HO CHI MINH CITY PEOPLE’S COMMITTEE

**SAIGON UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**

A blue button with white text

AI-generated content may be incorrect.

**SOFTWARE TESTING PROJECT REPORT**

**TESTING THE SHOESHOP E-COMMERCE WEBSITE**

Members:

Đỗ Phú Thành - 3122411189

Huỳnh Duy Khang – 3122411088

Võ Thành Danh – 3122411024

Huỳnh Minh Quân – 3122411167

Supervisor: Dr. Đỗ Như Tài

**HO CHI MINH CITY, 2025 – 2026**

# **Table of Contens**

[**Table of Contens** 1](#_Toc217145192)

[**CHAPTER 1 – INTRODUCTION** 4](#_Toc217145193)

[**Chapter Overview** 4](#_Toc217145194)

[**1.1. Topic Overvie** 4](#_Toc217145195)

[1.1.1. Background and Rationale for Topic Selection 4](#_Toc217145196)

[1.1.2. Objectives of the Shoeshop System 5](#_Toc217145197)

[1.1.3. Project Scope and Testing Scope 5](#_Toc217145198)

[1.1.4. Software Development Process (Agile Scrum) 6](#_Toc217145199)

[1.1.5. CI/CD Overview and Its Role in Testing 6](#_Toc217145200)

[1.1.6. Software Testing Overview 6](#_Toc217145201)

[**1.2. System Requirements Analysis** 7](#_Toc217145202)

[1.2.1. Current State Assessment and Business Pain Point 7](#_Toc217145203)

[1.2.2. System Functional Scope 7](#_Toc217145204)

[1.2.3. Technologies Used (ReactJS – Spring Boot – MySQL – Vercel/Railway 8](#_Toc217145205)

[**1.3. Business Processes** 9](#_Toc217145206)

[**1.4. Business Requirements** 10](#_Toc217145207)

[1.4.2. Acceptance Criteria (UAC Test) 11](#_Toc217145208)

[1.4.3. Functional Requirements 12](#_Toc217145209)

[1.4.4. Non-functional Requirements (NFR: Performance, Security, Usability) 13](#_Toc217145210)

[**1.5. Implementation Plan** 15](#_Toc217145211)

[1.5.1. Development Plan 15](#_Toc217145212)

[1.5.2. Overview Test Plan 15](#_Toc217145213)

[**CHAPTER 2 – SYSTEM ANALYSIS & DESIGN** 19](#_Toc217145214)

[**CHAPTER 2 OVERVIEW** 19](#_Toc217145215)

[**2.1. System Design** 19](#_Toc217145216)

[2.1.1. Functional Design 19](#_Toc217145217)

[2.1.2. Data design 28](#_Toc217145218)

[2.1.3. Screen design 30](#_Toc217145219)

[**2.2. Architecture Design** 34](#_Toc217145220)

[2.2.1. Architecture Overview 34](#_Toc217145221)

[2.2.2. Component Analysis 35](#_Toc217145222)

[2.2.3. Communication Analysis 47](#_Toc217145223)

[**CHAPTER 3 – TEST PLAN** 48](#_Toc217145224)

[**3.1. Test Plan Overview** 48](#_Toc217145225)

[3.1.1. Test Scope (In-scope/Out-of-scope) 48](#_Toc217145226)

[**3.2. Testing Methodology** 49](#_Toc217145227)

[3.2.1. Static Testing 49](#_Toc217145228)

[3.2.2. Dynamic Testing 49](#_Toc217145229)

[**3.3. Test Items** 50](#_Toc217145230)

[7. Usability 53](#_Toc217145231)

[10. Performance 53](#_Toc217145232)

[**3.4. Out-of-Scope Test Items** 54](#_Toc217145233)

[3.4.1. Third-Party System Functions 54](#_Toc217145234)

[3.4.2. Advanced Security Testing 55](#_Toc217145235)

[**3.5. Test Strategy** 55](#_Toc217145236)

[3.5.1. (Test Types) 56](#_Toc217145237)

[3.5.2. Test Stage Matrix 59](#_Toc217145238)

[**3.6. Test Environment** 61](#_Toc217145239)

[3.6.1 Hardware 61](#_Toc217145240)

[3.6.2 Software 61](#_Toc217145241)

[3.6.3. Human Resources 62](#_Toc217145242)

[**CHAPTER 4 TEST DESIGN** 64](#_Toc217145243)

[**4.1 Introduction** 64](#_Toc217145244)

[**4.2. Test Design Process Based on the V-Model** 64](#_Toc217145245)

[4.2.1. Requirements Analysis – 1a 64](#_Toc217145246)

[4.2.2. System Design – 2a 65](#_Toc217145247)

[4.2.3. Architecture Design – 3a 65](#_Toc217145248)

[4.2.4. Module Design – 4a 65](#_Toc217145249)

[4.2.5. Unit Testing – 1b 66](#_Toc217145250)

[4.2.6. Integration Testing – 2b 67](#_Toc217145251)

[4.2.7. System Testing – 3b 67](#_Toc217145252)

[4.2.8. Acceptance Testing – 4b 68](#_Toc217145253)

[**4.3. Test Design Techniques** 68](#_Toc217145254)

[4.3.1. White-Box Testing Technique 68](#_Toc217145255)

[4.3.2. Black-Box Testing Technique 75](#_Toc217145256)

[**Test Cases Derived from the State Transition Table** 78](#_Toc217145257)

[**4.4. Test Design Approaches** 79](#_Toc217145258)

[4.4.1. Manual Testing 79](#_Toc217145259)

[4.4.2. Automated Testing 79](#_Toc217145260)

[**4.5. Applying Generative AI in Test Case Generation** 81](#_Toc217145261)

[4.5.1. GenAI Inputs 81](#_Toc217145262)

[4.5.2. GenAI Processing Phase 82](#_Toc217145263)

[4.5.3. Test Case Refinement 83](#_Toc217145264)

[4.5.4. Prompt Engineering Techniques Applied to Test Case Generation 83](#_Toc217145265)

[Sample Prompt for Test Case Generation Using GenAI 84](#_Toc217145266)

[Prompt 2 – Generating Test Cases from Test Scenarios 84](#_Toc217145267)

[**CHAPTER 5. TESTING REPORT** 86](#_Toc217145268)

[**5.1. Overview of the Testing Process** 86](#_Toc217145269)

[**5.2. Test Case Report** 86](#_Toc217145270)

[5.2.1. Introduction 86](#_Toc217145271)

[5.2.2. Coverage Scope 87](#_Toc217145272)

[5.2.3. Execution Results 87](#_Toc217145273)

[**5.3. Defect Report** 87](#_Toc217145274)

# 

# 

# **CHAPTER 1 – INTRODUCTION**

## **Chapter Overview**

Chapter 1 provides an overall view of the Shoeshop system from a TESTING perspective, serving as a foundation for developing the Test Plan and Test Methodology in subsequent chapters.

The objective of this chapter is not to describe the system in detail, but to identify sufficient and relevant information necessary to support testing activities, including:

* Project objectives and scope → to define the testing scope.
* Business processes → to understand the workflow and identify critical testing points.
* Business requirements (User Stories, Acceptance Criteria) → to serve as a basis for constructing Test Scenarios and Test Cases.
* Functional and non-functional requirements → to determine the types of testing to be performed (functional, UI, API, performance, security, etc.).
* Development process (Agile, CI/CD) → to determine how testing is integrated within sprints and the CI/CD pipeline.

## **1.1. Topic Overvie**

### **1.1.1. Background and Rationale for Topic Selection**

In the context of rapid digital transformation, e-commerce has become an indispensable component of modern business operations, particularly in the fashion and footwear sectors. Consumers today increasingly prefer online shopping due to the convenience of searching, comparing prices, placing orders, and making payments from home. However, practical surveys indicate that many existing online sales systems still suffer from software quality limitations, leading to suboptimal user experiences and operational risks. Therefore, in order to ensure software quality before deployment and provide confidence for enterprises prior to usage, the project team selected the testing project of an online shoe retail website – ShoeShop Store – with the aim of evaluating, verifying, and assuring the software quality of a realistically simulated e-commerce platform. The topic focuses on developing a test plan, designing test cases, executing tests, and analyzing defects in order to assess the system’s stability, accuracy, and operational readiness before official deployment.

### **1.1.2. Objectives of the Shoeshop System**

The ShoeShop Store system is developed with the objective of becoming a specialized e-commerce platform for the footwear industry, addressing the following aspects:

For Customers: Providing a smooth, convenient, and secure online shopping experience, covering product search and selection, cart management, payment (COD, VNPay), and order tracking.

For Operations (Staff, Manager): Optimizing internal business processes, including real-time order management, accurate inventory control, and flexible management of promotional programs.

For Administration (Admin): Ensuring data security and safety through detailed authorization mechanisms for four roles (Customer, Staff, Manager, Admin), and providing tools for overall system monitoring.

The objective of the testing project is to verify that the Shoeshop system has achieved the above goals.

### **1.1.3. Project Scope and Testing Scope**

Project Scope: The project scope focuses on the ShoeShop Store online shoe retail website system, including functionalities developed for four roles: Customer, Staff, Manager, and Admin, with primary emphasis on four core functions: product search, cart and payment, order management, and security and authorization. The project does not include third-party systems, marketing activities, or external physical logistics processes.

Testing Scope: The testing scope is defined in close alignment with the project scope, focusing on verifying and evaluating the quality of core functionalities. For Customers: Testing end-to-end user workflows, including registration, login, product search, cart management, discount code application, payment (COD & VNPay), order tracking, and order cancellation. For Staff: Testing business workflows related to order processing, including order confirmation, status updates (from CREATED to DELIVERED), and verification of inventory accuracy during processing. For Admin: Testing system administration functionalities, including authorization, user account management, and access security. For Manager: Testing business management functionalities, including creation and management of discount codes, order monitoring and intervention, and revenue reporting. Testing activities include both Functional Testing and Non-functional Testing.

### **1.1.4. Software Development Process (Agile Scrum)**

The project is developed using the Agile model, specifically the Scrum methodology. The project is divided into short development cycles (Sprints) with the goal of delivering complete functional components after each Sprint. Testing is conducted continuously throughout the Sprint (testing in-sprint) to ensure quality, detect defects early, and provide rapid feedback to the development team (Dev) before the Sprint concludes. This approach helps minimize risks and ensures that the product remains aligned with user requirements.

### **1.1.5. CI/CD Overview and Its Role in Testing**

The project applies CI/CD automation across the software development lifecycle, from source code changes to deployment. Continuous Integration (CI) automatically merges code from multiple developers into a shared repository. More importantly, with each merge, the system automatically executes basic test suites (such as Unit Tests and Integration Tests) to detect conflicts or defects early. Continuous Delivery (CD) automatically deploys changes that pass CI to the testing environment (Testing/Staging).

CI/CD is utilized in the project to ensure that new changes do not break existing functionalities (through automated regression testing), provide a stable and continuously updated staging environment for QA to perform manual testing of new features, and significantly shorten the defect feedback cycle.

### **1.1.6. Software Testing Overview**

The project evaluates the software system in order to identify discrepancies between defined requirements and actual outcomes. The objective is not only to detect defects (bugs) but also to ensure that the software meets requirements related to functionality, performance, security, and user experience. Processes including requirements analysis, test planning, test design, test execution, defect management, and reporting conducted throughout the development lifecycle will be presented in detail in this report.

## **1.2. System Requirements Analysis**

### **1.2.1. Current State Assessment and Business Pain Point**

General context: In the strong trend of digital transformation, e-commerce has become an indispensable part of business operations, especially in the fashion and footwear industry.

User needs: Consumers increasingly prefer online shopping due to the convenience of searching, comparing prices, placing orders, and making payments from home.

Project objective: This project is selected to test the online shoe retail website – ShoeShop Store – in order to evaluate, verify, and ensure the software quality of a realistically simulated e-commerce platform.

Survey methodology: The survey process is conducted through observing operational workflows at traditional retail stores and referencing popular e-commerce websites to identify real-world operational flows.

### **1.2.2. System Functional Scope**

Product Catalog Service: View product lists; filter by criteria (brand, size, color); search products by keywords. View product details, including descriptions, images, and inventory status. Add, edit, and delete products and related information. Manage categories, brands, sizes, and colors.

Shopping Cart Service: Add/edit/remove product quantities in the cart. Apply discount codes. Recalculate the cart total after applying promotions. Enter delivery information and select shipping methods. Integrate online payment gateways or select Cash on Delivery (COD).

Order Service: View the list of placed orders. Track detailed processing status of each order. Receive, confirm, and update order statuses from customers. Track, filter, and search orders based on multiple criteria.

User Management Service: Register/Login/Logout for customers and staff. Manage/update personal profiles and change passwords. Administrators can view and manage the user list.

Access Control Service: Define and manage roles such as Admin, Manager, Staff, and Customer. Ensure that each role can only access functions according to its assigned permissions.

### **1.2.3. Technologies Used (ReactJS – Spring Boot – MySQL – Vercel/Railway**

The ShoeShop E-commerce system is built using a modern full-stack architecture, clearly separating the Frontend and Backend to ensure flexibility and scalability.

1. Frontend (User Interface): The user interface components are developed using ReactJS. This is a powerful JavaScript library that enables the development of highly interactive interfaces in the form of Single Page Applications (SPA). The use of ReactJS helps optimize page load speed and provides a smooth, user-friendly shopping experience for end users.
2. Backend (Business Logic Processing):The core business logic is implemented using the Spring Boot framework with the Java programming language. Spring Boot is responsible for providing RESTful APIs to manage all system logic, including product management, shopping cart, payment processing, and administrative functions. Choosing Spring Boot ensures high system performance, strong security, and ease of scalability as business size grows.
3. Database: The MySQL relational database management system is used to store persistent system data. MySQL is selected for its stability, high reliability, and strong capability to handle complex transactions related to orders, inventory, and customer information.
4. Deployment: The application is deployed on cloud platforms to ensure availability and ease of management. Specifically, the Frontend (ReactJS) is hosted on Vercel due to its CI/CD (Continuous Integration/Continuous Deployment) integration capabilities and high content delivery performance. The Backend (Spring Boot) and MySQL are deployed on Railway, a flexible hosting platform that provides strong support for microservices/API architectures. In addition, Docker Compose is used in the development environment to package and manage Backend and Database components, ensuring consistency across working environments.

## **1.3. Business Processes**

Product Catalog Customers accessing the system can browse the list of available shoe products. The system supports search tools and advanced filters, allowing users to filter products by brand, size, color, or desired price range. When selecting a specific product, the system displays a detailed page including images, descriptions, technical specifications, and real-time inventory status, enabling customers to obtain sufficient information before making a purchase decision.

Access Control Process To ensure security and accurate authorization, the system requires users (including Customers, Staff, Managers, and Administrators) to log in to access corresponding functionalities. New customers can register an account by providing basic personal information. The system authenticates login credentials and grants access based on roles: Customers can access shopping features; Staff and Managers can access order management functions; Admin can access high-level administrative functions. At the end of a session, users log out to secure their accounts.

Shopping Cart Process After selecting desired products, customers add items to the shopping cart. Here, customers can review the product list, adjust purchase quantities, or remove unnecessary items. The system automatically recalculates the cart subtotal whenever quantity changes occur. If customers possess a discount code, they can apply it at this stage to view the discounted value before proceeding to checkout.

Order Management Process The ordering process begins when customers confirm the cart and provide delivery information. The system supports two payment methods: Cash on Delivery (COD) or online payment via VNPay. After successful order placement, the order is initialized with the status CREATED. Staff or Managers verify inventory and confirm the order (CONFIRMED). The processing workflow continues through preparation (PREPARING), ready for delivery (READY\_FOR\_DELIVERY), and handover to the shipping unit (OUT\_FOR\_DELIVERY). When the customer receives the goods, the order status changes to DELIVERED. In cases of order cancellation or returns, the system performs status updates and refund processing (for VNPay orders) according to regulations.

User Management Process This module allows users (Customers) to manage and update personal information such as full name, delivery address, and password for convenience in future purchases. For Administrators (Admin), the system provides the highest level of privileges to manage the entire user list. Admins can view details, create new accounts, update roles, or disable (lock) accounts that violate policies or belong to resigned staff members.

Discount Code Management Process The promotion management page is dedicated to Managers for setting up business campaigns. Managers can create new discount codes (Coupons) with detailed parameters such as code value, discount amount (percentage or fixed value), start/end dates, quantity limits, and minimum order value. During checkout, customers enter the code, and the system automatically validates its eligibility (validity period and conditions) and directly deducts the discount from the total payment if the code is valid.

## **1.4. Business Requirements**

**1.4.1. User Story (As a… I want… so that…)**

|  |  |  |
| --- | --- | --- |
| **Role (As a)** | **Desire (I want)** | **Purpose (So that)** |
| **Customer** | View the product list and filter by brand, size, and price range | So that I can quickly find shoes that match my needs and budget |
| View product details (description, images, inventory) | Để có đủ thông tin ra quyết định mua So that I have sufficient information to make a purchasing decision |
| Add products to the cart, update quantities, or remove products | So that I can manage the items I want to purchase before checkout |
| Enter discount codes in the cart or during checkout | So that I can receive promotions and reduce the total order cost |
| Pay for orders via COD or VNPay | So that I can flexibly choose the most convenient payment method |
| Track order status (Created → Delivered) and view order history | So that I can monitor delivery progress and manage my spending |
| **Staff** | View the order list and confirm orders based on inventory availability | So that sufficient stock is ensured before proceeding to the packing process |
| Update order status according to the delivery workflow | So that customers and managers can track the order journey |
| **Manager** | Create, edit, and disable discount codes | So that promotional and seasonal marketing campaigns can be executed |
| **Admin** | Manage users, assign permissions, and lock/unlock accounts | So that access control and system security are ensured |

### 1.4.2. Acceptance Criteria (UAC Test)

1. **UAC 1 – Order Logic**

* Order status must follow the sequence: CREATED → CONFIRMED → PREPARING → READY\_FOR\_DELIVERY → DELIVERED.
* Customers are only allowed to cancel orders when the status is CREATED or CONFIRMED.
* Order modification is not permitted once the status has changed to READY\_FOR\_DELIVERY.

