A black background with blue and purple text

Description automatically generated

**FINAL REPORT**

**COURSE: SOFTWARE TESTING**

**PROJECT : APPLYING TESTING TO A CAKE SELLING WEBSITE**

**Instructor: Do Nhu Tai**

**Member name:**

+ Huynh Anh Tuan Student ID: 3121411220

+ Phung Tan Nhut Student ID: 3121411159

**School year: 2024 - 2025**

**INDEX**

[CHAPTER 1: OVERVIEW OF BAKERY WEBSITE TESTING 7](#_Toc185517511)

[1. Introduction 7](#_Toc185517512)

[2. Business context 8](#_Toc185517513)

[3. Business process 8](#_Toc185517514)

[3.1. Import process row 8](#_Toc185517515)

[3.2. Purchasing process​ 9](#_Toc185517516)

[4. Problem solving subsystems 11](#_Toc185517517)

[4.1. Import process 11](#_Toc185517518)

[4.2. Purchasing process 12](#_Toc185517519)

[5. Conceptual Model 13](#_Toc185517520)

[6. User Stories 14](#_Toc185517521)

[7. Business requirements 15](#_Toc185517522)

[7.1. Functional requirements 15](#_Toc185517523)

[7.2. Non-functional requirements 15](#_Toc185517524)

[8. System participants 17](#_Toc185517525)

[CHAPTER 2: SOFTWARE REQUIREMENTS SPECIFICATION 18](#_Toc185517526)

[1. Deployment diagram 18](#_Toc185517527)

[1.1. Import goods 18](#_Toc185517528)

[1.2. Purchase 18](#_Toc185517529)

[2. Overview diagram of system functions 19](#_Toc185517530)

[3. General database diagram 20](#_Toc185517531)

[4. Import goods 21](#_Toc185517532)

[4.1. Focused UseCase 21](#_Toc185517533)

[4.2. Activity Diagram 24](#_Toc185517534)

[4.3. Sequence Diagram 25](#_Toc185517535)

[4.4. Class Diagram 26](#_Toc185517536)

[4.5. Interface illustration 26](#_Toc185517537)

[5. Purchase​ 29](#_Toc185517538)

[5.1. Focused UseCase 29](#_Toc185517539)

[5.2. Activity Diagram 33](#_Toc185517540)

[5.3. Sequence Diagram 34](#_Toc185517541)

[5.4. Class Diagram 35](#_Toc185517542)

[5.5. Illustrative interface​ 36](#_Toc185517543)

[CHAPTER 3: TEST PLAN 39](#_Toc185517544)

[1. Introduction 39](#_Toc185517545)

[1.1. Purpose 39](#_Toc185517546)

[1.2. Background 39](#_Toc185517547)

[1.3. Scope 39](#_Toc185517548)

[1.4. Project identification 39](#_Toc185517549)

[1.5 References 40](#_Toc185517550)

[2. Features to test 41](#_Toc185517551)

[2.1. Function (Customer) 41](#_Toc185517552)

[2.1.1. Add products to cart 41](#_Toc185517553)

[2.1.2. Remove products from cart 41](#_Toc185517554)

[2.1.3. Update products in cart 41](#_Toc185517555)

[2.1.4. Shopping cart payment 41](#_Toc185517556)

[2.2. Function (Manager) 42](#_Toc185517557)

[2.2.1. Add products to inventory 42](#_Toc185517558)

[2.2.2. Remove product from inventory 42](#_Toc185517559)

[2.2.3. Update products in stock 42](#_Toc185517560)

[2.3. Availability 42](#_Toc185517561)

[2.4. Design limitations 42](#_Toc185517562)

[2.5. Interface 43](#_Toc185517563)

[2.5.1. User interface 43](#_Toc185517564)

[3. Features that do not need testing 43](#_Toc185517565)

[3.1. Function (Customer) 43](#_Toc185517566)

[3.2. Function (Manager) 44](#_Toc185517567)

[4. Testing strategy 47](#_Toc185517568)

[4.1. Types of testing 47](#_Toc185517569)

[4.2. Tools 47](#_Toc185517570)

[5. Resources 47](#_Toc185517571)

[6. Project milestones 48](#_Toc185517572)

[7. Output products 48](#_Toc185517573)

[7.1 Testing model 48](#_Toc185517574)

[7.2 Test log 48](#_Toc185517575)

[CHAPTER 4: TESTING METHODS 49](#_Toc185517576)

[1. Unit Testing 49](#_Toc185517577)

[1.1. TestPlan 50](#_Toc185517578)

[1.2. Import process 53](#_Toc185517579)

[1.3. Purchasing process 56](#_Toc185517580)

[2. Integration Testing 58](#_Toc185517581)

[2.1. Test Design 58](#_Toc185517582)

[2.2. White box testing​ 62](#_Toc185517583)

[2.2.1. Import process 64](#_Toc185517584)

[2.2.2. Purchasing process 75](#_Toc185517585)

[2.3. Black box testing 85](#_Toc185517586)

[2.3.1. Import process 87](#_Toc185517587)

[2.3.2. Purchasing process 94](#_Toc185517588)

[3. System Testing 99](#_Toc185517589)

[3.1. TestCase List 100](#_Toc185517590)

[3.2. Import process 102](#_Toc185517591)

[3.3. Purchasing process 105](#_Toc185517592)

[3.4 . Non-functional 107](#_Toc185517593)

[4. Acceptance Testing 107](#_Toc185517594)

[CHAPTER 5: TEST RESULTS 109](#_Toc185517595)

[1. Test Report 109](#_Toc185517596)

[2. Bug Report 110](#_Toc185517597)

[3. DefectList 111](#_Toc185517598)

[4. Auto Test 111](#_Toc185517599)

[CHAPTER 6. CONCLUSION 116](#_Toc185517600)

**LIST OF CATEGORIES, IMAGES**

[Figure 1. Import process diagram 11](#_Toc185517677)

[Figure 2. Purchasing process diagram 12](#_Toc185517678)

[Figure 3. Conceptual model 13](#_Toc185517679)

[Figure 4. System participants 17](#_Toc185517680)

[Figure 5. Import Deployment Diagram 18](#_Toc185517681)

[Figure 6. Purchasing Deployment Diagram 18](#_Toc185517682)

[Figure 7. General UseCase 19](#_Toc185517683)

[Figure 8. Database Diagram 20](#_Toc185517684)

[Figure 9. Activity diagram of the Import process 24](#_Toc185517685)

[Figure 10. Sequence diagram of the Import process 25](#_Toc185517686)

[Figure 11. Class diagram of the Import process 26](#_Toc185517687)

[Figure 12. Product deletion interface Import goods 26](#_Toc185517688)

[Figure 13. Add product interface Import goods 27](#_Toc185517689)

[Figure 14. Product update interface Import goods 28](#_Toc185517690)

[Figure 15. Activity diagram of Purchasing process 33](#_Toc185517691)

[Figure 16. Purchase Process Sequence Diagram 34](#_Toc185517692)

[Figure 17. Purchase Process Class Diagram 35](#_Toc185517693)

[Figure 18. Default interface for adding a product to cart when purchasing 36](#_Toc185517694)

[Figure 19. Interface for adding products with optional quantity when purchasing 37](#_Toc185517695)

[Figure 20. Interface for viewing and updating shopping cart 37](#_Toc185517696)

[Figure 21. Interface to delete products from cart 38](#_Toc185517697)

[Figure 22. Shopping cart payment interface​ 38](#_Toc185517698)

[Figure 23. TestPlan Test Plan Table 50](#_Toc185517699)

[Figure 24. Unit Test Importing goods 53](#_Toc185517700)

[Figure 25. Purchase Unit Test 56](#_Toc185517701)

[Figure 26. Test Design Import 59](#_Toc185517702)

[Figure 27. Test Design Purchase 60](#_Toc185517703)

[Figure 28. White Box CheckList 62](#_Toc185517704)

[Figure 29. Control flow diagram Add product Import goods 64](#_Toc185517705)

[Figure 30. Product Update Control Flow Diagram 68](#_Toc185517706)

[Figure 31. Control flow diagram Delete product Import goods 73](#_Toc185517707)

[Figure 32. Control flow diagram Add product Shopping cart 75](#_Toc185517708)

[Figure 33. Control flow diagram of Update Shopping Cart 78](#_Toc185517709)

[Figure 34. Control flow diagram Delete product Shopping cart 80](#_Toc185517710)

[Figure 35. Control flow diagram of Checkout Shopping Cart 82](#_Toc185517711)

[Figure 36. Black Box CheckList 85](#_Toc185517712)

[Figure 37. Decision table Add product 87](#_Toc185517713)

[Figure 38. Product Update Decision Table 90](#_Toc185517714)

[Figure 39. Product Delete Decision Table 93](#_Toc185517715)

[Figure 40. Decision table Add product to Cart 94](#_Toc185517716)

[Figure 41. Decision table Update quantity in Shopping Cart 95](#_Toc185517717)

