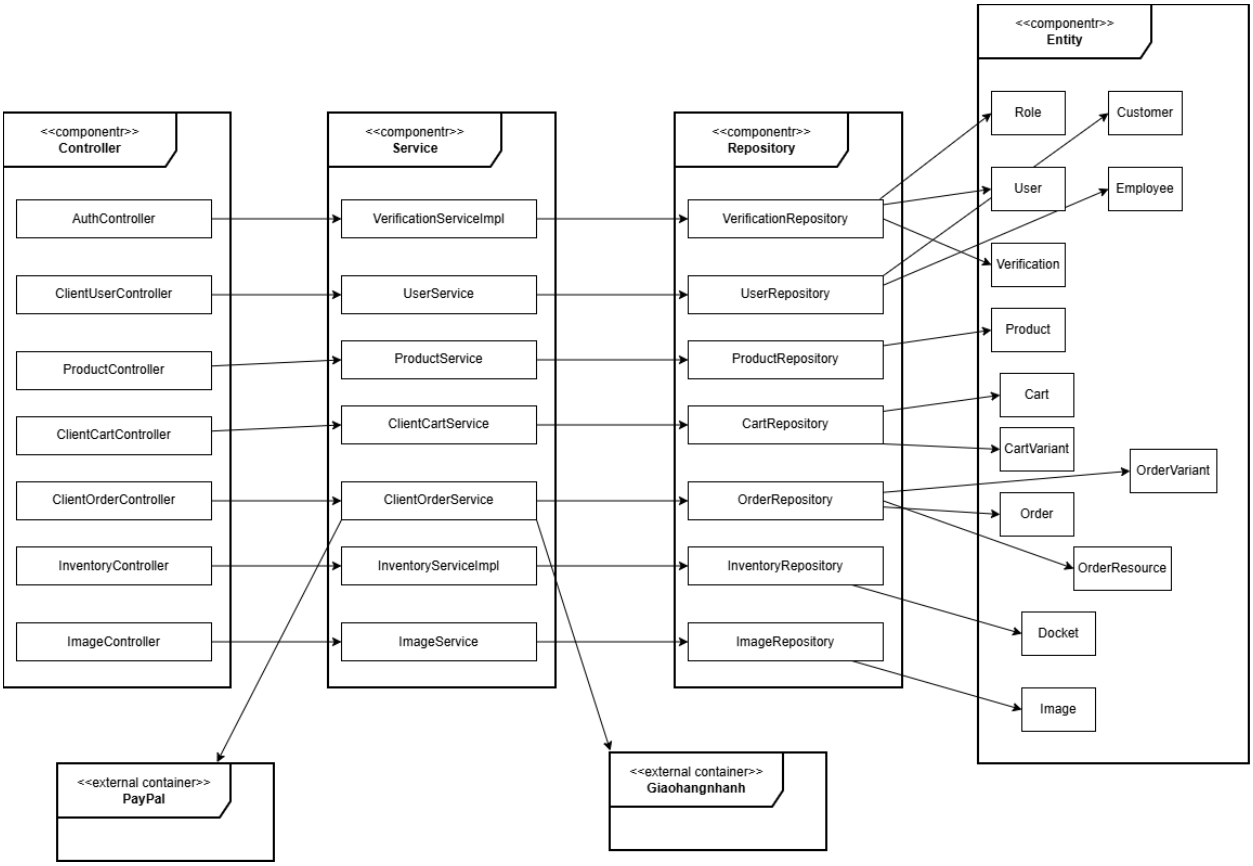
The application is divided into four main layers:

- Controller Layer: Handles client requests and routes API calls.
- Service Layer: Implements business logic and integrates with external systems such as PayPal (payment processing) and GiaoHangNhanh (delivery management).
- Repository Layer: Provides data access operations.
- Entity Layer: Maps objects in the source code to tables in the database.

The diagram also illustrates the relationships between the backend and other components, including the frontend (React), the MySQL database, and external services.

Data flows are transmitted using the HTTPS protocol and JSON format, ensuring security and ease of integration.

Module-Level



2.4.4.1. Theoretical Foundation

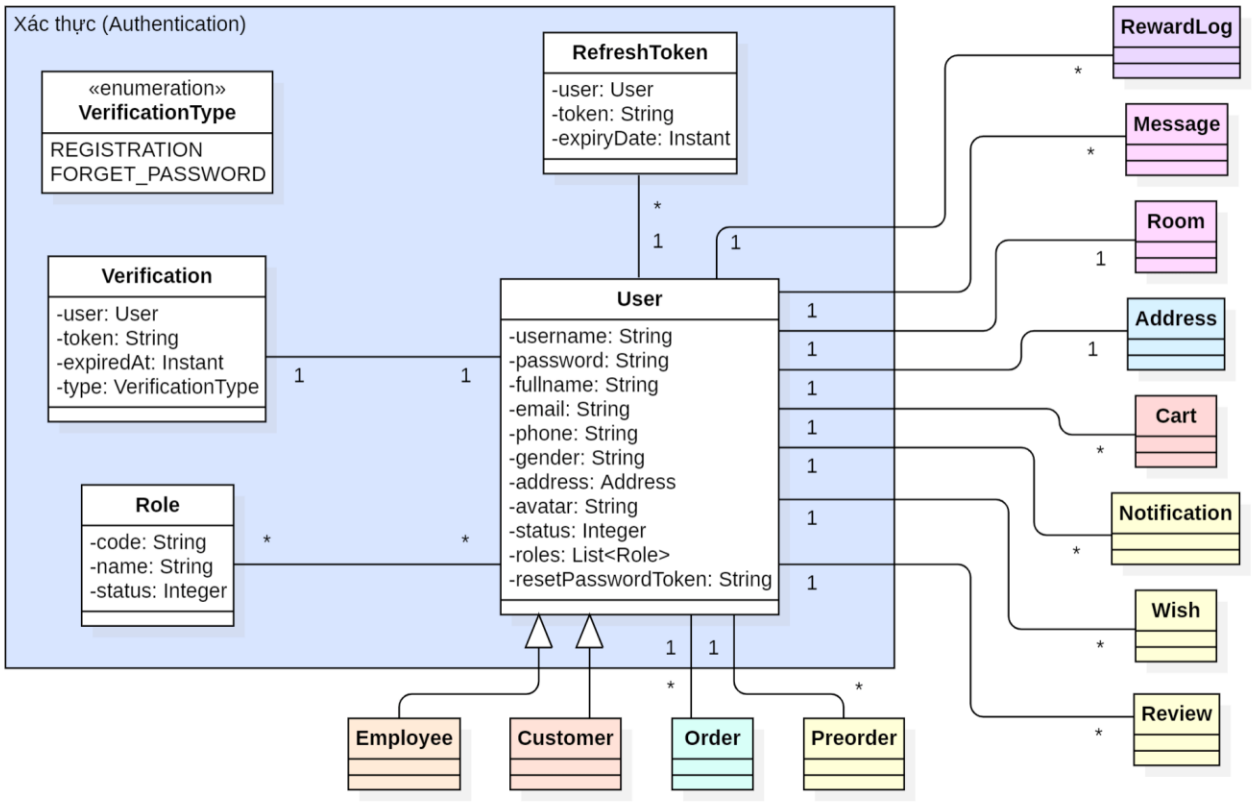
The Code Level (Code / Implementation Level) is the most detailed level in the C4 Model, describing the internal structure of components defined at the C3 level.

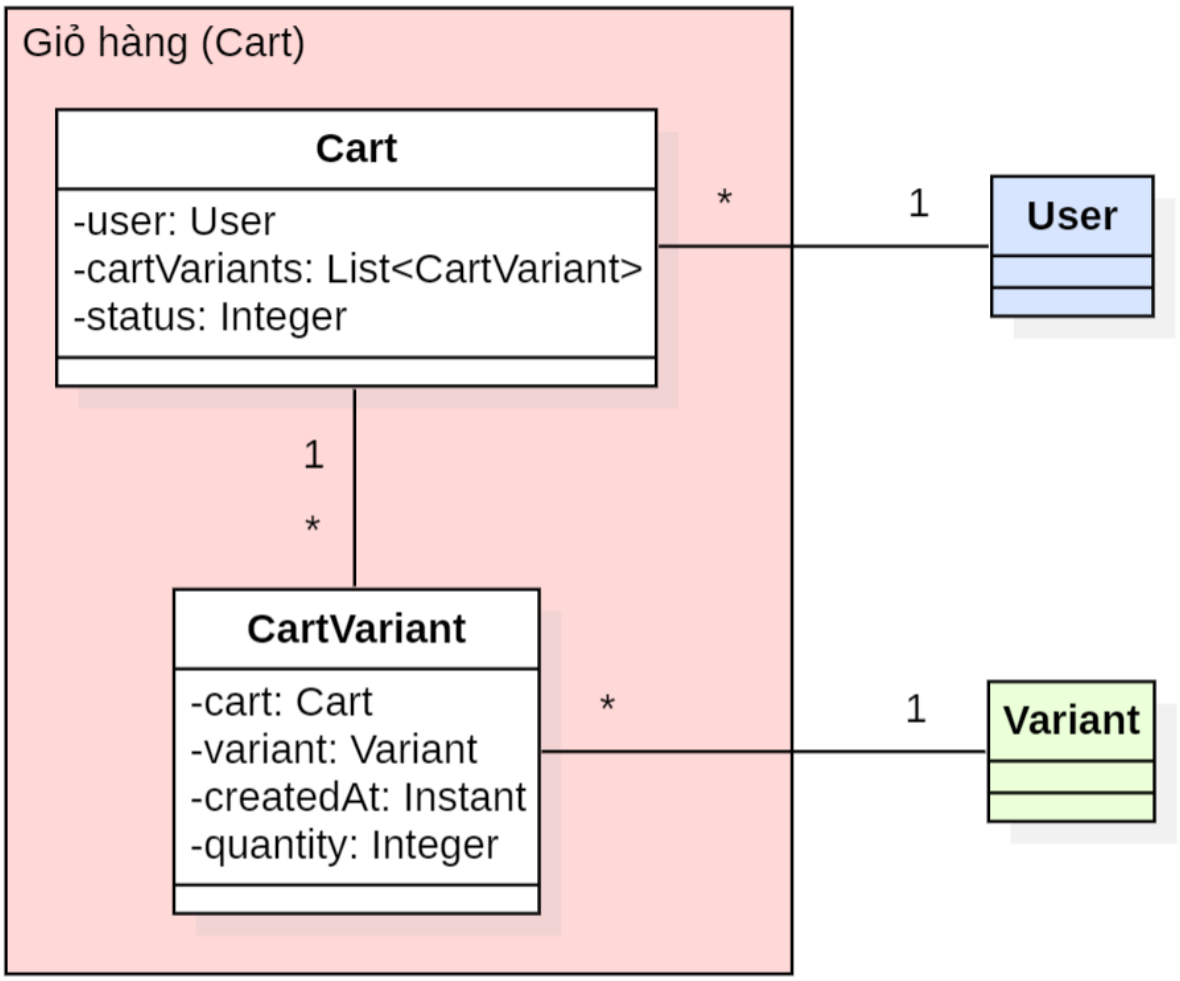
While the Component Diagram shows major functional blocks such as Controller, Service, and Repository, the Code Level goes deeper by illustrating the relationships between classes, interfaces, methods, and specific source code elements that make up each component.