1. **UAC 2 – Inventory Logic:**

* When an order is successfully created, inventory quantity must be automatically deducted corresponding to the order quantity.
* Staff cannot confirm an order if inventory is insufficient.

1. **UAC 3 – Discount Logic:**

* A discount code can only be applied when all three conditions are satisfied:
* Within the valid period
* Order meets the minimum value requirement
* Usage count has not reached the limit
* The system must display specific error messages if the code is invalid or expired.

1. **UAC 4 – Payment Logic:**

* If VNPay is selected, the system must redirect to the Sandbox payment gateway and update the order status after receiving a successful transaction response.

### **1.4.3. Functional Requirements**

1. **Product Catalog Management:**
   * The system displays a product list with filtering and sorting features.
   * The system displays product details including: Name, Price, Description, Size, Brand, Images, and Inventory.
   * Admin has CRUD permissions.
2. **Shopping Cart:**
   * The system calculates total amount, taxes, shipping fees, and discounts in real time when users change quantities.
   * The system stores the cart state of logged-in users in the Database.

**3. Checkout & Payment:**

* + The system allows users to enter delivery addresses and select payment methods (COD/VNPay).
  + The system creates order records (CustomerOrder) and order details (OrderDetail) in the Database upon successful checkout.

**4. Order Management:**

* + Staff/Admin can change order statuses.
  + Admin/Manager can view revenue reports and order status summaries.

**5. User Management & Security:**

* + The system supports Registration, Login, and Logout.
  + Admin manages the user list and activates or disables accounts.
  + Users can update personal information.

### **1.4.4. Non-functional Requirements (NFR: Performance, Security, Usability)**

* **Performance:**
  + Page load time < 2 seconds.
  + Checkout processing time < 3 seconds.
  + The system ensures stable operation with 1,000–2,000 concurrent users.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Main Activities** | **Deliverables** |
| **1.Requirements Analysis** | Collect and analyze business requirements; define User Stories, Use Cases, FR/NFR; identify four roles (Customer, Staff, Manager, Admin) | Completed SRS and User Stories; Use Case list; defined testing scope. |
| **2. System Design** | Design architecture (C4), database (ERD), APIs, UI Flow and Wireframes; describe workflows for Cart – Checkout – Order; design API architecture and Sequence Diagrams for core flows. | Complete architectural model; data diagrams; UI Flow; API list. |
| **3. Development** | Develop Backend (Spring Boot) and Frontend (ReactJS); integrate MySQL; implement core APIs: Authentication, Product, Cart, Order, Discount; integrate payment; package using Docker Compose. | Fully functional system modules: Login, Catalog, Cart, Checkout, Order, RBAC; stable APIs and UI. |
| **4. Testing** | Design Test Cases and Test Data; execute testing following the V-Model: Unit Test, Integration Test, System Test, and UAT; run Automation using Selenium; perform Regression and UAT. | Complete Test Case set; Test Plan; Test Report; bug list with severity levels; system quality confirmation. |
| **5. Deployment & Evaluation** | Deploy the system to Vercel (Frontend) and Railway (Backend + MySQL); integrate CI/CD with GitHub Actions: build → test → deploy; evaluate trial run results. | Stable deployment version; system running in Production/Staging environments; final sprint evaluation report. |

* **Security:**
  + User passwords must be encrypted.
  + Online payments must be conducted over HTTPS and standard SSL/TLS-secured gateways.
  + Role-based access control (Admin, Manager, Staff, Customer).
* **Usability:**
  + The interface supports Responsive Design and displays well on Desktop and Mobile devices.
* **Reliability:**
  + System uptime reaches 99%.
  + Recovery time after incidents (Recoverability) < 5 minutes.

## **1.5. Implementation Plan**

### **1.5.1. Development Plan**

The ShoeShop Store system is developed using the Agile Scrum model with Sprint cycles. Each Sprint delivers a complete functional increment to support continuous testing. The V-Model is applied to divide software development into design, implementation, integration, and quality verification. This systematic strategy ensures that each development phase has a clearly defined testing activity, thereby contributing to high-quality software. Testing at each phase helps detect defects early, accelerate project completion, reduce costs, and improve overall quality.

### **1.5.2. Overview Test Plan**

#### **1.5.2.a. Test Objectives**

The primary purpose of testing for the ShoeShop Store system is to ensure comprehensive software quality before go-live, specifically including:

* Verify business functionality: Ensure that all e-commerce business processes operate correctly according to defined requirements. Key modules include: Product Catalog, Shopping Cart and Checkout, Order Management, User Management and Access Control, and Discount Management.
* Non-functional evaluation: Validate system performance, security, usability, reliability, and compatibility.
* Risk identification and mitigation: Detect defects (bugs) as early as possible through continuous testing within each Sprint (testing in-sprint) and the V-Model approach, thereby reducing defect-fix costs and operational risks.
* Stability assurance: Confirm that the system meets the criteria for end-user acceptance (UAT) through End-to-End testing flows.

**1.5.2.b. Testing Approach**

The test plan applies a combination of methods to ensure end-to-end quality, particularly integrating Agile Scrum and the V-Model to ensure testing is conducted throughout the lifecycle—from requirements analysis through development, integration, and product acceptance.

* Test Levels

The testing process is executed sequentially through four primary levels, corresponding to the Validation phase of the V-Model, ensuring quality from the smallest components to the overall system. It begins with Unit Testing, performed by the development team. Using tools such as JUnit and Mockito, developers verify the correctness of individual functions and services, focusing on core business logic such as total cost calculation, inventory validation, and account authentication. Once code units are stable, the process moves to Integration Testing to validate seamless interaction between critical modules (e.g., data flow from Shopping Cart to Checkout) as well as data integrity between the Backend and the Database.

After the components are fully integrated, the QA team conducts System Testing. This phase performs End-to-End testing based on real user business workflows—from login and product search to completing payment and tracking orders—to comprehensively evaluate stability, performance, and UI/UX experience. Finally, before product handover, Acceptance Testing (UAT) is conducted in the Staging environment. This step validates the system from the end-user perspective to confirm that it fully meets the defined business requirements.

* Testing Techniques and Types

To maximize effectiveness, the project applies a flexible combination of manual testing and automated testing:

For complex business flows that require human logical reasoning or involve UI/UX and user perception, the team applies Manual Testing. This method helps identify usability issues, navigation problems, and the correctness of error messages that automation may not reliably detect.

Conversely, Automation Testing is prioritized for frequently repeated tasks to save resources and ensure consistency. Specifically, Selenium WebDriver is used to automate UI scenarios such as login and checkout, while Postman/Newman is responsible for sending automated requests to validate JSON structures and HTTP status codes of Backend APIs. In addition, Performance Testing is emphasized using Apache JMeter to measure load/stress capacity of high-traffic APIs such as product listing and order placement.

* Integrating Testing into the CI/CD Process

To ensure the system remains deployment-ready, testing activities are deeply integrated into the GitHub Actions pipeline. The system is configured to automatically trigger Unit Tests and Integration Tests whenever a Pull Request is created or code is merged. In parallel, API test suites are executed automatically to validate data consistency. This workflow enables early defect detection, minimizes regression risks, and ensures that every new change does not break stable existing features before deployment to the Staging environment.

|  |  |
| --- | --- |
| Content Group | Methodology Summary |
| Test Levels | * Unit Test: Developers test individual functions using JUnit/Mockito. * Integration Test: Verify interactions between modules and between Backend and Database. * System Test: QA performs End-to-End testing across the entire business workflow. * Acceptance Test (UAT): Validate business requirements in the Staging environment before handover. |
| Testing Techniques & Types | * Manual Testing: Applied to UI/UX, complex business flows, and user experience validation. * Automation Testing: Selenium for UI; Postman/Newman for APIs; suitable for repetitive test cases. * Performance Testing: JMeter is used to conduct load, stress, and response time testing for critical APIs. |
| CI/CD Integration | * Testing is integrated into GitHub Actions. * Automatically execute Unit Tests and Integration Tests on Pull Requests or code merges. * Automatically run API Tests to validate data consistency. * Ensure no regression defects occur before deployment to the Staging environment. |

# 

# **CHAPTER 2 – SYSTEM ANALYSIS & DESIGN**

## **CHAPTER 2 OVERVIEW**

Chapter 2 presents the system design and technical architecture of Shoeshop with the objective of directly supporting software testing activities. The content focuses on describing system operations from two perspectives: business (System Design) and technical (Architecture Design), thereby helping to identify test objects, critical data flows, and risk points that need to be prioritized when designing test scenarios.

At the business level, the chapter describes the system's functional decomposition, the main processing flows for customers and administrators, along with the conceptual data model. => This helps testers clearly understand "what" the system does and "in what sequence" it processes, serving as a basis for constructing Test Scenarios and identifying critical testing areas.

At the technical level, the chapter describes the system architecture comprising the Frontend – Backend – Database layers, communication mechanisms via APIs, and internal data flows. This information supports the selection of appropriate testing types, such as API testing, integration testing, database testing, or performance testing.

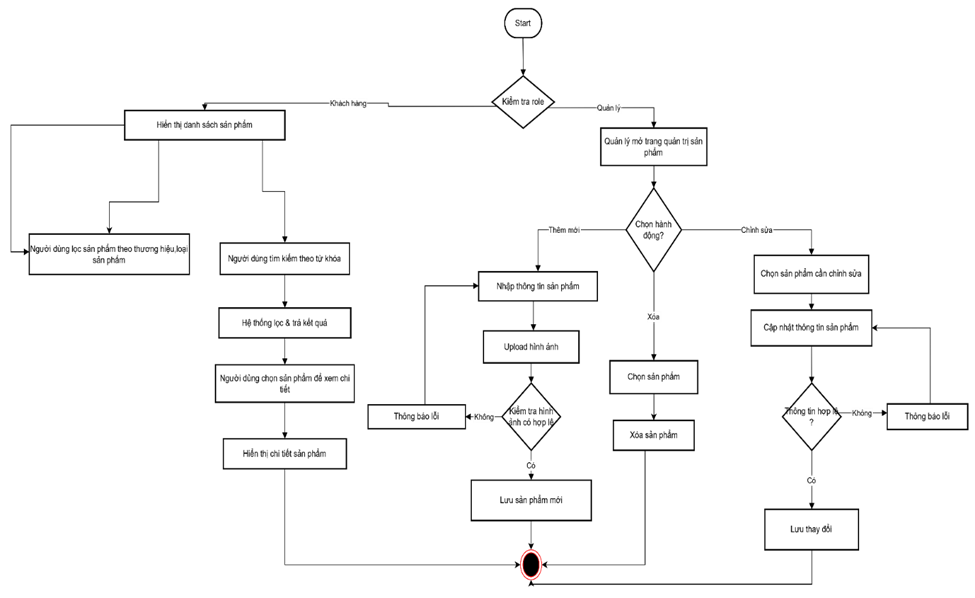
Chapter 2 serves as a foundation for Chapter 3 and Chapter 4: the designs, processing flows, and architectures presented here are used to develop the Test Plan, Test Strategy, and Testing Methodology for the entire project. Detailed models such as the full ERD, sequence diagrams, API list, and interface mockups will be presented in the Appendix.

## **2.1. System Design**

### **2.1.1. Functional Design**

**1. Product Catalog**

Description: This module acts as a central management and displays all data related to store products. This functionality enables customers to easily access products through search and filtering tools, while simultaneously providing managers with the capability to accurately control and update merchandise information.



*Image 1.Activity Product*

**Users (Customers) can:**

* + **Filter products:** Customers can use available filters such as brand and product category. The system will apply these criteria and display only the list of products satisfying the filtering conditions selected by the user.
  + **Search for products:** Customers enter a name or keyword related to the desired product into the search box. The system performs a query within the database and returns a list of products whose names or characteristics match the keyword.
  + **View product details:** Including description, images, and information regarding stock quantity (Availability).

**Managers can:**

* Perform operations to Add, Edit, and Delete products and related information (price, description, images).

**2. Shopping Cart & Checkout Service**

This module manages the entire process from the moment the customer selects products and applies promotions until payment completion and order creation.



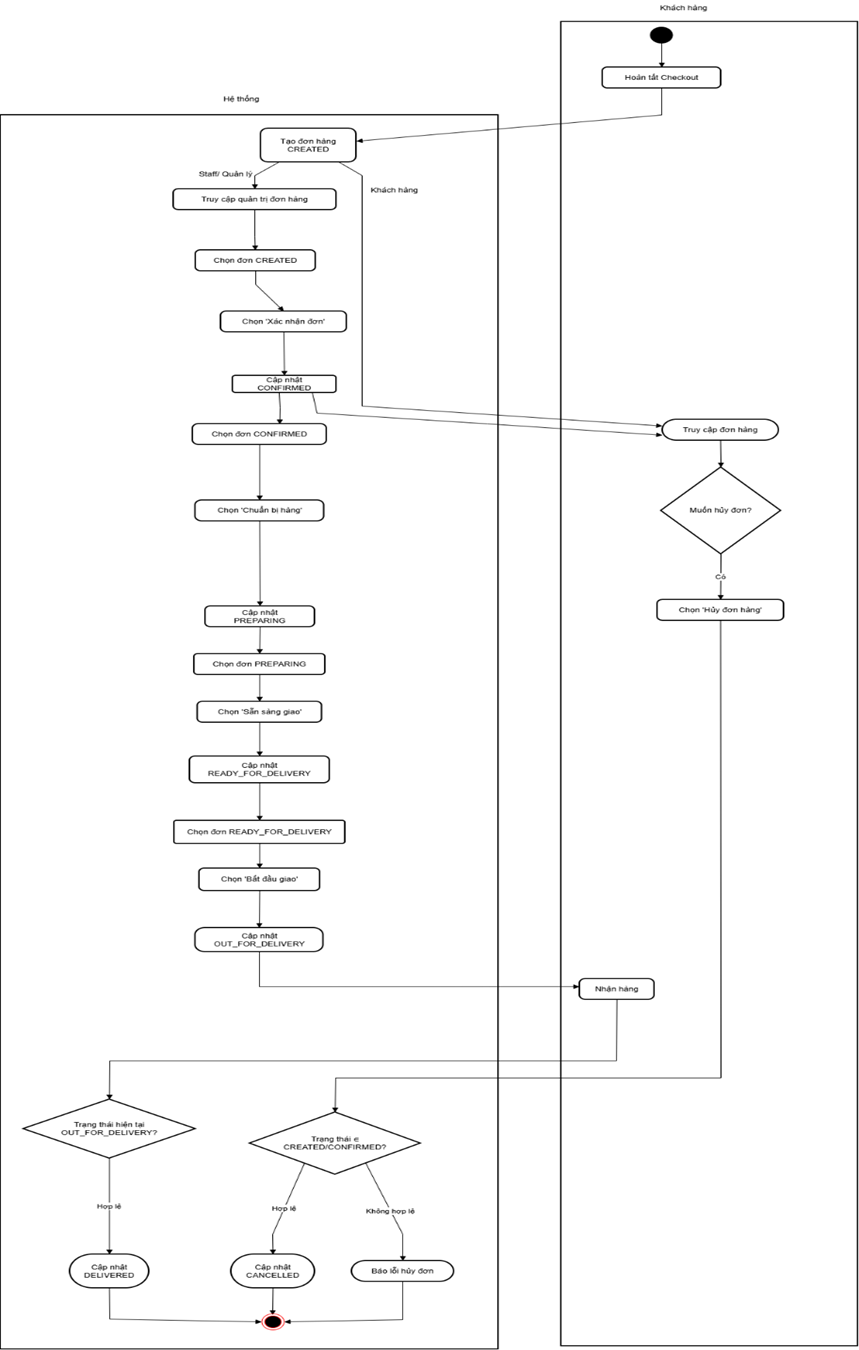
*Image 2.Activity Cart*

**Users (Customers) can:**

* + **Manage shopping cart:** Customers can flexibly perform Add/Edit/Delete operations on products within the shopping cart. When the user changes quantities or removes products, the system automatically updates the list and the provisional total value of the shopping cart.
  + **Apply discount code:** Customers can enter a discount code if available. The system verifies the validity of the code (expiration, applicable conditions) and automatically recalculates the total amount to be paid after deducting the promotional value.
  + **Enter shipping information:** At the checkout stage, customers must provide full delivery details (name, phone number, specific address) and select a shipping method suitable for their needs and costs.
  + **Select payment method:** Use integrated online payment gateways or select Cash on Delivery (COD).

**3. Order Management**

Description: This module plays a central role in ensuring that order processing proceeds smoothly and transparently. This functionality enables both customers and staff to closely monitor the order progress from creation to successful delivery.