[Figure 42. Decision table to Remove product from shopping cart 96](#_Toc185517718)

[Figure 43. Cart Checkout Decision Table 97](#_Toc185517719)

[Figure 44. List of test cases 100](#_Toc185517720)

[Figure 45. Module and test results for the import process 102](#_Toc185517721)

[Figure 46. Data Test for the import process 102](#_Toc185517722)

[Figure 47. TestCase of import process 104](#_Toc185517723)

[Figure 48. Module and test results for the purchasing process 105](#_Toc185517724)

[Figure 49. Test Data for the purchasing process 105](#_Toc185517725)

[Figure 50. Purchase Process TestCae 106](#_Toc185517726)

[Figure 51. Test results for non-functionality 107](#_Toc185517727)

[Figure 52. Test Data for Non-Functional 107](#_Toc185517728)

[Figure 53. Non-functional TestCase 107](#_Toc185517729)

[Figure 54. Q&A 108](#_Toc185517730)

[Figure 55. TestReport of TestCases (UnitTest 109](#_Toc185517731)

[Figure 56. TestReport of TestCases (white box and black box) 110](#_Toc185517732)

[Figure 57. DefectList 111](#_Toc185517733)

# CHAPTER 1: OVERVIEW OF BAKERY WEBSITE TESTING

## 1. Introduction

Software testing is an important step in the process of developing a Bakery Website, ensuring that the system operates correctly, stably, and provides the best experience for users. The Bakery Website is designed with many functions such as product management, import, and online shopping, meeting the operational needs of administrators and customers.

Website Bakery software testing focuses on verifying all system features, detecting errors, ensuring compatibility with different environments, and fully meeting business requirements.

**Objective of testing**

* **Ensure accuracy:** Test features like adding products, updating carts, and checkout to ensure they work as expected.
* **Error Detection and Correction:** Identify potential problems in the system before going live.
* **Enhance user experience:** Ensure the interface is easy to use and functions work smoothly across multiple devices.
* **Performance Evaluation:** Tests the system's ability to handle concurrency and responsiveness under high load conditions.

**The Importance of Testing Bakery Website**

* Testing helps ensure that the Bakery Website operates smoothly and without errors during use.
* Ensure product information, orders, and customer data are always stored accurately.
* Create trust and satisfaction for both administrators and customers.
* Minimize business risk by preventing critical errors before systems are officially deployed.

## 2. Business context

**Bakery Website** has basic features for **Importing and Purchasing**

With **Import Goods** , administrators can add, edit, and delete products in the system. After adding or editing products, the system will automatically update the product list with information such as product name, description, price, inventory quantity, and image. When necessary, administrators can delete products that are no longer for sale. Changes will be automatically saved and updated in the system, helping to ensure that the product list is always accurate and complete.

With **Buy** , customers can add cakes to their cart from the product list or from the product detail page. After adding products to their cart, customers can view their updated cart with information such as the total cost of the cake **.** and the total amount to be paid. Customers can adjust their cart by updating quantities or removing items. When ready, customers can proceed to checkout to complete their purchase.

Some initial setup work needs to be completed when the Bakery Website is launched, including creating an administrator account, two customer accounts, and sample data for bakery products, inventory, and reviews for a few cakes.

## 3. Business process

### 3.1. Import process row

**Add product:**

* **The manager** enters product information (name, price, image, description, quantity in stock, category, brand).
* Information is checked for validity.
* The product is added to the database and displayed in the product list.

**Product Update :**

* **The administrator** selects the product to edit.
* Update information (name, price, image, quantity in stock...).
* Update changes to the database and display the new information on the interface.

**Delete product:**

* **The administrator** selects the product to delete.
* The system checks if the product is in an unprocessed order:
  + If yes: No can be deleted.
  + If not: Delete the product from the database.

### 3.2. Purchasing process​

**Add products to cart:**

* **User** selects product and quantity to buy.
* Product added to cart:
  + If the product is already in the cart: Update quantity.
  + If not: Add product to cart list.
* Show list of products in cart with total quantity
* The system automatically calculates the total price based on the quantity and price of the products in the shopping cart.

**Update quantity of products in cart:**

* **User** adjusts the quantity of products in the cart.
* Inventory checking system:
  + If sufficient: Update cart.
  + If not: Notify and keep the old quantity.

**Delete product in Cart:**

* **User** selects product to delete.
* The product is removed from the cart and the total is updated.

**Bar shopping cart :**

* **User** confirms order in cart.
* Testing system :
  + Delivery information (address, phone number).
* If eligible:
  + Payment successful.
  + The system reduces the quantity of products in stock accordingly.
  + Create order code and save to purchase history.
* If not eligible:
  + Report the error and keep the cart.