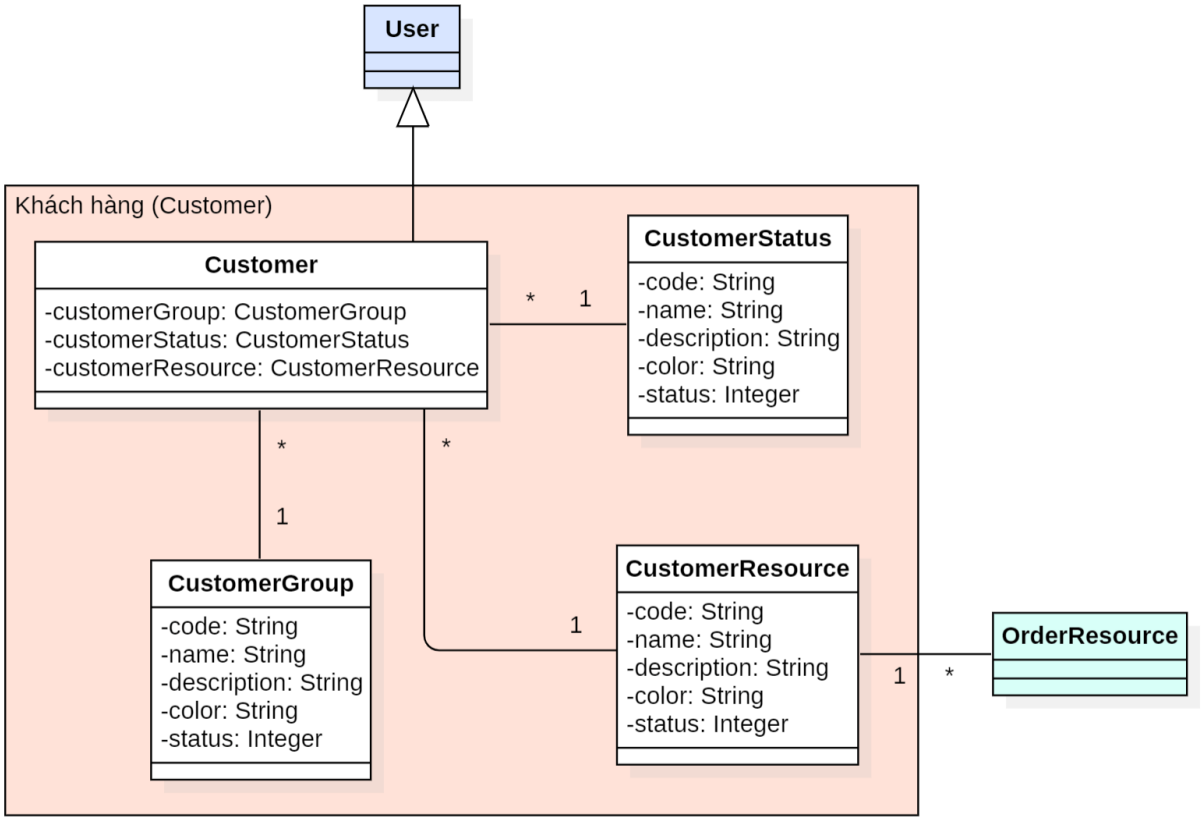
The objectives of this level are to help developers, software engineers, and maintainers clearly understand:

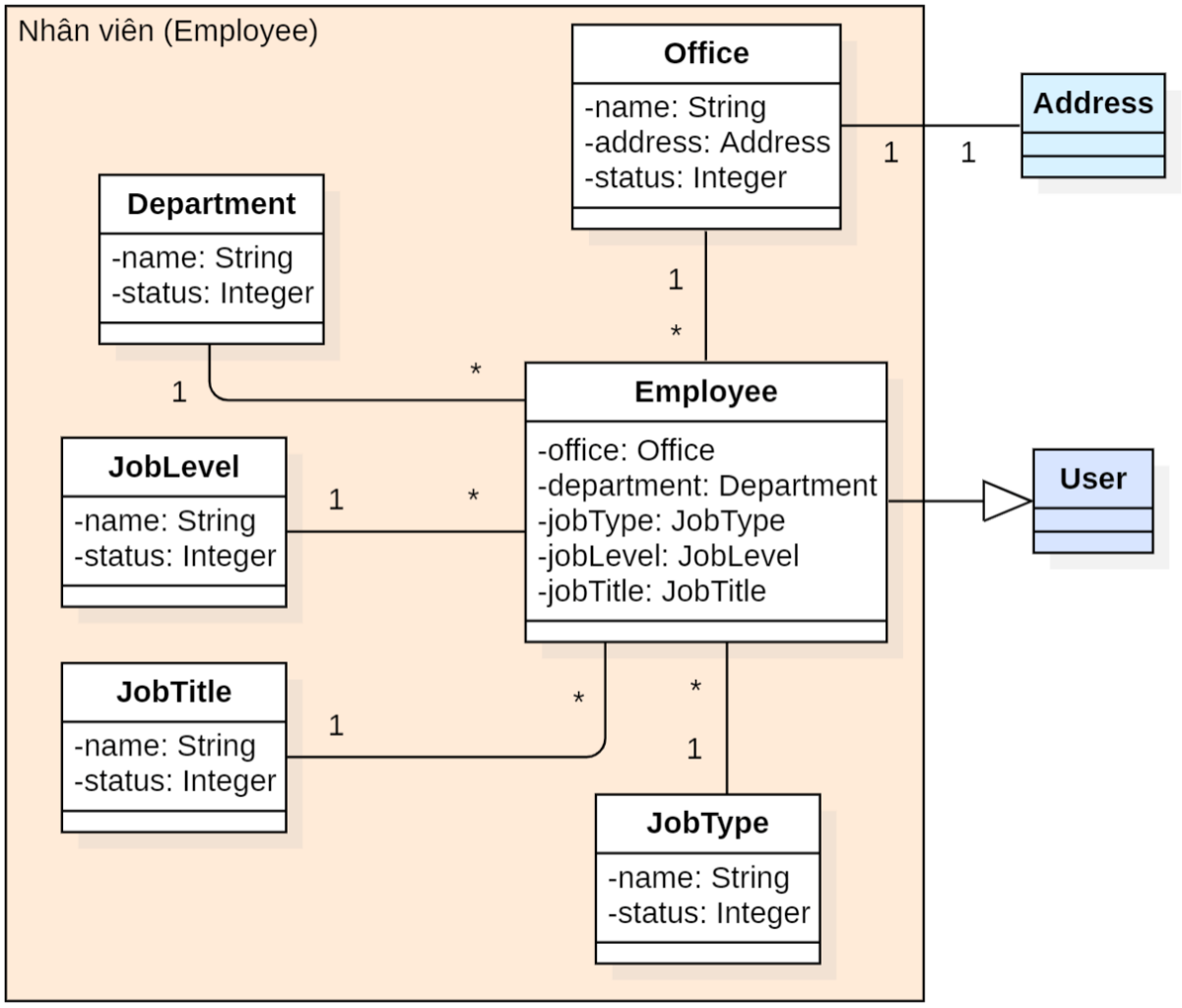
- How classes are organized and interact within a component.
- Dependencies between classes, interfaces, entities, and external libraries.
- The detailed processing flow from business requirements down to source code execution.