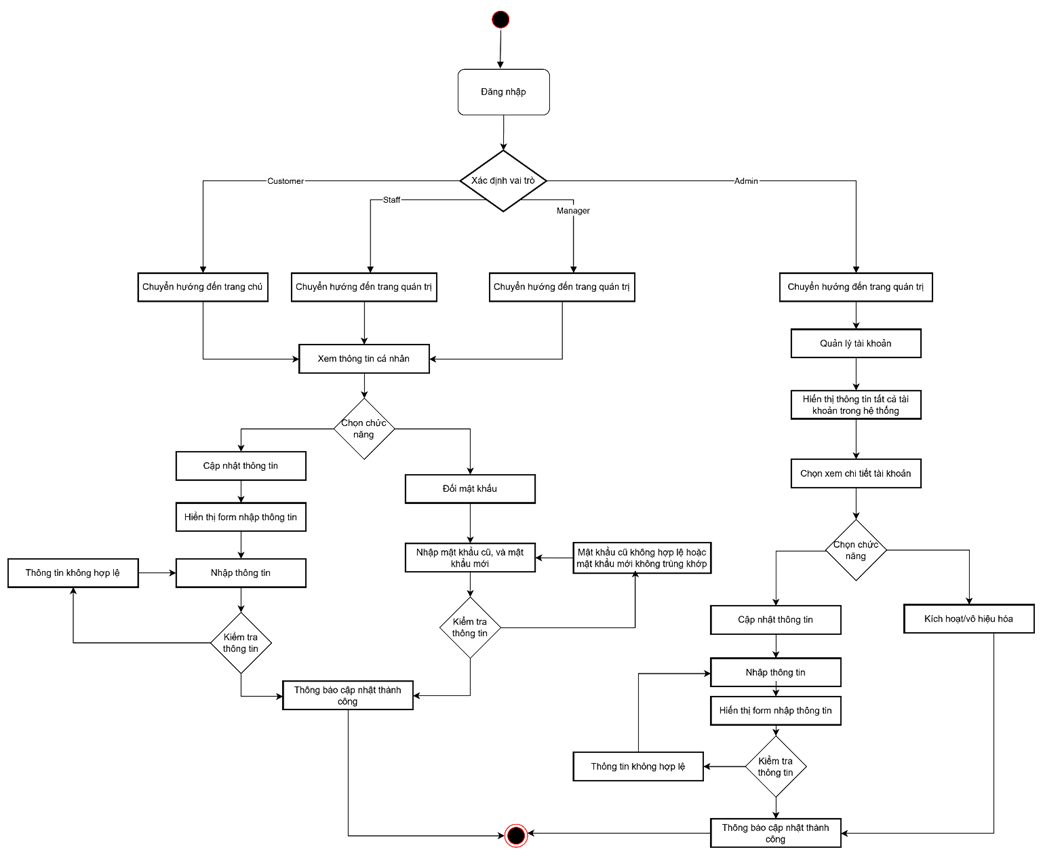
*Image 3. Activity Order*

**Users (Customers) can:**

* + **View order history:** View the list of orders placed in history.
  + **Track status:** Customers can monitor the detailed processing status of each order in real-time (e.g., Pending, Preparing, Shipping, Successfully Delivered
  + **Cancel order:** In case of a change of mind, customers can actively cancel an order if it is still in the "Pending" status (not yet processed by staff).
* **Managers/Staff can:**
  + **Update status:** Staff execute business processes such as Pending-> Confirm (inventory check) -> PREPARING-> DELIVERY. Each step is updated on the system for customers to track.
  + **Manage lists:** Managers can use advanced Filter and Search tools to look up orders based on various criteria (by order date, order status, customer name), making the management of large volumes of orders efficient.

**4. User Management**

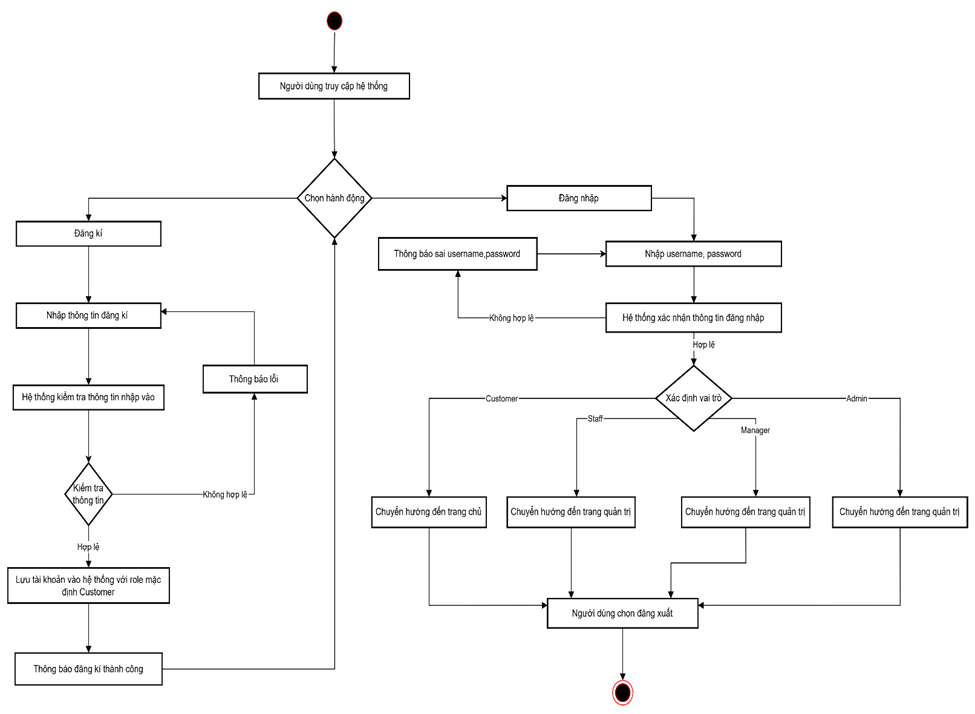
This module manages account information and user profiles within the system.



* **General Functionality:**
  + Support the Registration/Login/Logout processes for both customers and staff.
  + Allow users to Manage/update personal profiles and change passwords.
* **Admins can:**
  + View and manage the list of user accounts, including both customers and staff.

**5. Access Control**

Access Control (Access Control Service) Description: This module plays a crucial role in protecting the system and ensuring data integrity. The primary function is to authenticate user identities and grant access to appropriate resources based on their roles.



*Hình 4. Activity Access Control*

**Main Functions:**

* + **Registration:** New users can create an account by providing personal information (email, password). The system will verify the validity and uniqueness of the information before creating the account with the default role of Customer.
  + **Login:** Users access the system using a username and password. The system authenticates the information; if correct, the user is granted access (usually via a token or session).

**Authorization:** After successful login, the system will automatically navigate the user to the working interface appropriate for their role:

o **Customer:** Access the homepage, view products, shopping cart, order history

o **Staff:** Access the order management page for processing.

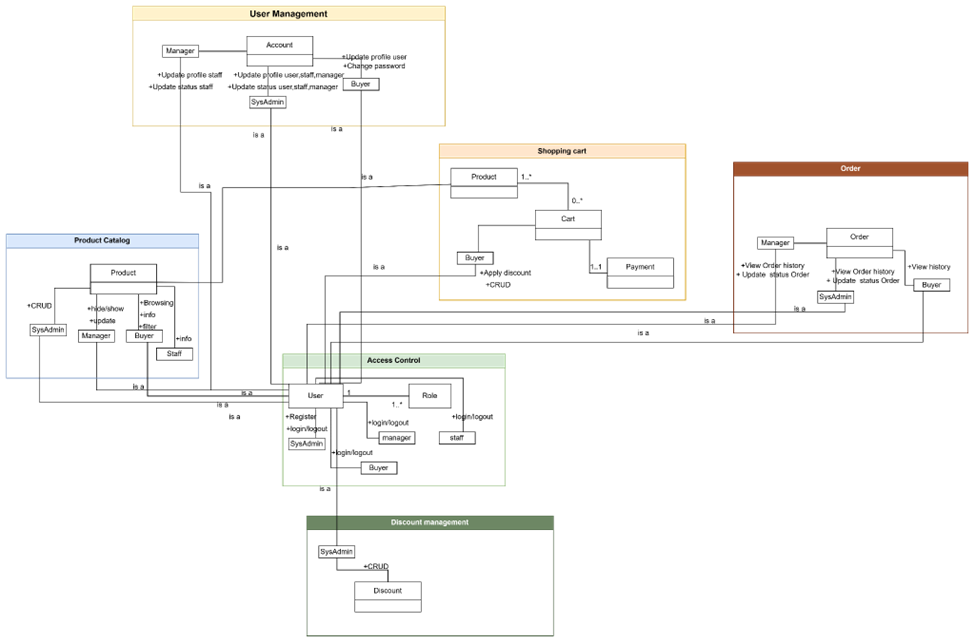
o **Manager:** Access product management, discount,order functions.

o **Admin:** Access full system privileges, including user management and system configuration.

* + **Logout:** Users can securely end the working session. The system will destroy the current token/session and redirect the user to the login page or homepage.

### **2.1.2. Data design**

Conceptual Data Model



*Image 5. Conceptual Data Model*

The ERD describing the system's data structure is tightly organized according to 5 main functional groups:

- **Access Control**: The central User entity combines with Role to implement authorization. Specific roles such as Buyer, Manager, and SysAdmin all have an "is a" inheritance relationship with the Account entity for centralized login information management.

- **Product Catalog**: Focuses on the Product entity. This data is directly linked to user actions: Buyer performs Browsing/Filter and Manager performs Create/Update/Delete.

- **Shopping Cart**: Cart plays the role of temporarily storing the list of Products selected by the Buyer (1..\* relationship). This entity has a 1-1 association with Payment to record payment information before conversion into an order.

- **Order Management**: Order is the primary data entity generated following the purchasing process. This data is accessed by the Buyer to view history (View history) and acted upon by the Manager to update the processing status (Update status).

- **Discount Management**: The Discount entity contains discount code information initiated by the Manager (CRUD), and is linked to the purchasing process when the Buyer applies the code (Apply discount).

ERD logic



*Hình 6. ERD Logic*

Note: data dictionary sheet file.

### **2.1.3. Screen design**

* + **UI Flow (Customer → Cart → Payment → Orders)**
  + **Product List):** Users view and search for products (Select product)
  + **Product Detail:** View information, select size, quantity → Click "Add to Cart".
  + **Cart (Cart Page):** Review list, edit quantity, enter discount code → Click "Checkout".
  + **Checkout (Checkout Page):** Enter address, Select payment method (COD/VNPay) → Click "Place Order".
    - **(If VNPay is selected):** Redirect to VNPay payment page → Return.
    - **Success (Order Success):** Order success notification.
  + **Order History:** Users view order processing status.

**Wireframes of main pages**

Product List Screen

* Header: Logo, Cart Icon (with quantity badge), User Avatar (Login/Register).

A screenshot of a website

AI-generated content may be incorrect.

*Image 7. UI Home Page*

* Sidebar (Filters):
  + Filter by shoe brand (Nike, Adidas...)
  + Filter by shoe category (Running, Casual...)
  + Filter by gender (Men, Women, Unisex)
* Main Content (Product Grid): Displayed in Grid format. Each product card includes:
  + Image.
  + Product name.
  + Selling price and struck-through Original price.
  + Quick button: "View details".

A screenshot of a website

AI-generated content may be incorrect.

*Image 8. UI Product*

Cart Screen (Cart UI)

* List Table:
  + Product Column: Thumbnail image + Name + Selected Size.
  + Unit Price Column.
  + Quantity Column: Numeric input or +/- buttons (Real-time price update).
  + Total Amount Column (Unit Price \* Quantity).
  + Delete Button.
* Order Summary Table:
  + Subtotal.
  + Discount Code: Code input field + "Apply" button. Displays discounted amount.
  + Total: Final amount to be paid.

A screenshot of a computer

AI-generated content may be incorrect.

*Image 9. UI Cart*

Checkout Screen

* Shipping Information: Input form (Full name, Phone number, Detailed address). Requires validation to ensure fields are not left empty.
* Order Review: Summarize the list of items and total amount one last time before finalizing.
* Payment methods:
  + Cash on Delivery (COD).
  + VNPay Payment (Redirect to payment gateway).

A screenshot of a computer

AI-generated content may be incorrect.

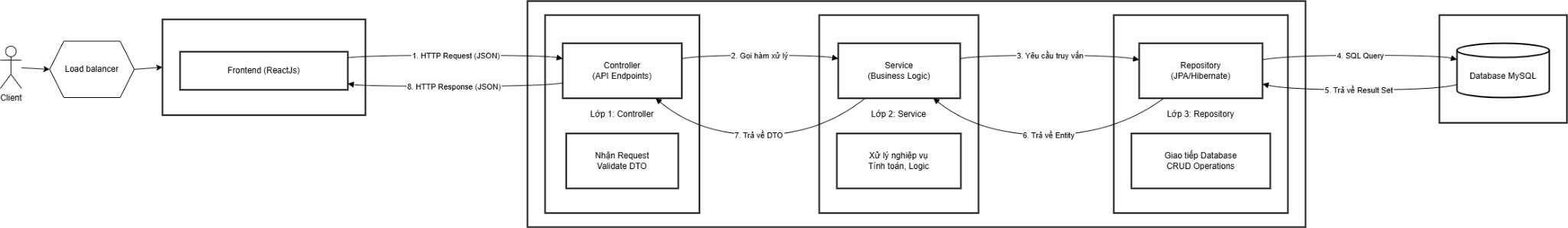
*Image 10. Checkout*

**Mapping UI → Interface Testing Group**

|  |  |  |  |
| --- | --- | --- | --- |
| UI | Element | Test Scenario | Hình ảnh |
| Product List | Filter | Select "Adidas" Brand → Only Adidas shoes are displayed. |  |
| Product Detail | "Add to Cart" button | Click "Add" → Cart icon on the Header must update the count (+1). |  |
| Cart | Discount code | Enter expired/incorrect code → Red error message "Invalid Coupon". |  |
| Order History | Order Status | The newly placed order must have the status CREATED (or PENDING if not yet paid). |  |

## **2.2. Architecture Design**

### **2.2.1. Architecture Overview**

****

*Image 11. Architecture Design*

* The data flow will be: Client (Frontend) -> Controller -> Service -> Repository -> Database
* Client Block (User Side): Represented by the web browser. Its function is to send HTTP requests (GET, POST, PUT, DELETE) to the Server, send data in JSON format (e.g., order information, login information), and receive responses from the Server to display to the user.
* **Controller Layer Block (Interface/Routing Layer):** Represented in the code by classes: ShoeController, CartController, AuthController. **Function:** Receives requests sent from the Client, validates input data, and does not process complex logic; it merely coordinates "who will perform what actions". It calls down to the Service Layer to request business logic processing and returns results (APIResponse, HTTP Status 200, 400, 500) to the Client.

#### **Service Layer Block (Business Layer):** Represented in the code by classes ShoeService, CartService, AuthService. **Function:** Acts as the "Brain" of the system. Contains the entire business logic. Examples: Checking inventory, calculating cart totals, applying discount codes, encrypting passwords, transforming data between DTOs (Data Transfer Objects) and Entities, and calling down to the Repository Layer to retrieve or save data.

* **Repository Layer Block (Data Access Layer):** Represented in the code by interfaces such as ShoeRepository, CartRepository, UserRepository. Function**:** Directly interacts with the Database, executes query commands such as: findById, save, delete, findByStatusTrue…, and maps data from tables in the Database into Java Objects (Entities) via Hibernate/JPA.

#### **Database Block (Database):** Represented by MySQL. Function: To persistently store all data (User, Shoe, Order, Cart tables...).

* Data Flow Example for "Create Product" Function:

1. **Client:** Sends POST /shoes with shoe information in JSON format.
2. Controller (ShoeController):

* Receives the request.
* Performs validation using @Valid (e.g., name is not empty, price > 0).
* Calls shoeService.createShoe(request).

1. Service (ShoeService):

* Receives ShoeCreateRequest.
* Checks if the Brand exists.
* Maps ShoeCreateRequest → Shoe Entity.
* Gọi shoeRepository.save(shoe).

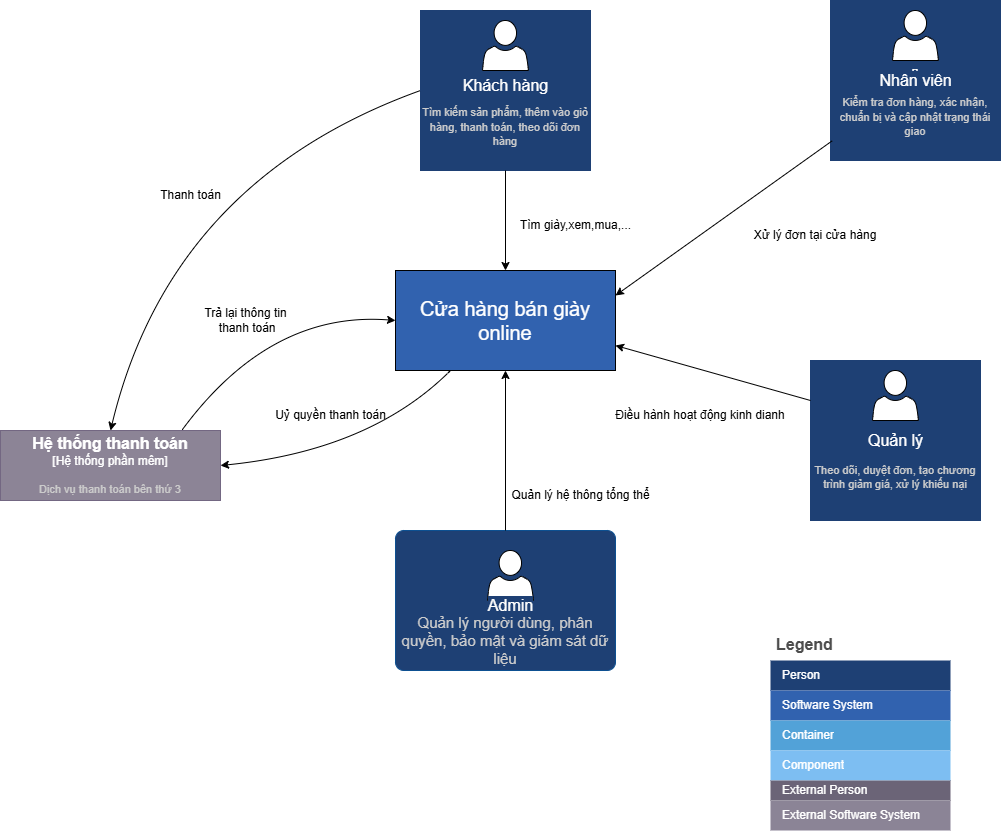
1. Repository (ShoeRepository):

* Generates INSERT INTO shoe ... statement.
* Sends query to the Database.