## 4. Problem solving subsystems

### 4.1. Import process

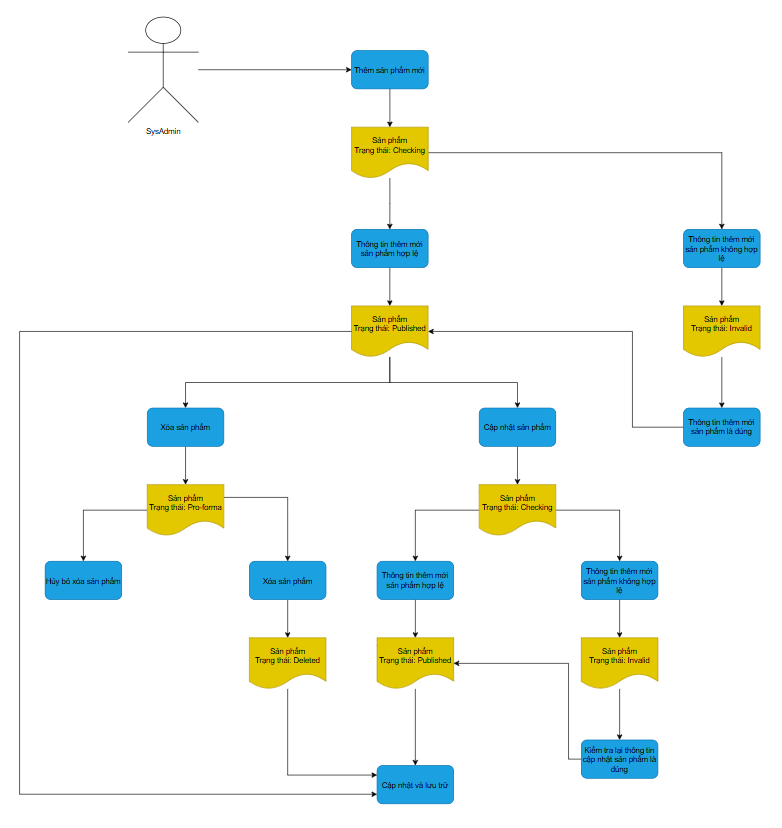


Figure 1. Import process diagram

### 4.2. Purchasing process

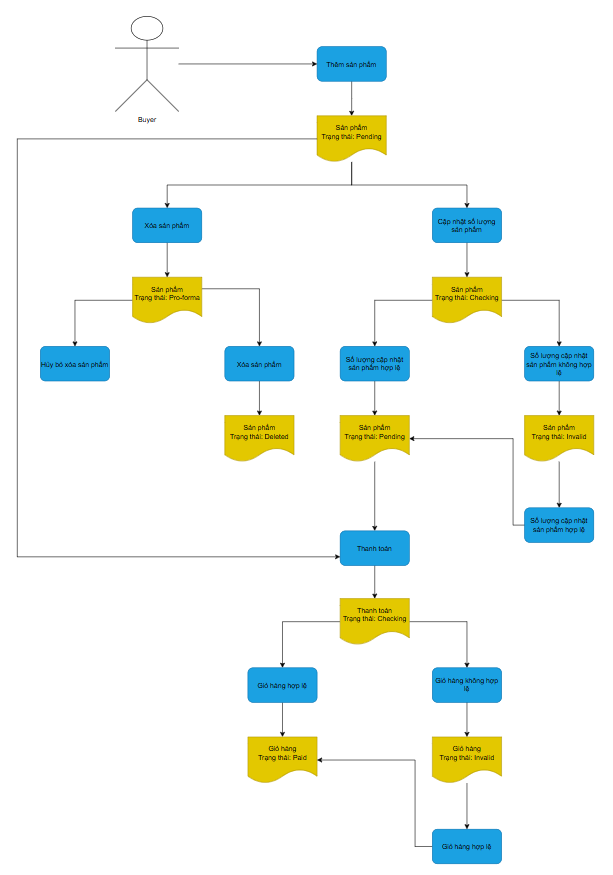


Figure 2. Purchasing process diagram

## 5. Conceptual Model

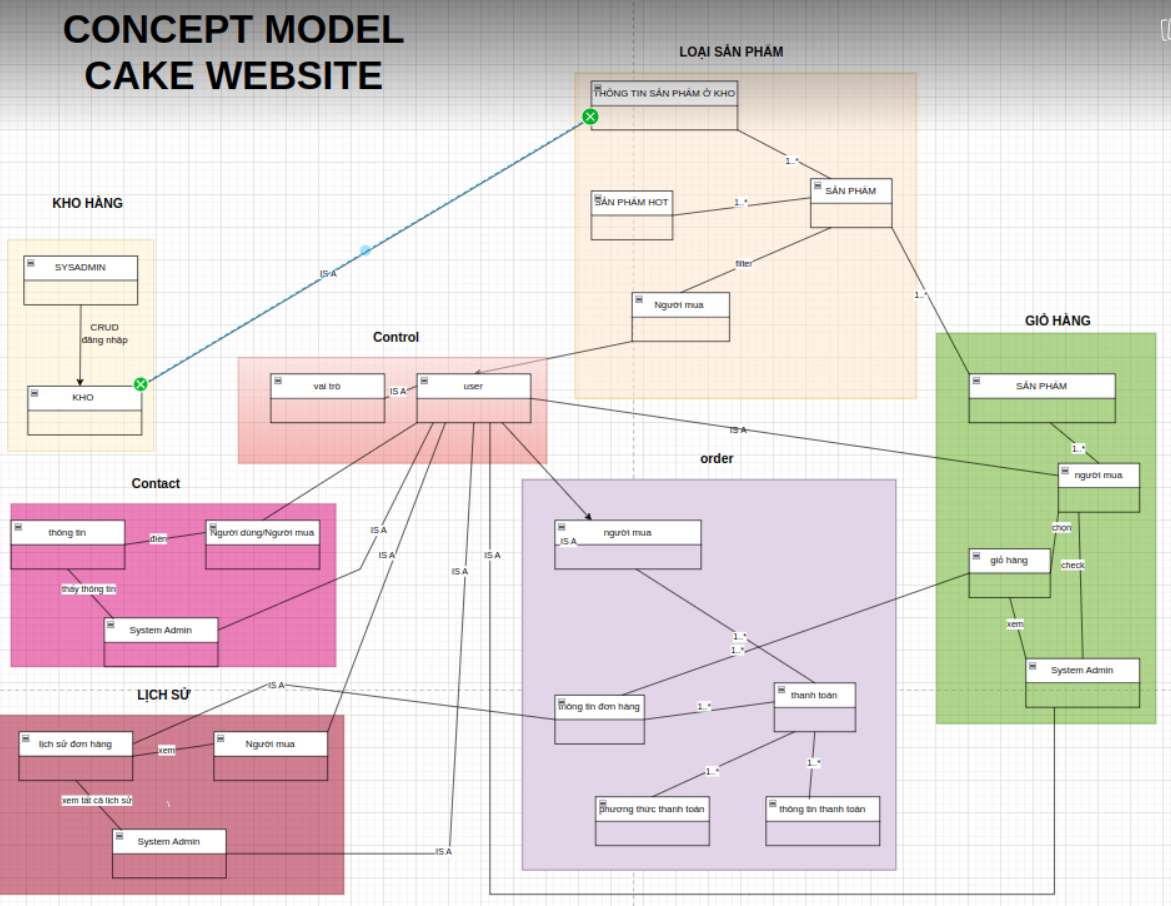


Figure 3. Conceptual model

## 6. User Stories

**Import goods**

* **As an administrator** , I want to add products to the system, including name, category, brand, selling price, import price, description, image and stock quantity.
* **As an administrator** , I want to delete a product from the system when it is no longer available for sale or out of stock.
* **As an admin** , I want to edit product information, including updating price, description, inventory quantity, images, and other attributes.
* **As a system** , I want to check the input data when adding or editing products to ensure the information is valid.
* **As a system** , I want to save product edit history so that I can look up and check past changes.

**Purchase​**

* **As a buyer** , I want to buy any product on the product category page.
* **As a buyer** , I want to see product details to decide to buy the product. If I like the product and it is in stock, I can add it to my cart.
* **As a buyer** , I want to see the list of products in the cart and the total cart cost.
* **As a buyer** , I want to update the quantity of products in the shopping cart. If the product is out of stock The system will give an error message and not update the quantity . When I increase or decrease the quantity, the total cost of the cart will be automatically updated.
* **As a buyer** , I want to remove unnecessary products from the shopping cart. After deletion, the total cost and product list will be updated immediately.
* **As a buyer** , I want to check out my cart. If the cart is empty (product quantity is 0), I will get a message that I cannot check out.
* **As a buyer** , I want to save my shopping cart information so I can return to shopping after I have not paid.
* **As a system** , I will send a confirmation email to the buyer upon successful purchase .

## 7. Business requirements

### 7.1. Functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Request Code** | **Requestor** | **Request content** | **Essential level** |
| **WB1** | Administrator | **Import goods :** Add, delete, edit products. | High |
| **WB1.1** | Administrator | Add products with information: name, category, brand, price, description, quantity, image. | High |
| **WB1.2** | Administrator | Delete product, only do this when there are no related orders. | High |
| **WB1.3** | Administrator | Edit product information, save change history. | High |
| **WB2** | User | **Purchase :** Add products, update quantity, delete products, checkout | High |
| **WB2.1** | User | More products in cart | High |
| **WB2.2** | User | Update quantity of products in cart, check inventory. | High |
| **WB2.3** | User | Remove product from cart . | High |
| **WB2.4** | User | Confirm information before payment . | High |
| **WB2.5** | User | After successful payment, reduce the quantity of products in stock accordingly. | High |

### 7.2. Non-functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Request Code** | **Requestor** | **Description of requirements** | **Priority level** |
| **WB5** | Administrator | **Load Handling** | High |
| **WB5.1** | Administrator | The system must be able to handle a large number of concurrent users without any loss of performance. | High |
| **WB6** | Administrator | **Security** | High |
| **WB6.1** | Administrator | The system must use strong authentication mechanisms (e.g. 2FA – 2-factor authentication) when users log in and perform important transactions. | High |
| **WB6.2** | Administrator | The system must have protections against attacks such as SQL Injection, XSS, CSRF, DoS/DDoS. | High |
| **WB6.3** | Administrator | All user connections and transactions must be encrypted and protected via HTTPS. | High |