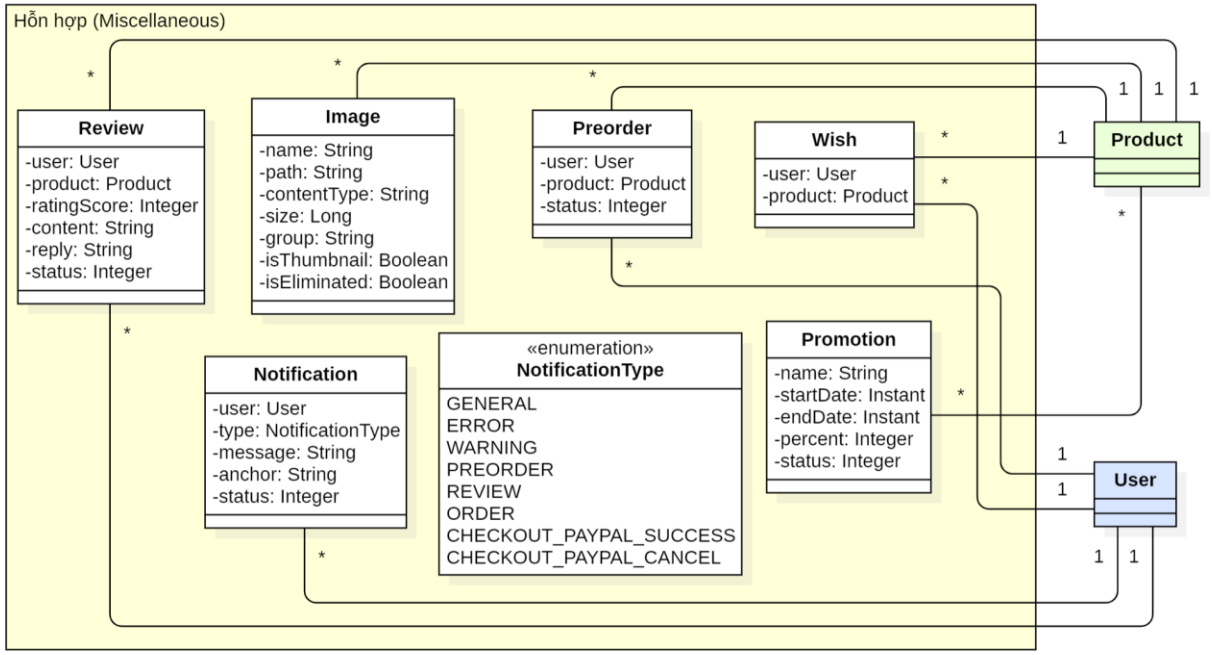
2.4.4.2. Diagram Design

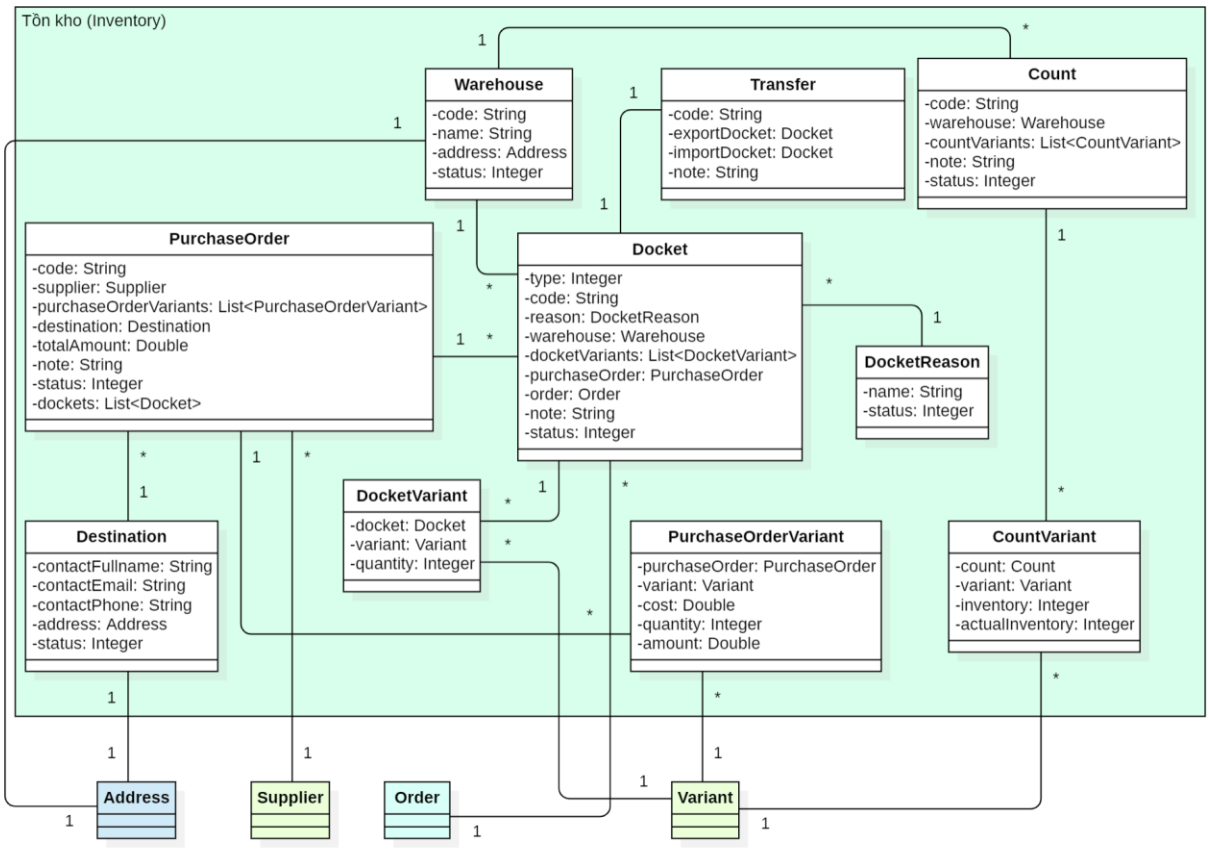


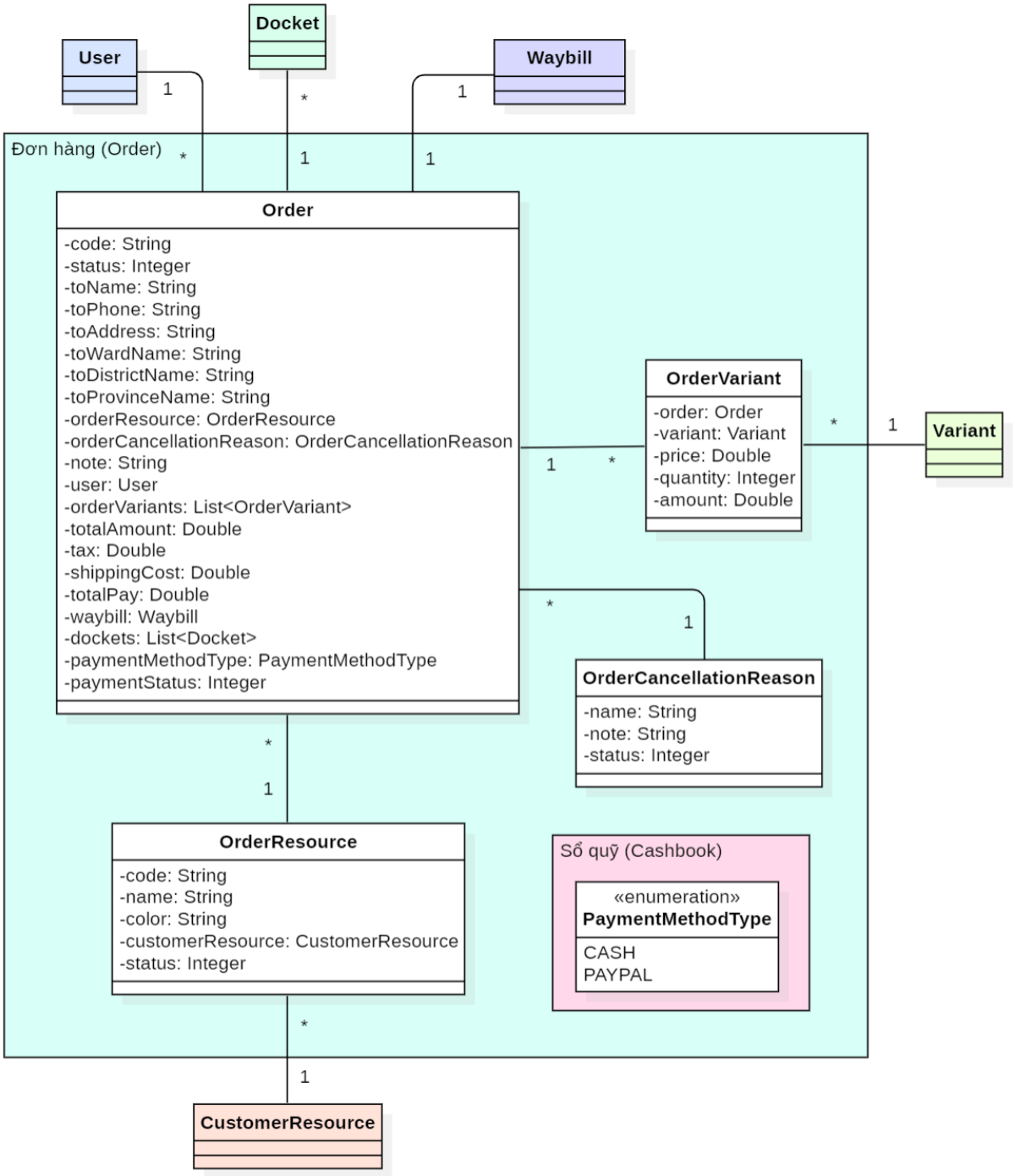

Giỏ hàng (Cart)
Cart
-user: User
-cartVariants: List<CartVariant>
-status: Integer
*
1
User
1
*
CartVariant
-cart: Cart
-variant: Variant
-createdAt: Instant
-quantity: Integer
*
1
Variant

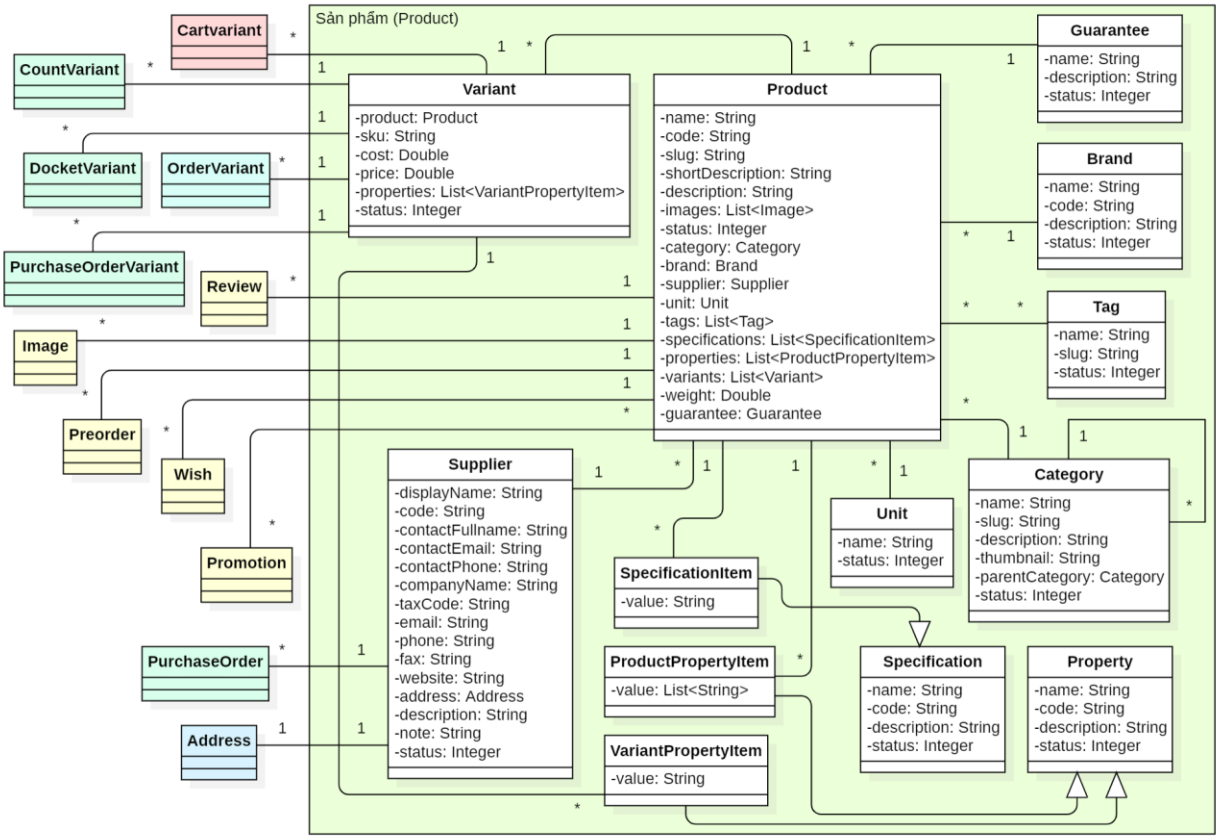

User
Khách hàng (Customer)
Customer
-customerGroup: CustomerGroup
-customerStatus: CustomerStatus
-customerResource: CustomerResource
CustomerStatus
-code: String
-name: String
-description: String
-color: String
-status: Integer
CustomerGroup
-code: String
-name: String
-description: String
-color: String
-status: Integer
CustomerResource
-code: String
-name: String
-description: String
-color: String
-status: Integer
OrderResource
*
1
*
1
*
1
1
*

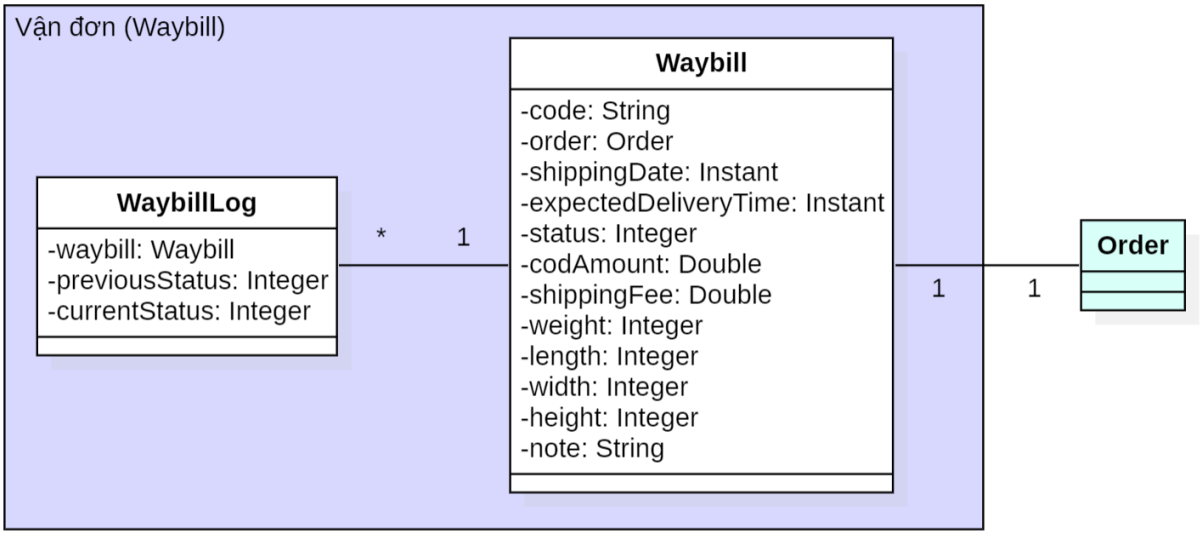

Nhân viên (Employee)
Office
-name: String
-address: Address
-status: Integer
Address
1
1
Department
-name: String
-status: Integer
1
*
1
*
Employee
-office: Office
-department: Department
-jobType: JobType
-jobLevel: JobLevel
-jobTitle: JobTitle
User
JobLevel
-name: String
-status: Integer
1
*
JobTitle
-name: String
-status: Integer
1
*
*
1
JobType
-name: String
-status: Integer


Hỗn hợp (Miscellaneous)
Review
-user: User
-product: Product
-ratingScore: Integer
-content: String
-reply: String
-status: Integer
Image
-name: String
-path: String
-contentType: String
-size: Long
-group: String
-isThumbnail: Boolean
-isEliminated: Boolean
Preorder
-user: User
-product: Product
-status: Integer
Wish
-user: User
-product: Product
Product
Notification
-user: User
-type: NotificationType
-message: String
-anchor: String
-status: Integer
«enumeration»
NotificationType
GENERAL
ERROR
WARNING
PREORDER
REVIEW
ORDER
CHECKOUT_PAYPAL_SUCCESS
CHECKOUT_PAYPAL_CANCEL
Promotion
-name: String
-startDate: Instant
-endDate: Instant
-percent: Integer
-status: Integer
User


Tồn kho (Inventory)
Warehouse
-code: String
-name: String
-address: Address
-status: Integer
Transfer
-code: String
-exportDocket: Docket
-importDocket: Docket
-note: String
Count
-code: String
-warehouse: Warehouse
-countVariants: List<CountVariant>
-note: String
-status: Integer
PurchaseOrder
-code: String
-supplier: Supplier
-purchaseOrderVariants: List<PurchaseOrderVariant>
-destination: Destination
-totalAmount: Double
-note: String
-status: Integer
-dockets: List<Docket>
Docket
-type: Integer
-code: String
-reason: DocketReason
-warehouse: Warehouse
-docketVariants: List<DocketVariant>
-purchaseOrder: PurchaseOrder
-order: Order
-note: String
-status: Integer
DocketReason
-name: String
-status: Integer
Destination
-contactFullname: String
-contactEmail: String
-contactPhone: String
-address: Address
-status: Integer
DocketVariant
-docket: Docket
-variant: Variant
-quantity: Integer
PurchaseOrderVariant
-purchaseOrder: PurchaseOrder
-variant: Variant
-cost: Double
-quantity: Integer
-amount: Double
CountVariant
-count: Count
-variant: Variant
-inventory: Integer
-actualInventory: Integer
Address
Supplier
Order
Variant


Docket
User
Waybill
1
*
1
Đơn hàng (Order)
*
1
1
Order
-code: String
-status: Integer
-toName: String
-toPhone: String
-toAddress: String
-toWardName: String
-toDistrictName: String
-toProvinceName: String
-orderResource: OrderResource
-orderCancellationReason: OrderCancellationReason
-note: String
-user: User
-orderVariants: List<OrderVariant>
-totalAmount: Double
-tax: Double
-shippingCost: Double
-totalPay: Double
-waybill: Waybill
-dockets: List<Docket>
-paymentMethodType: PaymentMethodType
-paymentStatus: Integer
OrderVariant
-order: Order
-variant: Variant
-price: Double
-quantity: Integer
-amount: Double
1
*
*
1
Variant
*
1
OrderCancellationReason
-name: String
-note: String
-status: Integer
*
1
OrderResource
-code: String
-name: String
-color: String
-customerResource: CustomerResource
-status: Integer
Sổ quỹ (Cashbook)
«enumeration»
PaymentMethodType
CASH
PAYPAL
*
1
CustomerResource


Sản phẩm (Product)
Cartvariant
CountVariant
DocketVariant
OrderVariant
PurchaseOrderVariant
Review
Image
Preorder
Wish
Promotion
PurchaseOrder
Address
Variant
-product: Product
-sku: String
-cost: Double
-price: Double
-properties: List<VariantPropertyItem>
-status: Integer
Product
-name: String
-code: String
-slug: String
-shortDescription: String
-description: String
-images: List<Image>
-status: Integer
-category: Category
-brand: Brand
-supplier: Supplier
-unit: Unit
-tags: List<Tag>
-specifications: List<SpecificationItem>
-properties: List<ProductPropertyItem>
-variants: List<Variant>
-weight: Double
-guarantee: Guarantee
Guarantee
-name: String
-description: String
-status: Integer
Brand
-name: String
-code: String
-description: String
-status: Integer
Tag
-name: String
-slug: String
-status: Integer
Supplier
-displayName: String
-code: String
-contactFullname: String
-contactEmail: String
-contactPhone: String
-companyName: String
-taxCode: String
-email: String
-phone: String
-fax: String
-website: String
-address: Address
-description: String
-note: String
-status: Integer
Unit
-name: String
-status: Integer
Category
-name: String
-slug: String
-description: String
-thumbnail: String
-parentCategory: Category
-status: Integer
SpecificationItem
-value: String
ProductPropertyItem
-value: List<String>
VariantPropertyItem
-value: String
Specification
-name: String
-code: String
-description: String
-status: Integer
Property
-name: String
-code: String
-description: String
-status: Integer


Vận đơn (Waybill)
WaybillLog
-waybill: Waybill
-previousStatus: Integer
-currentStatus: Integer
Waybill
-code: String
-order: Order
-shippingDate: Instant
-expectedDeliveryTime: Instant
-status: Integer
-codAmount: Double
-shippingFee: Double
-weight: Integer
-length: Integer
-width: Integer
-height: Integer
-note: String
Order

2.5. ERD

2.5.1. Theoretical Foundation

ERD stands for Entity Relationship Diagram. It is a type of visual diagram that describes how entities (such as customers, products, and orders) and the relationships between them are organized within a database system.

