**HO CHI MINH CITY PEOPLE'S COMMITTEE**

**SAIGON UNIVERSITY**

A blue circle with white text

AI-generated content may be incorrect.

**PROJECT**

**SOFTWARE TESTING**

**COMPUTER SALES WEBSITE**

**Instructor:** Dr. Do Nhu Tai

**Students perform**

3122411168 – Nguyen Minh Quan

3122411037 – Dao Cu Dat

3122411074 – Ngo Gia Hung

3122411178 – Nguyen Ngoc Son

**Ho Chi Minh City, September 2025**

Commitment

I hereby declare that this course project, titled "Testing the SkyWind E-commerce Website," is the result of my group's research and implementation, under the guidance of the course instructor. All content in this project has been researched, synthesized, analyzed, and developed based on my learned knowledge and reliable reference materials and sources.

All data, results, and images presented in this project are truthful, obtained during the project, and have never been used or published in any other work. If any copying, fraud, or copyright infringement is discovered, our team will be fully responsible to the university and our supervising lecturer.

This project aims to apply knowledge of software testing, requirements analysis, and functional testing techniques to evaluate the quality of an e-commerce website selling laptops. Throughout the process, our team has strived to understand and experiment with testing tools, analyze results, and draw practical conclusions to enhance our understanding of software quality assurance processes.

I would like to express my sincere gratitude to my supervising lecturer, Dr. Do Nhu Tai, for his dedicated assistance and for creating favorable conditions for me to complete this project.

**Index**

[Declaration 7](#_Toc217446708)

[Chapter 1. Project Overview 8](#_Toc217446709)

[**1.** **Project Description** 8](#_Toc217446710)

[**2. Current System Analysis** 8](#_Toc217446711)

[**2.1. Business processes** 8](#_Toc217446712)

[**3.** **Technologies Used** 10](#_Toc217446713)

[**4.** **User requirements** 11](#_Toc217446714)

[**4.1. Functional requirements** 11](#_Toc217446715)

[**4.2. Non-functional Requirements** 13](#_Toc217446716)

[**5.** **Deployment Plan** 14](#_Toc217446717)

[**5.1. Scope** 14](#_Toc217446718)

[**5.2. Testing Strategy** 14](#_Toc217446719)

[**6.** **Objectives** 15](#_Toc217446720)

[**Chương 2. Software Design and Architecture** 16](#_Toc217446721)

[**1. Software Design** 16](#_Toc217446722)

[**1.1. Business Context** 16](#_Toc217446723)

[**1.2. Business Process Diagrams** 17](#_Toc217446724)

[1.3. System Actors 20](#_Toc217446725)

[**1.4. Context Diagram** 21](#_Toc217446726)

[**1.5. Use cases** 22](#_Toc217446727)

[**1.6. User stories** 41](#_Toc217446728)

[**1.7. Domain driven design** 42](#_Toc217446729)

[**1.8. Data model** 45](#_Toc217446730)

[**1.9. Wireflow** 47](#_Toc217446731)

[**1.10. User Interface Design** 47](#_Toc217446732)

[2. Architecture Design 52](#_Toc217446733)

[**2.1. Block Diagram** 52](#_Toc217446734)

[**2.2. C4 Architecture Model** 52](#_Toc217446735)

[**2.3. Communication Flow Diagram** 56](#_Toc217446736)

[**2.4. Sơ đồ triển khai** 56](#_Toc217446737)

[Chương 3. Test Plan 57](#_Toc217446738)

[1. Introduction 57](#_Toc217446739)

[**1.1. Purpose** 57](#_Toc217446740)

[**1.2. Scope** 57](#_Toc217446741)

[2. Test Items 57](#_Toc217446742)

[**2.1. Functional Testing** 57](#_Toc217446743)

[**2.2. Usability Testing** 59](#_Toc217446744)

[**2.3. Compatibility Testing** 59](#_Toc217446745)

[**2.4. User Interface Testing** 59](#_Toc217446746)

[**2.5. Security Testing** 59](#_Toc217446747)

[**2.6. Database Testing** 59](#_Toc217446748)

[**2.7. Regression Testing** 60](#_Toc217446749)

[**2.8. API Testing** 60](#_Toc217446750)

[**2.9. Error Handling and Boundary Data Testing** 60](#_Toc217446751)

[**2.10. Performance Testing (Basic Level)** 60](#_Toc217446752)

[3. Test Items Not Included 61](#_Toc217446753)

[**3.1. Third-Party System Functions** 61](#_Toc217446754)

[**3.2. Advanced Security Testing** 61](#_Toc217446755)

[**3.3. Advanced Usability Testing** 61](#_Toc217446756)

[4. Acceptance Test Criteria 61](#_Toc217446757)

[**4.1. Test Coverage** 61](#_Toc217446758)

[**4.2. Test Case Pass Rate** 61](#_Toc217446759)

[**4.3. Số lượng trường hợp kiểm thử** 62](#_Toc217446760)

[**4.4. Số lượng lỗi** 62](#_Toc217446761)

[**4.5. Mật độ kiểm thử đơn vị** 62](#_Toc217446762)

[**4.6. Độ bao phủ mã nguồn** 62](#_Toc217446763)

[**4.7. Quy trình CI/CD** 62](#_Toc217446764)

[5. Test Strategy 62](#_Toc217446765)

[**5.1. Testing Approach** 62](#_Toc217446766)

[**5.2. Test Types** 63](#_Toc217446767)

[**5.3. Test Levels** 64](#_Toc217446768)

[6. Resources 65](#_Toc217446769)

[**6.1. Human Resources** 65](#_Toc217446770)

[**6.2. Test Schedule** 66](#_Toc217446771)

[7. Test Environment 67](#_Toc217446772)

[**7.1. Hardware** 67](#_Toc217446773)

[**7.2. Software** 67](#_Toc217446774)

[**7.3. Infrastructure** 67](#_Toc217446775)

[8. Deliverables 68](#_Toc217446776)

[Chương 4. Test Design 69](#_Toc217446777)

[**1. Introduction** 69](#_Toc217446778)

[**2. Test Design Process Based on the V-Model** 69](#_Toc217446779)

[**2.1. Requirements Analysis – Phase 1a** 69](#_Toc217446780)

[**2.2. System Design – Phase 2a** 70](#_Toc217446781)

[**2.3. Architecture Design – Phase 3a** 70](#_Toc217446782)

[**2.4. Module Design – Phase 4a** 70](#_Toc217446783)

[**2.5. Unit Testing – Phase 1b** 70](#_Toc217446784)

[**2.6. Integration Testing – Phase 2b** 71](#_Toc217446785)

[**2.7. System Testing – Phase 3b** 72](#_Toc217446786)

[**2.8. Acceptance Testing – Phase 4b** 72](#_Toc217446787)

[**3. Test Design Techniques** 73](#_Toc217446788)

[**3.1. Black-Box Testing** 73](#_Toc217446789)

[**3.2. White-Box Testing** 73](#_Toc217446790)

[**4. Test Design Methods** 73](#_Toc217446791)

[**4.1. Manual Testing** 73](#_Toc217446792)

[**4.2. Automated Testing** 73](#_Toc217446793)

[**5. Applying GenAI for Test Case Generation** 74](#_Toc217446794)

[**5.1. GenAI Inputs** 74](#_Toc217446795)

[**5.2. GenAI Processing Phase** 74](#_Toc217446796)

[**5.3. Test Case Finalization** 74](#_Toc217446797)

[Chương 5. Test Report 75](#_Toc217446798)

[**1. Overview of the Testing Process** 75](#_Toc217446799)

[**2. Test Case Report** 75](#_Toc217446800)

[**2.1. Introduction** 75](#_Toc217446801)

[**2.2. Test Coverage** 75](#_Toc217446802)

[**2.3. Test Execution Results** 77](#_Toc217446803)

[**3. Defect Report** 78](#_Toc217446804)

[**3.1. Defect Classification by Severity** 78](#_Toc217446805)

[**3.2. Defect Statistics** 78](#_Toc217446806)

[**3.3. Defect Resolution Process** 79](#_Toc217446807)

# Declaration

I hereby declare that this course project entitled “Software Testing of the SkyWind E-commerce Website” is the result of our team’s research and implementation, conducted under the guidance of the course instructor. All contents of this report have been studied, collected, analyzed, and developed based on the knowledge acquired during the course, along with reliable reference materials and sources.

All data, results, and figures presented in this project are truthful, produced during the project implementation process, and have not been previously used or published in any other work. In the event of any plagiarism, academic misconduct, or copyright infringement, our team fully accepts responsibility before the university and the supervising lecturer.

This project was carried out with the objective of applying knowledge in software testing, requirements analysis, and functional testing techniques to evaluate the quality of an e-commerce website specializing in laptop sales. Throughout the project, our team made continuous efforts to study and experiment with various testing tools, analyze testing results, and draw practical conclusions in order to enhance our understanding of the software quality assurance (SQA) process.

I would like to express my sincere gratitude to our supervisor, Dr. Đỗ Như Tài, for his dedicated guidance, support, and valuable assistance, which enabled us to successfully complete this project.

# Chapter 1. Project Overview

# **Project Description**

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

The testing process includes requirements analysis, test case design and execution, as well as the implementation of unit testing, integration testing, system testing, and acceptance testing, in order to ensure that the system operates in accordance with the specified requirements prior to deployment.

# **2. Current System Analysis**

## **2.1. Business processes**

**Shopping Cart Processing Workflow**

After logging into the system and selecting desired products, customers can add laptops to the shopping cart. On the shopping cart interface, the system displays a list of added laptops along with images, product titles, prices, discounts, and purchase quantities. Customers can modify quantities or remove products from the cart at any time. Each change automatically updates the total cost, enabling users to easily control expenses before proceeding to order placement and payment.

**Order Placement and Payment Workflow**

After customers add products to the shopping cart, the system proceeds to the checkout step and requests delivery information (full name, phone number, and address), as well as the selection of a payment method: Cash on Delivery (COD) or Momo/Bank Transfer.

If customers choose payment via Momo or bank transfer, the system temporarily stores the order and generates a payment QR code for customers to scan in order to complete the transaction.

Upon receiving a response from Momo or the bank, if the payment is successful and the available laptop inventory is sufficient, the system automatically updates the order status, deducts the corresponding product quantities, and notifies the customer of successful order placement.

Conversely, if the payment is successful but the available inventory is insufficient, the system automatically processes a refund, displays an order failure notification along with the reason, and deletes the **order to ensure data consistency and accuracy.**

**System Administration Workflow**

The administration dashboard is designed for system administrators and sales staff to monitor and manage website operations. System administrators have full access to all system functionalities, while sales staff have restricted permissions and are not allowed to perform user management functions.

Key administrative functions include:

* Laptop Management: Adding, deleting, or updating laptop information (laptop name, CPU, GPU, RAM, specifications, description, product images, and available quantity) for customer browsing.
* Category and Component Management: Adding, updating, or deleting laptop categories and core components to ensure structured data organization and support efficient product searching.
* Order Management: System administrators and sales staff can monitor order lists and manually update order statuses. For orders paid via Momo or bank transfer, if an order is canceled or returned, the system triggers a refund request to the payment gateway.
* Payment Management: Displaying lists of successful payment transactions and refunded transactions processed through the Momo or bank payment gateway.
* User Management: Only system administrators are authorized to add, delete, update, or lock administrator and customer accounts.