1. **Database:** Saves data and returns the newly created ID.
2. **Service:** Receives the saved Entity, maps it to ShoeResponse.
3. **Controller:** Wraps ShoeResponse into APIResponse and returns HTTP status 200.

### **2.2.2. Component Analysis**

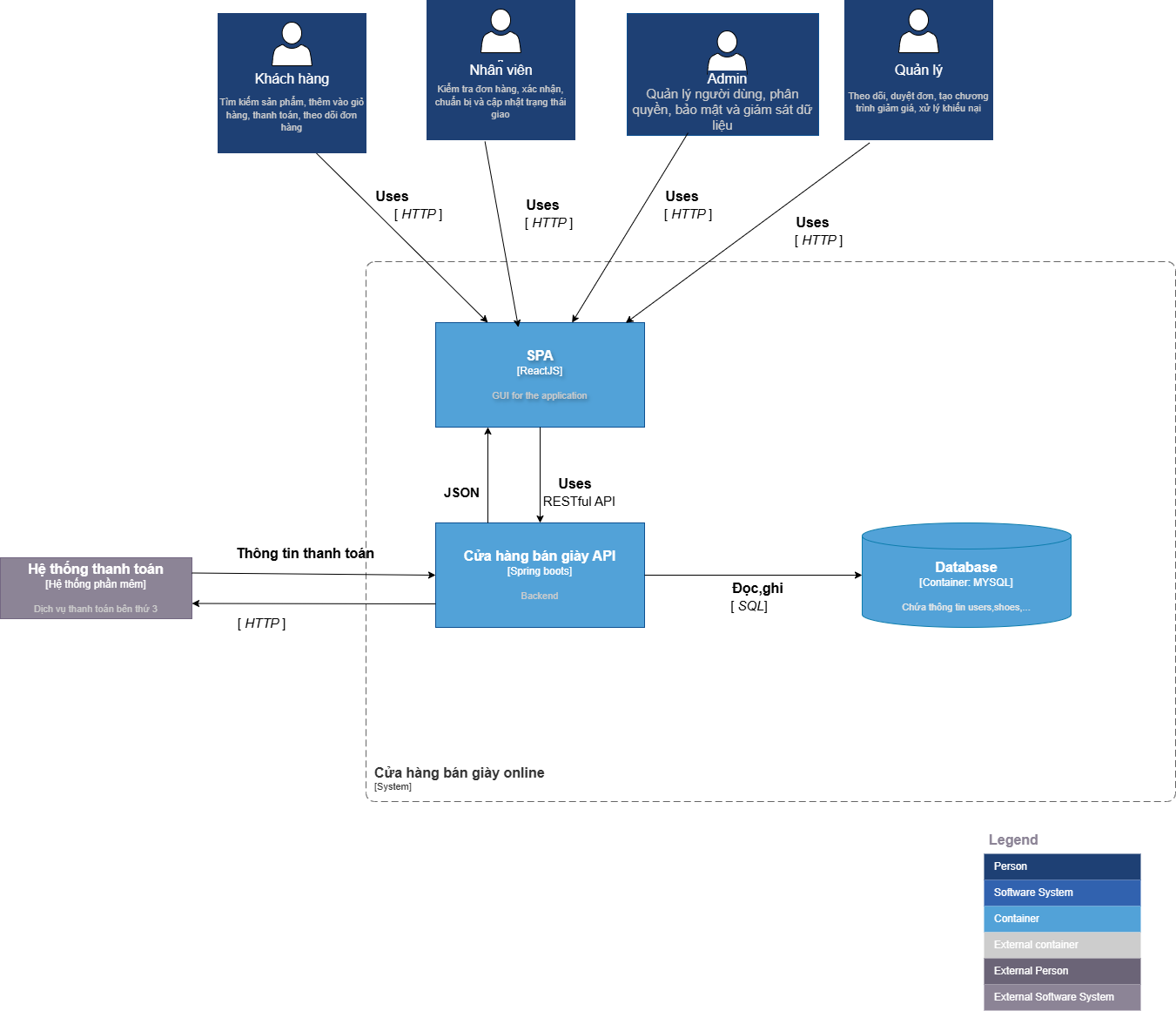
1. C1



*Image 12. C1 Model*

* The system consists of 4 distinct user roles: Admin, Customer, Manager, and Staff.
* Customer is the end-user who shops for products; they can search for products, add to the cart, make payments, and track orders.
* Store Staff is the person who processes orders at the store; they can check orders, confirm, prepare, and update delivery status.
* Store Manager is the person responsible for operating business activities; they can monitor, approve orders, create discount programs, and handle complaints.
* Admin is the person who manages the overall system; they can manage users, assign permissions, maintain security, and monitor data.

1. C2



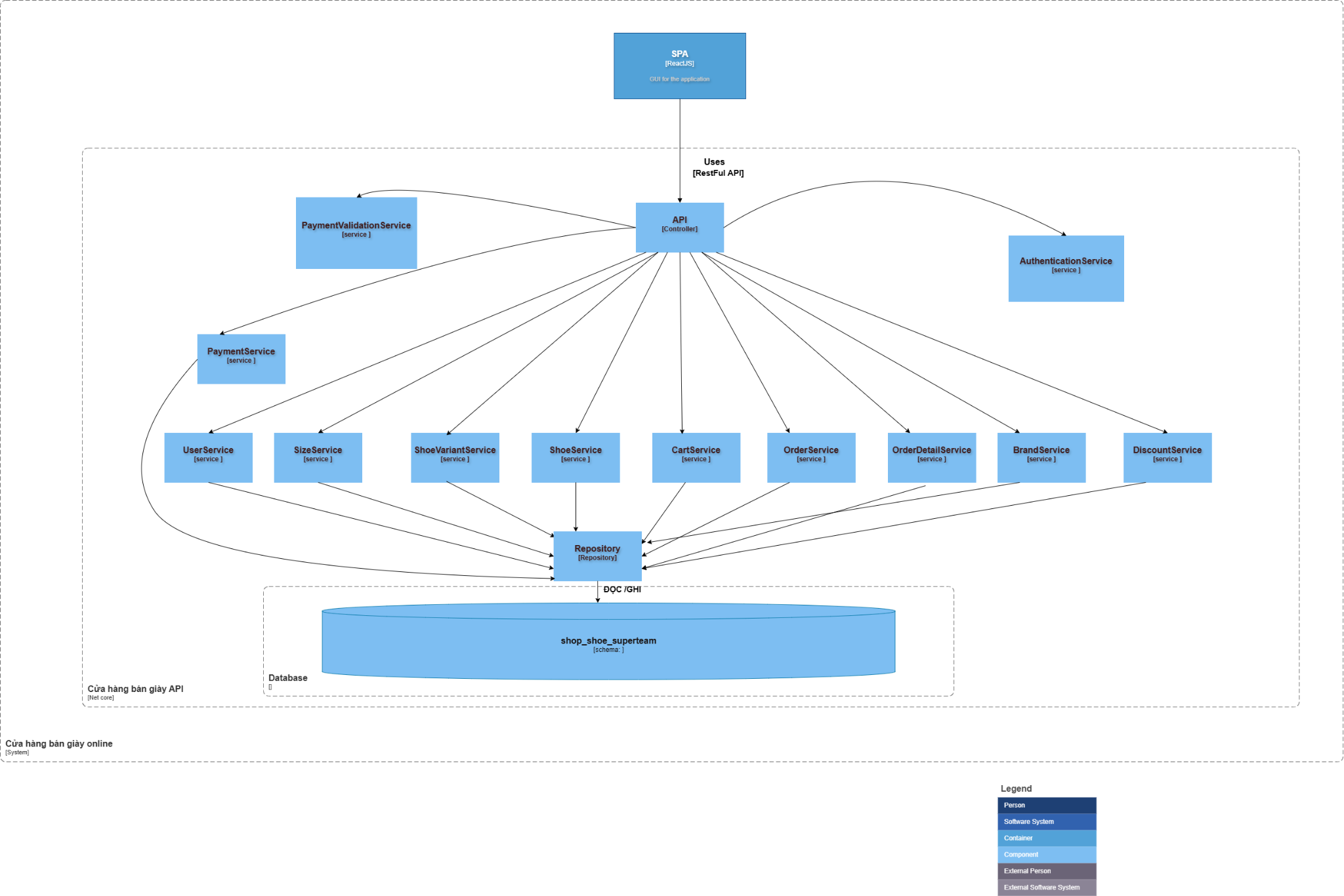
*Image 12. C2 Model*

* SPA (Single Page Application):
* Type: Container (represented by a light blue rectangle).
* Technology: ReactJS.
* Role: Provides the Graphical User Interface (GUI) for the entire application. This is where users (Customer, Staff, Manager, Admin) interact directly.
* Interaction: Receives requests via HTTP from all 4 user types. Communicates with the "Shoe Store API" via JSON to send/receive data.
* Shoe Store API (Backend) + Type: Container (represented by a light blue rectangle).
* Technology: Spring Boot (Java language).
* Role: This is the heart of the system, handling the entire business logic of the application.
* It provides RESTful API services for the SPA to call and interact with.
* Interaction: Receives requests (Uses) RESTful API from the SPA. Reads/Writes (Read/write) data to the "Database". Interacts (Payment information) with the external "Payment System".
* External Software System: The system interacts with an external software system, which is the payment system (Banking System):
* Type: External Software System (represented by a gray rectangle).
* Technology: Third-party payment service.
* Role: Processes payment transactions.
* Interaction: "Shoe Store API" sends payment requests (HTTP) to this system and receives the results.

#### Database

* Type: Container (represented by a dark blue cylinder).
* Technology: MySQL.
* Role: Stores all information regarding shoes, users, orders, etc. (Contains users, shoes information...).
* Interaction: "Shoe Store API" is the only component that reads from and writes directly to the database.
* Relationships and Protocols
* Arrows: Represent information flow or dependency
* Labels on arrows: Describe the type of information transmitted or the protocol used (HTTP, JSON, RESTful API, Read/write, Payment information).

1. C3

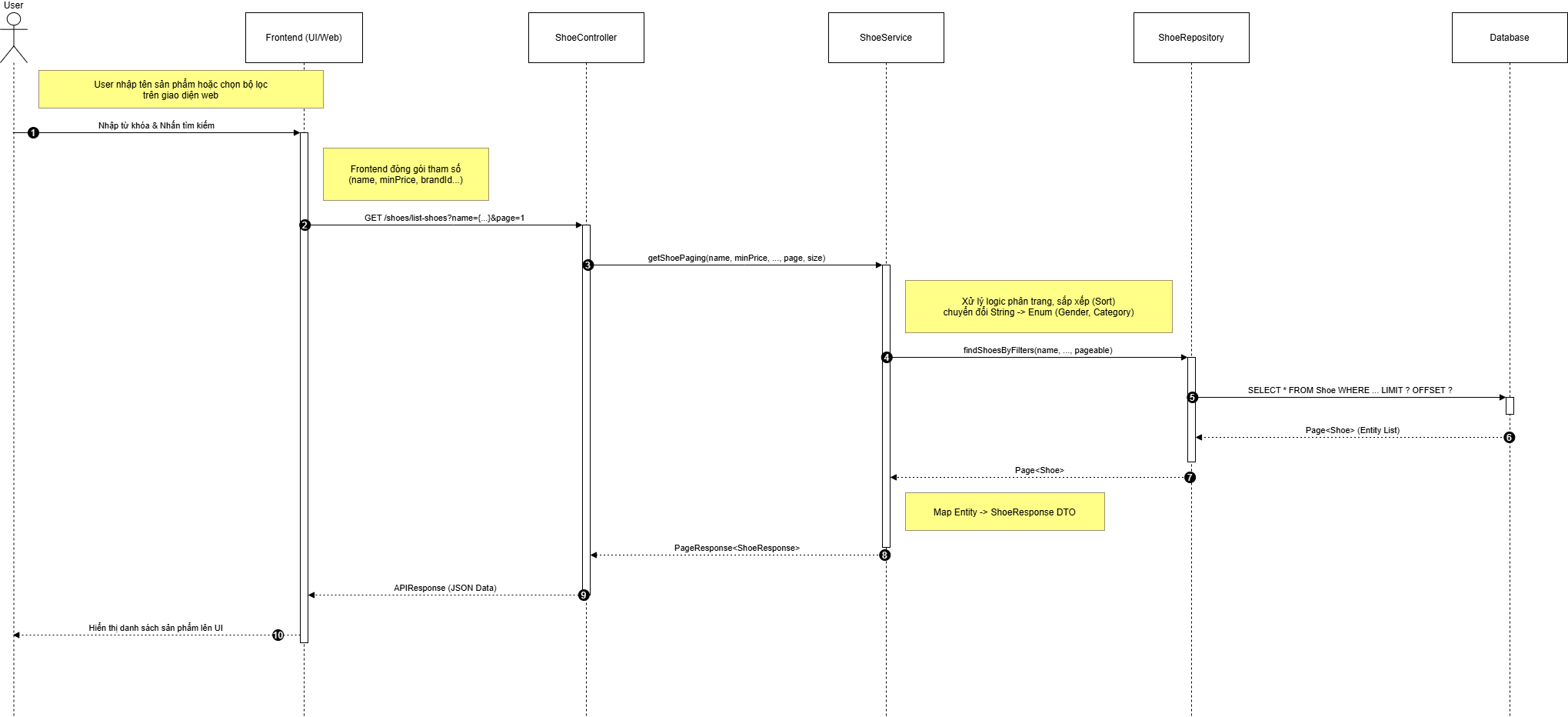


*Image 13. C3 Model*

* Data flows from the Web Browser into the API Application via the HTTP protocol using the JSON data format.
* Within the API section, the system is divided into distinct Controller components handling specific business logic; for example, AuthenticationController is responsible for authentication, authorization, login, and account registration.
* ShoeController manages functions related to shoe products such as viewing lists, details, and searching. CartController processes cart operations such as adding products, updating quantities, and deleting. OrderController manages the order lifecycle from creation and confirmation to delivery. PaymentController processes payment transactions, DiscountController manages the creation and application of discount codes for orders. UserController provides functions for managing personal information and user accounts. BrandController manages information regarding brands and product classification.
* To perform their functions, these controllers call the corresponding services; services needing to interact with the database call the corresponding repositories.