ERDs are widely used in database design to clearly visualize the logical structure of the database and the rules governing data relationships.

Main Components of an ERD

Entity:

- An object or concept within the system, usually represented by a rectangle (e.g., Customer, Product).

Attribute:

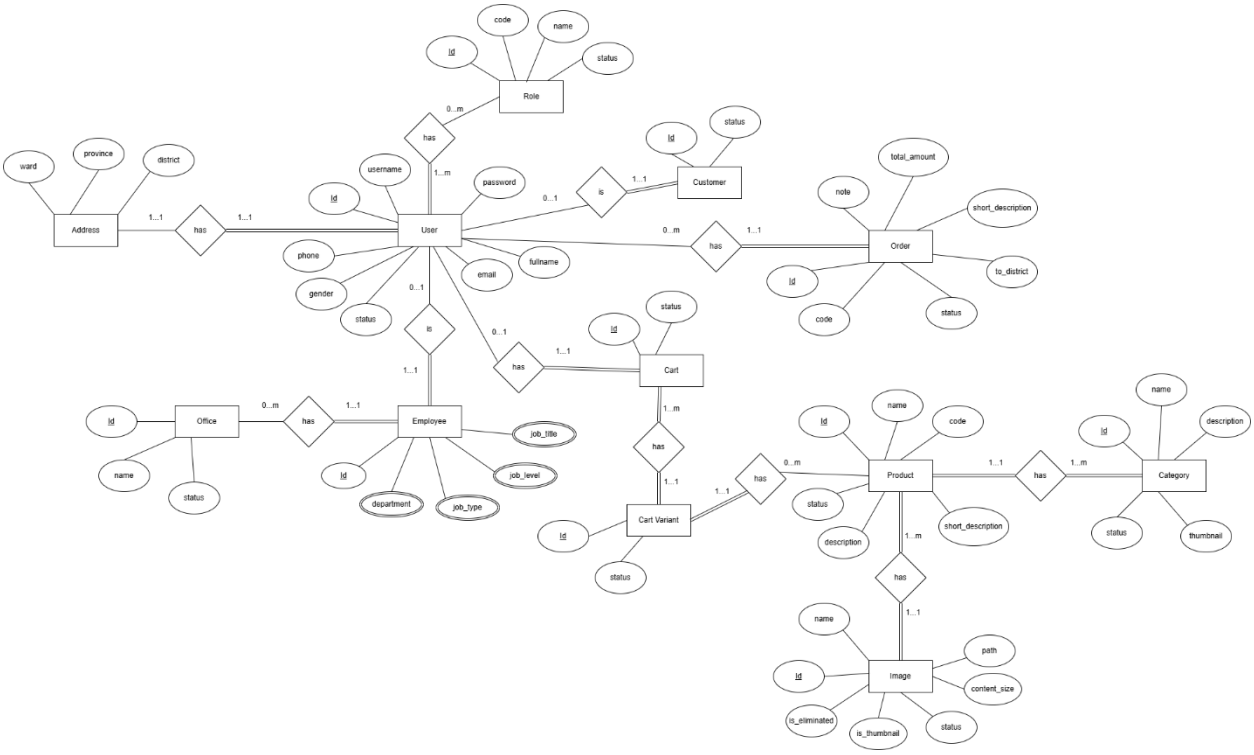
- Characteristics or descriptive information of an entity (e.g., Customer Name, Product Price).

Relationship:

- The association between two or more entities (e.g., a Customer can have multiple Orders).

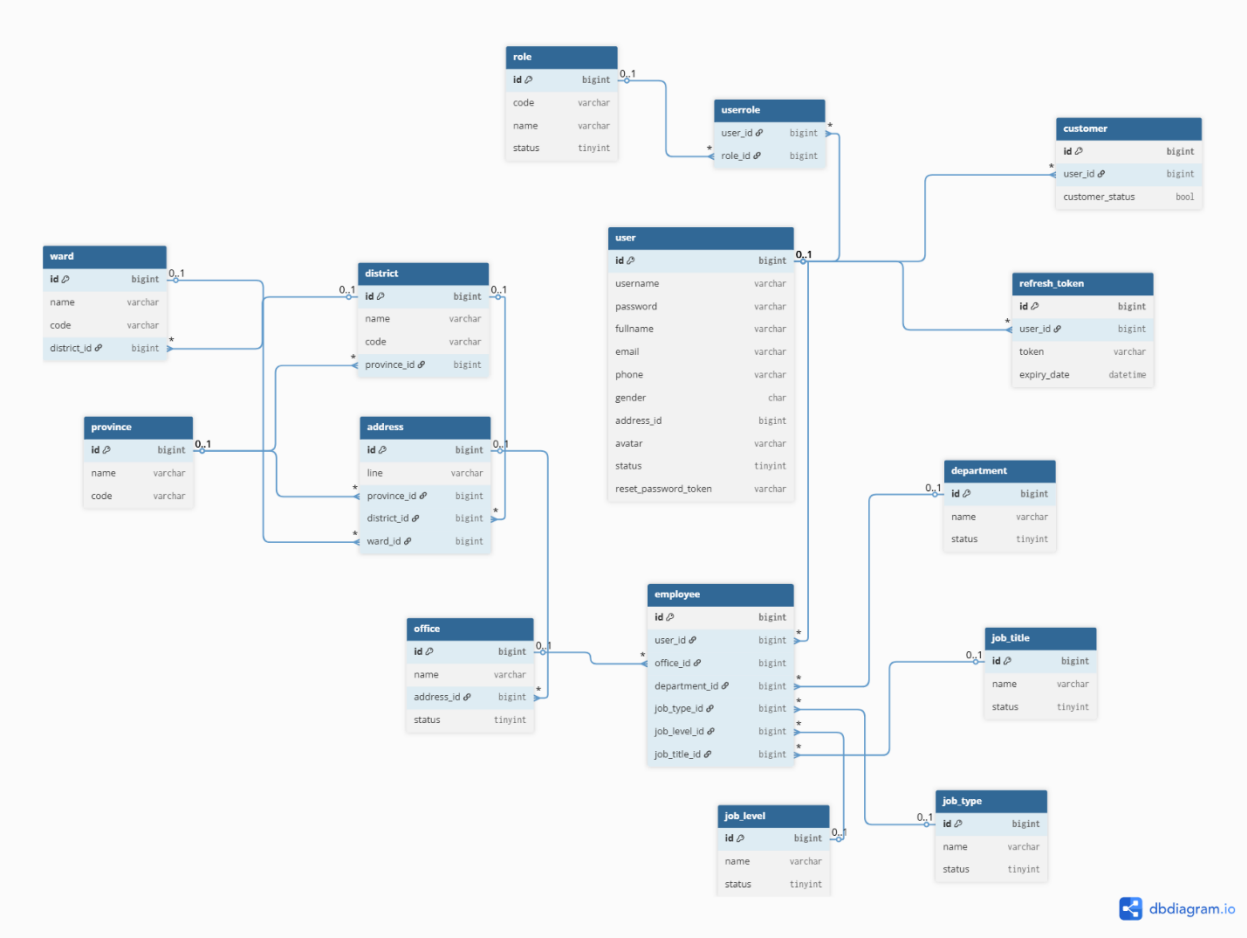
2.5.2. Design Diagram:

Conceptual:



Logical

2.5.2.1. Group relations of User – Customer – Employee:



1. Table: user

Attribute	Data Type	Description
id	bigint	Primary key identifying the user.
username	varchar	User's login name.
password	varchar	Encrypted password.
fullname	varchar	Full name of the user.
email	varchar	Registered email address.
phone	varchar	Contact phone number.
gender	char	User's gender (M/F).
address_id	bigint	Foreign key referencing the address table.
avatar	varchar	Path to the user's avatar image.
status	tinyint	Account status (0: locked, 1: active).
reset_password_token	varchar	Token used for password reset.

2. Table: userrole

Attribute	Data Type	Description
user_id	bigint	Foreign key referencing the user table.
role_id	bigint	Foreign key referencing the role table.

3. Table: refresh_token

Attribute	Data Type	Description
id	bigint	Primary key of the token.
user_id	bigint	Foreign key linking to the user (user.id).
token	varchar	Token used for access authentication.
expiry_date	datetime	Expiration date of the token.

4. Table: address

Attribute	Data Type	Description
id	bigint	Primary key of the address.
line	varchar	Detailed address (house number, street, etc.).
province_id	bigint	Foreign key to the province table.
district_id	bigint	Foreign key to the district table.
ward_id	bigint	Foreign key to the ward table.

5. Table: province

Attribute	Data Type	Description
id	bigint	Primary key of the province/city.
name	varchar	Name of the province/city.
code	varchar	Administrative code.

6. Table: district

Attribute	Data Type	Description
id	bigint	Primary key of the district.
name	varchar	Name of the district.
code	varchar	Administrative code.
province_id	bigint	Foreign key to the province table.

7. Table: ward

Attribute	Data Type	Description
id	bigint	Primary key of the ward/commune.
name	varchar	Name of the ward/commune.
code	varchar	Administrative code.
district_id	bigint	Foreign key to the district table.

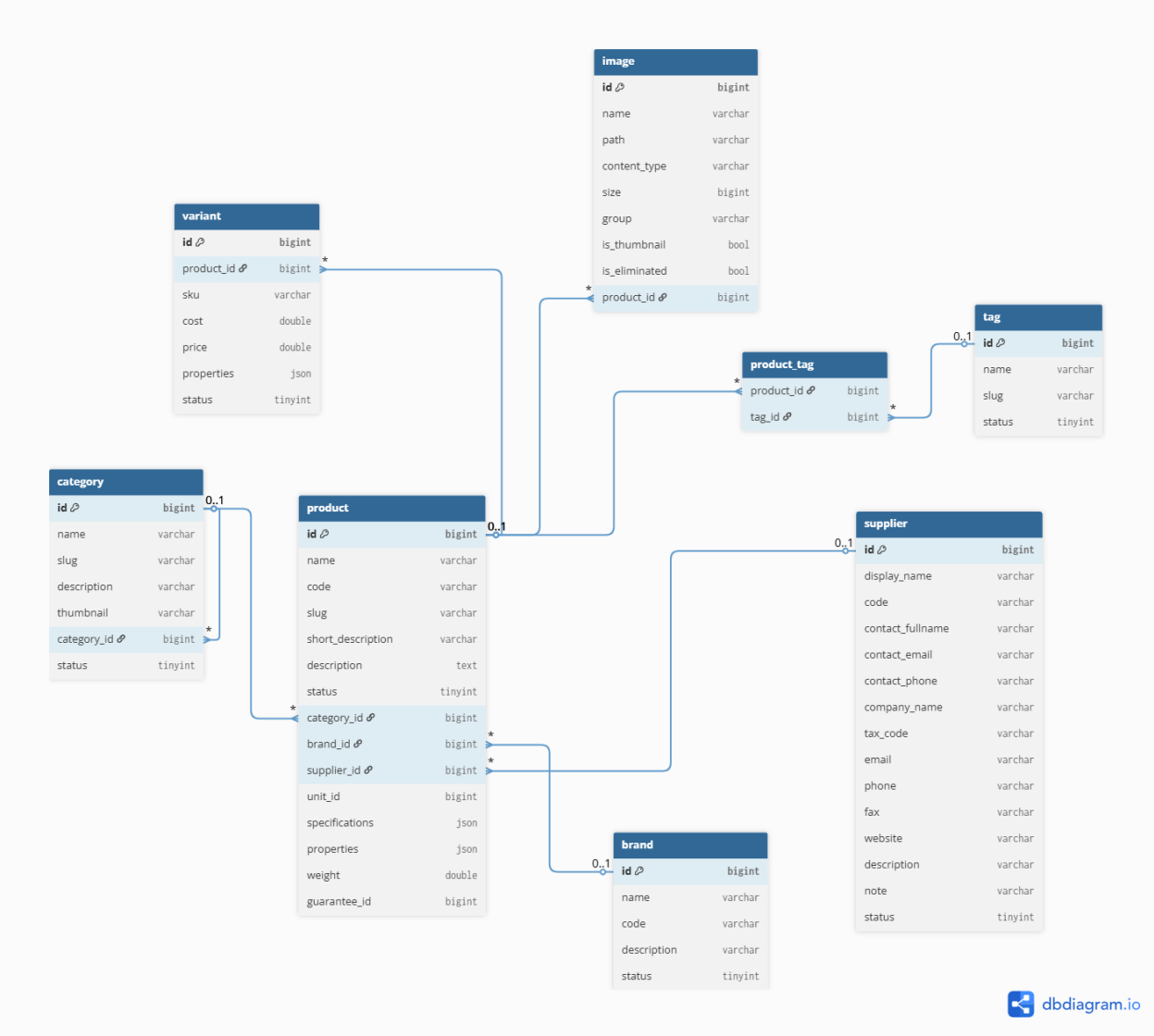
8. Table: employee

Attribute	Data Type	Description
id	bigint	Primary key of the employee.
user_id	bigint	Reference to the user table.
office_id	bigint	Reference to the office table.
department_id	bigint	Reference to the department table.
job_type_id	bigint	Reference to the job_type table.
job_level_id	bigint	Reference to the job_level table.
job_title_id	bigint	Reference to the job_title table.