# **Technologies Used**

The SkyWind website is developed based on the **Client–Server architecture**, applying modern technologies to ensure high performance, security, and scalability in an e-commerce environment.

* 1. Frontend (User interface)
* Languages: HTML5, CSS3, JavaScript (ES6+).
* Framework: ReactJS – enables flexible, high-performance, and maintainable user interfaces.
* Supporting Libraries:
  + Axios – API communication with the backend
  + React Router DOM – client-side routing without full page reload
  + ailwindCSS / Bootstrap – modern, responsive, and cross-platform UI design
* Build tools: Vite / Webpack – optimize the compilation and deployment.
  1. Backend (Business Logic Processing)
* Languages: Python.
* Framework: FastAPI (or Django REST Framework)
  + Provides high-performance, scalable, and easily integrable RESTful APIs
* Core Functionalities:
  + User, product, shopping cart, and order management
  + Payment processing, authentication, and authorization
  + Secure data exchange via APIs compliant with HTTP/HTTPS standards
  1. Database
* Database management system:
  + PostgreSQL – used for relational data.
* ORM:
  + SQLAlchemy (with FastAPI) or Django ORM (with Django REST Framework) – efficient and secure database operations.
  1. Authentication & Security
* JWT (JSON Web Token) – user authentication and session management.
* OAuth2 – secure authentication and third-party integration.
* Passlib / BCrypt – password hashing and encryption.
* CORS Middleware & HTTPS – secure communication between client and server.
  1. Tools & DevOps
* Git / GitHub – source code management and team collaboration.
* Postman – API testing during development.
* Docker – application containerization for deployment.
* Render / Vercel / Railway / AWS – online deployment platforms for frontend and backend.
  1. Testing & Monitoring
* Pytest / Unittest – backend functional testing.
* Google Analytics / Firebase – user behavior and experience monitoring.

# **User requirements**

## **4.1. Functional requirements**

|  |  |  |
| --- | --- | --- |
| **ID** | **Requester** | **Content** |
| **BR1** | **Product Management** | |
| BR1.1 | Customers | Search, filter, and sort products based on criteria such as price, title, category, CPU, RAM, GPU, storage, and display  View detailed information of each laptop |
| BR1.2 | System Administrators | Manage laptops (add, delete, or update laptop information)  Manage components (CPU, GPU, RAM, storage, display specifications)  Manage categories (add, delete, or update categories) |
| **BR2** | **Shopping Cart Management** | |
| BR2.1 | Customers | Add products to the shopping cart  View the list of products in the cart  Modify quantities or remove products from the cart |
| **BR3** | **Payment Management** | |
| BR3.1 | Customers | Receive payment result notifications (successful or failed) |
| BR3.2 | System Administrators | Monitor lists of successful and refunded payment transactions |
| **BR4** | **Order Management** | |
| BR4.1 | Customers | View order history and order details  Place new orders |
| BR4.2 | System Administrators | Manually update order statuses  Monitor all orders in the system |
| **BR5** | **Access Control** | |
| BR5.1 | System Administrators | Add, delete, update, or lock user accounts (excluding their own account) |
| BR5.1 | System Administrators | Register, log in, and log out of the system |

## **4.2. Non-functional Requirements**

|  |  |
| --- | --- |
| **BR6** | **System Deployment** |
| BR6.1 | The software operates in an Internet environment and supports concurrent access by multiple users. |
| **BR7** | **User Interface** |
| BR7.1 | The interface is well-structured, visually appealing, and compliant with web standards. |
| **BR8** | **Security** |
| BR8.1 | All data must maintain integrity constraints, and sensitive data must be encrypted. |

# **Deployment Plan**

## **5.1. Scope**

This document serves as the basis for testing, acceptance, and evaluation of the SkyWind E-commerce Website project. It covers the verification of website functionalities, user interface behavior, and the validity of data and database integrity.

## **5.2. Testing Strategy**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Performer** | |  | | --- | |  |  |  | | --- | | **Role** | | **Responsibilities** |
| 1 | Nguyễn Minh Quân | Tester / Developer | Test planning, system testing |
| 2 | Đào Cư Đạt | Tester / Developer | Test case design, unit testing, integration testing |
| 3 | Ngô Gia Hưng | Tester / Developer | CI/CD implementation using GitHub Actions, test documentation |
| 4 | Nguyễn Ngọc Sơn | Tester / Developer | |  | | --- | |  |  |  | | --- | | Review checklist preparation, test documentation | |

**Environment and Tools**

|  |  |
| --- | --- |
| **Purpose** | **Components** |
| Test documentation | Microsoft Word, Microsoft Excel |
| Browsers | Chrome, Firefox, Safari |
| Frontend development | Vite, ReactJS, JavaScript |
| Backend development | Python |
| Source code management | GitHub |
| CI/CD | GitHub Actions |
| Production deployment | Frontend & Backend on Render |
| Database | PostgreSQL hosted on Aiven |

# **Objectives**

The objective of the SkyWind website testing activities is to verify that the system operates correctly according to specified requirements, detect defects early in the development process, and ensure software quality prior to deployment.

Testing focuses on validating core business functionalities, basic system stability, and compatibility across popular browsers and devices. Test results will serve as a foundation for evaluating system completeness and supporting software deployment, operation, and maintenance.

# **Chương 2. Software Design and Architecture**

# **1. Software Design**

## **1.1. Business Context**

The SkyWind website is developed to meet users’ needs for online laptop shopping, providing core functionalities such as Product Catalog, Shopping Cart, Payment Process, Inventory Management, Product Rating, and Access Control.

With the Product Catalog, customers can browse a list of laptops using filters such as brand, price, and specifications (CPU, RAM, storage, graphics card, etc.). When selecting a product, customers can view detailed information including name, images, technical specifications, stock availability, and reviews from other buyers (if available). System administrators (SysAdmin) are authorized to add, edit, or delete products and assign them to appropriate inventory categories.

With the Shopping Cart, users can add laptops to the cart or proceed with immediate purchase. On the cart page, customers can view selected products along with a summary table displaying product name, price, quantity, order value, promotions, shipping fees, and the total payable amount. The cart is automatically updated whenever products are added or removed. Customers can then initiate the checkout process by clicking the Checkout button. SysAdmins can monitor and handle abnormal carts to ensure data validity.

With the Payment Process, after customers place an order by clicking the Checkout button, the system validates product information, processes online payments, and sends a confirmation email along with an electronic invoice to the customer.

With Inventory Management, SysAdmins can manage laptop quantities in stock, update product status, and monitor inventory levels to ensure product availability.

With the Rating feature, customers can submit ratings (from 1 to 5 stars) and comments after purchasing products, helping build store credibility and support other customers’ purchasing decisions.

With Access Control, users are required to log in to place orders, manage orders, and submit ratings. After logging in, SysAdmins can access the administration panel to manage products, orders, and customers. (Customers and SysAdmins log in through separate interfaces.)

In addition, several initialization tasks are required during system deployment, such as creating administrator accounts, sample customer accounts, and importing product, inventory, and test rating data.

## **1.2. Business Process Diagrams**

**Shopping Cart Processing Flow**

After logging in and searching/selecting a laptop, customers can add products to the shopping cart. On the cart interface, customers can review the list of selected products, adjust quantities, or remove laptops from the cart. Customers then proceed to place an order.

**A diagram of a flowchart

AI-generated content may be incorrect.**

Hình 2.0.1 Shopping Cart Processing Flow Diagram

**Order Placement and Payment Flow**

After entering delivery information, customers choose a payment method: Cash on Delivery (COD) or Momo/Bank Transfer.

* With COD, the system proceeds directly to inventory verification. If sufficient stock is available, the order is successfully placed.
* With Momo/Bank Transfer, the system generates a QR code for customers to scan.
  + If payment fails, customers return to select another payment method.
  + If payment is successful, the system checks inventory availability. If sufficient, the order is completed successfully; otherwise, the system automatically requests a refund from Momo/Bank.

A diagram of a flowchart

AI-generated content may be incorrect.

Hình 2.0.2 Order Placement and Payment Flow Diagram

**System Administration Flow**

After logging in, system administrators and sales staff can perform management functions related to categories, products, components, orders, and payments. User management functions are restricted to system administrators only.

During order management, if the status of an order paid via Momo is updated to Canceled or Returned, the system automatically sends a refund request to Momo for transaction processing.

A diagram with white text and black lines

AI-generated content may be incorrect.

Hình 2.0.3 System Administration Flow Diagram

## 1.3. System Actors

|  |  |  |
| --- | --- | --- |
| Actor | Role | Primary Permissions |
| Buyer (Customer) | Primary system user | View, search, purchase products, make payments, submit product ratings |
| System Admin (Quản trị viên) | |  | | --- | |  |  |  | | --- | | System administrator | | Perform all administrative functions |
| Momo/Ngân hàng | Third-party payment gateway | Provide QR codes for customer payments |

A group of people with white text

AI-generated content may be incorrect.

Hình 2.0.4 System Actors Diagram

## **1.4. Context Diagram**

Customer: Interacts directly with SkyWind to view product lists, log in, manage the shopping cart, place orders, make payments, and view order history.

Momo: External payment gateway interacting with SkyWind to generate QR codes, process refunds, and return transaction results.

System Administrator: Manages the system through functions such as product, category, order, payment, and user management.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 2.5 System Context Diagram

## **1.5. Use cases**

**Use case summary**

Customers perform shopping cart management, product browsing, and access control.

System Administrators perform product management, payment management, order management, and access control.

Momo/Bank performs payment processing.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 2.6 Use Case Summary Diagram

**UC1 Product Management**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Use Case Number:** | **UC1** | | |
| **Use Case Name:** | Product Management | | |
| **Actor (s):** | Admin | | |
| **Maturity:** | Focused | | |
| **Summary:** | Admin manages product categories with add, edit, and delete | | |
| **Basic Course of Events:** | **Actor Action** | **System Response** | |
|  | 1. Admin logs into the system |  | |
|  | 2. System verifies credentials and displays the appropriate interface | |
| 3. Admin selects “Products” |  | |
|  | 4. System processes the request | |
|  | 5. Admin adds, edits, or deletes a product **E1** |  | |
|  |  | 6. The system displays updated results. | |
|  |  | 7. System displays update results and synchronizes with the frontend | |
| **Alternative Paths:** | **A1.** Admin updates product information (price, description); the system updates and displays changes on the frontend. | | |
| **Exception Paths:** | **E1.** Admin enters invalid or incomplete data; the system displays error messages and requests corrections. | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Assumptions** | None | | |
| **Preconditions** | None | | |
| **Post Conditions** | Changes are successfully saved  Updated information is displayed on the frontend | | |
| **Authors** |  | | |
| **Activity Diagram** | | |  |
| **Sequence** | | |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Use Case Number:** | **UC1** | | |
| **Use Case Name:** | Product Catalog | | |
| **Actor (s):** | Customer | | |
| **Maturity:** | Focused | | |
| **Summary:** | Customers search, filter, and view product details. | | |
| **Basic Course of Events:** | **Actor Action** | **System Response** | |
|  | 1. Actor logs into the system |  | |
|  | 2. System verifies credentials and displays the appropriate interface | |
| 3. Search and filter products |  | |
|  | 4. The system displays updated results. | |
|  | 5. Customer view detailed information of selected laptops **E1** |  | |
|  |  | 6. The system displays corresponding product details. | |
| **Alternative Paths:** |  | | |
| **Exception Paths:** |  | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Assumptions** | None | | |
| **Preconditions** | None | | |
| **Post Conditions** |  | | |
| **Authors** |  | | |
| **Activity Diagram** | | |  |
| **Sequence** | | |  |