## **8. System participants**

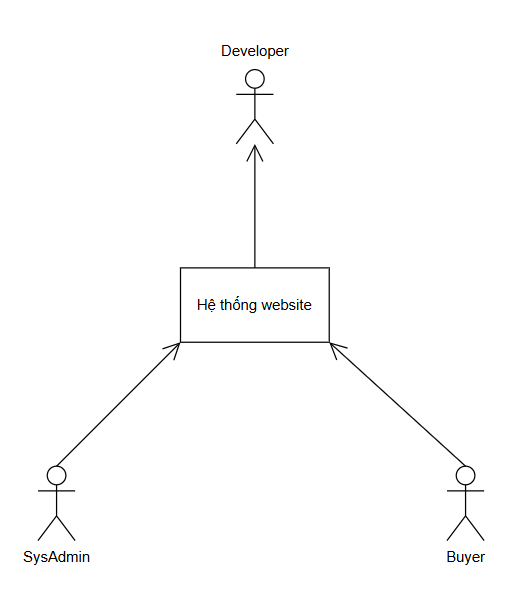


Figure 4. System participants

# CHAPTER 2: SOFTWARE REQUIREMENTS SPECIFICATION

## 1. Deployment diagram

### 1.1. Import goods



Figure 5. Import Deployment Diagram

### 1.2. Purchase

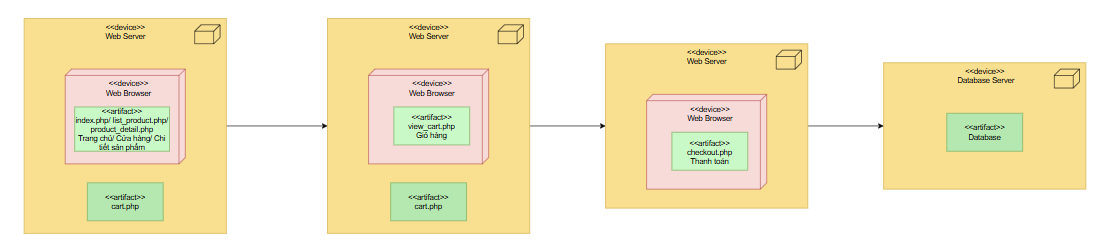
****

Figure 6. Purchasing Deployment Diagram

## 2. Overview diagram of system functions

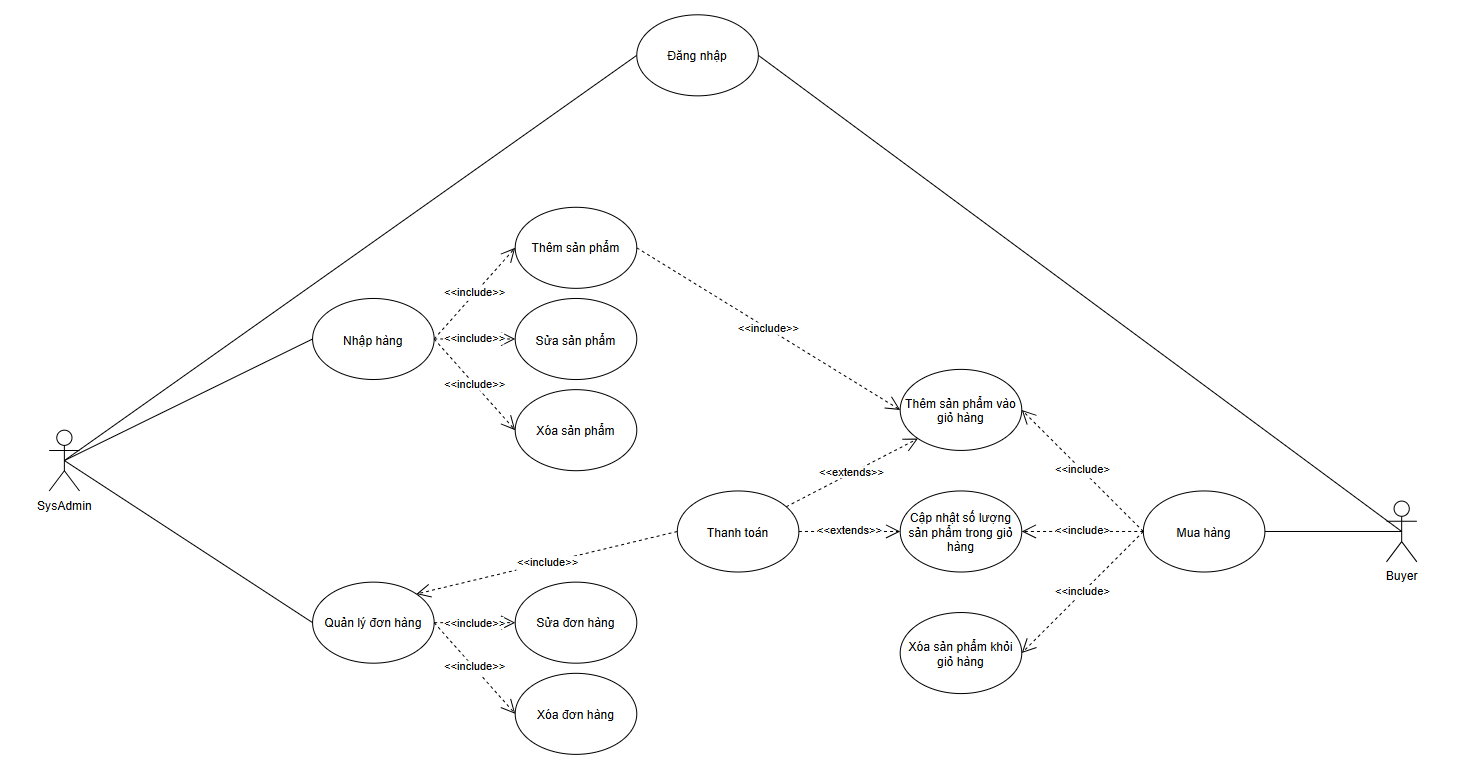


Figure 7. General UseCase

## 3. General database diagram

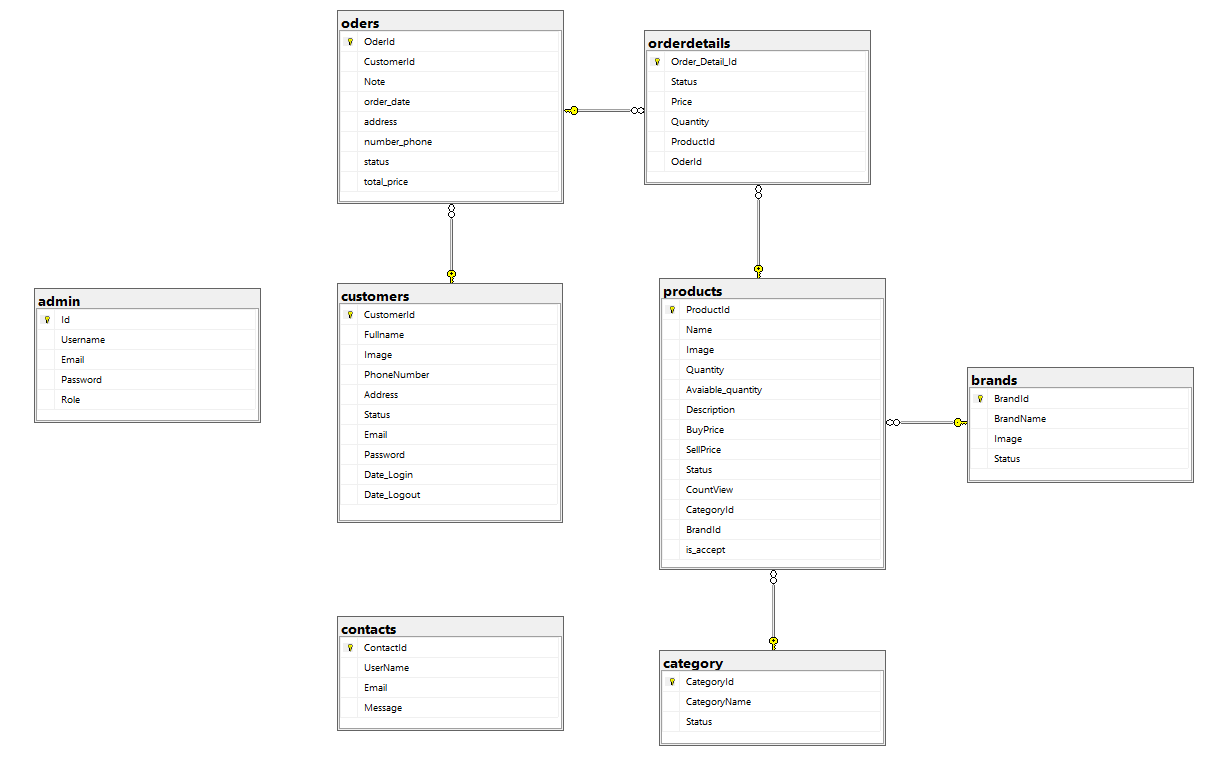


Figure 8. Database Diagram

## 4. Import goods

### 4.1. Focused UseCase

|  |  |  |
| --- | --- | --- |
|  | | |
| **Use case number** | **UC1** | |
| **Use case name** | **Import and export** | |
| **Actor(s)** | **Manager** | |
| **Maturity** | **Focused** | |
| **Summary** | *Use case describes the process of importing and exporting goods of the manager in the system.* | |
| **Basic Course of events** | **Actor Action** | **System response** |
| 1. Execute (Login Authentication) |  |
|  | 2. Home page display system |
| 3. The Use Case begins when the Actor selects “ Product ” |  |
|  | 4. The system displays a list of products. |
| 1.Actor clicks “Add new” **A1 A2** |  |
|  | 2. The system displays the form to add new products. |
| 3. Actor enters new information |  |
|  | 4. Implementation system  **{Check text box} E 1** |
|  | 5. System updates data into database |
| **Alternative Paths** | **A1** |  |
| **Actor Action** | **System response** |
| 1.Actor clicks “Edit” |  |
|  | 2. The system displays the product update form |
| 3. Actor enters updated information |  |
|  | 4. Implementation system  **{Check text box} E 1** |
|  | 5. System updates data into database |
| **A2** |  |
| **Actor Action** | **System response** |
| 1.Actor clicks “Delete” |  |
|  | 2. The system requires confirmation. |
| 3. Actor selects “OK” **A4** confirmation |  |
|  | 6. The system updates data into the database. |
| **A4** |  |
| **Actor Action** | **System response** |
| 1. Actor can click “ Cancel” to operate |  |
|  | 2 .The system deletes the confirmation request . Return to step 4 of Basic Course of Events. |
| **Exception Path** | **E1 .** At **{Check text cell}** , check if the text cell is empty, has the correct data type and constraint, if so, report the corresponding error message | |
| **Triggers:** | When actor wants to import or export goods | |
| **Assumption:** | None | |
| **Preconditions:** | None | |
| **Post Conditions:** | None | |