9. Table: customer

Attribute	Data Type	Description
id	bigint	Primary key of the customer.
user_id	bigint	Foreign key to the user table.
customer_status	bool	Customer status (active or inactive).

2.5.2.2. Group of relations Product:



1. Table: product

Attribute	Data Type	Description
id	bigint	Primary key of the product.
name	varchar	Product name.
code	varchar	Product identifier (SKU or system code).
slug	varchar	Friendly URL path.
short_description	varchar	Short description of the product.
description	text	Detailed product description.
status	tinyint	Display status (1: active, 0: hidden).

category_id	bigint	Foreign key referencing the category table.
brand_id	bigint	Foreign key referencing the brand table.
supplier_id	bigint	Foreign key referencing the supplier table.
unit_id	bigint	Unit of measurement code (if a separate unit table exists).
specifications	json	Detailed technical specifications.
properties	json	Product attributes (color, size, etc.).
weight	double	Product weight.
guarantee_id	bigint	Foreign key to warranty policy (if available).

2. Table: category

Attribute	Data Type	Description
id	bigint	Primary key of the category.
name	varchar	Category name.
slug	varchar	Friendly URL for the category.
description	varchar	Short description of the category.
thumbnail	varchar	Path to the category thumbnail image.
category_id	bigint	Self-referencing foreign key (parent category).
status	tinyint	Display status of the category.

3. Table: brand

Attribute	Data Type	Description
id	bigint	Primary key of the brand.
name	varchar	Brand name.
code	varchar	Brand identifier code.
description	varchar	Detailed description of the brand.
status	tinyint	Display status of the brand.

4. Table: supplier

Attribute	Data Type	Description
id	bigint	Primary key of the supplier.
display_name	varchar	Display name of the supplier.

code	varchar	Supplier code.
contact_fullname	varchar	Contact person’s full name.
contact_email	varchar	Contact person’s email.
contact_phone	varchar	Contact person’s phone number.
company_name	varchar	Supplier company name.
tax_code	varchar	Company tax code.
email	varchar	Official company email.
phone	varchar	Company phone number.
fax	varchar	Fax number (if any).
website	varchar	Company website.
address_id	bigint	Foreign key referencing the address table.
description	varchar	Additional description of the supplier.
note	varchar	Internal notes.
status	tinyint	Operational status of the supplier.

5. Table: variant

Attribute	Data Type	Description
id	bigint	Primary key of the product variant.
product_id	bigint	Foreign key referencing the product table.
sku	varchar	SKU code for the variant.
cost	double	Cost price of the variant.
price	double	Selling price of the variant.
properties	json	Variant attributes (e.g., color, size).
status	tinyint	Display status.

6. Table: image

Attribute	Data Type	Description
id	bigint	Primary key of the image.
name	varchar	Image file name.
path	varchar	Storage path of the image.
content_type	varchar	Content type (image/png, image/jpeg, etc.).

size	bigint	File size (in bytes).
group	varchar	Image group (e.g., gallery, thumbnail).
is_thumbnail	bool	Whether this is a thumbnail image.
is_eliminated	bool	Whether the image has been logically deleted.
product_id	bigint	Foreign key referencing the product table.

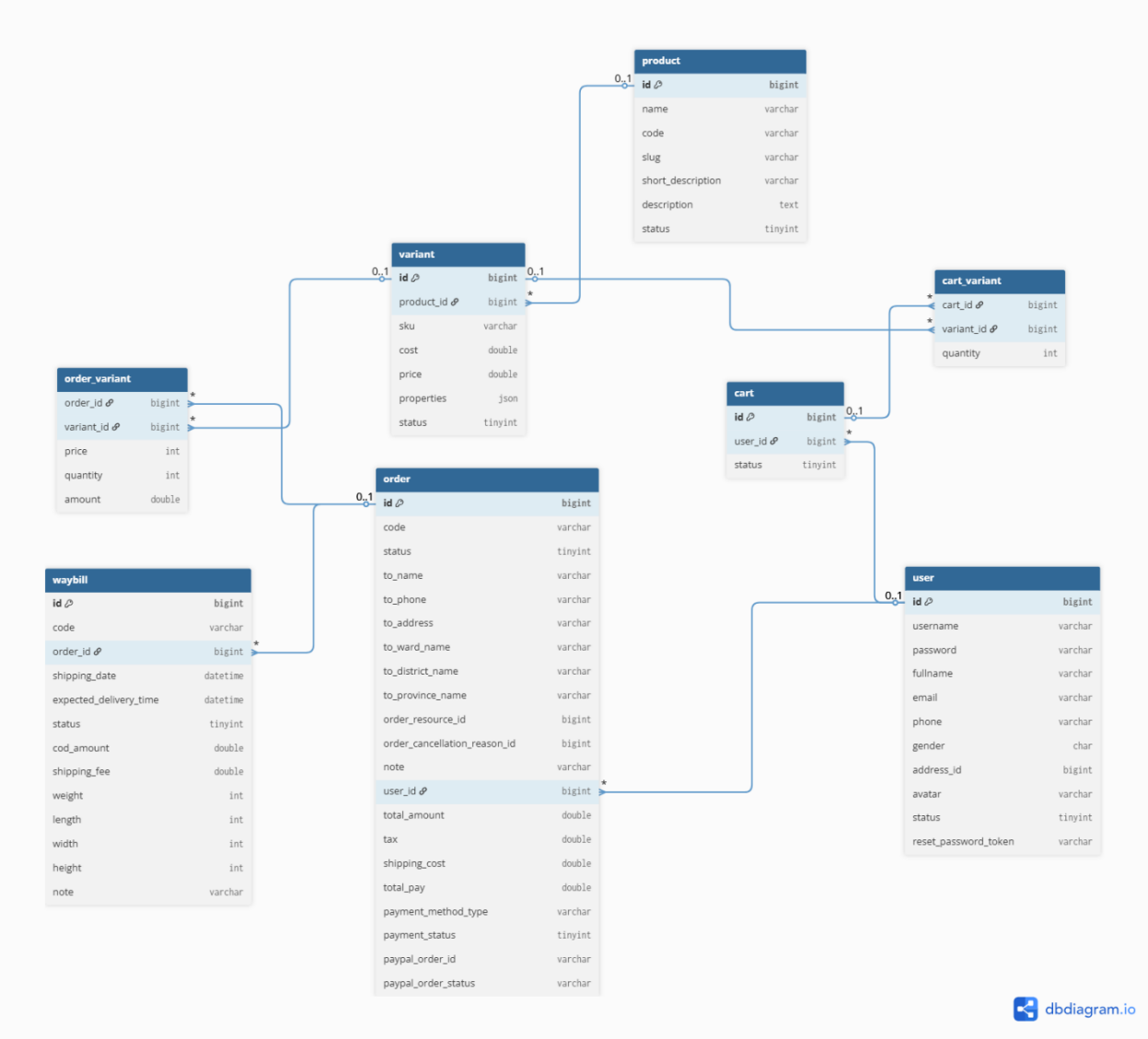
7. Table: tag

Attribute	Data Type	Description
id	bigint	Primary key of the tag.
name	varchar	Tag name.
slug	varchar	Friendly URL for the tag.
status	tinyint	Operational status of the tag.

8. Table: product_tag

Attribute	Data Type	Description
product_id	bigint	Foreign key to the product table.
tag_id	bigint	Foreign key to the tag table.

2.5.2.3. Group of relations Order/Payment/Waybill:



1. Table: order

Attribute	Data Type	Description
id	bigint	Primary key of the order.
code	varchar	Order identifier code.
status	tinyint	Order status (e.g., 0 – pending, 1 – in delivery, 2 – completed).
to_name	varchar	Recipient's name.
to_phone	varchar	Recipient's phone number.
to_address	varchar	Delivery address.
to_ward_name	varchar	Name of ward/commune for delivery.

to_district_name	varchar	Name of district for delivery.
to_province_name	varchar	Name of province/city for delivery.
order_resource_id	bigint	Order source (e.g., website, mobile app).
order_cancellation_reason_id	bigint	Reason for order cancellation (if any).
note	varchar	Notes from user or staff handling the order.
user_id	bigint	Foreign key referencing the user table.
total_amount	double	Total product value in the order.
tax	double	Tax applied to the order.
shipping_cost	double	Shipping fee.
total_pay	double	Total amount to be paid.
payment_method_type	varchar	Payment method (PayPal, COD, etc.).
payment_status	tinyint	Payment status (0 – unpaid, 1 – paid).
paypal_order_id	varchar	PayPal order ID (if paid via PayPal).
paypal_order_status	varchar	PayPal order status.

2. Table: order_variant

Attribute	Data Type	Description
order_id	bigint	Foreign key referencing the order table.
variant_id	bigint	Foreign key referencing the variant table.
price	int	Price of the variant at the time of order.
quantity	int	Quantity of the product ordered.
amount	double	Total amount (price × quantity).

3. Table: waybill

Attribute	Data Type	Description
id	bigint	Primary key of the waybill.
code	varchar	Waybill code (issued by system or delivery unit).
order_id	bigint	Foreign key referencing the order table.
shipping_date	datetime	Scheduled shipping date.
expected_delivery_time	datetime	Expected delivery time.
status	tinyint	Shipping status (e.g., in transit, delivered, returned).

cod_amount	double	Cash on delivery amount (if COD).
shipping_fee	double	Delivery fee.
weight	int	Package weight.
length	int	Package length.
width	int	Package width.
height	int	Package height.
note	varchar	Notes for the delivery service.

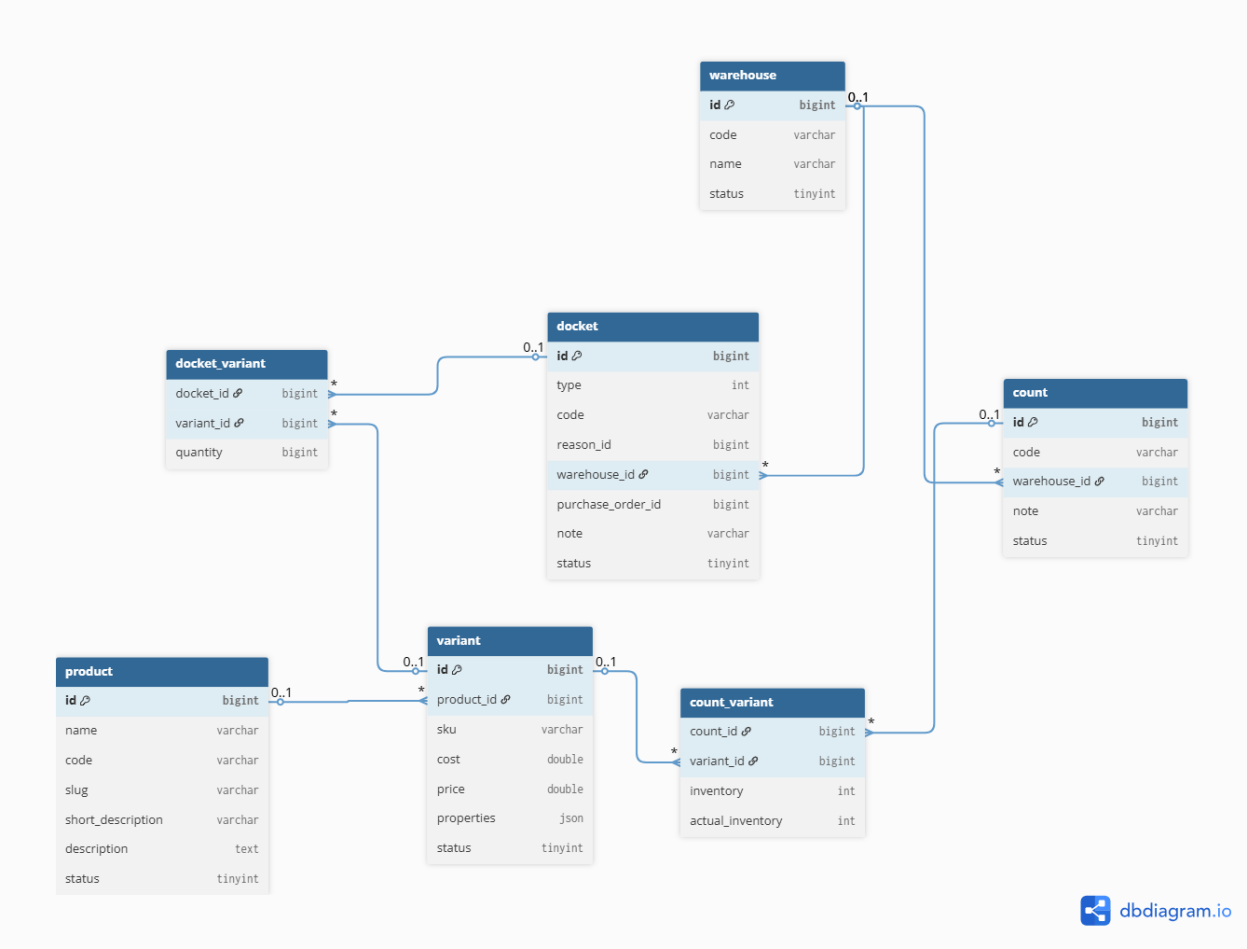
4. Table: cart

Attribute	Data Type	Description
id	bigint	Primary key of the cart.
user_id	bigint	Foreign key referencing the user table.
status	tinyint	Cart status (e.g., 0 – active, 1 – checked out).