**UC2 Shopping Cart Management**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Use Case Number:** | **UC2** | | |
| **Use Case Name:** | Shopping Cart Management | | |
| **Actor (s):** | Customer | | |
| **Maturity:** | Focused | | |
| **Summary:** | Customers add products to the cart, confirm orders, and proceed with online payment. | | |
| **Basic Course of Events:** | **Actor Action** | **System Response** | |
|  | 1. Customers browse products |  | |
|  | 2. The store's product display system. | |
| 3. Buyer views product details |  | |
|  | 4. Product details display system | |
|  | 5. Buyer adds product to cart |  | |
|  |  | 6. The system displays updated shopping cart information. | |
|  | 7. Buyer checks product information in the shopping cart. **A1** |  | |
|  |  | 8. Product quantity display system | |
|  | 9. Buyer clicks “Checkout” |  | |
|  |  | 10. The system redirects to the “Payment” page E1 | |
| **Alternative Paths:** | **A1**. Buyer updates product quantity -> System updates product quantity | | |
| **Exception Paths:** | **E1.** The system checks the product quantity to see if it exceeds the inventory limit -> the system alerts when the quantity exceeds the limit. | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Assumptions** | None | | |
| **Preconditions** | The user is logged into the app.  The shopping cart is not empty. | | |
| **Post Conditions** | The order has been confirmed and successfully paid.  The system saves order information, status, and payment details.  The status "Completed" is displayed on the app. | | |
| **Authors** |  | | |
| **Activity Diagram** | | |
| **Sequence** | | |

**UC3 Order Management**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Use Case Number:** | UC3 | | |
| **Use Case Name:** | Order Management | | |
| **Actor (s):** | Admin | | |
| **Maturity:** | Focused | | |
| **Summary:** | Admin views, confirms, and updates order status. | | |
| **Basic Course of Events:** | Actor Action | System Response | |
|  | 1. Admins receive order notifications |  | |
|  | 2. Order list display system | |
| 3. Admin confirms and updates the status of order A1. |  | |
|  | 4. System for updating order status corresponding to E1 | |
|  |  | 5. The system records the status and sends notifications to customers. | |
| **Alternative Paths:** | A1. Admin cancels the order | | |
| **Exception Paths:** | E1. Canceled orders will be notified to the Buyer in the order history section. | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Assumptions** | None | | |
| **Preconditions** | Order confirmed  The system updates the status as progresses  The buyer interface shows the actual status of the order | | |
| **Post Conditions** | Admin must confirm the order and update the order status.  The system synchronizes order status over time. | | |
| **Authors** |  | | |
| **Activity Diagram** | | |
| **Sequence** | | |

**UC4 Payment Management**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Use Case Number:** | **UC4** | | |
| **Use Case Name:** | **Payment Management** | | |
| **Actor (s):** | **Customer** | | |
| **Maturity:** | **Focused** | | |
| **Summary:** | **Customers enter delivery information and select payment methods.** | | |
| **Basic Course of Events:** | **Actor Action** | **System Response** | |
|  | 1. Buyers click "Checkout" or "Buy Now" from the product details page. |  | |
|  | 2. Payment page display system | |
|  | 3. The system updates the default address information for Buyer A1. | |
| 4. Buyer checks the information and delivery address A2 |  | |
|  | 5. Information update system | |
|  | 6. Buyer chooses payment method |  | |
|  |  | 7. Payment method update system | |
|  | 8. Buyer clicks “Thanh toán ngay” |  | |
|  |  | 9. E1 Payment Success Notification System | |
| **Alternative Paths:** | A1. The system will retrieve the user's existing address information.  A2. If the buyer changes their address information, the system will update the new information. | | |
| **Exception Paths:** | E1. Buyer will notify you if a payment method is not selected. | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Assumptions** | None | | |
| **Preconditions** | None | | |
| **Post Conditions** | Orders that are successfully delivered or those that encounter problems will receive clear incident feedback.  Admin has the right to approve and make adjustments.  The system ensures continuous synchronization of status data between the Admin and the System. | | |
| **Authors** |  | | |
| **Activity Diagram** | | |  |
| **Sequence** | | |  |

**UC5 Access Control**

|  |  |  |  |
| --- | --- | --- | --- |
| A diagram of a system  AI-generated content may be incorrect. | | | |
| **Use Case Number:** | **UC5** | | |
| **Use Case Name:** | Access Control | | |
| **Actor (s):** | Customer, Admin | | |
| **Maturity:** | Focused | | |
| **Summary:** | Users log in and log out of the system. | | |
| **Basic Course of Events:** | **Actor Action** | **System Response** | |
|  | 1.Actors logging into the system |  | |
|  | 2. The system retrieves the data and account type from the system and then starts the E1 session. | |
|  | 3. The system displays the execution function based on the account type (Buyer, Admin). | |
|  | 4. The system directs users to the homepage. | |
| **Alternative Paths:** | None | | |
| **Exception Paths:** | E1. If the username or password is incorrect, the system will display a login error message, redirecting the user back to the login session. | | |
| **Extension Points:** | None | | |
| **Triggers:** | None | | |
| **Extension Points** | None | | |
| **Triggers** | The user accessed the page but did not start a session. | | |
| **Assumptions** | None | | |
| **Preconditions** | None | | |
| **Post Conditions** | None | | |
| **Authors** |  | | |
| **Activity Diagram** | | |
| **Sequence** | | |

## **1.6. User stories**

**Product Management**

As a customer, I want to search and filter laptops by name, brand, components, and price to easily find the product I need.

As a customer, I want to view detailed laptop information to make informed purchasing decisions.

As a system administrator, I want to add, edit, or delete products to keep the catalog accurate and up to date.

As a system administrator, I want to manage categories and components to organize products effectively.

**Shopping Cart Management**

As a logged-in customer, I want to add laptops to the cart to save items for purchase.

As a customer, I want to review items in my cart before checkout.

As a customer, I want to modify quantities or remove items from my cart.

**Payment Management**

a customer, I want to choose between COD and Momo/Bank payments for convenience.

As a customer, I want to receive confirmation when payment is successful.

As a system administrator, I want to manage successful and refunded payment transactions.

**Access Control**

As a visitor, I want to register a new account to access system features.

As a system administrator, I want to manage user accounts.

As a system administrator, I want to lock or unlock user accounts to control access.

## **1.7. Domain driven design**

**Business Domain Description**

The main reasons for selecting this business domain include:

1. A real-world system already exists, allowing comparison between implementation and an operational website.
2. The domain contains complex business logic beyond simple CRUD operations.
3. The domain is manageable in size, easy to understand, and feasible to implement.

Core entities include **Laptop, Component, and Category**. Customers can browse, search, and filter laptops, while system administrators and sales staff can perform CRUD operations on laptops, categories, and components such as CPU, GPU, RAM, and SSD.

Only logged-in customers can add products to the shopping cart. Customers can review, modify, or remove cart items and proceed to order placement and payment.

Orders consist of **Order** and **Order Details** entities. After adding products to the cart and entering delivery and payment information, an order is created. System administrators and sales staff can update order status.

Payments include entities such as **Payment**, **Payment Method**, and **Payment Status**. Customers can choose Momo/Bank payments, while system administrators can view transaction lists.

Access control includes **User** and **Role** entities. Users include customers and system administrators. System administrators manage user accounts.

**Conceptual Model**

Products: include laptops, main components, and categories; customers can view, search, and filter; system administrators can perform CRUD operations on the data.

Shopping Cart: Customers log in to add, edit, and delete products, and place orders.

Orders: include the Order Form and Order Details; customers create orders when placing them, and sales staff/system administrators update the status.

Payment: includes Payment, Payment Method, and Payment Status; customers can pay using Momo/Bank Transfer, and system administrators can view the transaction.

Access control includes Users (customers, system administrators) and Positions; system administrators manage accounts and user login/logout.

A computer screen with white lines and dots

AI-generated content may be incorrect.

Hình 2.7 Conceptual Model Diagram

## **1.8. Data model**

The conceptual Entity–Relationship model identifies core entities such as Laptop, ImageLaptop, Brand, Cart, Category, Order, User, and Payment, along with relationships such as *belongs to* and *has*.

The logical ER model translates conceptual entities into database tables with primary keys (PK), foreign keys (FK), attributes, and relationships.

The physical ER model defines actual data storage structures, including tables, data types, and constraints, ensuring data integrity and efficient querying.

A computer screen shot of a computer screen

AI-generated content may be incorrect.

Hình 2.8 Physical Entity–Relationship Model

## **1.9. Wireflow**

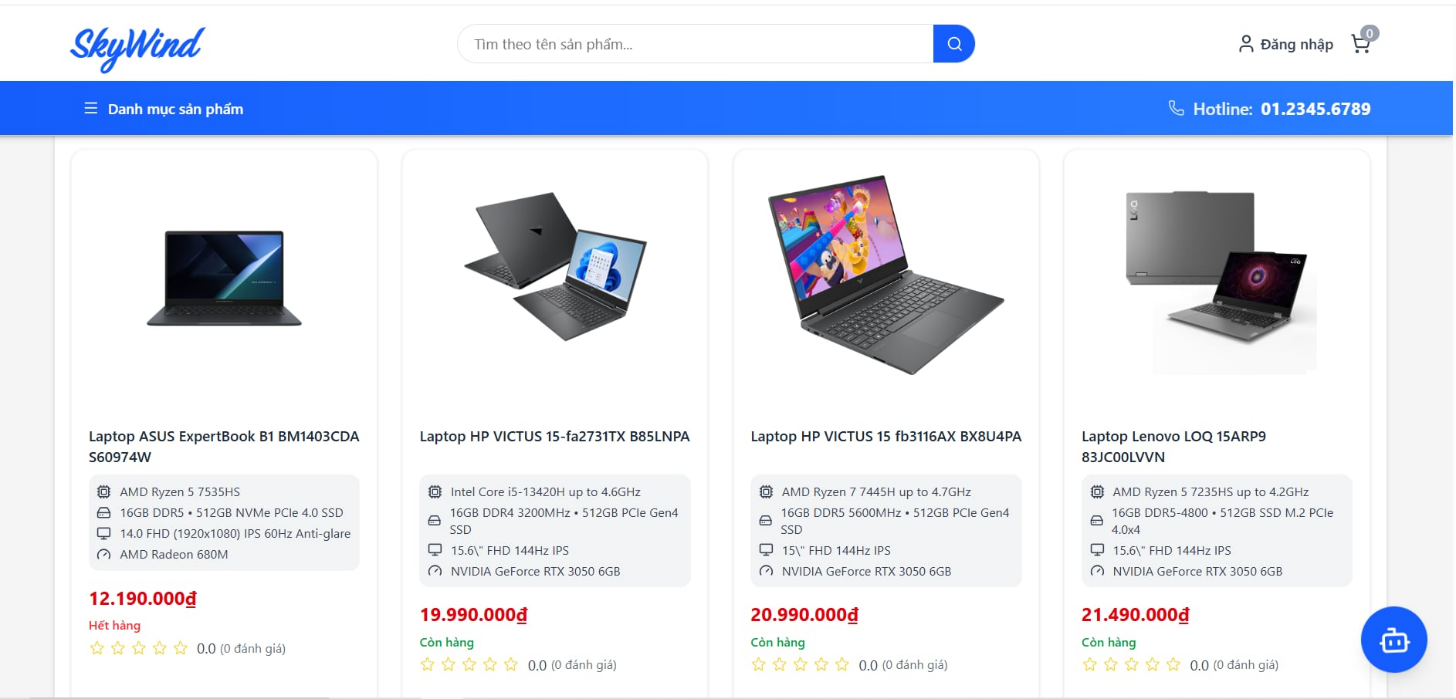
Customers access the SkyWind homepage to browse categories and featured products, search for laptops, and view detailed product information. After selecting Add to Cart, users are prompted to log in if not already authenticated.