### 4.2. Activity Diagram

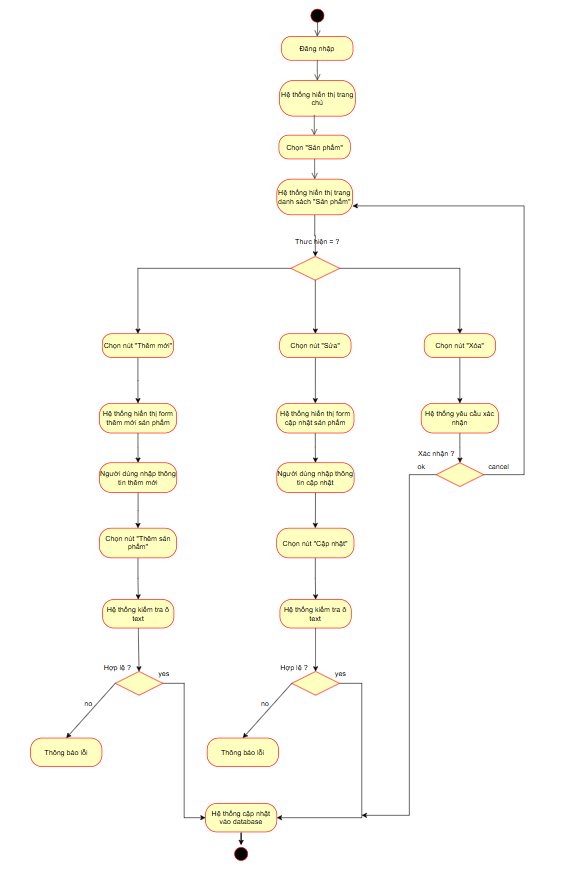
****

Figure 9. Activity diagram of the Import process

### 4.3. Sequence Diagram

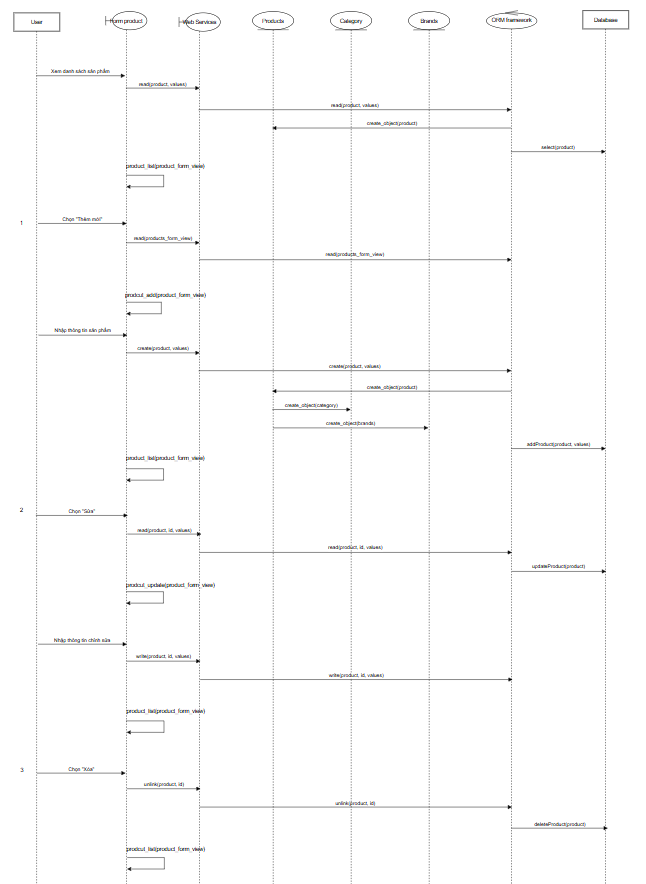


Figure 10. Sequence diagram of the Import process

### 4.4. Class Diagram

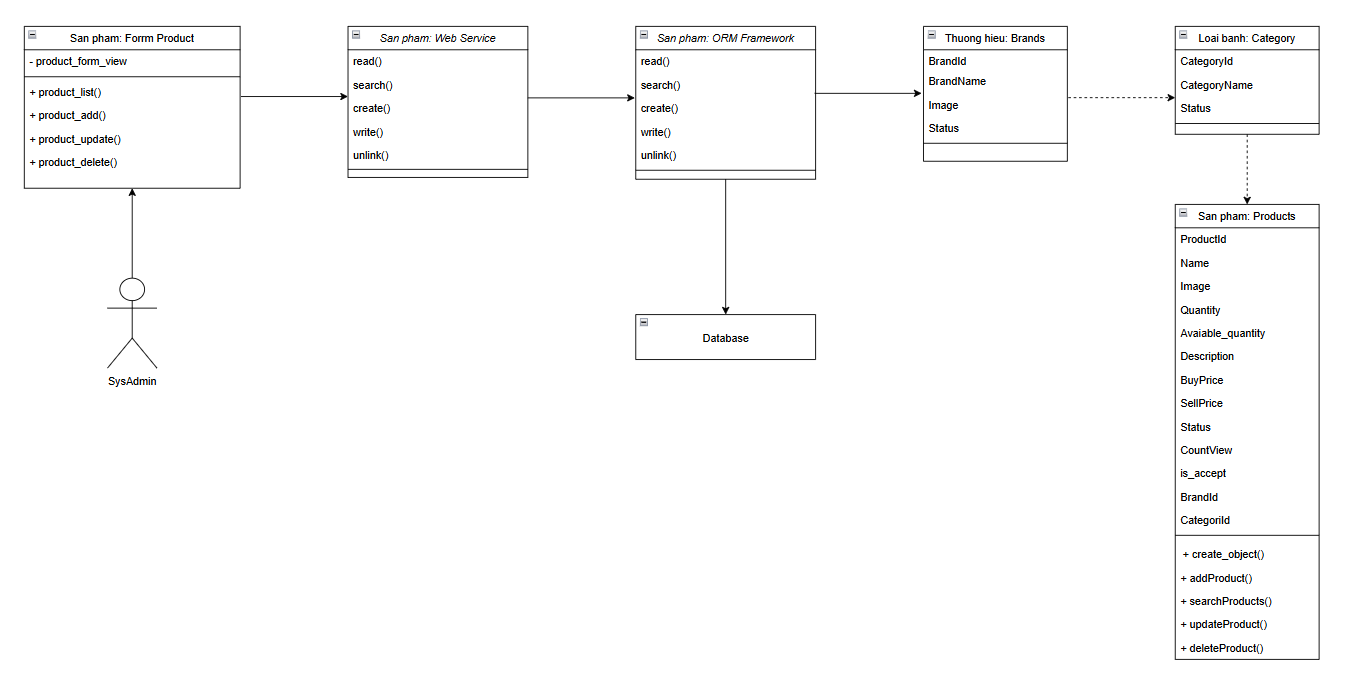


Figure 11. Class diagram of the Import process

### 4.5. Interface illustration

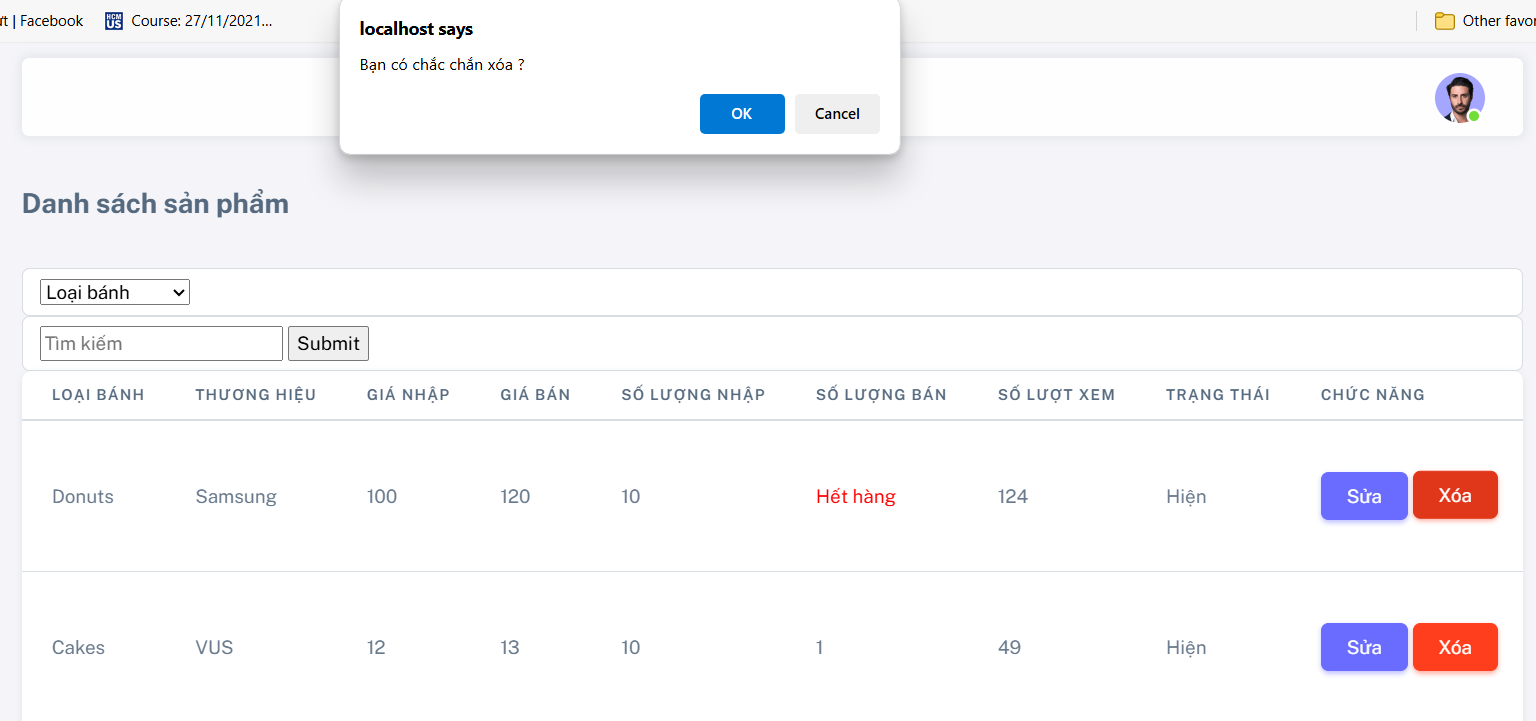
Delete product  


Figure 12. Product deletion interface Import goods

Add product



Figure 13. Add product interface Import goods

Product Updates

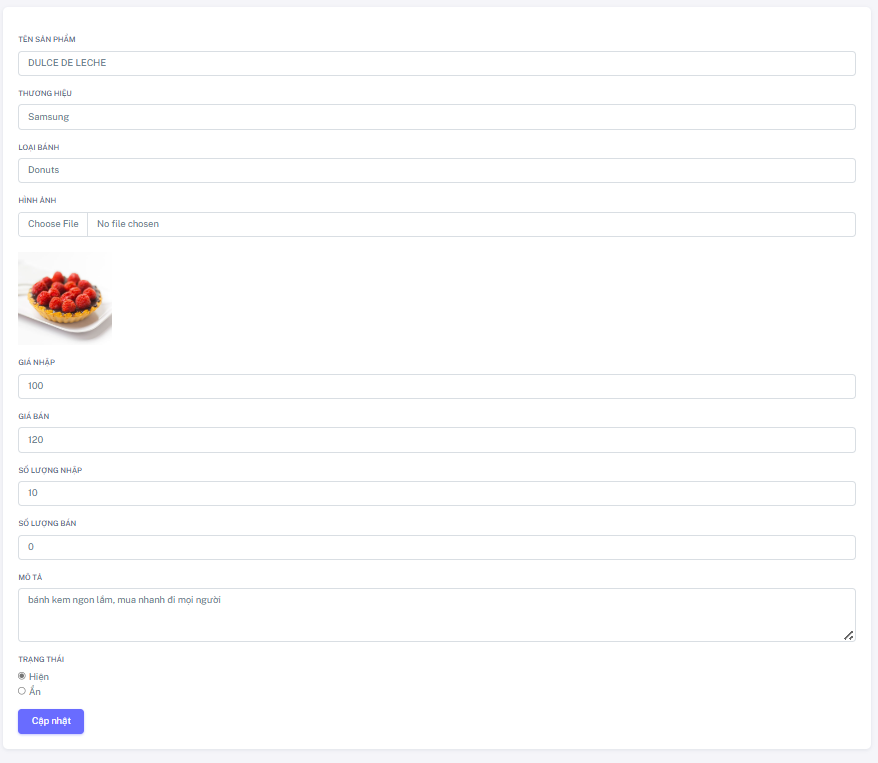


Figure 14. Product update interface Import goods

## 5. Purchase​

### 5.1. Focused UseCase

|  |  |  |
| --- | --- | --- |
|  | | |
| **Use case number** | **UC2** | |
| **Use case name** | **Shopping Cart** | |
| **Actor(s)** | **Shopper​** | |
| **Maturity** | **Focused** | |
| **Summary** | *Use case describes the user 's purchasing process in the system.* | |
| **Basic Course of events** | **Actor Action** | **System response** |
| 1. Use Case starts when Actor selects “Add to cart ” with default product **A1** |  |
|  | 2. Implementation system  **{Check quantity} E 1** |
|  | 3. The system displays a list of products added to the cart. |
| 4. Actor can make payment **A2 A3** |  |
|  | 5. The system displays the shopping cart payment form |
| 6. Actor enters payment information |  |
|  | 7. Implementation system  **{Check text box} E 1** |
|  | 8. The system updates order information into the database. |