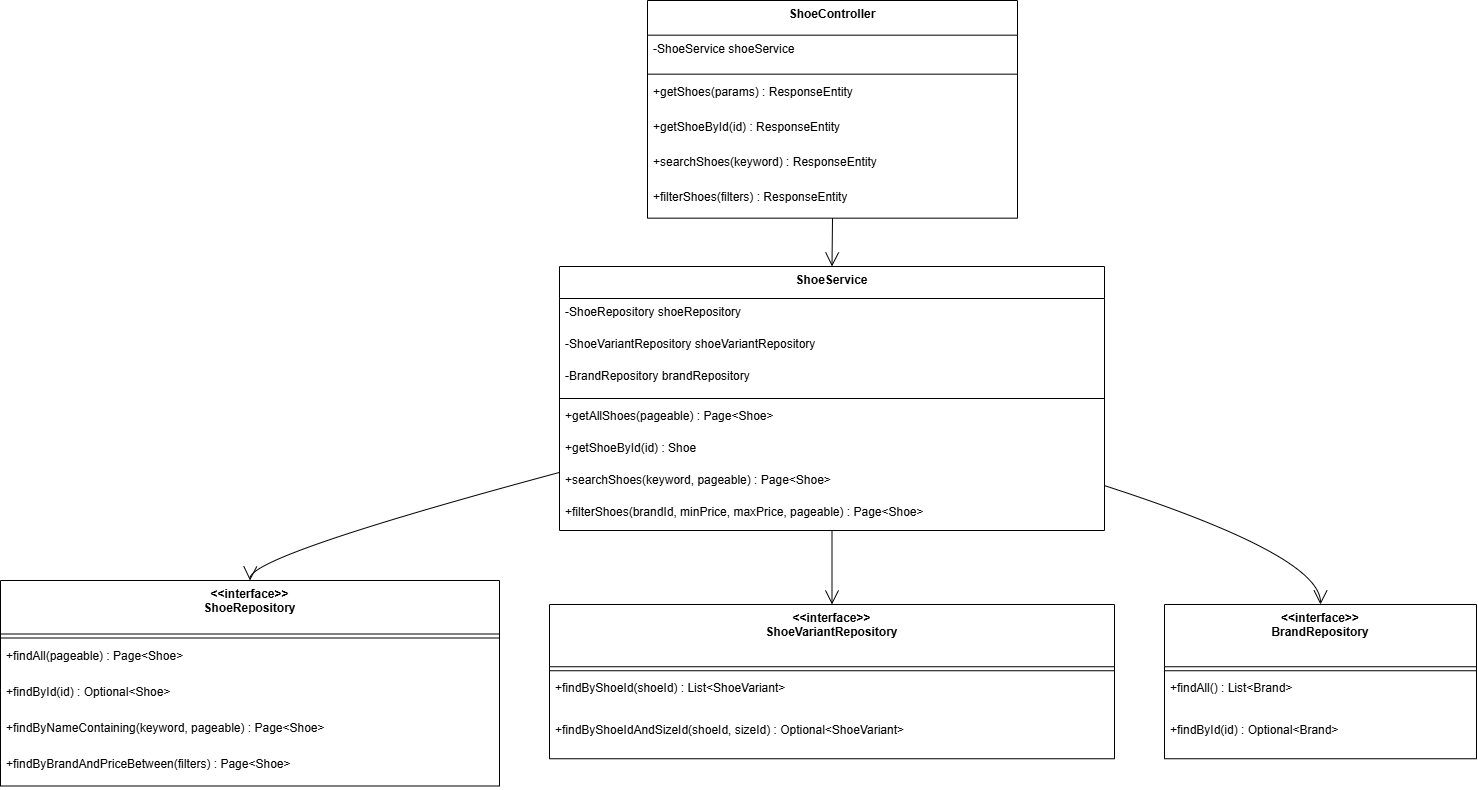
b) Data Flow Analysis

Product Search Sequence Diagram



*Image 14. Sequence Product Search*

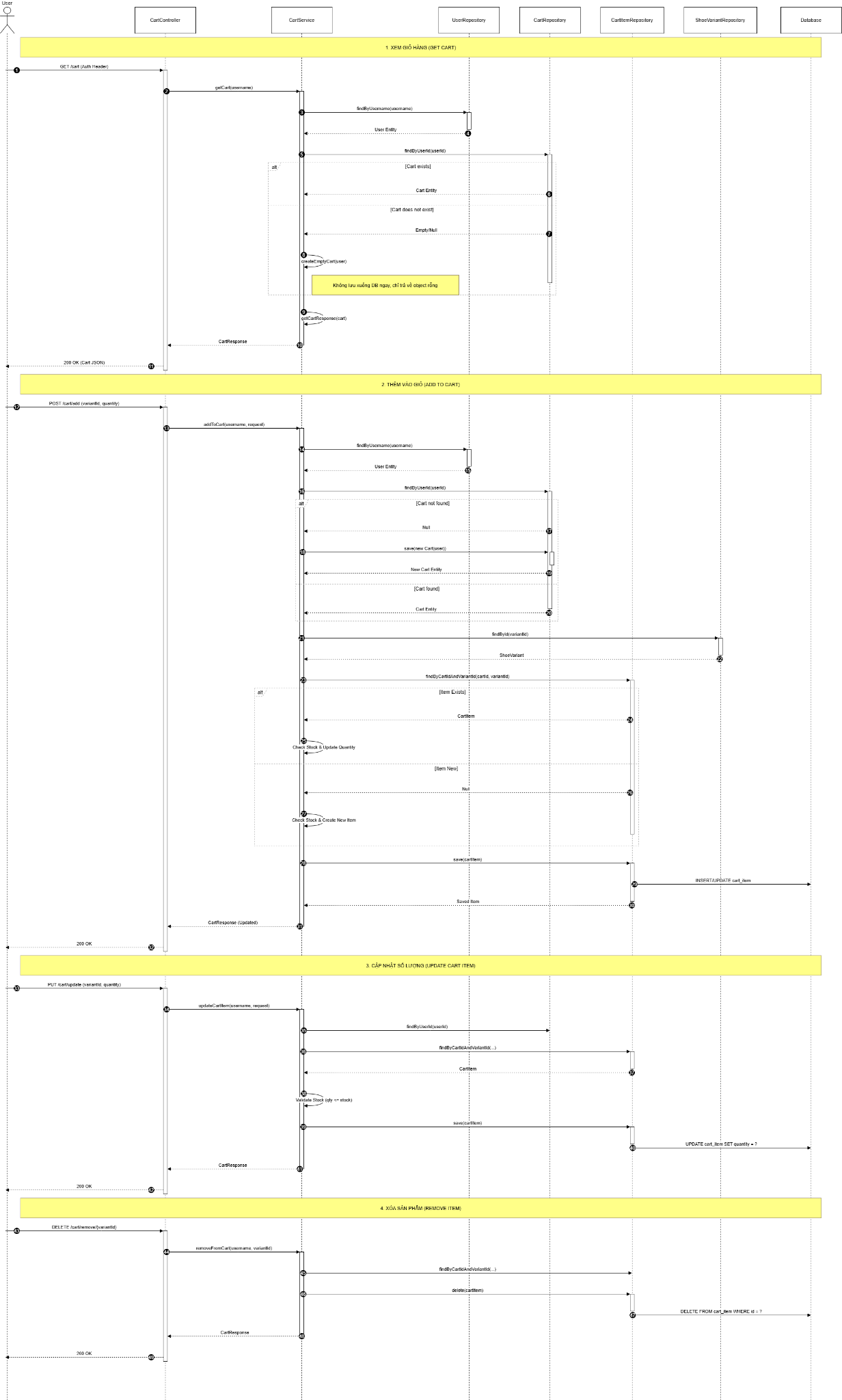
Product Search Class Diagram



*Image 15. Class Diagram Product Search*

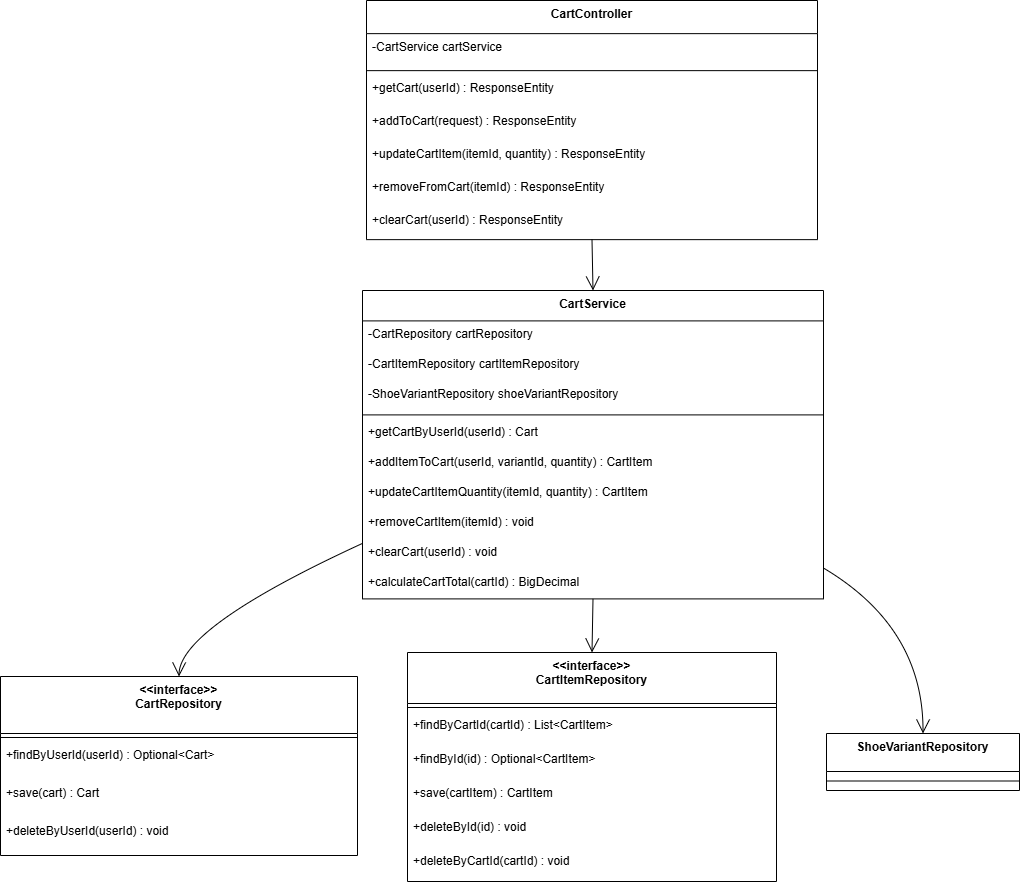
The diagram describes the process where the Customer searches for products using keywords. The Customer enters a keyword and clicks the search button; the Interface validates the keyword. If invalid, an error is displayed immediately. If valid, the Interface sends a request to the ProductController, the controller forwards it to the ProductService for processing, and the service queries the ProductRepository. The Repository executes an SQL statement to the Database to retrieve the list of products containing the keyword. Upon receiving the result, if the list is empty, the system notifies that no products were found; if data exists, the ProductController returns the list to the Interface to display to the Customer.

Cart Functionality Sequence Diagram



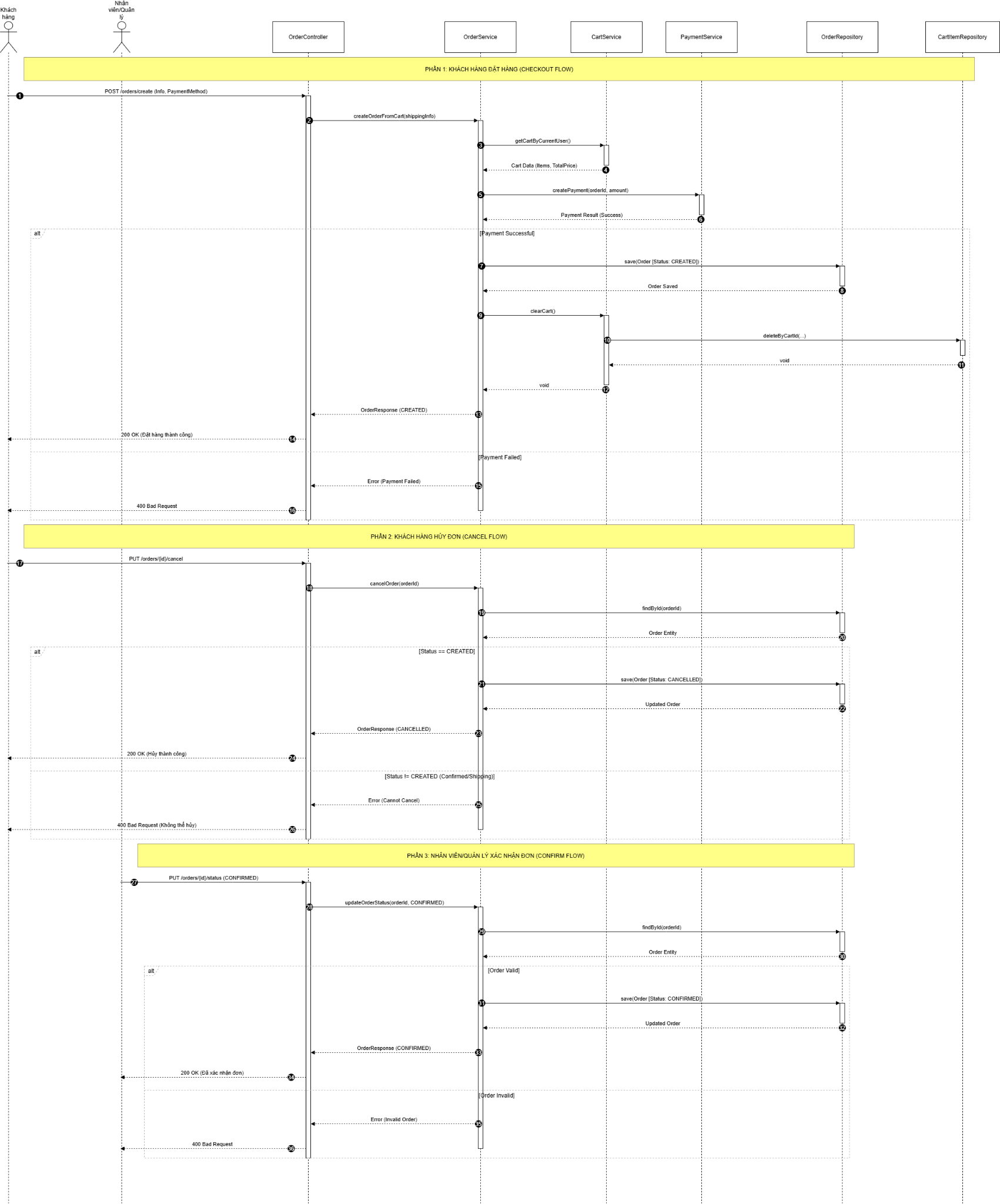
*Image 16. Sequence Cart*

Cart Functionality Class Diagram



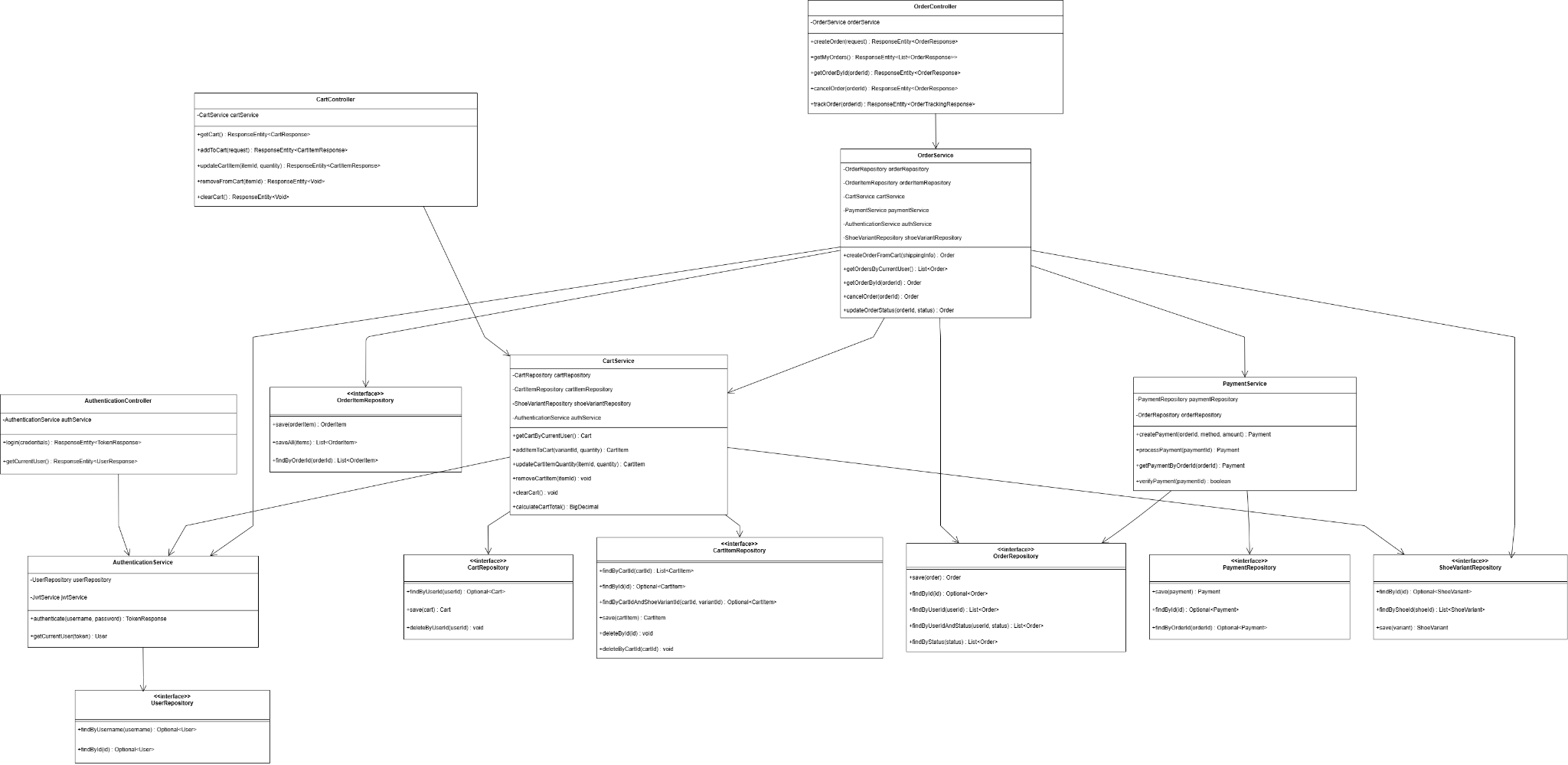
*Image 17. Class Diagram Cart*

* The diagram describes the process where the Customer adds a product to the shopping cart. Upon clicking "Add to Cart", the Interface sends an addToCart request (containing the product code and quantity) to the CartController, which then forwards it to the CartService. The Service verifies the product via the ProductRepository; if the Database does not return the product, the system reports a "Product does not exist" error. If it exists, the CartService retrieves the user's current shopping cart from the CartRepository; if a cart does not exist, the system creates a new one. Next, the Service iterates through the list of products in the cart: if the product already exists, the quantity is increased; otherwise, it is added as a new item. After updating, the CartRepository saves the data to the Database. Upon successful saving, the CartController returns a response for the Interface to display an "Add successful" notification to the Customer.