Customers review cart items, adjust quantities, proceed to checkout, enter delivery details, select payment methods, and complete payment. The system displays order results and allows customers to track orders and view detailed order information.

## **1.10. User Interface Design**

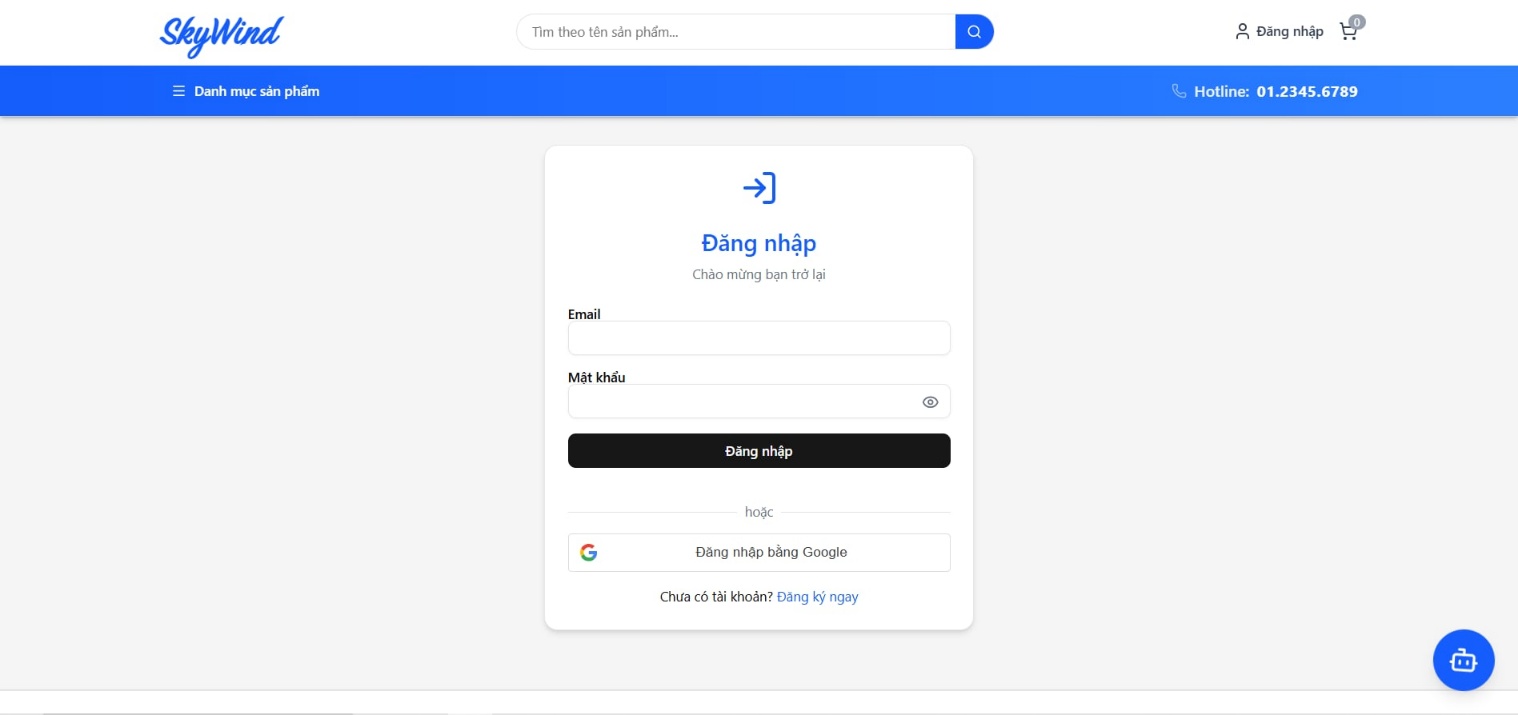
**Customer Interfaces**

First page displayed when customers access the SkyWind website.



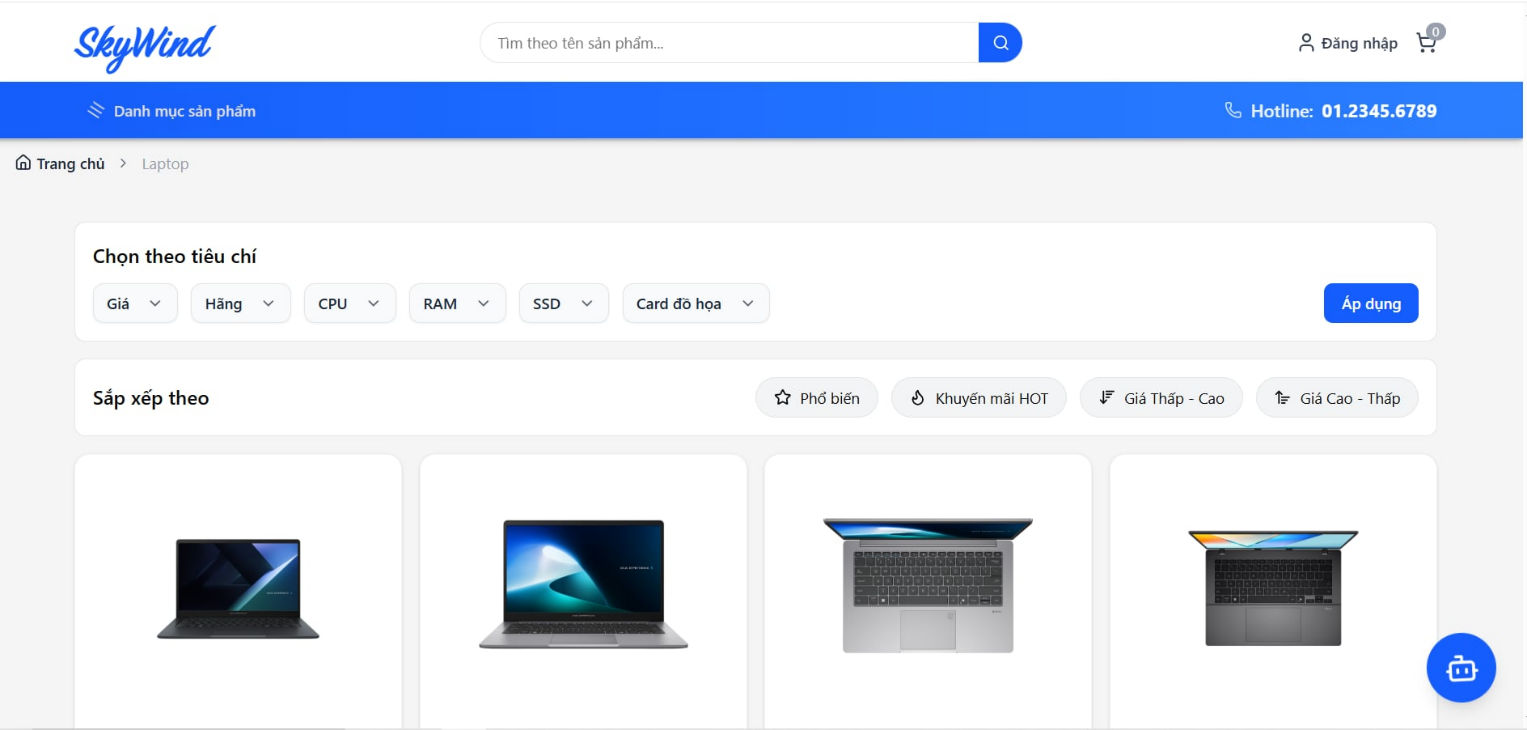
Hình 2.9 Giao diện trang chính

Requires customers to enter email and password to purchase products.



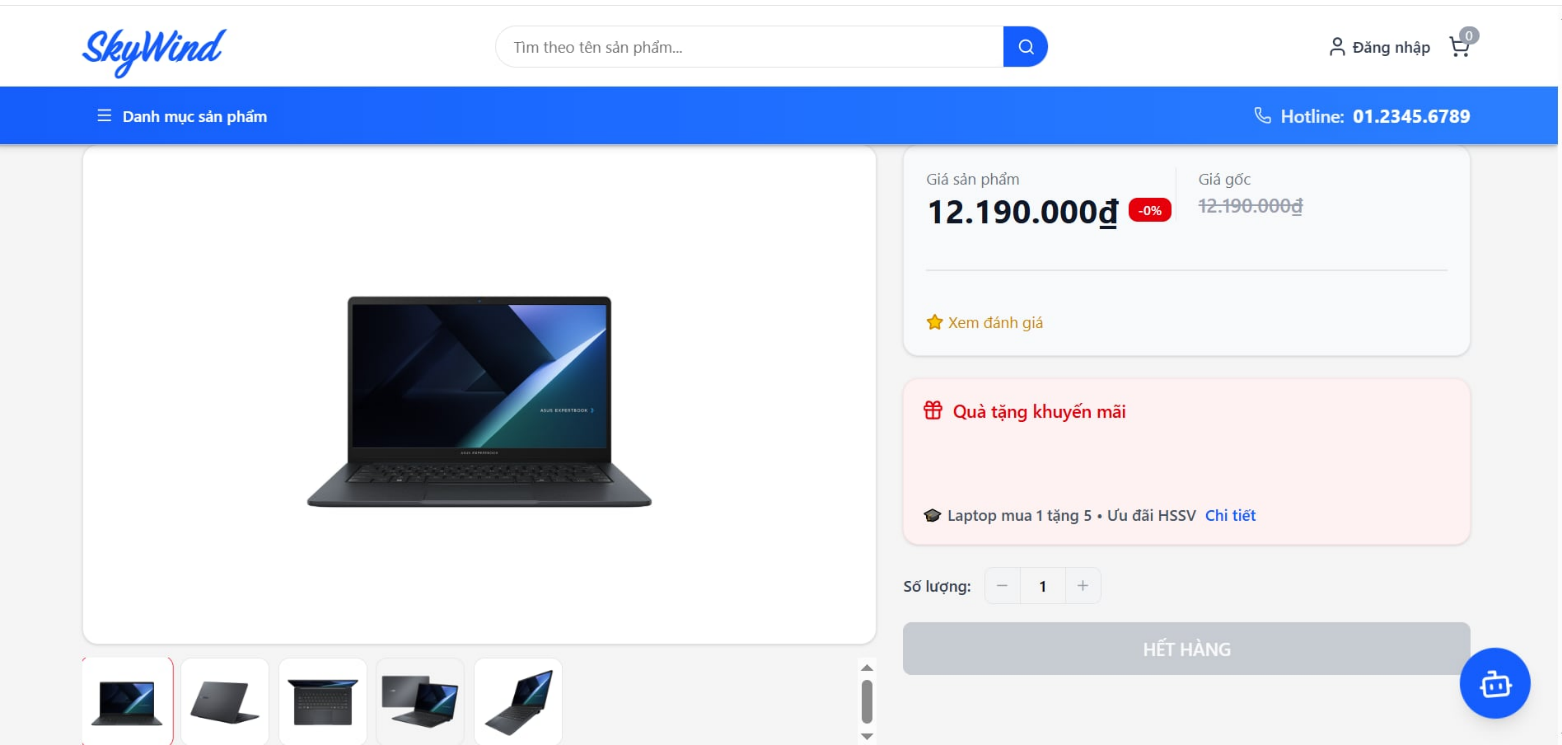
Hình 2.10 Customer Login Interface

Allows searching, category selection, and viewing discounted products.