|  | 9. The quantity of products in stock will decrease according to the order quantity. |
| **Alternative Paths** | **A1** |  |
| **Actor Action** | **System response** |
| 1.Actor clicks on the product image to add |  |
|  | 2. The system displays detailed product information including: Name, Price, Quantity, Cake type, Brand |
| 3. Actor customizes the number of products to add to cart |  |
|  | 4. Implementation system  **{Check quantity} E 1** |
| 5. Actor selects “Add to cart” |  |
|  | 6. The system displays a list of products added to the cart. |
| **A2** |  |
| **Actor Action** | **System response** |
| 1. Actor fills in the field the number of products to update |  |
| 2. Actor clicks “Update” |  |
|  | 3. Implementation system  **{Check quantity} E 1** |
|  | 4. The system displays the updated number of products in the shopping cart. |
| **A3** |  |
| **Actor Action** | **System response** |
| 1.Actor clicks “Delete” |  |
|  | 2. The system requires confirmation. |
| 3. Actor selects “OK” **A4** confirmation |  |
|  | 6. The system displays products removed from the cart. |
| **A4** |  |
| **Actor Action** | **System response** |
| 1. Actor can click “ Cancel” to operate |  |
|  | 2 .The system deletes the confirmation request . Return to step 2 of Basic Course of Events. |
| **Exception Path** | **E1 .** At **{Check quantity}** , check if the quantity is enough for the product in stock or not, if there is a corresponding error message | |
| **Triggers:** | When the actor wants to buy the item | |
| **Assumption:** | None | |
| **Preconditions:** | None | |
| **Post Conditions:** | None | |