5. Table: cart_variant

Attribute	Data Type	Description
cart_id	bigint	Foreign key referencing the cart table.
variant_id	bigint	Foreign key referencing the variant table.
quantity	int	Quantity of the product in the cart.

2.5.2.4. Group of relations Inventory (tồn kho):



1. Table: warehouse (kho hàng)

Attribute	Data Type	Description
id	bigint	Primary key. Unique identifier for the warehouse.
code	varchar	Internal code of the warehouse.
name	varchar	Name of the warehouse.
status	tinyint	Status of the warehouse.

2. Table: docket (phiếu nhập/xuất kho)

Attribute	Data Type	Description
id	bigint	Primary key. Unique identifier for the docket (stock in/out/transfer note).
type	int	Type of docket (1 – Import, 2 – Export).
code	varchar	Docket code/number.

warehouse_id	bigint	Foreign key referencing warehouse.id. Warehouse affected by this docket.
purchase_order_id	bigint	Linked to purchase order ID (if import docket), can be a foreign key.
note	varchar	Notes for the docket.
status	tinyint	Status of the docket.

3. Table: docket_variant (chi tiết phiếu nhập/xuất)

Attribute	**Data Type**	**Description**
docket_id	bigint	Primary key and foreign key referencing docket.id. Links to a specific docket.
variant_id	bigint	Primary key and foreign key referencing variant.id. Product variant in the docket.
quantity	bigint	Quantity of the product variant in this docket.

Chapter 3: Test Plan

3.1. Theoretical Background of Software Testing

3.1.1. Concept and Objectives of Software Testing

Concept

Software testing is the process of executing a program or system with the intent of finding errors (bugs/defects). It is a mandatory activity in the Software Development Life Cycle (SDLC) to assess product quality by checking and verifying that the software meets the specified requirements.

Testing is a process that includes not only executing the program but also activities such as planning, analysis, design, implementation, and result reporting.

Objectives

- The core objective of testing is not only to find defects but also to:
- Verification: Ensure that the software is built according to the technical requirements and initial design (building the product right).
- Validation: Ensure that the software meets real user needs and solves business problems (building the right product).
- Quality Assessment: Provide objective information about product quality and risk levels to stakeholders.
- Defect Prevention: Through early testing (Shift-Left Testing) and defect analysis, help improve the development process and prevent similar defects from recurring.

3.1.2. Concept of a Test Plan

A Test Plan is a document that describes the objectives, scope, strategy, approach, resources, and schedule of the upcoming testing activities. It is a living document, approved by stakeholders, used to guide and manage the overall testing effort of the project.

Main Components of a Test Plan

A standard Test Plan (based on IEEE 829) typically includes the following sections:

- Introduction: Purpose, system overview, definitions, and terminology.
- Test Scope: Features to be tested and not to be tested.
- Test Strategy: Testing approach, levels, and types of testing to be applied.
- Test Environment: Hardware, software, tools, and test data.
- Resources & Schedule: Staffing allocation, effort estimation, and key milestones.
- Deliverables: Testing artifacts to be produced (test cases, defect logs, test reports).

3.2. Overview of the Electro Project Test Plan

3.2.1. Purpose and Scope

Purpose

This test plan aims to:

- Identify the software components to be tested for the ShopEase e-commerce web application (Backend: Spring Boot, Frontend: React).
- Propose and describe specific testing strategies (functional and non-functional) to ensure product quality.
- Provide a framework for effort estimation, risk management, and test deliverables throughout the testing process.

Main Test Scope (Features to Be Tested)

The software will be comprehensively tested for the following features:

Category	**Functional**
Authentication & Acess control	Registration, Login (Buyer/SysAdmin), Logout, Access control (JWT token), User status management (Admin).
Product Catalog	View, filter, and search products (Buyer); add, edit, delete, and manage product categories (SysAdmin).
Cart & Checkout & Payment	Add/Remove/Update items in cart, calculate total cost, place orders (COD), track order status (Buyer); process/update/cancel orders (SysAdmin).
Inventory Management	View inventory details; create/view/edit/delete stock in/out records; view import–export history; manage purchase orders (SysAdmin).
Data Initialization	Verify initial creation of SysAdmin accounts, sample buyers, sample products, and sample orders.

Non-Functional Test Scope (Non-Functional)

Category	**Description**
Usability	User session management, responsive React UI, intuitive navigation.
Compatibility	Testing on major browsers (Chrome, Firefox, Edge, Safari) and operating systems (Windows 11/macOS).
Design Constraints	Data length limits, UTF-8 character encoding support.
Interfaces	Correct UI rendering and proper API endpoint interaction between Frontend and Backend.

Not in scope:

- Integration with real third-party payment gateways (payment will be simulated only).

- Testing of a standalone mobile application (focus is on web responsiveness only).
- Extreme stress testing and crash/recovery testing.

3.2.2. Test Strategy

The test strategy is designed based on the Agile (Scrum) model and the Continuous Testing approach, ensuring that quality is built into every stage of development (Shift-Left Testing) and continuously maintained through the CI/CD pipeline.

A. Test Stages

This strategy follows a feature-driven and Sprint-based testing model aligned with the project development plan.

Stage	**Objective / Mapping from Sprint Backlog**	**Responsible Team**	**Frequency**
Unit Test (UT)	Verify business logic at function/method level (Services, Controllers, Components). Ensure that Sprint Backlog tasks are correctly implemented.	Development Team	Automatically executed on every commit and when merging into the main branch.
Integration Test (IT)	Ensure correct interaction between modules (React ↔ Spring Boot APIs) and with the database (Spring Boot ↔ MySQL).	Development/Testing Team	Automatically executed after Unit Tests and before the build stage.
System Test (ST)	End-to-end testing of the whole system for selected User Stories in the Sprint. Ensure all business flows of the Sprint work as expected.	Testing Team	Automatically executed after a new build is deployed to the Staging environment, simulating real end-user flows.
Regression Test (RT)	Re-validate all functionalities completed in previous Sprints (e.g., Auth, Catalog) after new Sprint code is merged.	Testing Team	The full automated test suite (UT, IT, E2E) from previous Sprints is re-executed to protect existing features.
Acceptance Test (UAT)	Confirm that the Product Increment meets the Definition of Done (DoD) and the needs of SysAdmin/Buyer representatives.	Đại diện Người dùng	Performed at the end of each Sprint in the Staging environment.

B. Test Automation Strategy in CI/CD

This strategy focuses on automating tests to support the project's Continuous Integration (CI) and Continuous Delivery (CD) process:

1. Unit & Integration Test Automation (CI Focus):

- Objective: Ensure the stability of newly committed code.
- Trigger: Automatically executed when developers push code and when merging into the main branch.
- Action: If these tests fail, the CI pipeline is broken, preventing faulty builds from being generated.

2. System & Regression Test Automation (CD Focus):

- Objective: Protect existing functionalities (regression) and validate the correctness of the new build.
- Trigger: Automatically executed after the new build is successfully deployed to the Staging environment.

3. Performance Testing (Load) - Periodic:

- Objective: Evaluate API responsiveness and load-handling capability under moderate load (50–100 concurrent users).
- Frequency: Performed at the end of major Sprints (Sprint 3 and 4), after critical modules (Order, Inventory) are completed.

C. Testing Types:

Testing Type	**Primary Objective (per Product Backlog)**	**Applied Techniques**
Functional Testing	Verify each function according to User Stories (e.g., AUTH-01, CAT-02, INV-03).	Apply **Equivalence Partitioning** and **Boundary Value Analysis** for input fields (product quantity, inventory values, etc.).
Business Cycle Testing	Ensure complete business workflows (e.g., Place Order (ORDER-05) → Process Order (ORDER-09) → Dispatch/Stock-out (INV-03)).	Use core business flows (High-priority User Stories) as End-to-End test scenarios.
Data Integrity Testing	Ensure inventory data is correctly synchronized across tables such as docket_variant, variant, and count_variant.	Validate data constraints and post-transaction integrity (e.g., inventory never becomes negative after stock-out).
Performance Testing	Measure performance of critical APIs (e.g., Inventory list API INV-01, Order API ORDER-05).	Use load testing tools (JMeter/k6) to measure **Response Time** and **Error Rate**.

Security / Access Control Testing	Ensure access control rules (AUTH-05, AUTH-08) are strictly enforced by roles (Buyer vs. SysAdmin).	Attempt to access Admin resources with Buyer accounts and vice versa.

3.2.3. Test Environment and Resources

A. Test Environment)

The test environment is configured to closely simulate the Production environment and includes the following components:

Component	Technical Details
Web Server	Embedded Tomcat (version 11.0.13)
Backend	Spring Boot (3.5.6) running on Windows 11
Frontend	ReactJS (19) with Mantine UI
Database	MySQL (8.0.44)
Browsers	Chrome, Firefox (execution); Edge, Safari (certification)
Minimum Hardware	Intel Core i5 CPU, 4 GB RAM, 40 GB HDD (for Web server)

B. Human Resource

The testing team consists of members with clearly defined roles and responsibilities:

- **Trang Gia Huy (Tester):** Leads the testing team; responsible for test planning, designing and executing test cases for core functionalities (Authentication, Product Catalog).
- **Nguyễn Lê Quỳnh Hương (Tester):** Responsible for designing and executing test cases for performance and compatibility testing, and supporting Unit Testing for backend modules.

C. Test Milestones

The milestones are directly aligned with the Sprint schedule in the Project Plan, emphasizing iterative and incremental testing.

Activity	Project Phase	Planned Start	Planned End
I. DESIGN & PLANNING PHASE (PRE-SPRINT & SPRINT 1)	Initiation / Analysis		
Develop Master Test Plan	Project Initiation	2025-10-01	2025-10-06
Review & Update Test Plan (TP)	Project Initiation	2025-10-07	2025-10-08
Develop Unit Test Plan	Pre–Sprint 1	2025-10-09	2025-10-10
Review & Update Unit Test Plan	Pre–Sprint 1	2025-10-11	2025-10-11
Design Unit Test Cases (UTC)	Sprint 1 (Catalog)	2025-10-12	2025-10-17

Review & Update Unit Test Cases (UTC)	Sprint 1 (Catalog)	2025-10-18	2025-10-19
II. SPRINT-BASED TEST EXECUTION			
Execute UT, IT, ST & Defect Reporting (Catalog features)	End of Sprint 1	2025-10-20	2025-10-24
User Acceptance Testing (UAT) for Sprint 1	Post Sprint 1	2025-10-25	2025-10-27
Design Integration Test Cases (ITC) (Auth, Order)	Start of Sprint 2	2025-11-04	2025-11-08
Review & Update Integration Test Cases (ITC)	Start of Sprint 2	2025-11-09	2025-11-10
Design System Test Cases (STC) (Order, Inventory)	Start of Sprint 2	2025-11-11	2025-11-15
Review & Update System Test Cases (STC)	Start of Sprint 2	2025-11-16	2025-11-17
Execute UT, IT, ST, RT & Defect Reporting (S2: Auth, S3: Cart/Order)	End of Sprint 2	2025-11-28	2025-12-02
UAT for Sprints 2 & 3	Post Sprint 2	2025-12-04	2025-12-08
Execute UT, IT, ST, RT & Defect Reporting (Sprint 4: Inventory)	Following S2 UAT	2025-12-09	2025-12-13
Final User Acceptance Testing (Final UAT)	End of Project	2025-12-14	2025-12-16
Production Deployment & Post-Deployment Testing (PDT)	After UAT	2025-12-17	2025-12-18

D. Deliverables

The main deliverables to be provided after the completion of testing include:

- Test Plan.
- Manual Test Case Suite: Unit Test Cases, Integration Test Cases, and System Test Cases.
- Automated Test Suite: Source code (scripts) for automated API and E2E tests, stored together with the project source code repository.
- Detailed Defect Log.
- Test Reports:
    - Test Summary Report (per Sprint): Reports on the results of manual and automated testing for the User Stories in each Sprint.

- Final Test Report (end of project): A comprehensive quality report and defect detection rate.