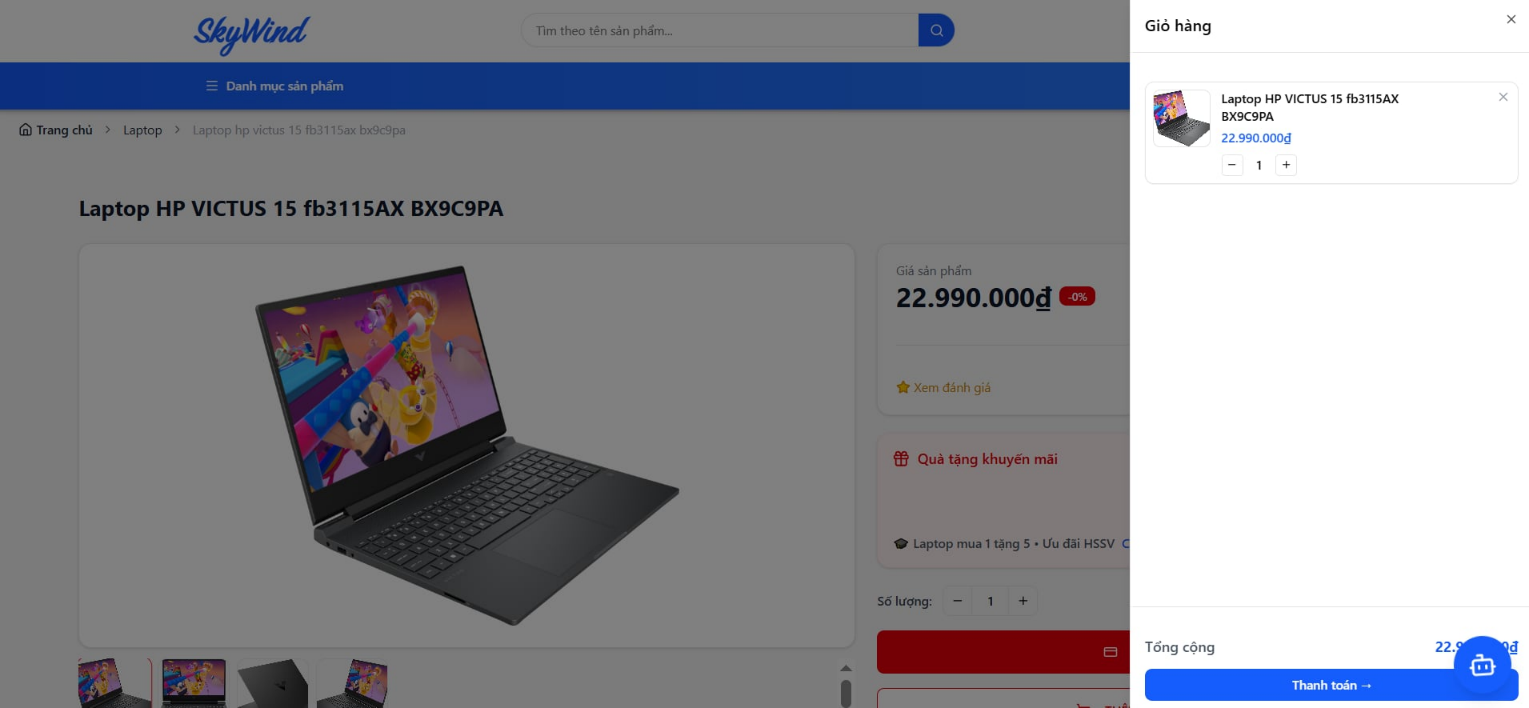
Hình 2.11 Product List Interface

Displays detailed laptop information and purchase options.



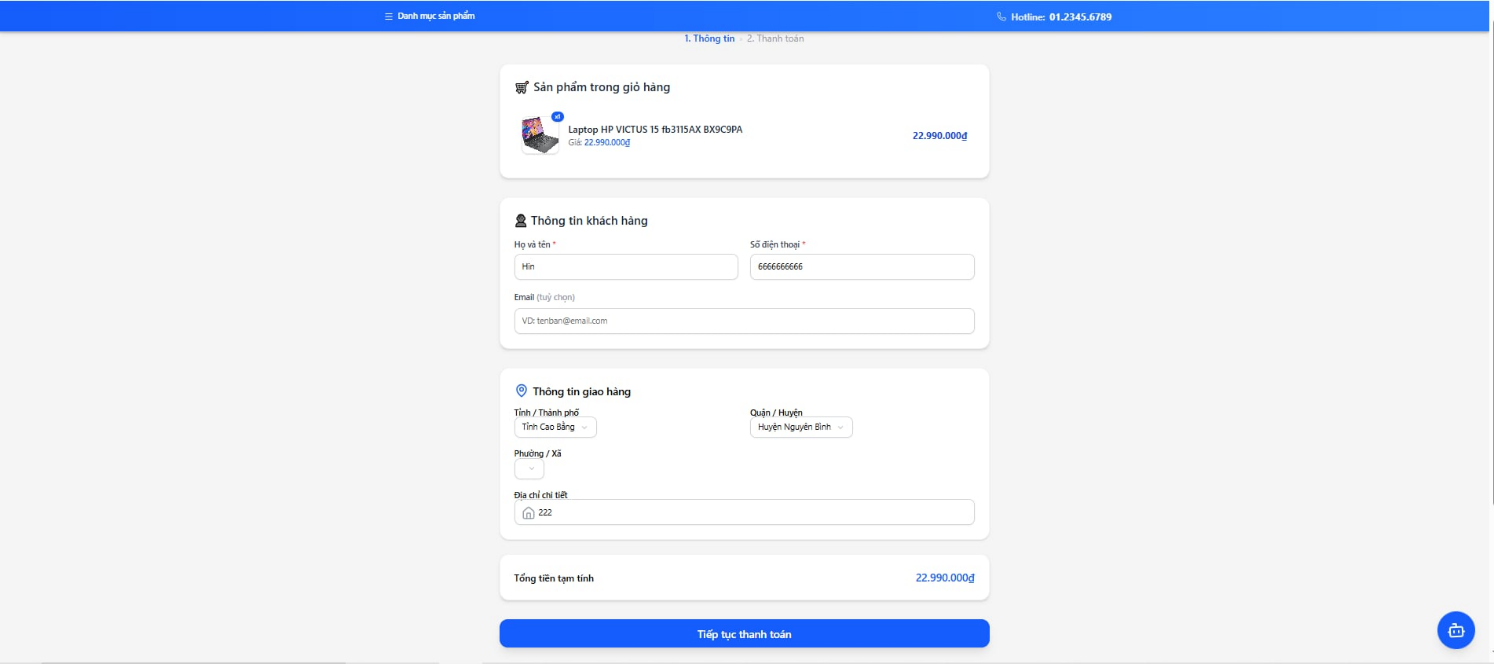
Hình 2.12 Product Detail Interface

Allows customers to manage cart items and proceed to checkout.

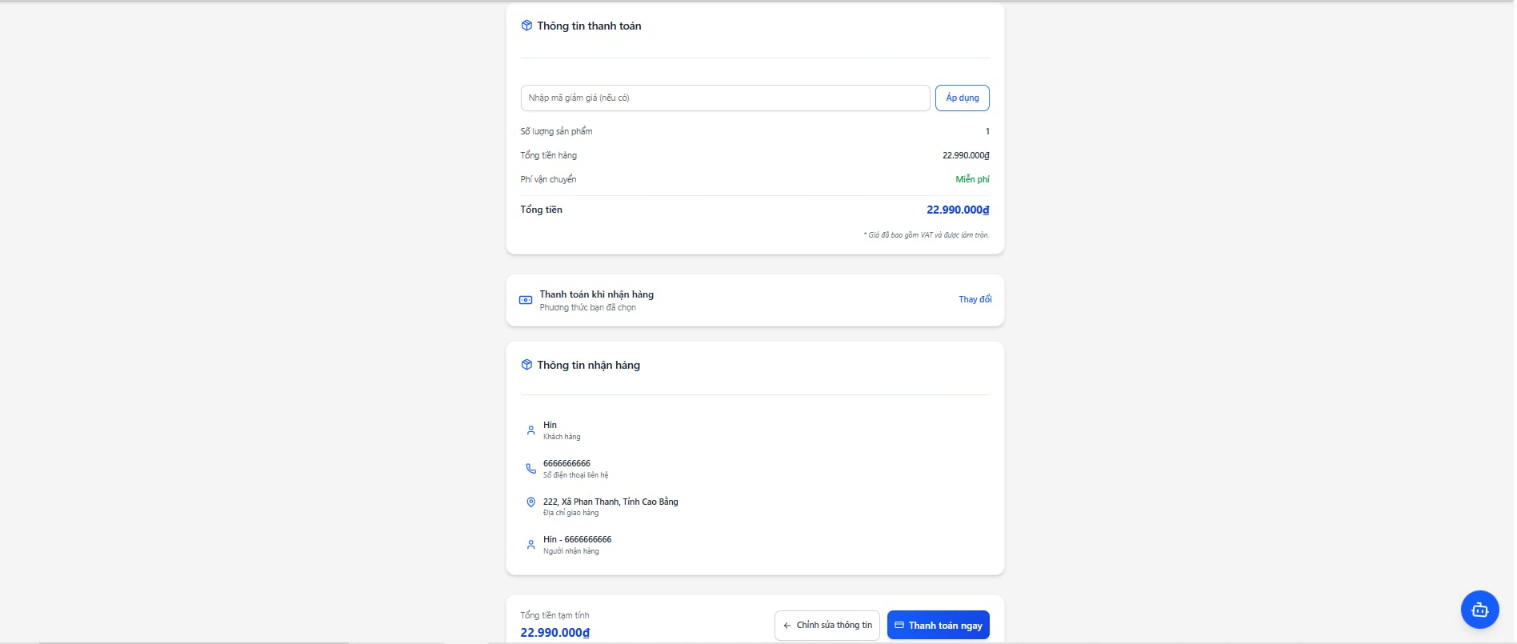


Hình 2.13 Shopping Cart Interface

Displays order summary and payment processing.