Order Functionality Sequence Diagram

*Hình 18. Sequence Order*

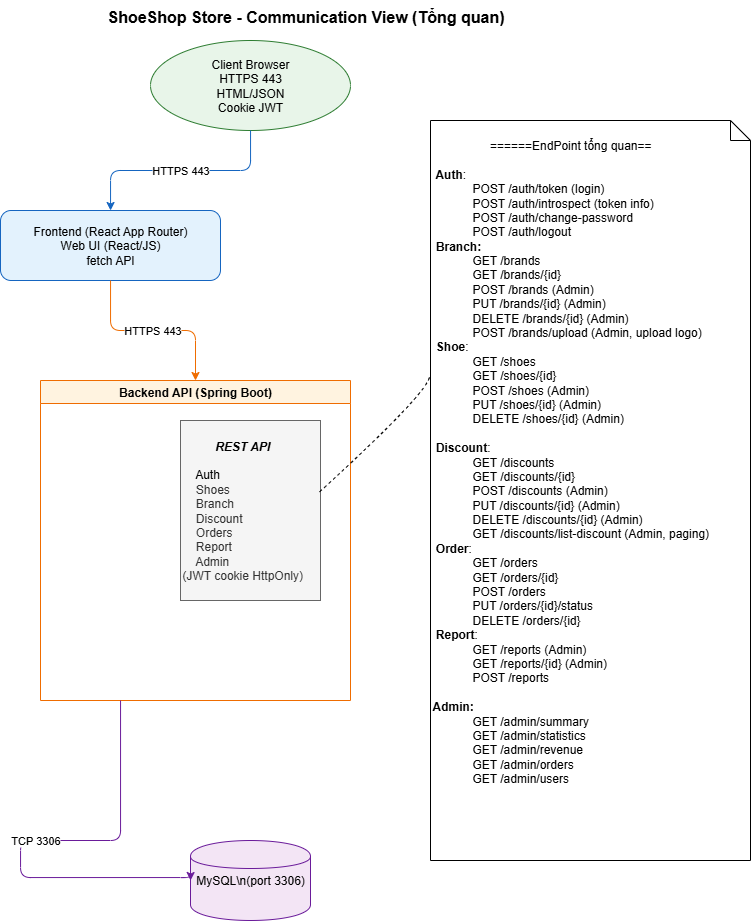
Class diagram Order



*Hình 19. Class Diagram Order*

* The diagram illustrates three primary processes: Customer placing an order, Customer canceling an order, and Staff confirming an order. When the Customer initiates order creation (POST), the OrderController invokes the OrderService; the service retrieves cart data from the CartService and processes the payment via the PaymentService. If the payment is successful, the OrderService saves the order with the status CREATED, clears the shopping cart, and returns 200 OK; if it fails, it returns 400 Bad Request. In the order cancellation process, the Customer submits a cancellation request (PUT cancel). The OrderService locates the order within the OrderRepository and verifies its status. If the order is in the CREATED status, the system updates it to CANCELLED and returns a success response; if it is in a different status, the system reports an error indicating that cancellation is not possible. In the order confirmation process, the Staff sends a request to update the status to CONFIRMED. The OrderService queries the order and validates the necessary conditions (such as inventory availability and payment status). If valid, the order is updated to CONFIRMED and saved to the database; if invalid, the system returns 400 Bad Request.

### **2.2.3. Communication Analysis**

****

*Image 10. Communication View*

# **CHAPTER 3 – TEST PLAN**

CHAPTER 3 OVERVIEW – TEST PLAN

Chapter 3 presents the overall test plan for the Shoeshop system, aiming to clearly define the testing scope, test objects, testing approach, and testing strategies applied throughout the project.

The chapter focuses on describing the testing methodology at a high level, including: testing scope (in-scope/out-of-scope), testing approaches (static – dynamic, white-box – black-box – gray-box), testing levels (Unit Test, Integration Test, System Test, Acceptance Test), acceptance criteria, testing environment, resources, and the Agile-based implementation roadmap.

## **3.1. Test Plan Overview**

### **3.1.1. Test Scope (In-scope/Out-of-scope)**

Testing objectives: Define the scope, strategy, environment, resources, and deliverables of testing activities for the SuperShoe system.

Overall purpose: Ensure that all e-commerce business processes (Product Catalog, Shopping Cart, Order, Discount, User Management, Access Control) operate correctly according to functional and non-functional requirements; minimize risks prior to official release.

Applied standards & tools:

|  |  |
| --- | --- |
| Standards/Tools | Purpose |
| Selenium | Automated UI testing |
| JMeter | Load and performance testing |
| Postman | API testing and backend data validation |

− **Functional scope**: Product Catalog, Shopping Cart, Order, Discount, User Management, Access Control.

− **Non-functional scope**: Performance, Security, Usability, Maintainability, Reliability, Compatibility.

## **3.2. Testing Methodology**

### **3.2.1. Static Testing**

Static testing techniques are performed without executing the program source code. This process focuses on reviewing documentation and analyzing source code structure.

**Objectives:**

- Detect early defects related to logic, design, and missing requirements.

- Ensure that the source code complies with coding conventions.

- Minimize defect fixing costs before moving to the dynamic testing phase.

### **3.2.2. Dynamic Testing**

Dynamic testing involves executing the ShoeShop system source code in a real environment to detect defects and verify functionality.

Approach: Dynamic testing is implemented through the combination of three strategies:

Black-box Testing:

* Validate functionality based on User Stories and the user interface (UI) without considering the internal code structure.
* Applied to ShoeShop: Frontend ReactJS interactions: Add to cart, product filtering, responsive display. Business workflows (End-to-End): Checkout process, discount code application, role-based access control (Customer/Staff/Admin/Manager).

White-box Testing:

* Examine internal structure, logical flows, and conditional branches within the source code.
* Applied to ShoeShop: Unit Testing: Use JUnit 5 to test business logic functions in the Spring Boot Backend. Business logic: Validate total price calculation algorithms, discount logic, and JWT token security.

Grey-box Testing:

* Combine external functional testing with an understanding of internal data structures.
* Applied to ShoeShop: API Testing: Use Postman to test HTTP methods (GET/POST/PUT). Database: Verify data integrity in MySQL (foreign key constraints, inventory deduction, order status updates).

Execution Types:Manual Testing: Manual testing for UI and complex business workflows.Automation Testing: Use Selenium (UI) and JUnit (Backend) for regression testing.

## **3.3. Test Items**

**Functional Requirements**

**1. Product Catalog**

– Verify that the system displays the product list with complete details: name, description, price, brand, size, and inventory status.

– Verify that the system allows customers to search products by keywords, filter by category/brand, and sort by price/name.

– Verify that the system accurately displays images and detailed information when users view a specific product.

– Verify that the system allows Managers to add, edit, delete (or hide) products and update inventory.

– Verify that the system displays appropriate error messages when invalid product data is entered (e.g., negative price, missing name).

**2. Shopping Cart & Checkout**

– Verify that the system allows customers to add products to the cart from the product detail page.

– Verify that the system correctly displays the list of products in the cart, including unit price, quantity, and subtotal.

– Verify that the system automatically updates the total payment amount immediately when users change quantities or remove products from the cart.

– Verify that the system allows users to apply valid discount codes and rejects invalid or expired codes.

– Verify that the system accurately stores and synchronizes the cart state when users log in again or reload the page.

**3. Order Management**

– Verify that the system allows customers to complete checkout with full delivery information and selected payment methods (COD/VNPay).

– Verify that the system correctly processes online payment transactions and updates the corresponding order status.

– Verify that the system allows customers to view order history and detailed processing status of each order.

– Verify that the system only allows customers to cancel orders when the order is in a valid status (Created/Confirmed).

– Verify that the system allows Staff/Managers to update order status according to the defined business workflow (Confirmed → Preparing → Delivery → Delivered).

– Verify that the system automatically deducts inventory quantity when an order is confirmed and restores inventory if the order is canceled.

**4. Discount Management**

– Verify that the system allows Managers to create new discount codes with parameters such as discount value (amount/percentage), validity period, and usage limit.

– Verify that the system allows editing or disabling active discount codes.

– Verify that the system correctly applies discounts to eligible orders and prevents application to ineligible orders.

**5. User Management**

– Verify that the system allows customers to update personal profile information (Full name, address, phone number) and change passwords.

– Verify that the system allows Admins to view the full user list and search accounts based on criteria.

– Verify that the system allows Admins to disable or enable user accounts and prevents disabled accounts from logging in.

**6. Access Control**

– Verify that the system allows login for valid accounts (Customer, Staff, Manager, Admin).

– Verify that the system allows customers to register new accounts and displays appropriate error messages for invalid information (duplicate email, incorrect format).

– Verify that the system allows secure logout and terminates the current session for all users.

– Verify that the system enforces correct access control, ensuring users can only access pages and APIs appropriate to their assigned roles.

– Verify that the system prevents unauthorized access to administrative pages from customer accounts or unauthenticated users.

### **7. Usability**

* Verify that the system provides clear navigation menus, allowing users to access main pages such as home, product list, cart, and orders.
* Verify that users can easily navigate back to the previous page or home page during usage.
* Verify that primary functional buttons are clearly displayed and easy to interact with.

**8. Compatibility**

* Verify that the website interface displays correctly and consistently across popular browsers such as Chrome, Firefox, and Safari.
* Verify that the system automatically adjusts the interface to fit different screen sizes (Desktop, Tablet, Mobile).
* Verify that UI components do not break or misalign when screen size changes.
* Verify that users can fully interact with system functionalities on different devices.

**9. User Interface**

* Verify that colors, fonts, and display styles are consistent with the design specifications.
* Verify that logos, images, and icons are displayed in correct positions without distortion.
* Verify that error messages, warnings, and status information are displayed appropriately according to business context.

### **10. Performance**

* Verify that the system responds quickly under normal load conditions.
* Verify that the system remains stable under high concurrent user load (load testing).
* Verify that the system can withstand peak load without crashing (stress testing).

**11. API**

* Verify that API endpoints function correctly according to design specifications.
* Verify that data returned from APIs is accurate, complete, and conforms to the required format.
* Verify that API access control and security are enforced, preventing unauthorized access.
* Verify that APIs interact correctly with the database, UI interface, and related endpoints.
* Verify that core business workflows executed via APIs remain stable after system changes.

## **3.4. Out-of-Scope Test Items**

### **3.4.1. Third-Party System Functions**

Internal processing functions of the VNPay payment gateway are outside the testing scope of this project. The team does not perform testing on business logic, payment processing algorithms, or internal security mechanisms managed by VNPay.

* The team’s testing scope only focuses on integration flows between the ShoeShop system and the VNPay payment gateway (sandbox/demo) via APIs, including:
* Sending payment transaction creation requests from the ShoeShop system to VNPay.
* Receiving and processing response results from VNPay (success, failure, transaction cancellation).
* Verifying the correctness of response data and updating order status within the ShoeShop system.

### 3.4.2. Advanced Security Testing

Advanced security testing activities, including penetration testing and in-depth security assessments, are not included in the scope of this project.

The team does not conduct security vulnerability assessments at the following levels:

* Operating system and server infrastructure.
* Network, firewall, and system security configurations.
* Internal security mechanisms of third-party services.

The team only performs basic security testing, focusing on access control and user authorization.

## **3.5. Test Strategy**

The project is implemented following the V-Model in software testing, in which testing activities are conducted in parallel with each development phase, enabling quality control throughout the software lifecycle.

Each development phase is associated with a corresponding testing level, including unit testing, integration testing, system testing, and acceptance testing, to ensure that the product fully meets the defined requirements.

Test Cases are developed based on functional requirements and system Use Cases. In addition, the project applies automated testing within the CI/CD pipeline using GitHub Actions, which automates the Build, Test, and Deploy stages, thereby improving the stability and efficiency of the development process.

### **3.5.1. (Test Types)**

The table below summarizes the test types applied in the project, along with their objectives and corresponding acceptance criteria.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Type** | **Objective** | **Test Content** | **Acceptance Criteria** |
| Functional Testing | Ensure the system operates correctly according to business requirements and Use Cases (BR1 – BR5) | Automated functional testing  Unit Testing using JUnit, Mockito  Integration Testing using Spring Boot Test, MockMvc, H2  System Testing using Selenium | * All functional Test Cases pass * No critical or blocking defects (Blocker/Critical) exist |
| UI/UX Testing | Evaluate user experience, navigation, and interface consistency of the ReactJS UI | - Verify interface layout, structure, and colors  - Verify navigation between screens  - Basic UI testing using Selenium on selected pages  - Manual testing across multiple browsers and devices | Interface displays correct layout.  Stable operation across common browsers and devices |
| API Testing | Ensure accurate, secure, and RESTful-compliant communication between Client and Server | Verify HTTP methods (GET, POST, PUT, DELETE)  Verify status codes (200, 400, 401, 403, 500)  Verify returned JSON structure  Verify Token/JWT authentication | APIs operate correctly according to business logic  Appropriate error codes returned for invalid input  Invalid tokens are blocked (401/403)  Response time < 1 second (internal network) |
| Performance Testing | Ensure stable response time, latency, and resource utilization  Evaluate system load handling under concurrent users | Load testing: system operates under expected load  Stress testing: test beyond capacity to identify limits  Load testing and stress testing using K6 | System handles expected load well (e.g., 100 users)  No crashes under overload  System degrades gracefully when exceeding capacity |
| Crowd Testing | Detect defects, compatibility issues, and evaluate real user experience | Manual testing across multiple devices and browsers  Collect feedback from multiple testers | Real users can perform full system functions  No critical defects  System operates stably and consistently |
| Regression Testing | Ensure stable functionalities are not affected after updates or bug fixes | Automated testing integrated with CI/CD and Selenium | Regression Test Case pass rate ≥ 98%  GitHub Actions pipeline executes successfully |

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Objective** | **Technique** | **Method** |
| Unit Testing | Verify the correctness of functions/methods | White-box | Automated testing using JUnit and Mockito in the Development environment |
| Integration Testing | Verify interactions between functional modules | White-box | Automated testing using Spring Boot Test, MockMvc, and H2 in the Development environment |
| System Testing | Verify business workflows and user interface | Black-box | Automated testing using Selenium (UI, E2E) and K6 (performance) in the Development environment |
| Acceptance Testing | Verify and approve the product | Black-box | Manual testing in the Production environment |

### **3.5.2. Test Stage Matrix**

The table below defines the testing stages at which each test type is applied

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test types** | **Static** | **Unit** | **Integration** | **System** | **Acceptance** |
| **Functional Testing** |  | X | X | X | X |
| **UI/UX & Usability Testing** |  |  |  | X | X |
| **API Testing** |  |  | **X** | **X** |  |
| **Performance Testing** |  |  |  | X | X |
| **Load & Stress Testing** |  |  |  | X |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Test Level** | | | |
| **Unit Testing** | **Integration Testing** | **System Testing** | **Acceptance Testing** |
| Functional Testing | x | x | x | x |
| UI Testing |  |  | x | x |
| Performance Testing |  |  | x |  |
| Crowd Testing |  |  | x | x |
| Regression Testing | x | x | x |  |
| API Testing | x | x |  |  |

## **3.6. Test Environment**

### **3.6.1 Hardware**

|  |  |  |
| --- | --- | --- |
| **Hardware** | **Quantity** | **Usage** |
| Application Server | 2 | Deploy backend (Spring Boot) |
| Database Server | 1 | MySQL test DB |
| Client Test Devices | 5 | PC for Tester, Chrome/Edge/ test |

### **3.6.2 Software**

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Purpose** |
| **Hệ điều hành** | Windows 10,11 (client) | Window cho server & client |
| **Backend** | Spring Boot 3.x | API backend |
| **Frontend** | React + Vite | UI web app |
| **Database** | MySQL 8.0 | Test data storage |
| **CI/CD** | Docker | Automated build/test/deploy |
| **Automation Test** | Selenium 4.x | UI testing |
| **Performance Test** | K6 | Load/Stress test |
| **Unit Test** | JUnit 5.x | Backend unit testing |

### **3.6.3. Human Resources**

Assign specific responsibilities to members of the testing team (QA Team):

|  |  |
| --- | --- |
| **Member** | **Role & Key Responsibilities** |
| **Võ Thành Danh** (Test Lead) | Develop and approve the Test Plan.  Assign tasks to testers.  Design and execute Test Cases for the Access Control module. |
| **Đỗ Phú Thành** (Tester) | Design and execute Test Cases for the Order Management module.  Prepare the Test Summary Report. |
| **Huỳnh Minh Quân** (Tester) | Responsible for the Discount Management module.  Validate the correctness and logic of discount codes. |
| **Huỳnh Duy Khang** (Tester) | Responsible for the Product Catalog and Shopping Cart modules.  Verify product display and shopping cart calculations. |

### 

# **CHAPTER 4 TEST DESIGN**

## **4.1 Introduction**

This chapter presents the test design process and the results of developing test cases for the ShoeShop system based on the V-Model testing approach. The test case design is carried out based on the documents analyzed in Chapter 1 (System Overview), Chapter 2 (System Analysis and Design), and Chapter 3 (Testing Strategy and Scope). The objective of this chapter is to clarify the relationship between requirements, system design, and testing, while demonstrating that all functionalities of the ShoeShop system are verified through appropriate testing levels in accordance with the V-Model.

## **4.2. Test Design Process Based on the V-Model**

The team’s test design process is fully developed based on the V-Model, a testing model that emphasizes the parallelism and symmetry between software development phases and their corresponding testing levels.

In this model, the analysis and design activities on the left side of the “V” are systematically validated by the testing activities on the right side. This approach ensures that all requirements and design decisions can be verified through concrete test cases, thereby improving the overall quality and reliability of the system

### **4.2.1. Requirements Analysis – 1a**

In the initial phase, the team conducts the collection, analysis, and clarification of the business requirements for the ShoeShop online shoe retail system.The objective of this phase is to ensure that the requirements are fully specified, within the defined scope, free of inconsistencies, and testable.

Based on the analysis results, the team develops the Software Requirements Specification (SRS), which serves as the foundation for all subsequent design and testing activities. The system is identified as having four main user roles: Customer, Staff, Manager, and Admin, with key functional modules including Product Management, Shopping Cart, Order Management, and Payment.

The outcome of the requirements analysis phase provides the basis for constructing acceptance test cases, in which each business requirement is transformed into acceptance criteria to verify that the system meets user expectations.

### **4.2.2. System Design – 2a**

After the requirements are clearly defined, the team proceeds with the overall system design. This phase focuses on describing the core functionalities, business workflows, use case diagrams, and the interactions between users and the ShoeShop system.