### 5.2. Activity Diagram

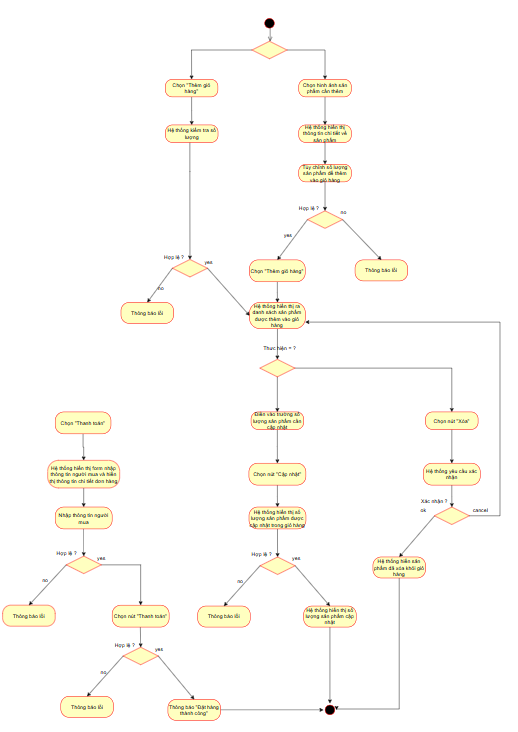


Figure 15. Activity diagram of Purchasing process

### 5.3. Sequence Diagram

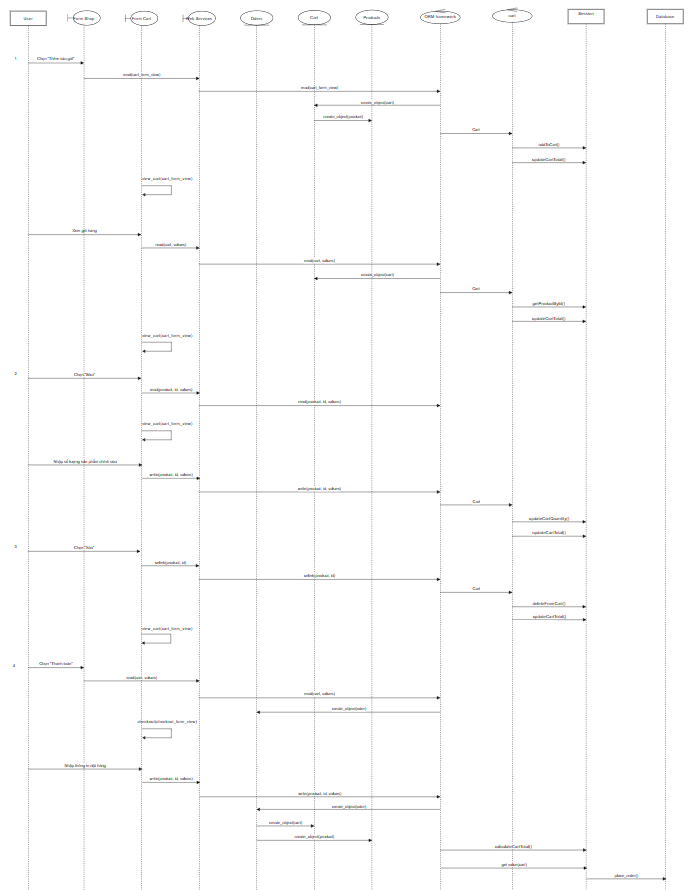


Figure 16. Purchase Process Sequence Diagram

### 5.4. Class Diagram

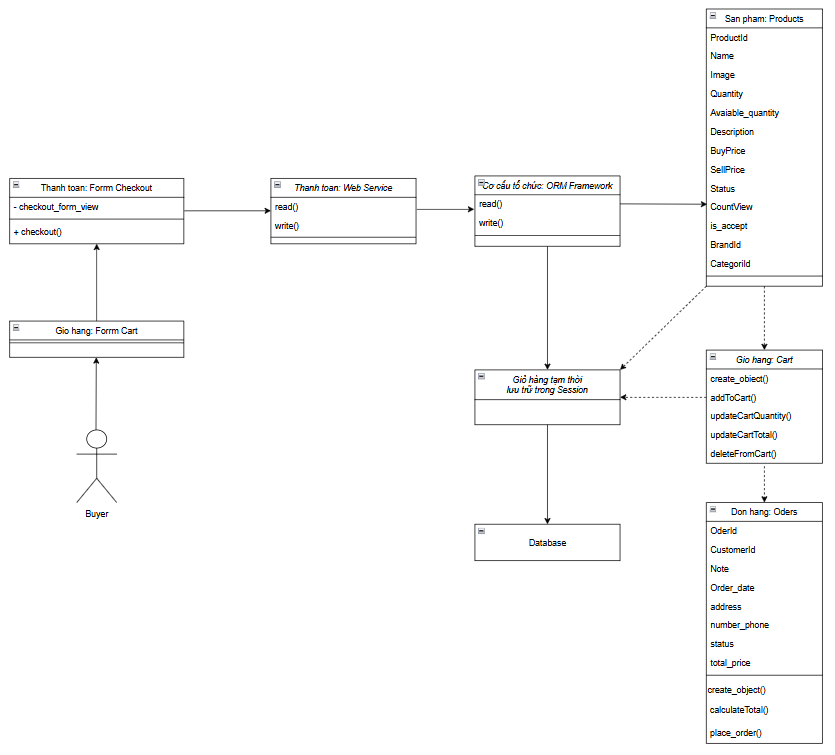


Figure 17. Purchase Process Class Diagram

### 5.5. Illustrative interface​

Add product to cart

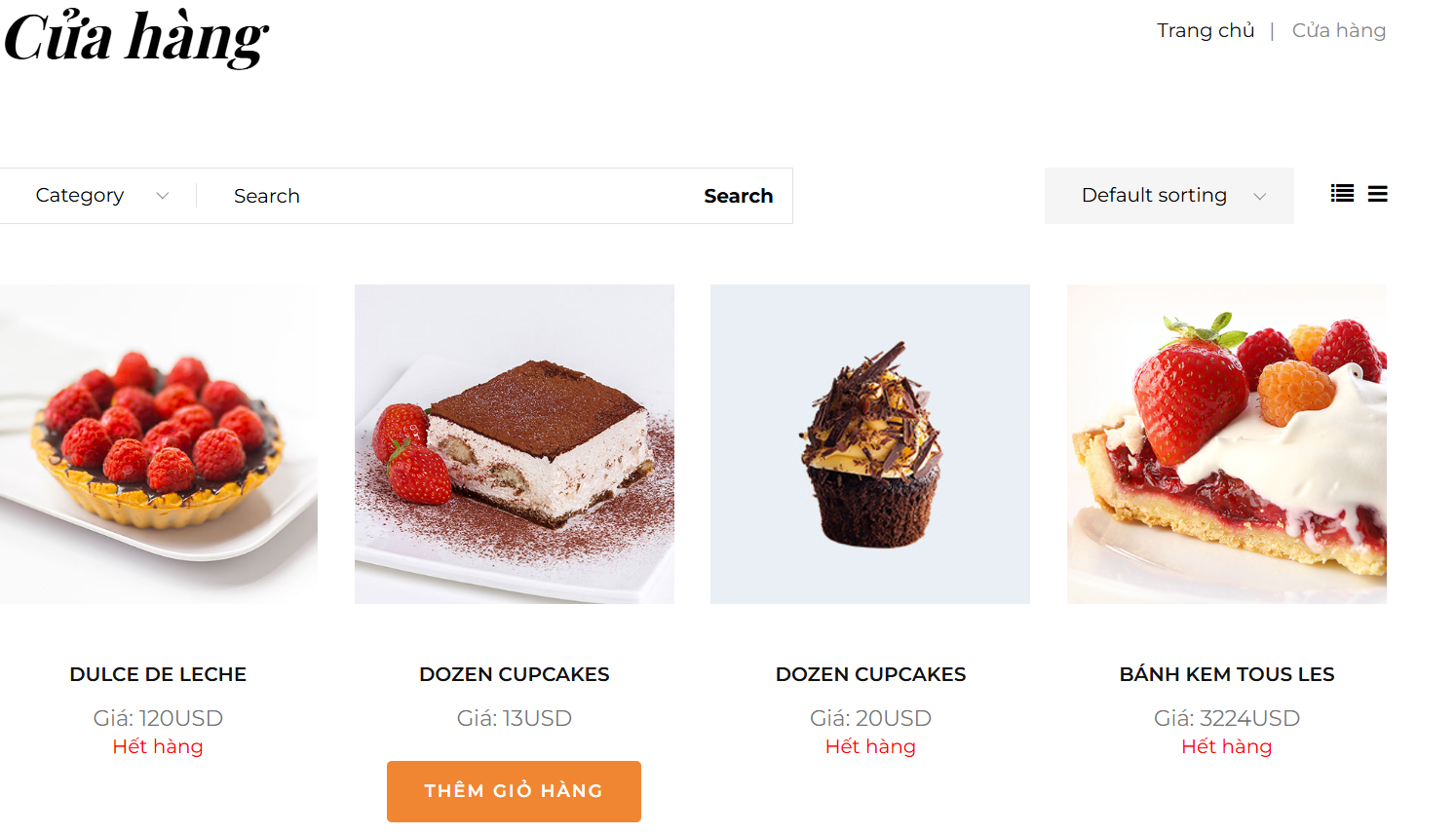