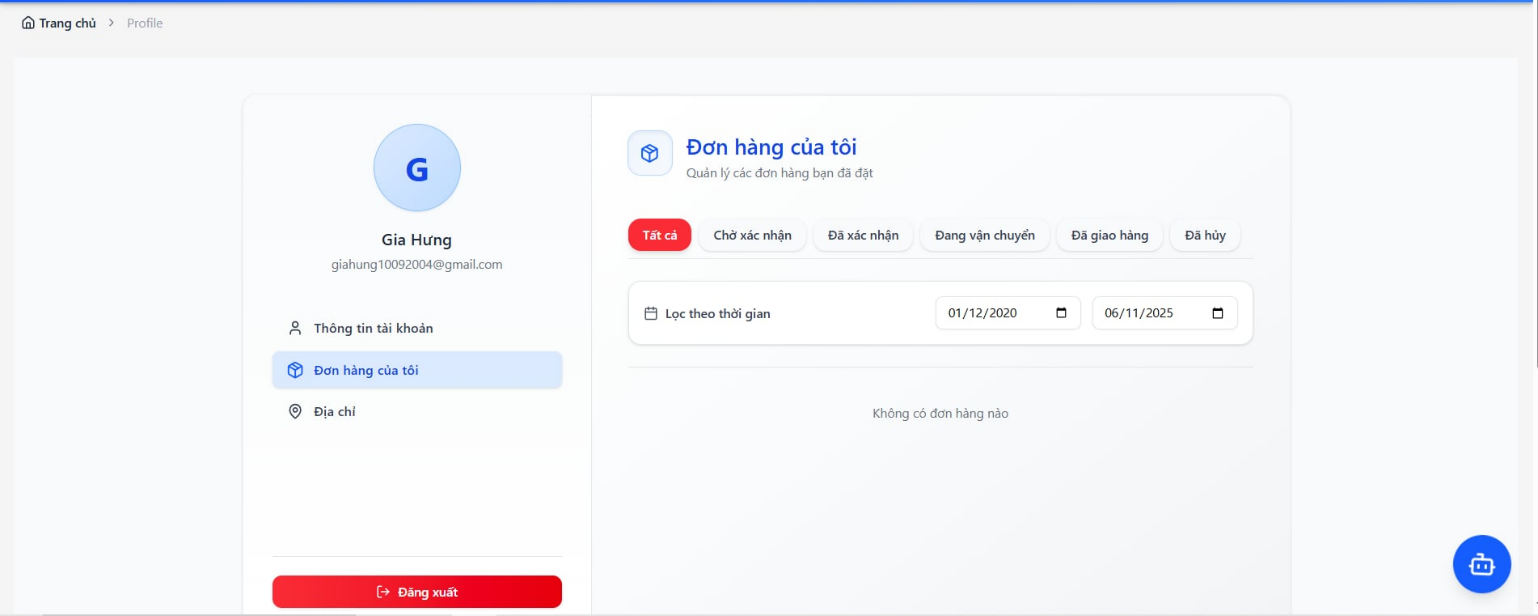


Hình 2.0.14 Payment Information Review Interface



Hình 2.15 Checkout Interface

Displays successfully placed orders.



Hình 2.16 Order List Interface

**Administrator Interfaces**

Requires administrators to authenticate before accessing the system.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 2.17 Admin Login Interface

The product listing page allows system administrators or sales staff to view a list of laptops, and to update their status or remove them from the list.

The order list page is where system administrators or sales staff can view a list of orders and update their status.

The order details page is where system administrators or sales staff can view the full details of an order.

The payment list page is where system administrators or sales staff can view a list of payment transactions.

The user list page is a system administrator page where you can view a list of administrator and client accounts, and you can also lock, unlock, or delete user accounts.

2. Architecture Design

## **2.1. Block Diagram**

## **2.2. C4 Architecture Model**

**C1 – System context**

Customers use the system to register/log in, search for products, view product details, place orders, and complete payments.

System administrators log into the system to manage laptops, categories, configurations, components, orders, and payments. Administrators are also authorized to manage user accounts.

The system integrates with the Momo/Bank payment gateway to process transactions, receive payment results, and display the payment interface to customers.

A diagram of people with text

AI-generated content may be incorrect.

Hình 2.18. C2 System Contex

**C2 – Container**

The ReactJS SPA Container serves as the user interface, allowing customers, administrators, and sales staff to access the system via HTTP. This container sends requests to the SkyWind System API (Python) to handle business logic.

The Python Backend Container is responsible for processing requests and querying data from the PostgreSQL Container, which stores information related to entities such as Laptop, Order, Category, and others.

The backend also communicates bidirectionally with the Momo/Bank Payment Gateway over HTTP/HTTPS to execute payments and receive transaction results.

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 2.18 C2 Container Diagram

**C3 – Component (high-level)**

The User Interface (ReactJS) connects to the API layer implemented in Python. The API layer is divided into separate business modules, including:

* Product Management
* Shopping Cart
* Order Management
* Payment Processing
* Access Control

Each module interacts with its corresponding data tables in the database.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 2.19 C3 Component Diagram

**C4 – Code**

**Class Diagram**

The class diagram illustrates the layered architecture consisting of:

* Controller classes that receive requests from users
* Service classes that handle business logic
* Repository classes responsible for data access
* Entity classes representing database tables

A black screen with white text

AI-generated content may be incorrect.

Hình 2.20 Class Diagram

## **2.3. Communication Flow Diagram**

The user’s browser communicates with the ReactJS Frontend over HTTP (port 3000) using HTML/JSON, with JWT used for authentication.

The Frontend (ReactJS SPA) utilizes Axios and useSWR libraries to invoke APIs over HTTP (port 8080) on the Python Backend.

The Python Backend, structured with Controller, Service, Repository, and Entity layers, interacts with the MySQL database via the TCP protocol (port 3306) using a MySQL driver / ORM framework. The backend also connects to the Momo payment gateway via HTTPS (port 443) to process payment transactions.

## **2.4. Sơ đồ triển khai**

# Chương 3. Test Plan

1. Introduction

## **1.1. Purpose**

The test plan document for the SkyWind project is developed to define the testing approach, scope, and required testing resources, ensuring that the software fully satisfies business and technical requirements. In addition, this document aims to facilitate early defect detection, improve product quality, provide consistent testing guidance, and enable effective control of the entire testing process.

## **1.2. Scope**

This test plan describes testing activities for the SkyWind website throughout its development and deployment lifecycle. The scope includes functional testing, unit testing, integration testing, system testing, and acceptance testing, based on the requirements specified in the Software Requirements Specification (SRS). Testing activities are conducted in both Development and Production environments according to the planned schedule.

2. Test Items

## **2.1. Functional Testing**

Login:

* Verify that the system allows login for customer and administrator roles.
* Verify that the system supports customer account registration.
* Verify that the system allows logout for all authenticated users.
* Verify that role-based access control is enforced, allowing users to access only the functions permitted for their roles.

Product Management:

* Verify that the system provides full **CRUD (Create, Read, Update, Delete)** operations for product management by administrators.
* Verify that the system supports management of product-related information, including product details, product status, and pricing.
* Verify that customers can access and view the product list.
* Verify that the system supports search, filtering, and sorting functionalities to help customers easily find desired products.
* Verify that product data remains consistent and accurate between the user interface and the backend system.

Shopping Cart Management:

* Verify that the system automatically updates and accurately displays shopping cart information, including product list, quantities, and total amount.
* Verify that shopping cart data remains consistent throughout user interactions and page navigation.
* Verify that business rules are enforced to limit the quantity of products added to the cart based on available inventory.
* Verify that the system allows customers to continue the purchasing process from the shopping cart to the order placement stage.

Order Management:

* Verify that the system provides complete order management functionality for customers, including placing orders, viewing order lists, and viewing order detail.
* Verify that the system provides full order management functions for administrators, including viewing, searching, filtering, and updating order status.
* Verify that the system supports the correct business workflow for order processing from order creation to completion.
* Verify that order data, including order status, product information, and total amount, is accurate and consistent.

Payment Management:

* Verify that the system provides multiple payment methods for customers, including Cash on Delivery (COD), Banking, and MoMo payments.
* Verify that payment data and order data remain accurate and consistent throughout the payment process.

## **2.2. Usability Testing**

Verify that the system provides clear navigation, allowing users to access main pages such as the homepage, product list, shopping cart, and order list.

Verify that users can easily navigate back to the previous page or the homepage during usage.

Verify that primary action buttons are clearly displayed, easily recognizable, and easy to operate.

## **2.3. Compatibility Testing**

Verify that the website interface displays correctly and consistently across popular browsers such as Chrome, Firefox, and Safari.

Verify that the system automatically adapts the interface to different screen sizes (Desktop, Tablet, Mobile).

Verify that UI components do not break or become misaligned when screen size changes.

Verify that users can fully perform system functions on different devices.

## **2.4. User Interface Testing**

Verify that colors, fonts, and display styles match the design specifications.

Verify that logos, images, and icons are displayed in the correct positions without distortion.

Verify that error messages, warnings, and status notifications are displayed correctly according to business context.

## **2.5. Security Testing**

Verify that user passwords are securely encrypted when stored in the system.

Verify that role-based access control is correctly enforced and prevents unauthorized access.

Verify that critical APIs are protected by appropriate authentication and authorization mechanisms.

## **2.6. Database Testing**

Verify that CRUD operations performed in the system are accurately reflected in the database.

Verify that data is stored using correct formats and data types according to the design.

Verify that relationships between database tables (primary keys, foreign keys, and constraints) are correctly maintained.

Verify that search, filtering, and reporting queries return correct results within acceptable response times.

Verify that the system can withstand peak load conditions without crashing (basic stress testing).

## **2.7. Regression Testing**

Verify that existing system functionalities continue to work correctly after changes or upgrades.

Verify that API endpoints, user interfaces, databases, and business processes remain compatible and accurate.

Verify that core business workflows continue to operate stably.

## **2.8. API Testing**

Verify that API endpoints function correctly according to the design specifications.

Verify that API responses are complete, accurate, and conform to the defined data formats.

Verify that APIs enforce proper access control and prevent unauthorized usage.

Verify that APIs correctly interact with the database, user interface, and related endpoints.

Verify that core business workflows through APIs remain stable after system changes.

## **2.9. Error Handling and Boundary Data Testing**

Verify that the system correctly handles invalid, empty, or out-of-range input data.

Verify that the system displays clear error messages without interrupting or crashing system operations.

Verify that the system prevents invalid business operations.

## **2.10. Performance Testing (Basic Level)**

Verify that response times for key functions such as product viewing, adding items to the cart, and placing orders are within acceptable thresholds.

Verify that the system remains stable under simulated concurrent access loads.

3. Test Items Not Included

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

Internal processing functions of the **Momo/Bank payment gateway** are excluded from the testing scope.

The testing team focuses only on integration flows between the SkyWind system and the Momo/Bank payment gateway via APIs, including payment requests and response handling.

The stability, performance, and internal security of the Momo system are outside the scope of this project.

## **3.2. Advanced Security Testing**

Advanced security testing such as penetration testing is not performed.

System-level, infrastructure-level, and network-level security vulnerabilities are outside the testing scope.

Only basic security testing is conducted, focusing on access control and user authorization.

## **3.3. Advanced Usability Testing**

In-depth usability testing (e.g., usability testing with real users, UX surveys) is not performed.

User perception, satisfaction, and behavioral analysis are outside the scope of this project.  
Only basic usability testing is conducted to verify that the system is operable and usable.

4. Acceptance Test Criteria

## **4.1. Test Coverage**

At least 95% of functional requirements must have test cases designed and executed.

All critical use cases must be tested.

## **4.2. Test Case Pass Rate**

100% test case mức Critical và Major phải đạt trạng thái Pass.