The information obtained from the system design phase is used to construct system test scenarios. Each use case, representing one or more business workflows, becomes a testing target to ensure that the system operates correctly in accordance with the designed business scenarios.

### **4.2.3. Architecture Design – 3a**

During the architecture design phase, the team applies the C4 Model (Context – Container – Component – Code) to describe and decompose the ShoeShop system from a high-level overview to detailed technical structures.

The architecture design clearly defines the following aspects:

– The separation of responsibilities between the frontend, backend, and database layers.

– The interaction points between modules through APIs.

– The data flow and communication mechanisms among system components.

The outcomes of this phase provide a critical foundation for integration testing, in which the interaction points between components are analyzed and transformed into test cases to evaluate the correctness and consistency of exchanged data.

### **4.2.4. Module Design – 4a**

In the module design phase, the team performs a detailed analysis of each functional module of the system. Class diagrams are used to describe data structures, attributes, methods, and the relationships among classes within each module.

These artifacts serve as the basis for identifying critical components that need to be tested at the unit level. A clear module design enables the team to develop unit test cases aimed at detecting defects early at the lowest level, before the modules are integrated with one another.

### **4.2.5. Unit Testing – 1b**

Based on the module design, the team develops and executes unit tests to evaluate the correctness of individual methods, classes, or functions in isolation.

The objective of unit testing is to ensure that the smallest components of the system operate correctly according to the design, thereby minimizing defects that may propagate to integration and higher-level testing phases.

Unit test cases are implemented using JUnit in combination with Mockito to mock dependent objects. The use of mocking techniques helps isolate the test logic from external components such as databases or other services, ensuring test accuracy and repeatability.

All unit tests are integrated into the CI/CD pipeline through GitHub Actions, allowing tests to be automatically executed whenever source code changes occur. This integration supports early defect detection and continuous maintenance of software quality..

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Function Name** | **Description** | **Input Data** | **Expected Result** | **Actual Result** | **Status** |
| addToCart\_existingItemExceedStock\_shouldThrowOutOfStock | Add an existing item exceeding available stock | username='unit\_test\_user', variantId='variant-001', oldQty=2, addQty=9, stock=10 | Throw AppException with ErrorCode.OUT\_OF\_STOCK | Exception: Product is out of stock (OUT\_OF\_STOCK) | PASS |
| addToCart\_exceedStock\_shouldThrowAppException | Add to cart exceeding available stock | username='unit\_test\_user', variantId='variant-001', quantity=15, stock=10 | Throw AppException withi ErrorCode.OUT\_OF\_STOCK | Exception: Product is out of stock (OUT\_OF\_STOCK) | PASS |
| addToCart\_userNotFound\_shouldThrowUserNotExisted | Add to cart when the user does not exist | username='unknown\_user' | Throw AppException withi ErrorCode.USER\_NOT\_EXISTED | Exception: User does not exist (USER\_NOT\_EXISTED) | PASS |
| addToCart\_variantNotFound\_shouldThrowVariantNotFound | Add to cart when the variant does not exist | username='unit\_test\_user', variantId='non\_existent\_variant', quantity=2 | Throw AppException with ErrorCode.VARIANT\_NOT\_FOUND | Exception: Variant Not Found (VARIANT\_NOT\_FOUND) | PASS |
| addToCart\_existingItem\_shouldUpdateQuantity | Add an item that already exists in the cart | username='unit\_test\_user', variantId='variant-001', oldQty=2, addQty=3, stock=10 | Update quantity from 2 to 5 | Updated existing item. New Quantity: 5 | PASS |
| addToCart\_newItem\_shouldSaveCartItem | Add a new item to the cart | username='unit\_test\_user', variantId='variant-001', quantity=2, stock=10 | Save a new CartItem with quantity = 2 | Saved new CartItem. Quantity: 2, Total Price: 400.0 | PASS |

### **4.2.6. Integration Testing – 2b**

After the modules have been verified as stable through unit testing, the team conducts integration testing to evaluate the interactions among system components within the ShoeShop system.

Integration testing focuses on verifying the following aspects:

– The processing flow between the Controller, Service, and Repository layers.

– Data transmission and communication through APIs.

– Compatibility between the backend and the database.

The team uses Spring Boot Test in combination with MockMvc to test APIs in a simulated environment that closely resembles real-world conditions. The production database is replaced with an in-memory H2 Database to ensure test independence, repeatability, and to prevent any impact on real data.

Integration test cases are designed based on the interaction points identified during the architecture design phase, ensuring comprehensive coverage of critical system interaction flows..

### **4.2.7. System Testing – 3b**

At a higher level, the team performs system testing to evaluate the entire ShoeShop system as an integrated whole. System testing is primarily conducted using manual testing, based on the test cases derived from use cases and key business workflows.

The testing system activities include:

– Functional testing: Verifying end-to-end business workflows such as login, purchasing, and payment.

– UI testing: Ensuring that the user interface is intuitive, consistent, and meets display and usability standards.

– Basic security testing: Validating login flows, role-based authorization, and access control mechanisms.

The results of this phase reflect the overall quality of the system before proceeding to acceptance testing.

### **4.2.8. Acceptance Testing – 4b**

Acceptance testing is the final phase of the testing process, aiming to evaluate the system from the end-user perspective. At this stage, acceptance criteria are directly validated against the Software Requirements Specification (SRS) to confirm that the ShoeShop system fully satisfies the defined requirements.

After completing internal testing, the system is handed over to the Product Owner to conduct acceptance testing. The Product Owner acts as a representative of end users, simulating real-world usage scenarios and evaluating the system in terms of business requirement fulfillment, usability, and overall stability.

The results of acceptance testing provide a critical basis for making the final decision on deploying the ShoeShop system into the production environment.

.

## **4.3. Test Design Techniques**

### **4.3.1. White-Box Testing Technique**

#### Test-case add product to cart

public CartResponse addToCart(String username, AddToCartRequest request) {

log.info("Adding item to cart for user: {}", username);

User user = userRepository.findByUsername(username)

.orElseThrow(() -> new AppException(ErrorCode.USER\_NOT\_EXISTED));

Cart cart = cartRepository.findByUserId(user.getId())

.orElseGet(() -> {

Cart newCart = Cart.builder()

.user(user)

.build();

return cartRepository.save(newCart);

});

ShoeVariant variant = shoeVariantRepository.findById(request.getVariantId())

.orElseThrow(() -> new AppException(ErrorCode.VARIANT\_NOT\_FOUND));

CartItem cartItem = cartItemRepository

.findByCartIdAndVariantId(cart.getId(), variant.getId())

.orElse(null);

if (cartItem != null) {

int newQuantity = cartItem.getQuantity() + request.getQuantity();

if (newQuantity > variant.getStockQuantity()) {

throw new AppException(ErrorCode.OUT\_OF\_STOCK);

}

cartItem.setQuantity(newQuantity);

} else {

if (request.getQuantity() > variant.getStockQuantity()) {

throw new AppException(ErrorCode.OUT\_OF\_STOCK);

}

cartItem = CartItem.builder()

.cart(cart)

.variant(variant)

.quantity(request.getQuantity())

.build();

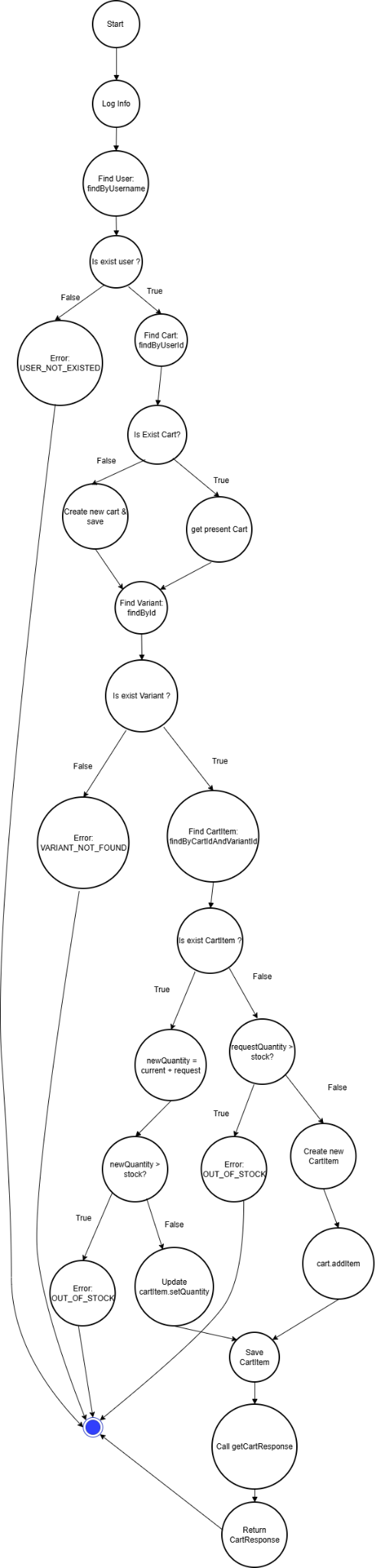
cart.addItem(cartItem);

}

cartItemRepository.save(cartItem);

return getCartResponse(cart);

}



*Image 21.*  *WorkFlow add product to Cart*

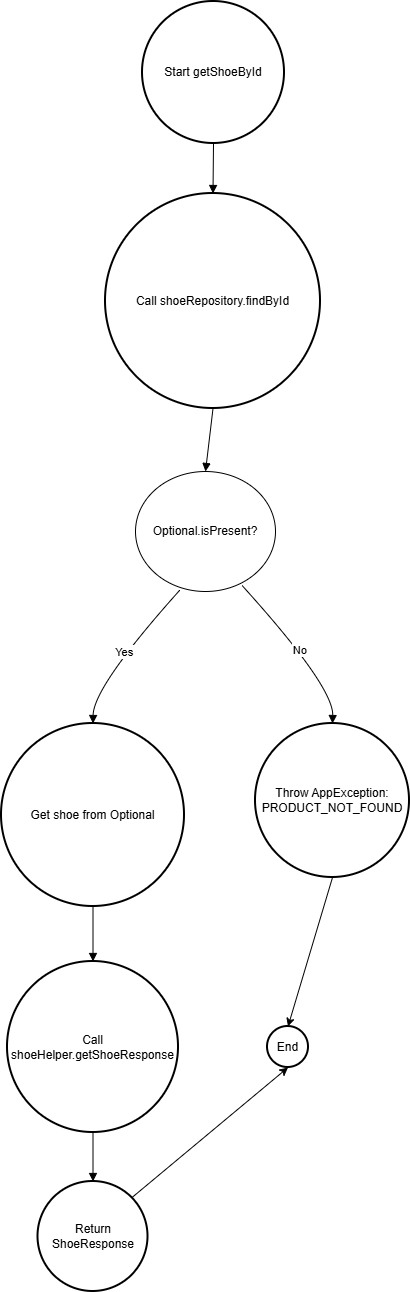
From the graph above, there are six main independent paths:

* User not exists
* Variant not exists
* New item, quantity ≤ stock
* New item, quantity > stock
* Existing item, newQuantity ≤ stock
* Existing item, newQuantity > stock

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Function Name** | **Description** | **Input Data** | **Expected Result** | **Actual Result** | **Status** |
| addToCart\_existingItemExceedStock\_shouldThrowOutOfStock | Add an existing item exceeding available stock | username='unit\_test\_user', variantId='variant-001', oldQty=2, addQty=9, stock=10 | Throw AppException with ErrorCode.OUT\_OF\_STOCK | Exception: Product is out of stock (OUT\_OF\_STOCK) | PASS |
| addToCart\_exceedStock\_shouldThrowAppException | Add to cart exceeding available stock | username='unit\_test\_user', variantId='variant-001', quantity=15, stock=10 | Throw AppException with ErrorCode.OUT\_OF\_STOCK | Exception: Product is out of stock (OUT\_OF\_STOCK) | PASS |
| addToCart\_userNotFound\_shouldThrowUserNotExisted | Add to cart when the user does not exist | username='unknown\_user' | Throw AppException with ErrorCode.USER\_NOT\_EXISTED | Exception: User does not exist (USER\_NOT\_EXISTED) | PASS |
| addToCart\_variantNotFound\_shouldThrowVariantNotFound | Add to cart when the variant does not exist | username='unit\_test\_user', variantId='non\_existent\_variant', quantity=2 | Throw AppException with ErrorCode.VARIANT\_NOT\_FOUND | Exception: Variant Not Found (VARIANT\_NOT\_FOUND) | PASS |
| addToCart\_existingItem\_shouldUpdateQuantity | Add an item that already exists in the cart | username='unit\_test\_user', variantId='variant-001', oldQty=2, addQty=3, stock=10 | Update quantity from 2 to 5 | Updated existing item. New Quantity: 5 | PASS |
| addToCart\_newItem\_shouldSaveCartItem | Add a new item to the cart | username='unit\_test\_user', variantId='variant-001', quantity=2, stock=10 | Save a new CartItem with quantity = 2 | Saved new CartItem. Quantity: 2, Total Price: 400.0 | PASS |

1. View details Product

public ShoeResponse getShoeById(int id) { Shoe shoe = shoeRepository.findById(id) .orElseThrow(() -> new AppException(ErrorCode.PRODUCT\_NOT\_FOUND)); return shoeHelper.getShoeResponse(shoe); }



*Image 22. WorkFlow view details prodcut*

The graph contains two independent paths:

* Product is found and detailed information is retrieved.
* Product is not found

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Function Name** | **Description** | **Input Data** | **Expected Result** | **Actual Result** | **Status** |
| getShoeById \_WhenNotExists \_ShouldThrowException | Retrieve a shoe using a non-existent ID | shoeId = 999 | Throw an AppException with ErrorCode.PRODUCT\_NOT\_FOUND | Exception: Product Not Found (PRODUCT\_NOT\_FOUND) | **PASS** |
| getShoeById \_WhenExists \_ShouldReturnShoe | Retrieve a shoe using an existing ID | shoeId = 1, name = "Air Jordan" | Return a ShoeResponse with name **"Air Jordan"** | Retrieved Shoe Name: Air Jordan | **PASS** |

### **4.3.2. Black-Box Testing Technique**

Black-box testing is used to verify system functionality based on business requirements without considering the internal structure or implementation of the software. In the ShoeShop project, this technique is applied primarily to system testing and acceptance testing, as the objective is to ensure that the software behaves as expected from the end-user perspective and satisfies the defined acceptance criteria.

The use of black-box testing enables the team to simulate real-world usage scenarios and validate complete end-to-end business workflows. This approach is well suited for both manual execution and UI-level automation testing.

**Application to Cart Management Test Cases**

The team applies two main black-box testing techniques to the Cart Management module:

*1. Decision Table Technique*

In the ShoeShop system, the following business rules are identified for the Cart Management module:

1. The user must be authenticated (logged in) to add products to the cart.
2. The product must be available in stock.
3. The requested quantity must be valid and must not exceed the available stock.
4. The discount code must be valid and satisfy all applicable conditions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rule** | **R1(Valid Add to Cart)** | **R2(Out of Stock)** | **R3(Invalid Quantity)** | **R4(Valid Discount Applied** |
| **Conditions** |  |  |  |  |
| **User is authenticated (logged in)** | ✓ | ✓ | ✓ | ✓ |
| **Product is in stock** | ✓ | ✗ | ✓ | ✓ |
| **Quantity is valid** | ✓ | – | ✗ | ✓ |
| **Discount code is valid** | – | – | – | ✓ |
| **Discount conditions are satisfied** | – | – | – | ✓ |
| **Result** |  |  |  |  |
| **System processes the request** | Add to cart successfully | Display out-of-stock error | Display quantity validation error | Apply discount successfully |

* **TC\_DT\_001: Add valid product → Success**
* **TC\_DT\_002: Add out-of-stock product → Error**
* **TC\_DT\_003: Quantity exceeds available stock → Error**
* **TC\_DT\_004: Apply valid discount → Success**

*2. State Transition Testing Technique*

Cart States in the ShoeShop System:

– Empty Cart (S1): The cart contains no items.

– Has Items (S2): The cart contains one or more products.

– Ready for Checkout (S3): The cart meets all required conditions and is ready for checkout.

– Error State (S4): The cart enters an error state due to invalid actions or business rule violations.

* 

*Image 23. State Transition*

State Transition Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Event** | **Action** | **Next State** |
| Empty Cart | addItem (valid)) | Add product and recalculate total amount | Has Items |
| Empty Cart | addItem (invalid) | Display error message | Error State |
| Has Items | updateQty | Update quantity and recalculate total amount | Has Items |
| Has Items | removeItem | Remove product and recalculate total amount | Empty Cart |
| Has Items | checkout | Prepare checkout data | Ready for Checkout |
| Error State | retry | Reset cart state | Empty Cart |

## **Test Cases Derived from the State Transition Table**

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Test Flow** | **Description** |
| TC\_ST\_001 | Empty Cart → addItem → Has Items | Add a valid product to an empty cart |
| TC\_ST\_002 | Has Items → updateQty → Has Items | Update product quantity when the cart already contains items |
| TC\_ST\_003 | Has Items → removeItem → Empty Cart | Remove a product from the cart |
| TC\_ST\_004 | Has Items → checkout → Ready for Checkout | Proceed to the checkout process |

## **4.4. Test Design Approaches**

### **4.4.1. Manual Testing**

In the ShoeShop project, manual testing is primarily applied during the acceptance testing phase to evaluate the system from the end-user perspective. This approach allows direct validation of core business workflows and verification of how well the system satisfies the requirements specified in the Software Requirements Specification (SRS).

Manual test cases are executed based on the predefined Test Cases and Review Checklists. During execution, actual results are recorded, compared, and validated against the expected results to assess the completion level of each requirement before the system acceptance process is performed.