CHAPTER 4. TEST DESIGN

4.1. Introduction

This chapter presents the process and results of designing detailed Test Cases based on the V-Model strategy, using the documents from Chapters 1, 2, and 3.
The objectives of this chapter are to ensure:

- Every business requirement is mapped to specific test objects.
- Main and exceptional business flows are fully covered.
- Testing activities can be traced backward from Test Case → Workflow → Requirement.

4.2. Test Design Process Using V-Model

The test design process of the team is entirely based on the V-Model – a model emphasizing symmetry and mutual correspondence between development and testing activities.

In the V-shaped structure, each analysis and design phase on the left branch has a corresponding testing activity on the right branch to ensure every requirement can be verified through a specific test case.

4.2.1. Requirement Analysis – 1a

At the initial phase, the team collects, analyzes, and clarifies the business requirements of the online sales system.
The objectives of this step are to ensure that all requirements are:

- Fully described
- Within scope
- Testable

The outcome of this phase is the SRS document, which serves as the basis for all subsequent test design activities.

From the requirement analysis, the team identifies the main system actors:

- Customer
- Administrator
- Warehouse Staff
- Supplier
- Delivery Service

The system is decomposed into the main functional modules:

- Authentication & Authorization
- Product Catalog
- Cart – Order – Payment

- Order Management
- Inventory Management

Business requirements at this stage are transformed into Test Scenarios and acceptance criteria, ensuring each requirement has at least one corresponding test scenario.

4.2.2. System Design – 2a

Once requirements are clearly defined, the team proceeds with high-level system design. The design includes:

- Use Case diagrams
- Main and exceptional business flows
- Relationships between actors and the system
- From Use Case diagrams and business processes, the team:
- Builds Business Process Steps
- Maps each business step to a Scenario ID
- Designs Test Workflows for each important business flow

Test Workflows are designed for:

- Customer purchase process
- Order processing process
- Inventory management process
- Exceptional flows such as out-of-stock, order cancellation, failed delivery

4.2.3. Architecture Design – 3a

During the architecture design phase, the team applies the C4 Model (Context – Container – Component – Code) to decompose the system from high-level to component-level.

Architecture design helps to:

- Clearly define boundaries between frontend, backend, and database
- Identify interaction points between modules
- Provide a basis for integration test design

From this design, integration test points are clearly identified, for example:

- Controller ↔ Service
- Service ↔ Repository
- Backend ↔ Database

4.2.4. Module Design – 4a

In the module design step, the team analyzes each functional module in detail using:

- Class Diagrams

- Relationships between classes
- Key attributes and methods

The outcome of this step serves as the basis for developing Unit Tests for key classes and methods in each module.

4.2.5. Unit Testing – 1b

Test Case ID	Unit Under Test	Test Description	Test Data	Expected result	Result
UT-CTL-AUTH-001	POST /api/auth/login	Đăng nhập thành công	Body: { username: "jasmine", password: "123123123" } Mock: authenticationManager.authenticate("jasmine","123123123") → Authentication	HTTP 200 OK JSON trả jwt + refreshToken	Passed
UT-CTL-AUTH-002	POST /api/auth/login	Đăng nhập thất bại do sai mật khẩu	Body: {username: "jasmine", password: "12345"} Mock: authenticationManager.authenticate("jasmine","123123123") → authenticate throw exception	HTTP 401 Unauthorized	Passed
UT-CTL-AUTH-003	POST /api/auth/refresh-token	Refresh token thành công	Body: refreshToken hợp lệ Body: { refreshToken: "refresh-xyz" } Mock: refreshTokenService.findByToken("refresh-xyz") → RefreshToken	HTTP 200 OK JWT mới được trả về	Passed
UT-CTL-AUTH-004	POST /api/auth/refresh-token	Refresh token hết hạn	Body: refreshToken expired Body: { refreshToken: "refresh-expired" } Mock: refreshTokenService.findByToken("refresh-expired") → Optional.empty()	HTTP 403 Forbidden	Passed

4.2.6. Integration Testing – 2b

After modules are stabilized through Unit Testing, the team performs Integration Testing to evaluate interactions between system components.
The team uses:

- Spring Boot Test
- MockMvc
- H2 Database (in-memory)

Integration test cases focus on:

- API processing flows
- Data transfer between layers
- Data consistency between backend and database

Example integration tests in the project:

Integration Test Case	Test Method
IT_PROD_01	testGetAllProducts_Pagination()
IT_PROD_02	testSearchProducts_ByKeyword()
IT_PROD_03	testFilterProducts_ByCategory()
IT_PROD_04	testGetNewestProducts()
IT_PROD_05	testGetProductDetail_Success()
IT_PROD_06	testGetProductDetail_NotFound()
IT_PROD_07	testGetRelatedProducts()
IT_CAT_01	testGetAllCategories_Client()
IT_CAT_02	testGetCategoryBySlug_Success()

4.2.7. System Testing – 3b

At a higher level, System Testing evaluates the software as a whole. It is conducted manually based on Test Cases created during test design, covering all use cases and main user business flows.

System testing activities include:

- Functional testing: verifying end-to-end business flows such as login, purchase, payment
- Interface testing: ensuring a consistent, user-friendly interface
- Basic security testing: verifying login flows, authorization, and access control

System testing results reflect the overall software quality before acceptance testing.

4.2.8. Acceptance Testing – 4b

Acceptance Testing is the final phase of testing, evaluating the system from the end-user perspective.
At this step, the team compares acceptance criteria with the SRS to ensure the system meets customer requirements.

After internal system testing, the software is delivered to the Product Owner for Acceptance Testing. The Product Owner acts as the end user, simulates real scenarios, and assesses usability, stability, and business suitability.

Acceptance testing results serve as a key basis for the final decision on deploying the system into the production environment.

4.3. Test Design Techniques

4.3.1. Black-box Testing

Black-box testing is used to verify system functionality based on business requirements without considering internal software structure.

In the project, this method is applied for System Testing and Acceptance Testing to ensure software behaves as expected for end users and meets acceptance criteria.

Equivalence partitioning:
In the Access Control test case, input data is divided into valid and invalid classes based on account status and user information validity.

Login Function:
Valid class:

- Existing username
- Correct password
- Active account

Invalid class:

- Wrong password
- Locked account
- Non-existent username

Registration Function:
Valid class:

- Username not existing
- Correct email format
- All required fields entered

Invalid class:

- Email already exists
- Required fields missing
- Invalid email/phone format

Mapping to actual test cases:

- TC_AC_LOGIN_VALID_*
- TC_AC_LOGIN_INVALID_*
- TC_AC_REGISTER_VALID_*
- TC_AC_REGISTER_INVALID_*

State Transition Technique:
In the Access Control module, key states include:

- User account state: normal or locked
- Session state: logged in or logged out

Test cases check:

- Transition from logged-in → logged-out
- Account state transitions: normal → locked → unlocked
- System behavior after state changes

Example Test Cases:

- TC_AC_007: successful logout
- TC_AC_017: lock user account
- TC_AC_018: login with locked account
- TC_AC_019: unlock user account

Decision Table:

Step 1: Define conditions

Symbol	Condition	Values
C1	Valid login information	Y/N
C2	Account locked	Y/N
C3	User role	Customer/Admin
C4	Accessed page	User/Admin

Step 2: Define actions

Symbol	Action
A1	Allow login
A2	Allow page access
A3	Show error message

Step 3: Number of rules

2 x 2 x 3 x 2 = 24 rules

Step 4: Full decision table *(partial view)*

Rule	R1	R2	R3	R4
C1: Valid login	N	Y	Y	Y
C2: Account locked	DC	Y	N	N
C3: Role	DC	DC	Customer	Admin
C4: Page	DC	DC	User	Admin

Step 5: Map Rule → Test Case

Rule	Test Case
R1	TC_AC_LOGIN_INVALID
R2	TC_AC_LOGIN_DISABLED

R3	TC_AC_ACCESS_DENIED
R4	TC_AC_ACCESS_ALLOWED

4.3.2. White-box Testing

White-box testing is used to verify internal code logic, data flows, condition branches, and loops, helping detect errors at the module or method level.

In the project, this method is applied for Unit Testing and Integration Testing to ensure components operate correctly and interact properly.

Example: deleteAddress function analysis

Node	Description
1	Check if address exists
2	Throw exception
3	Perform deletion
4	End

Edge	Description
1 → 2	Does not exist
1 → 3	Exists
2 → End	Ends with error
3 → 4	Ends successfully

4.4. Test Design Methods

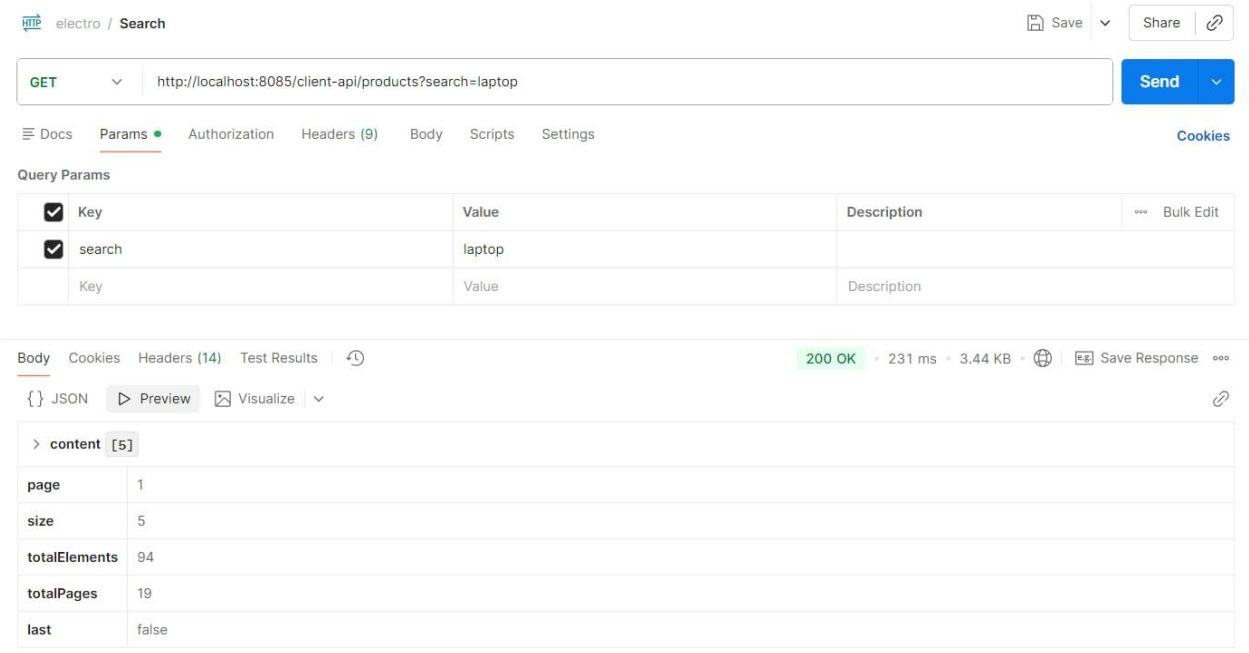
4.4.1. Manual Testing

Manual testing is applied for Acceptance Testing to evaluate the system from the end-user perspective. Testers verify main business flows and confirm system compliance with requirements.

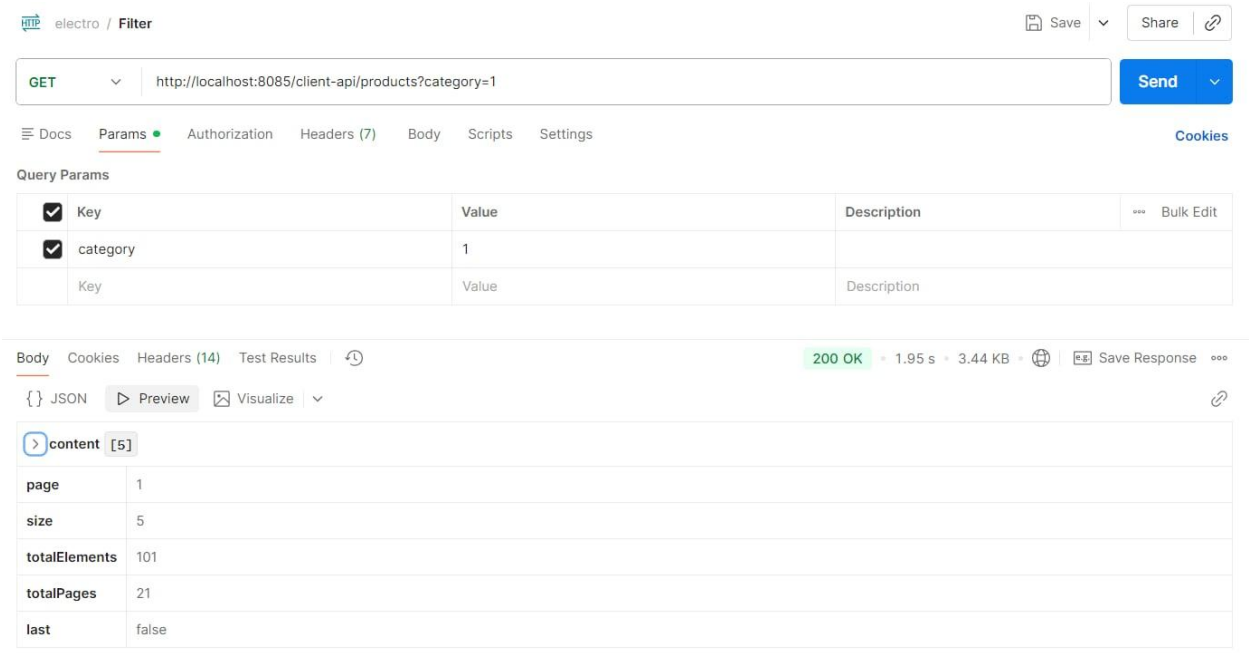
Manual test cases are executed based on Test Cases and Review Checklists, and results are compared with expected outcomes before system acceptance.