Tổng tỷ lệ test case đạt phải đạt ≥ 98%.

## **4.3. Số lượng trường hợp kiểm thử**

Toàn bộ test case ở các cấp độ sau phải được thực thi:

* Kiểm thử đơn vị (Unit Test).
* Kiểm thử tích hợp (Integration Test).
* Kiểm thử hệ thống (System Test).

## **4.4. Số lượng lỗi**

Không được tồn tại lỗi mức Critical tại thời điểm chấp nhận.

Số lỗi mức Major không vượt quá 2 lỗi và phải có giải pháp khắc phục được thống nhất.

Các lỗi mức Minor không ảnh hưởng đến chức năng chính của hệ thống.

## **4.5. Mật độ kiểm thử đơn vị**

Mật độ unit test phải đạt tối thiểu 80 Unit Test Case trên mỗi 1000 dòng mã nguồn (80 UTC/KLOC).

## **4.6. Độ bao phủ mã nguồn**

Các chỉ số bao phủ mã nguồn phải đạt:

* Statement Coverage ≥ 90%.
* Branch Coverage = 100%.
* Path Coverage = 100%.

## **4.7. Quy trình CI/CD**

Quy trình GitHub Actions CI/CD phải chạy hoàn tất với trạng thái thành công.

Không được tồn tại bất kỳ lỗi kiểm thử nào trong pipeline tự động.

5. Test Strategy

## **5.1. Testing Approach**

The testing approach for the SkyWind project combines manual testing and automated testing to ensure comprehensive validation of system functionality, usability, security, and performance.  
Manual testing is applied primarily to user interface validation, usability assessment, and exploratory testing.  
Automated testing is applied to backend logic, APIs, regression scenarios, and CI/CD pipelines to ensure testing efficiency and consistency.

## **5.2. Test Types**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Type | Objective | Techniques / Tools | Acceptance Criteria |
| Functional Testing | Ensure access control, product management, shopping cart, order, payment, and address management function correctly according to business requirements | Manual Testing; Automated Testing with pytest (Backend), Selenium (UI) | Functions operate correctly; no critical defects affecting the purchasing workflow |
| Usability Testing | Ensure users can easily operate, navigate, and use the website | Manual UI testing on ReactJS interface | User-friendly interface; clear business flows; easy-to-use interactions |
| Compatibility Testing | Ensure correct display and functionality across multiple browsers and devices | Manual testing on Chrome, Firefox, Safari; Responsive testing | UI displays correctly without layout issues; full functionality available |
| UI Testing | Ensure UI elements conform to design specifications and information is accurately presented | Manual Testing | Colors, fonts, images, and notifications match the design |
| Security Testing | Ensure user data security and access control enforcement | API authentication and authorization testing with **pytest** | No unauthorized access; passwords securely encrypted |
| Database Testing | Ensure data is stored, updated, and linked correctly in the DBMS | Integration testing with pytest and PostgreSQL | Data accuracy, integrity, and correct PK/FK constraints |
| Regression Testing | Ensure existing functionalities remain unaffected after system changes or upgrades | Automated testing with pytest and Selenium | Core business workflows remain stable |
| API Testing | Ensure API endpoints function correctly, securely, and consistently | API testing with pytest and requests library | Correct responses, valid formats, and proper access control |
| Error Handling & Boundary Testing | Ensure correct handling of invalid or boundary input data | Manual testing; automated testing with **pytest** | Appropriate error messages; no system crashes |
| Performance Testing (Basic Level) | Ensure acceptable response times for core functions | Manual testing; response time measurement | Response times within acceptable thresholds; system stability |

## **5.3. Test Levels**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Type** | **Test Levels** | | | |
| **Unit Test** | **Integration Test** | **System Test** | **Acceptance Test** |
| Functional Testing | x | x | x | x |
| UI Testing |  |  | x | x |
| Database Testing | x | x |  |  |
| Compatibility Testing |  |  | x | x |
| Security Testing | x | x | x |  |
| Performance Testing |  |  | x |  |
| Usability Testing |  |  | x | x |
| Regression Testing | x | x | x |  |
| API Testing | x | x |  |  |

6. Resources

## **6.1. Human Resources**

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Responsibilities** |
| Nguyễn Mình Quân | Tester / Developer | Test planning, system testing execution |
| Đào Cư Đạt | Tester / Developer | Test case design, unit and integration testing |
| Ngô Gia Hưng | Tester / Developer | CI/CD implementation using GitHub Actions, test documentation |
| Nguyễn Ngọc Sơn | Tester / Developer | Review checklist preparation, test documentation |
| Dr. Đỗ Như Tài | Product Owner | Acceptance testing execution |

## **6.2. Test Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Responsible** | **Start Date** | **End Date** |
| Test plan preparation | Nguyễn Minh Quân | 01/10/2025 | 12/10/2025 |
| Test case design | Đào Cư Đạt | 13/10/2025 | 22/10/2025 |
| Review checklist preparation | Nguyễn Ngọc Sơn | 23/10/2025 | 31/10/2025 |
| Unit and integration testing | Đào Cư Đạt | 01/11/2025 | 15/11/2025 |
| System testing | Nguyễn Minh Quân | 16/11/2025 | 20/11/2025 |
| CI/CD implementation (GitHub Actions) | Ngô Gia Hưng | 21/11/2025 | 25/11/2025 |
| Test documentation | Ngô Gia Hưng, Nguyễn Ngọc Sơn | 26/11/2025 | 16/12/2025 |
| Acceptance testing | Dr. Đỗ Như Tài | 21/12/2025 | 26/12/2025 |

7. Test Environment

## **7.1. Hardware**

|  |  |  |
| --- | --- | --- |
| **Category** | **Components** | **Purpose** |
| Development | Personal computers | Development, unit testing, and integration testing |
| Devices | Desktop, Android, iOS | UI and compatibility testing |

## **7.2. Software**

|  |  |  |
| --- | --- | --- |
| **Category** | **Components** | **Purpose** |
| Development | VSCode, ReactJS, Python, Local PostgreSQL | Development, unit and integration testing |
| Browsers | Chrome, Firefox, Safari | UI and compatibility testing |

## **7.3. Infrastructure**

|  |  |  |
| --- | --- | --- |
| **Category** | **Components** | **Purpose** |
| Production | Frontend and Backend deployed on Render | Production deployment, system and acceptance testing |
| PostgreSQL Hosting | PostgreSQL hosted on Aiven | Production database |
| CI/CD | GitHub Actions | Automated build, test, and deployment pipeline |

8. Deliverables

Word documents including: Architecture Design, Database Design, Test Plan, Software Requirements Specification (SRS), Bug Reports, and the overall software and testing report.

Excel documents including: UI specifications, Use Case specifications, Review Checklists, Test Reports, Test Cases, and Test Design documents.

Source code repository hosted on GitHub, including unit, integration, and system test scripts, as well as the CI/CD pipeline configuration (.github/workflows/ci-cd.yml).

# Chương 4. Test Design

# **1. Introduction**

This chapter presents the process and results of designing detailed test cases based on the V-Model testing strategy, using the documents from Chapter 1, Chapter 2, and Chapter 3 as primary inputs.

The objective of this chapter is to demonstrate how testing activities are systematically derived from requirements analysis, system design, and architectural design to ensure full traceability and test coverage throughout the software development lifecycle.

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

The team’s test design process is fully based on the V-Model, a development and testing model that emphasizes symmetry and close collaboration between development activities and testing activities.

In the V-shaped structure, each analysis and design phase on the left side corresponds to a specific testing activity on the right side. This ensures that every requirement can be verified through a corresponding and well-defined test case.

## **2.1. Requirements Analysis – Phase 1a**

In the initial phase, the team collects, analyzes, and clarifies the business requirements of the online laptop sales system. The objective of this step is to ensure that all requirements are fully defined, within scope, and testable.

The team produces the Software Requirements Specification (SRS) document, which serves as the foundation for all subsequent testing activities.

Based on the requirements analysis, the system is identified as having two primary user roles: Customer and Administrator, and is decomposed into six functional modules: Product Management, Shopping Cart, Order Management, Payment, Access Control.

The output of this phase directly supports acceptance test case design. Business requirements are transformed into acceptance criteria, ensuring that all expected user functionalities can be validated through testing.

## **2.2. System Design – Phase 2a**

After the requirements are clearly defined, the team proceeds with high-level system design. This phase includes defining major system functions, business workflows, use case diagrams, and the relationships between actors and the system.

The artifacts from this phase are used to design system test scenarios. Each function described in the use case diagrams becomes a test target, ensuring that the system operates correctly according to the overall business workflows.

## **2.3. Architecture Design – Phase 3a**

During the architecture design phase, the team applies the C4 Model (Context – Container – Component – Code) to decompose the system from the contextual level down to implementation components.

The objective is to clearly define: Communication between system layers, Separation of responsibilities, Data flow between architectural components

The architecture design outputs serve as the basis for integration testing. Interaction points between modules—such as API-to-database and frontend-to-backend—are identified and analyzed to design test cases that verify correct data exchange and interaction behavior.

## **2.4. Module Design – Phase 4a**

In the module design phase, the team analyzes and designs each system module in detail. Class diagrams are used to describe data structures, attributes, methods, and relationships between classes.

The documentation produced in this phase is used to design unit test cases. Key classes and methods of each module are identified for automated testing, enabling early defect detection at the lowest level before module integration.

## **2.5. Unit Testing – Phase 1b**

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

The goal of unit testing is to ensure that each smallest component behaves correctly according to the design specification, thereby minimizing the risk of defect propagation to integration and deployment stages.

Unit test cases are implemented using PyTest, combined with Mockito for mocking dependent objects. Mocking allows the team to isolate the test logic from external components such as databases or other services, enabling precise validation of individual functions, especially in complex or data-dependent scenarios.

All unit tests are integrated into the CI/CD pipeline using GitHub Actions, allowing automated test execution on every code update. This helps detect defects early and maintain software quality throughout the development process.

Example unit test cases in the project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case ID** | **Unit under test** | **Test description** | **Test data** | **Result** |
| UT-AUTH-003 | registerUser | Register a new user. | Email address does not exist. | Status 201 |
| UT-AUTH-004 | requireAdmin | Verify valid admin privileges. | Role = admin | Allow access |

## **2.6. Integration Testing – Phase 2b**

After individual modules are stabilized through unit testing, the team performs integration testing to evaluate interactions between system components.

Integration testing focuses on validating:

* Data transmission through APIs
* Interaction between Controller – Service – Repository layers
* Compatibility between backend services and the database

The team uses PyTest combined with MockMvc to test API integration in a near-realistic simulated environment. MockMvc allows sending requests, receiving responses, and validating the entire processing flow without running the full application on a server.

During testing, the production database is replaced with an H2 in-memory database, ensuring test isolation, repeatability, and no impact on real data.

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

Example integration test cases in the project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case ID** | **Unit under test** | **Test description** | **Test data** | **Result** |
| IT-PAY-001 | Payment API | Create payment | POST /api/payment | 201 Created |
| IT-PROD-001 | Product API | Get the product list | GET /api/products | 200 OK |

## **2.7. System Testing – Phase 3b**

At a higher level, the team conducts system testing to evaluate the software as a complete and integrated system.

System testing is performed manually based on the test cases designed during the test design phase and covers all use cases and primary business workflows.