### **4.4.2. Automated Testing**

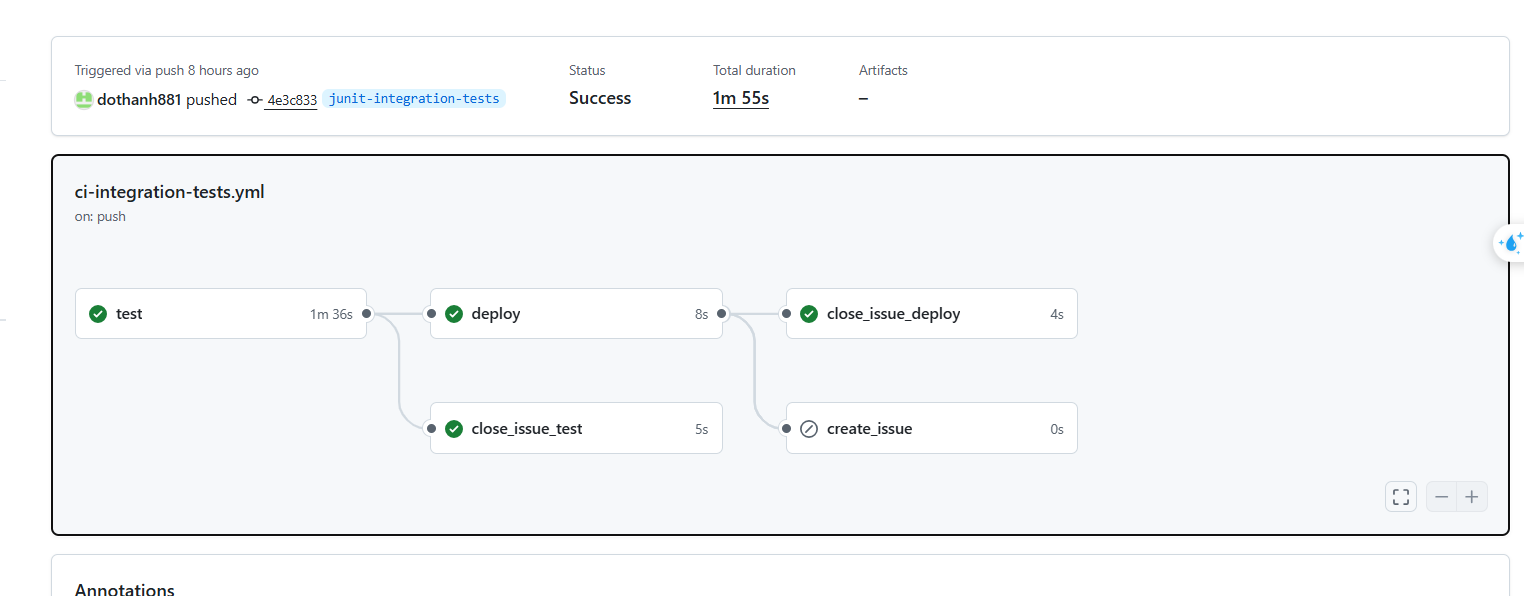
In addition to manual testing, the ShoeShop project applies automated testing by integrating a CI/CD pipeline using GitHub Actions to automate the build, test, and deployment processes.

Automated testing is used primarily for white-box testing activities, including unit testing, integration testing, and partial system testing. Whenever source code changes are pushed to the GitHub repository, the CI/CD pipeline is automatically triggered to execute testing jobs. These jobs build the project and run all predefined automated test suites.

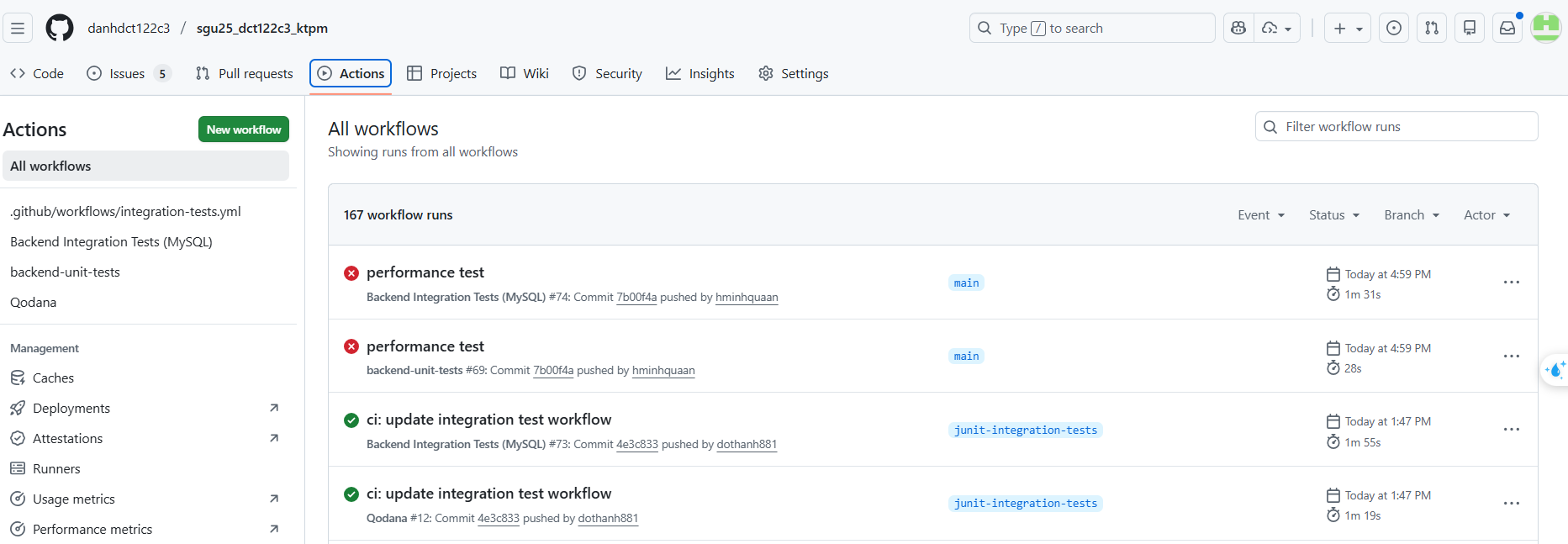
If a testing job fails, the pipeline automatically creates a GitHub Issue to record the detected defects, enabling the development team to track and resolve issues in a timely manner.

If the testing job succeeds, the pipeline automatically updates the build status and closes related issues associated with previous testing failures, ensuring that system quality information is accurately reflected.

When the source code is merged into the main branch, the CI/CD pipeline continues by triggering deployment jobs, which connect to the Render platform to redeploy the ShoeShop system to the operational environment.



*Image 24. CI/CD Pipeline Integration Test*



*Image 25. List of GitHub Actions workflow runs*

## **4.5. Applying Generative AI in Test Case Generation**

In this project, Generative AI (GenAI) is applied as a supporting tool during the test case design phase. GenAI does not replace the role of testers; instead, it assists in requirements analysis and in generating initial test scenarios and test cases. All outputs produced by GenAI are reviewed, refined, and approved by testers before being officially used in the testing process.

The application of GenAI provides the following benefits: – Reducing the time required for manual test case design. – Expanding test coverage. – Minimizing the risk of missing boundary cases and error scenarios.

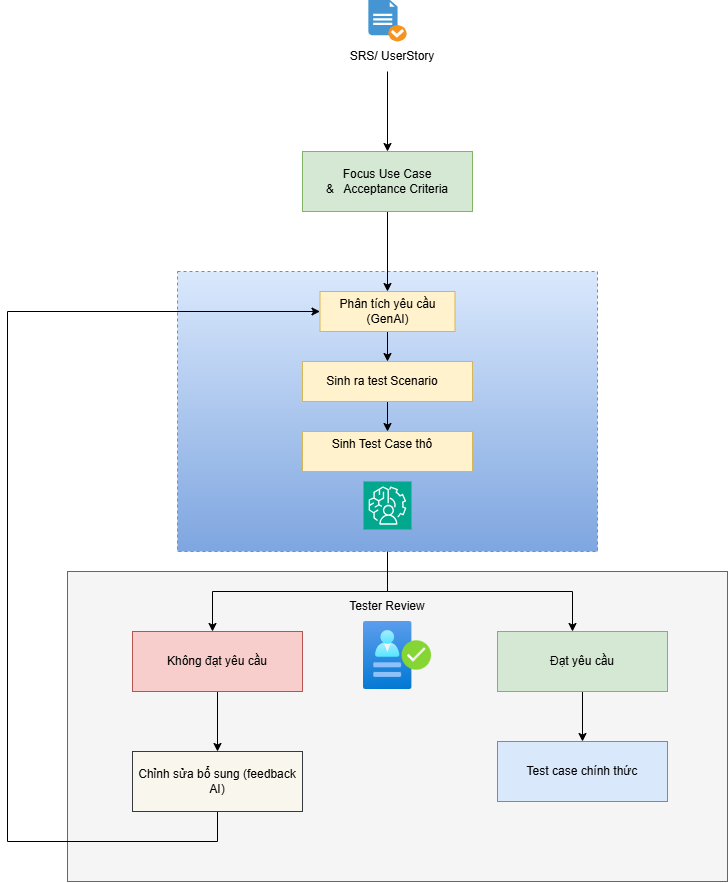
### **4.5.1. GenAI Inputs**

The inputs provided to GenAI include the system’s business and technical documentation, specifically: – Business requirement documents. – Software Requirements Specification (SRS). – User Stories and Use Cases.

These documents provide essential information regarding: – System functionalities. – Primary and alternative business workflows. – Business rules, constraints, and conditions.

The above materials are supplied to GenAI through prompts, serving as contextual information that enables GenAI to accurately understand the testing scope and objectives.

### **4.5.2. GenAI Processing Phase**



*Image 26. GenAI Test Case Generation Process*

During the processing phase, Generative AI (GenAI) performs a sequence of steps to support testers in the test design process. First, GenAI conducts semantic analysis of requirements, in which the contents of documents such as the Software Requirements Specification (SRS), user stories, and use cases are analyzed to clearly identify system functional objectives, required business processes, and relevant constraints. This analysis enables GenAI to understand the business context, functional scope, and key factors that directly affect system behavior.

Next, based on the analyzed requirements and identified use cases, GenAI generates test scenarios. These test scenarios are designed to comprehensively cover critical business workflows of the system, including main flows, alternate flows, and error or exception flows. Generating scenarios across multiple execution paths helps expand test coverage and reduces the risk of missing potential situations that may occur during real-world system operation.

From the generated test scenarios, GenAI continues to produce draft test cases. These draft test cases are created at an initial proposal level and include various types of testing situations, such as basic test cases, boundary cases, and error cases. However, these draft test cases are not used directly for test execution. Instead, they serve as reference inputs for testers to review, refine, and finalize before becoming official test cases of the system.

### **4.5.3. Test Case Refinement**

After GenAI generates draft test cases, testers perform the following activities: – Review the test case content. – Refine and adjust test steps to align with the actual system behavior. – Add test data and preconditions. – Remove duplicate or irrelevant test cases.

Once refined, the test cases become official testing and are used for manual testing or automated testing activities.

### **4.5.4. Prompt Engineering Techniques Applied to Test Case Generation**

In this project, the team applies several Prompt Engineering techniques to improve the quality of test cases generated by Generative AI (GenAI). First, the role-based prompting technique is used to clearly define the role of GenAI at each processing stage, such as acting as a Tester or a Quality Assurance (QA) Engineer. Defining a specific role helps guide how GenAI analyzes requirements and generates content that aligns with testing objectives.

In addition, the team applies context-based prompting by providing GenAI with sufficient contextual information through documents such as the Software Requirements Specification (SRS), user stories, and Business Requirement documents. Providing detailed context enables GenAI to accurately understand the business scope and system functionality, thereby reducing the generation of generic or irrelevant test cases.

Furthermore, the step-by-step prompting technique is used to instruct GenAI to generate content in a structured, sequential manner. Specifically, GenAI is guided to first generate test scenarios based on use cases, and then produce detailed test cases derived from those scenarios. This step-by-step approach allows better control over the test generation process and ensures consistency among testing artifacts.

Finally, the team applies constraint-based prompting by defining explicit constraints within the prompts, including requirements on test case structure, generation scope, and the prohibition of assuming system behaviors beyond what is specified in the requirements. These constraints help ensure that the generated test cases are well-structured, closely aligned with business requirements, and suitable for further review, refinement, and completion by testers.

### **Sample Prompt for Test Case Generation Using GenAI**

**Prompt 1 – Generating Test Scenarios from Use Cases (with attached documents)**

**You are acting as a software tester.** Based on the following use case, generate test scenarios.

**Requirements:** – Cover the main flow, alternate flows, and error flows. – Do not generate detailed test cases. – Ensure the scenarios are appropriate for the ShoeShop system.

**Use Case:** *A customer places an order for shoes through the online ShoeShop system.*

### **Prompt 2 – Generating Test Cases from Test Scenarios**

**You are acting as a QA Engineer.** Convert the following test scenario into detailed test cases.

**Requirements:** – Use the structure: **Preconditions – Test Steps – Expected Results**. – Do not assume any system behavior that is not explicitly described. – Each test case must be independent.

**Test Scenario:** *A customer places an order with an out-of-stock product.*

# **CHAPTER 5. TESTING REPORT**

## **5.1. Overview of the Testing Process**

After completing the test design phase in Chapter 4, the team proceeded to execute testing for the ShoeShop system in accordance with the established Test Plan. This included all testing levels: unit testing, integration testing, and system testing.

Testing activities focused on all core business functionalities of the ShoeShop system. Test Cases were executed in a systematic and detailed manner, and the results were fully recorded in testing documents to support quality evaluation and system handover.

## **5.2. Test Case Report**

### **5.2.1. Introduction**

The Test Cases in the ShoeShop project are divided into two groups:

• Positive Tests: verify valid behaviors and correct business workflows of the system.

• Negative Tests: verify error handling, exceptions, and invalid conditions.

|  |  |  |
| --- | --- | --- |
| **Function** | **Number of TCs** | **Result** |
| Access Control | 16 | 16 Pass |
| Product | 13 | 12 Pass - 1 fail |
| Shopping Cart | 23 | 21 Pass - 2 Fail |
| Order | 21 | 21 Pass |

### **5.2.2. Coverage Scope**

The team developed a total of 73 Test Cases covering most of the core business functionalities of the ShoeShop system. As the project scope focuses on functional testing, non-functional requirements were not included in the testing scope.

### **5.2.3. Execution Results**

A total of 73 Test Cases were executed with the following results:

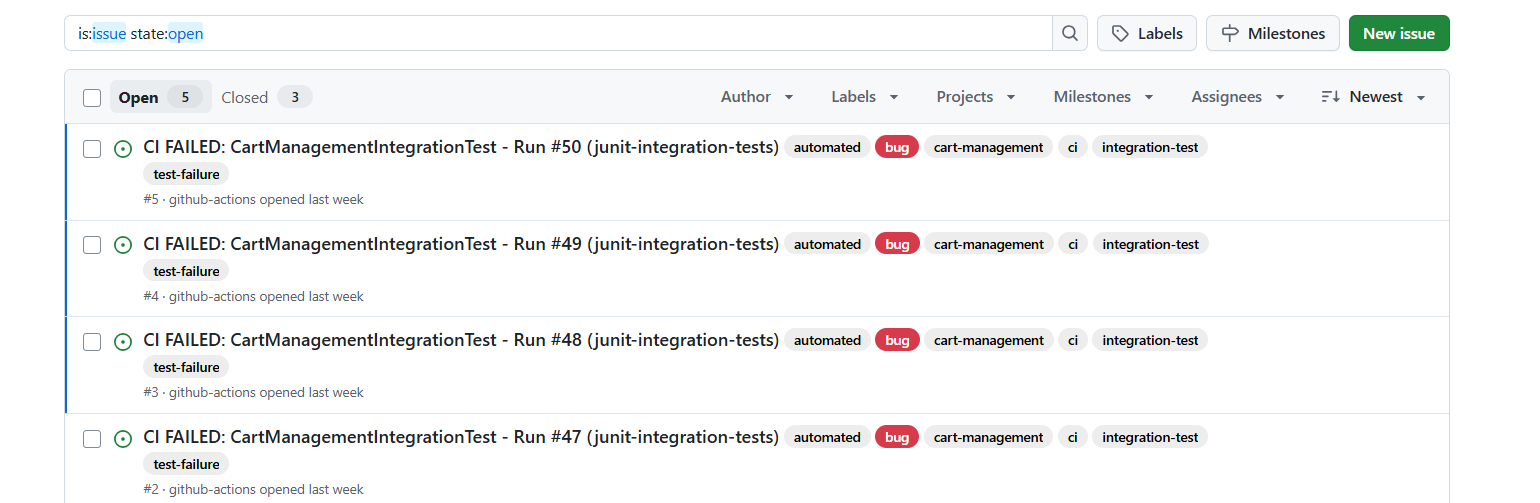
* **70 Test Case Passed (95.9%)**
* **3 Test Case Failed (4.1%)**
* **0 Test Case Not Run**

The results indicate that the ShoeShop system achieved a high level of stability before moving to the acceptance testing phase.

## **5.3. Defect Report**

During manual testing, the team identified three defects, classified by severity as follows:

* Duplicate quantity issue after clicking the plus button on the shopping cart page (TC\_CART\_007) (medium).
* Inventory overflow not validated on the shopping cart page (TC\_CART\_008) (medium).
* Total product quantity not displayed and only shown by size on the product management page (TC\_PRODUCT\_009) (low).



*Image 27. Issues Git Actions*