Postman is used for API testing:

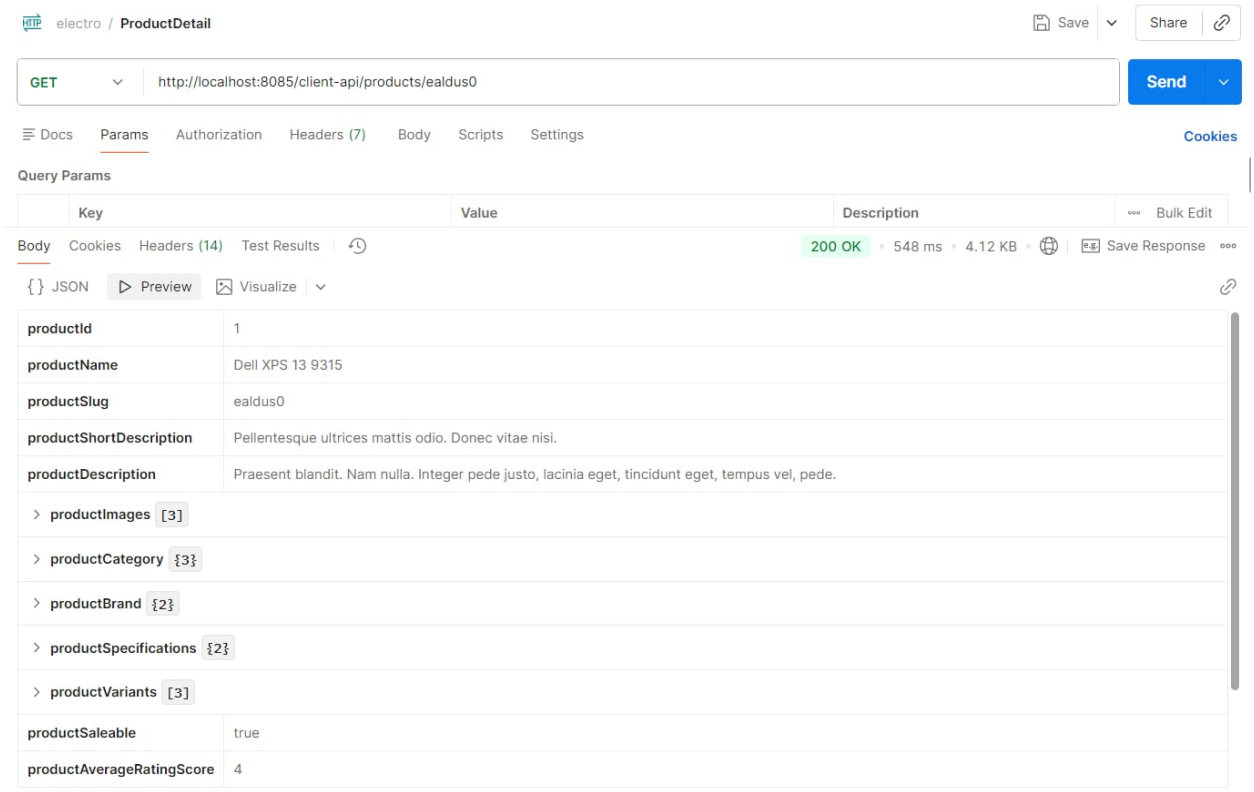
API_PROD_02



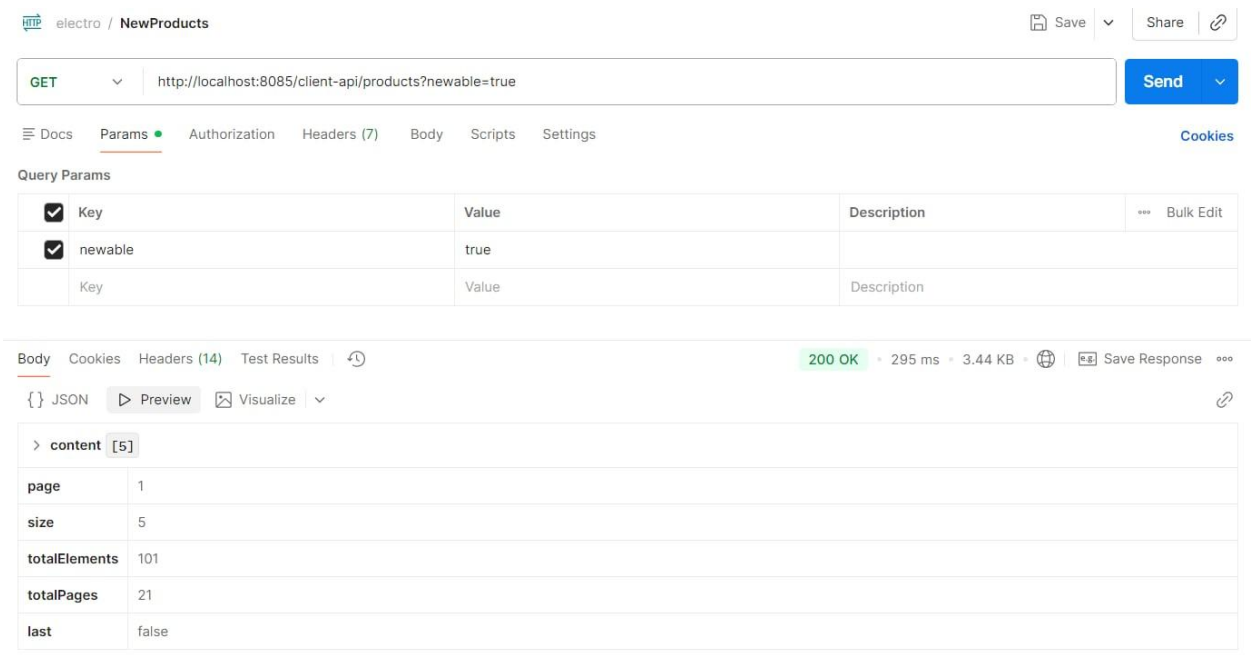
API_PROD_03



API_PROD_07



API_PROD_05



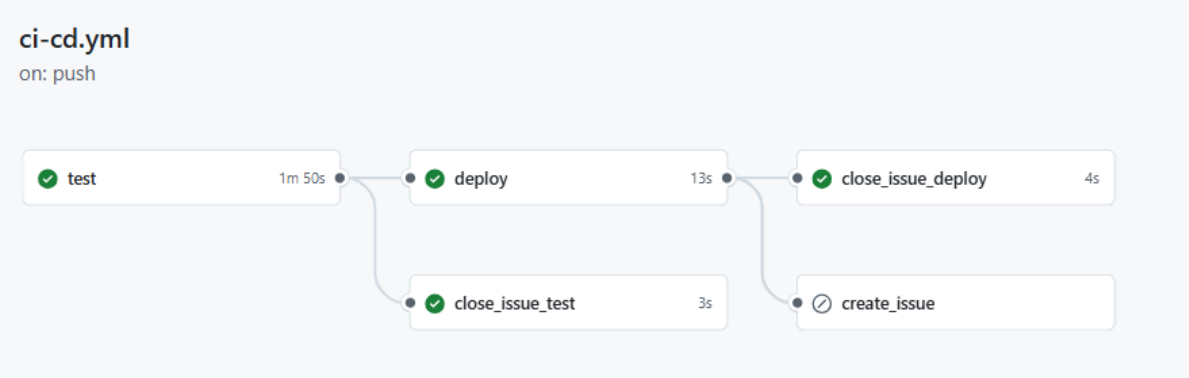
4.4.2. Automated Testing

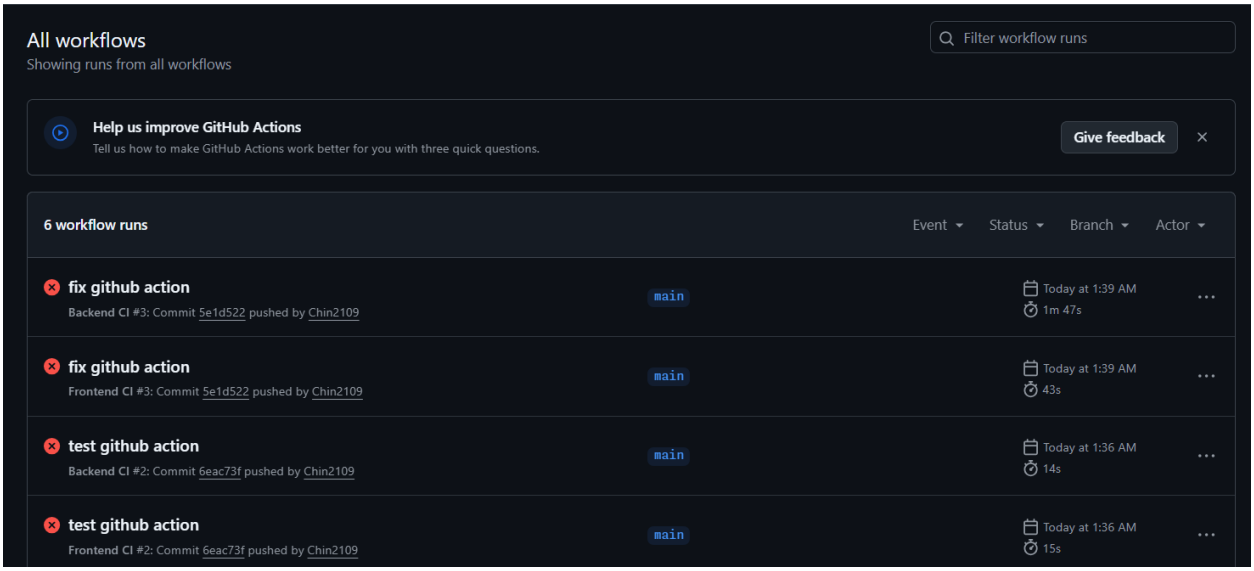
Automated testing is applied through CI/CD pipelines using GitHub Actions to automate build, testing, and deployment. Automated tests are used for White-box tests, including Unit Testing, Integration Testing, and System Testing.

Whenever source code is pushed to GitHub, the CI/CD pipeline triggers a test job that builds the project and runs all automated tests.

- If the test job fails, an Issue is automatically created on GitHub to track the problem.
- If the test job passes, related Issues are automatically closed.

When code is merged into the main branch, the CI/CD pipeline triggers the deployment job to call the Render platform and redeploy the system.





CHAPTER 5. TEST REPORT

5.1. Overview of the Testing Process

After completing the test design phase in Chapter 4, the team conducted testing for the Electro E-Commerce system according to the planned schedule. The testing process was carried out over 6 weeks and included the following stages: Unit Testing, Integration Testing, System Testing, and Acceptance Testing.

The team focused on testing all main business functions of the system. Test Cases were executed carefully and in detail, with all results fully documented in the handover documents.

5.2. Test Case Report

5.2.1. Introduction

These test cases are categorized into two groups:

- **Positive**: verifying system behavior as expected.
- **Negative**: checking the system's handling of errors, exceptions, and invalid conditions.

The test suite was built for the system's 6 main functional modules, including Access Control, Products, Cart, Orders, Payment, and Address Book, totaling **72 Test Cases**.

Module	Code	Number of Test Cases	Description	Result
Authentication & Authorization	AC	23	Registration, login, logout, profile update, Admin/Buyer authorization	20 Pass, 3 Fail
Product & Catalog Management	PR	19	View, filter, search, add/edit/delete products and categories	19 Pass
Cart & Order Management	OR	18	Cart operations, checkout, order tracking and management	18 Pass
Inventory Management	INV	20	Stock in/out, inventory history, purchase orders	20 Pass
Total	–	80	Covers all main system functions	77 Pass, 3 Fail

5.2.2. Coverage Scope

The team developed **80 Test Cases**, covering **100% of the system's main functional requirements**.

Covered functional groups include:

- User authentication and authorization
- User management
- Product and catalog management

- Cart and order processing
- Inventory management, stock in/out, and purchase orders

Non-functional requirements such as:

- System performance
- Load capacity
- Advanced security

were **not tested** within the scope of this project.

Coverage ratio:

- Functional: 100%
- Non-functional: 0%

5.2.3. Execution Results

Detailed statistics:

- Total Test Cases: 80
- Passed: 77
- Failed: 3
- Blocked: 0
- Not run: 0

Percentage results:

- 96.25% Test Cases passed
- 3.75% Test Cases failed
- 0% Test Cases blocked
- 0% Test Cases not run