Testing activities include:

* Functional testing: End-to-end workflows such as login, purchasing, and payment
* UI testing: Ensuring a consistent, intuitive, and responsive user interface
* Basic security testing: Verifying authentication, authorization, and access control

System test results reflect the overall software quality before proceeding to acceptance testing.

## **2.8. Acceptance Testing – Phase 4b**

Acceptance testing is the final phase of the testing process, aimed at evaluating the system from the end-user perspective.

At this stage, acceptance criteria are validated against the SRS to ensure that the system fully meets customer requirements.

After completing internal system testing, the software is delivered to the Product Owner for acceptance testing. The Product Owner acts as an end user, simulates real-world scenarios, and evaluates system usability, stability, business suitability, and requirement fulfillment.

Acceptance test results serve as the final basis for deciding whether the system is ready for deployment into the operational environment.

# **3. Test Design Techniques**

## **3.1. Black-box Testing**

Black-box testing is the primary testing technique applied throughout the SkyWind project. With this technique, the system is evaluated based on external behavior through input data and output results, without considering the internal structure or implementation logic of the source code.  
Black-box testing is used for all functional test cases, including login, registration, authorization, product management, cart, order, and payment functionalities.  
For API testing, endpoints are validated using both valid and invalid requests to verify HTTP status codes, response data, and error messages.  
The application of black-box testing ensures that the system meets business requirements and aligns with the end-user perspective.

## **3.2. Equivalence Partitioning**

Equivalence Partitioning is applied to reduce the number of test cases while maintaining adequate test coverage. Input data is divided into valid and invalid equivalence classes, from which representative values are selected for testing.  
This technique is widely used in authentication and data input functionalities, including login, account registration, user profile updates, product creation and modification, and payment information input.  
For example, in the login function, input data is divided into classes such as valid credentials, non-existent accounts, incorrect passwords, and empty inputs.  
Applying equivalence partitioning helps detect defects effectively without testing every possible input combination.

## **3.3. Boundary Value Analysis**

Boundary Value Analysis is applied to functions with data constraints to ensure the system correctly handles edge conditions.  
This technique is mainly used in features related to product quantities in the cart, inventory limits, purchase quantities, and required form fields.  
Test cases such as quantity equal to zero, equal to stock limit, exceeding stock limit, or missing required fields are executed to identify potential boundary-related defects.  
Applying boundary value analysis helps minimize critical errors that may directly affect data integrity and the purchasing process.

## **3.4. Workflow-based Testing**

Workflow-based testing is implemented by designing **TestDesign\_Workflow** based on the system’s activity diagrams. Each workflow represents a complete business process from the beginning of user interaction to the completion of a function.  
This technique is applied to core functionalities such as login, browsing and searching products, adding items to the cart, placing orders, payment processing, and order management by administrators.  
Test cases are directly mapped to each step in the workflow to ensure all execution paths, including both success and failure flows, are thoroughly tested.  
Workflow-based testing helps evaluate process integrity and coordination between the frontend, backend, and database.

## **3.5. Role-based Testing**

Role-based testing is applied to ensure the system enforces correct access control and authorization mechanisms. Test cases are designed separately for each role, such as Buyer and Admin, to verify feature accessibility, permission boundaries, and handling of unauthorized access attempts.  
This technique is used for accessing the admin dashboard, managing products, managing orders, and administrator-specific APIs.  
Role-based testing ensures system security, prevents unauthorized access, and ensures compliance with defined business requirements.

## **3.6. API Testing**

API testing is conducted to verify backend services independently of the user interface. APIs related to authentication, products, cart, orders, and payment are tested using tools such as **Postman** and **MockMvc**.  
Test cases focus on validating HTTP methods, request payloads, response data, status codes, and error handling.  
For third-party integrations such as **MoMo payment**, scenarios including redirection, successful transactions, and failure cases are tested to ensure proper system handling.  
API testing helps detect backend issues early and provides strong support for integration testing.

## **3.7. Usability Testing**

Usability testing is performed manually to evaluate the overall user experience. Usability test cases are designed for core functionalities such as login, product search, cart operations, address input, and payment.  
Testing focuses on interface layout, button visibility, feedback messages, and form clarity.  
Applying usability testing ensures that the system is user-friendly and effectively meets real-world usage requirements.

# **4. Test Design Methods**

## **4.1. Manual Testing**

Manual testing is applied primarily during the acceptance testing phase. This method evaluates the system from the end-user perspective, validating key business workflows and confirming compliance with specified requirements.

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

## **4.2. Automated Testing**

The project applies automated testing through CI/CD integration using GitHub Actions, automating the build, test, and deployment processes.

Automated testing is used for white-box testing, including unit testing, integration testing, and selected system testing scenarios.

Whenever source code changes are pushed to the GitHub repository, the CI/CD pipeline automatically triggers a test job that builds the project and executes all automated test suites.

* If the test job fails, the pipeline automatically creates a GitHub Issue to record the defect and support timely tracking and resolution.
* If the test job succeeds, the pipeline automatically closes related testing issues, ensuring accurate system status updates.

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

# **5. Applying GenAI for Test Case Generation**

In this project, Generative AI (GenAI) is used as a tool to support the test case design phase. GenAI does not replace the role of testers but plays a role in analyzing requirements and suggesting initial test scenarios, thereby helping to improve test efficiency and coverage.

## **5.1. GenAI Inputs**

GenAI's input includes documents such as System Requirements, SRS, and User Stories. These documents provide information about the system's functionality, business logic, and constraints.

## **5.2. GenAI Processing Phase**

During the GenAI Processing phase, GenAI performs the following steps:

* Semantic analysis of requirements to understand the functional and business objectives of the system.
* Generating test scenarios based on the main and sub-flows of the function.
* Generating rough test cases, including basic test cases, boundary cases, and error cases.

## **5.3. Test Case Finalization**

After being modified and supplemented, the test cases that meet the requirements will become the official test cases and will be used in the testing process (manual or automated).

# Chương 5. Test Report

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

After completing the test design phase described in Chapter 4, the team conducted testing activities for the SkyWind system in accordance with the established test plan. The testing process was carried out over a period of six weeks, covering multiple phases including unit testing, integration testing, system testing, and acceptance testing.

The team focused on validating all core business functionalities of the system. Test cases were executed carefully and thoroughly, and all test results were fully documented in the project deliverables.

# **2. Test Case Report**

## **2.1. Introduction**

These test cases are categorized into two groups: Positive (testing behavior that matches system expectations) and Negative (testing the ability to handle errors, exceptions, and invalid conditions).

The test suite was built for the five main functional modules of the system: access control, products, shopping cart, orders, payment, and address book, with a total of 79 test cases.

## **2.2. Test Coverage**

The team built 79 test cases covering 100% of the system's core functions. Non-functionality coverage was 0%.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module** | **Code** | **Number of test case** | **Description** | **Result** |
| Authentification | TC-AUTH-SYS | 11 | Verify login and registration workflows; verify user management operations including create, update, and delete users | 11 Pass |
| Product Management | TC-PROD-SYS | 12 | Verify product display, search, and filtering features for customers; verify create, update, and delete product operations for administrators | 11 Pass  1 Fail |
| Cart | TC-CART-SYS | 8 | Verify add, update, and remove cart items; validate stock quantity when adding products to the cart | 8 Pass |
| Order | TC-ORDER-SYS | 5 | Verify order management and order status update operations for administrators | 5 Pass |
| Payment | TC-ORDER-SYS | 8 | Verify payment methods (Cash on Delivery and MoMo); validate transaction result handling scenarios | 8 Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module** | **Code** | **Number of Test Cases** | **Discription** | **Result** |
| Authentification | UT-AUTH | 5 | Unit test | 5 Pass |
| Product Management | UT-PROD | 7 | Unit test | 7 Pass |
| Product Management | IT\_PROD | 4 | Intergration test | 4 Pass |
| Order | UT-ORDER | 6 | Unit test | 6 Pass |
| Payment | UT-PAY | 4 | Unit test | 4 Pass |
| Payment | IT-PAY | 1 | Intergration test | 1 Pass |
| User | IT-User | 2 | Intergration test | 2 Fail |
| Selenium | SEL | 2 | Selenium test | 2 Pass |
| Performance | UT-PERM | 3 | Performance test | 3 Pass |

## **2.3. Test Execution Results**

The pie chart shows the execution percentage of all test cases.

A total of 78 test cases were executed, with 75 successful and 3 unsuccessful.

• 97.2% of test cases passed.

• 2.8% of test cases failed.

• 0% of test cases were blocked.

• 0% of test cases did not run.

Hình 5.1. Biểu đồ tròn tỷ lệ phần trăm thực thi Test Case.

# **3. Defect Report**

## **3.1. Defect Classification by Severity**

During the test case review, the team discovered and recorded 3 errors. These errors are categorized by severity as follows:

Severity 2 – High: There is 1 error, this is a system-related error. A fairly serious error is that the administrator cannot delete a product.

Severity 4 – Low: There is 1 minor error; during integration testing, a new user cannot be created, but a new user can still be added to the user's product page.

Severity 4 – Low: There is 1 minor error; during integration testing, a new user ID cannot be obtained, but a new user can still be added to the user's product page.

## **3.2. Defect Statistics**

The bar chart shows the number of errors in the Test Case by severity level.

A total of 3 errors were detected, of which:

• 66% are 2 low-level errors.

• 33% are 1 high-level error.

• 0 critical and medium-level errors.

The number of errors fixed is 2 low-level errors and 1 high-level error.

The number of remaining errors is 0 at any level.

A graph with blue bars

AI-generated content may be incorrect.

Hình 5.2. Biểu đồ cột số lượng lỗi theo mức độ nghiêm trọng Test Case.

## **3.3. Defect Resolution Process**

The defect handling process is conducted according to standardized procedures to ensure that all identified defects are properly recorded, classified, and resolved in a timely manner throughout the project. The detailed steps are as follows:

**Step 1: Defect Logging**  
During the testing process, all detected defects are documented with the following information: defect description, related module, associated Test Case/Test Script, severity level, defect reporter, and the time of detection.

**Step 2: Defect Classification**  
The testing team classifies defects based on their severity levels.  
This classification enables the development team to prioritize defect resolution according to their criticality and impact on the system.

**Step 3: Defect Assignment**  
Defects are assigned to the developer or development team responsible for the related module. Each defect is assigned a status (Assigned) and a designated owner responsible for fixing it.

**Step 4: Defect Fixing**  
The development team proceeds to fix defects based on the identified root causes:

* **Data-related defects**: modifying business logic or validating input data.
* **UI defects**: adjusting the user interface or adding missing functionalities.
* **CI/CD defects**: correcting class names and refactoring code to comply with the Google Java Spotless formatting rules.

**Step 5: Defect Retesting**  
After the defect is fixed, the testing team performs retesting to verify that the defect has been successfully resolved. If the defect still persists, it is reassigned and returned to Step 3 for further correction.

**Step 6: Defect Closure**  
A defect is marked as **Closed** when:

* The defect has been completely fixed and successfully retested.
* The fix does not introduce any adverse impact on other system functionalities.

All closed defects are documented in the final test summary report.

**Step 7: Defect Reporting and Tracking**  
The testing team consolidates defect statistics based on severity levels, resolution status, and related Test Case/Test Script references.  
These reports are used to assess overall software quality and to improve the testing and defect management process in subsequent project phases.