Figure 18. Default interface for adding a product to cart when purchasing

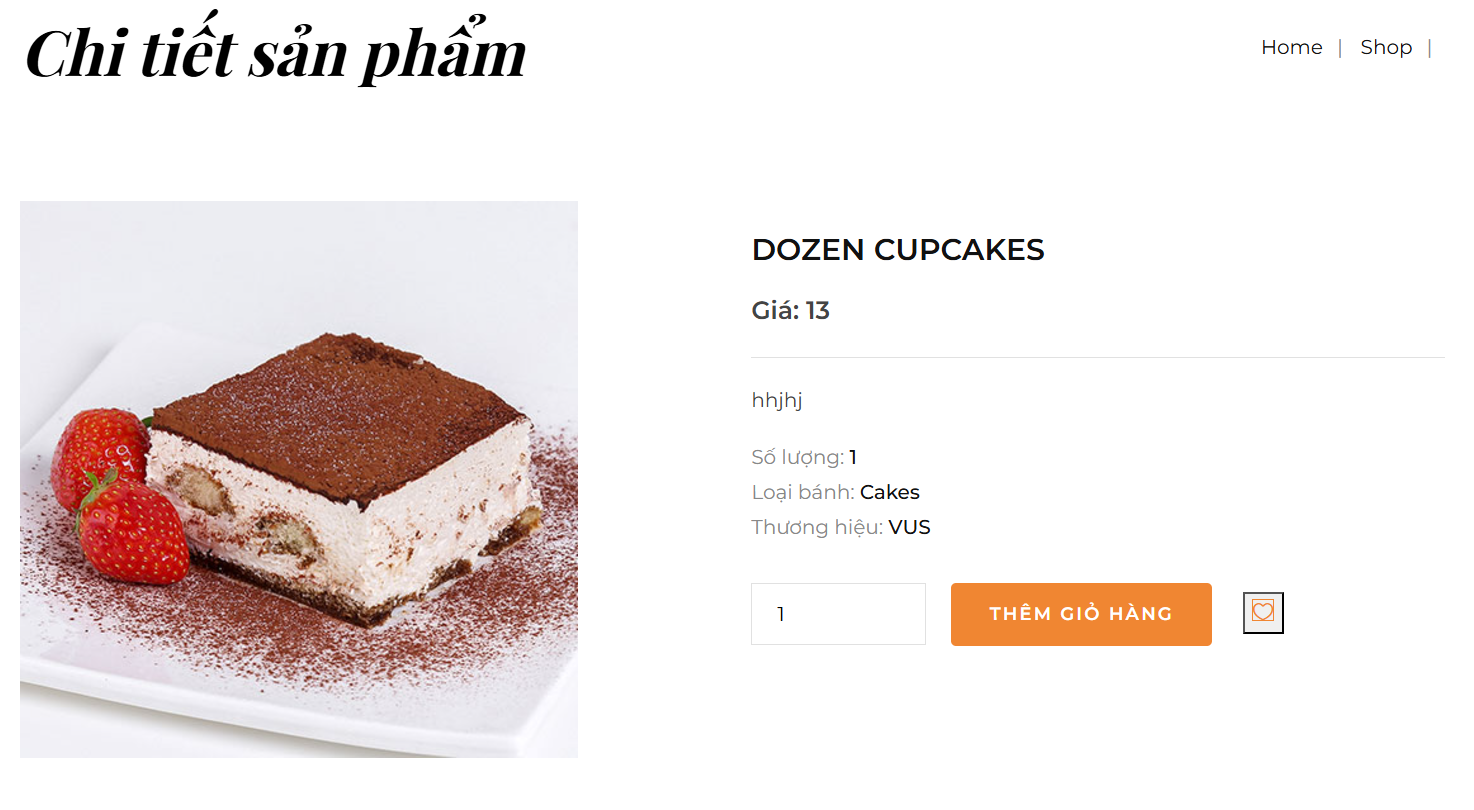


Figure 19. Interface for adding products with optional quantity when purchasing

Update quantity of products in cart

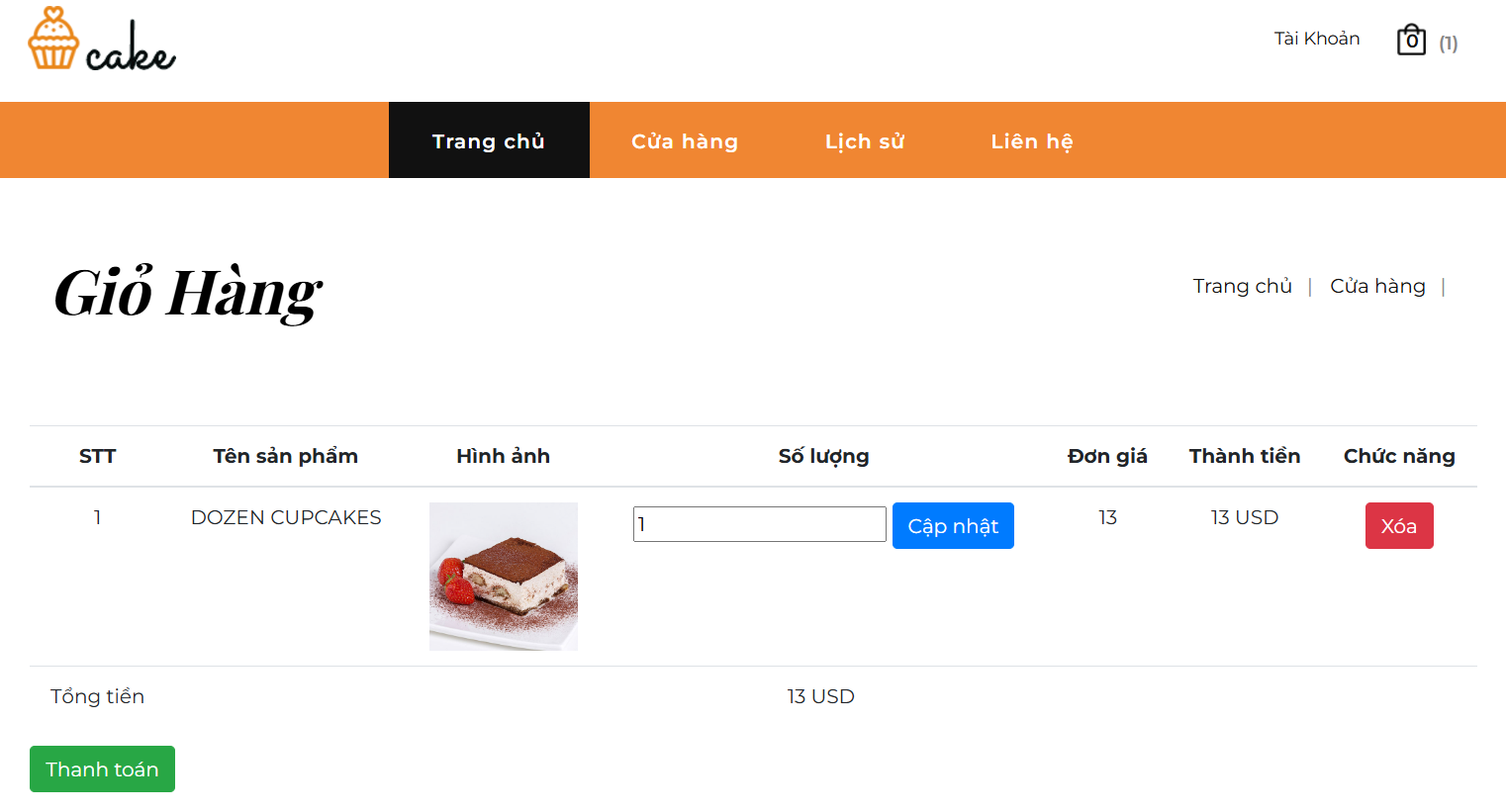


Figure 20. Interface for viewing and updating shopping cart

Remove product from cart

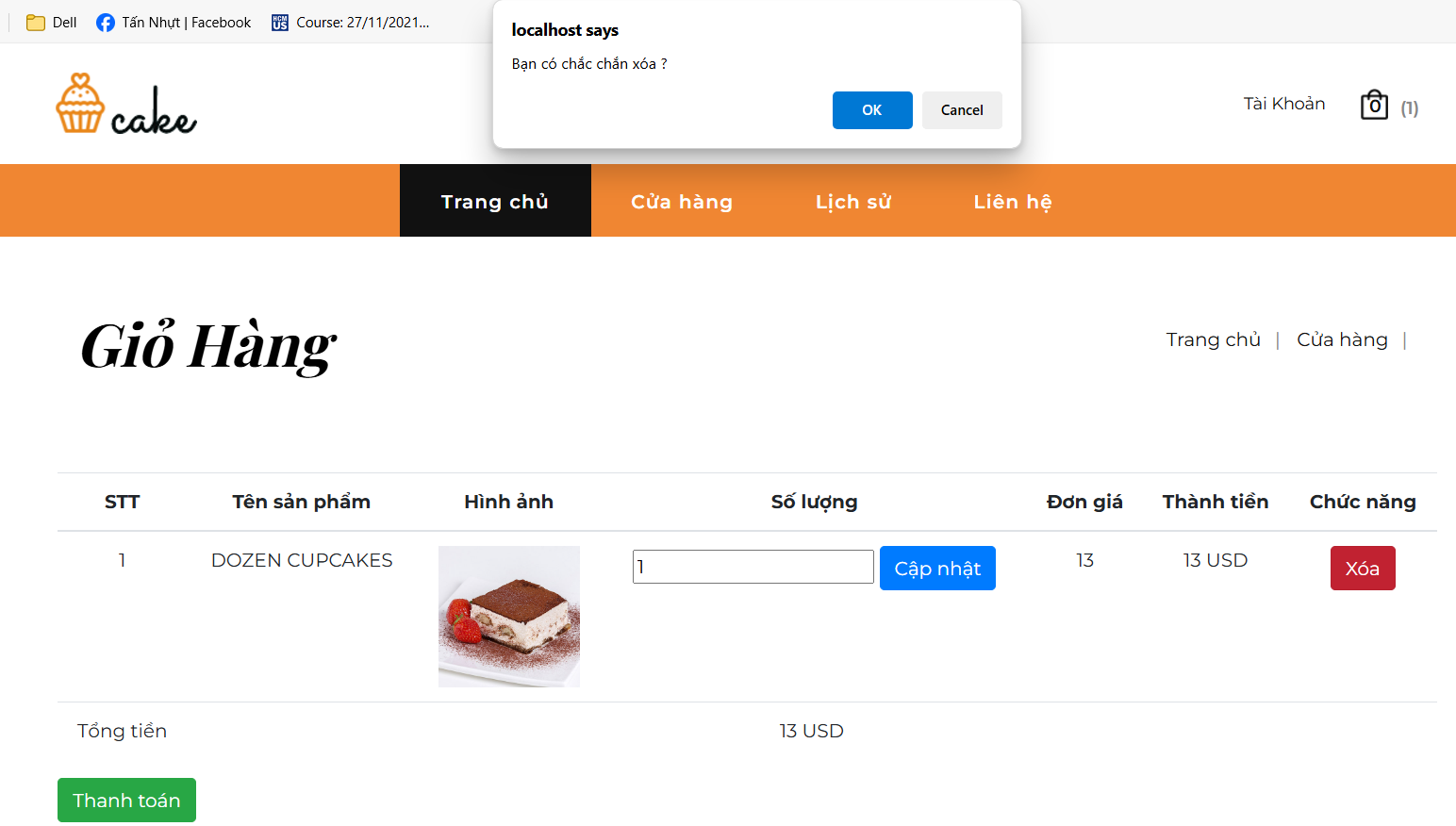


Figure 21. Interface to delete products from cart

Checkout cart

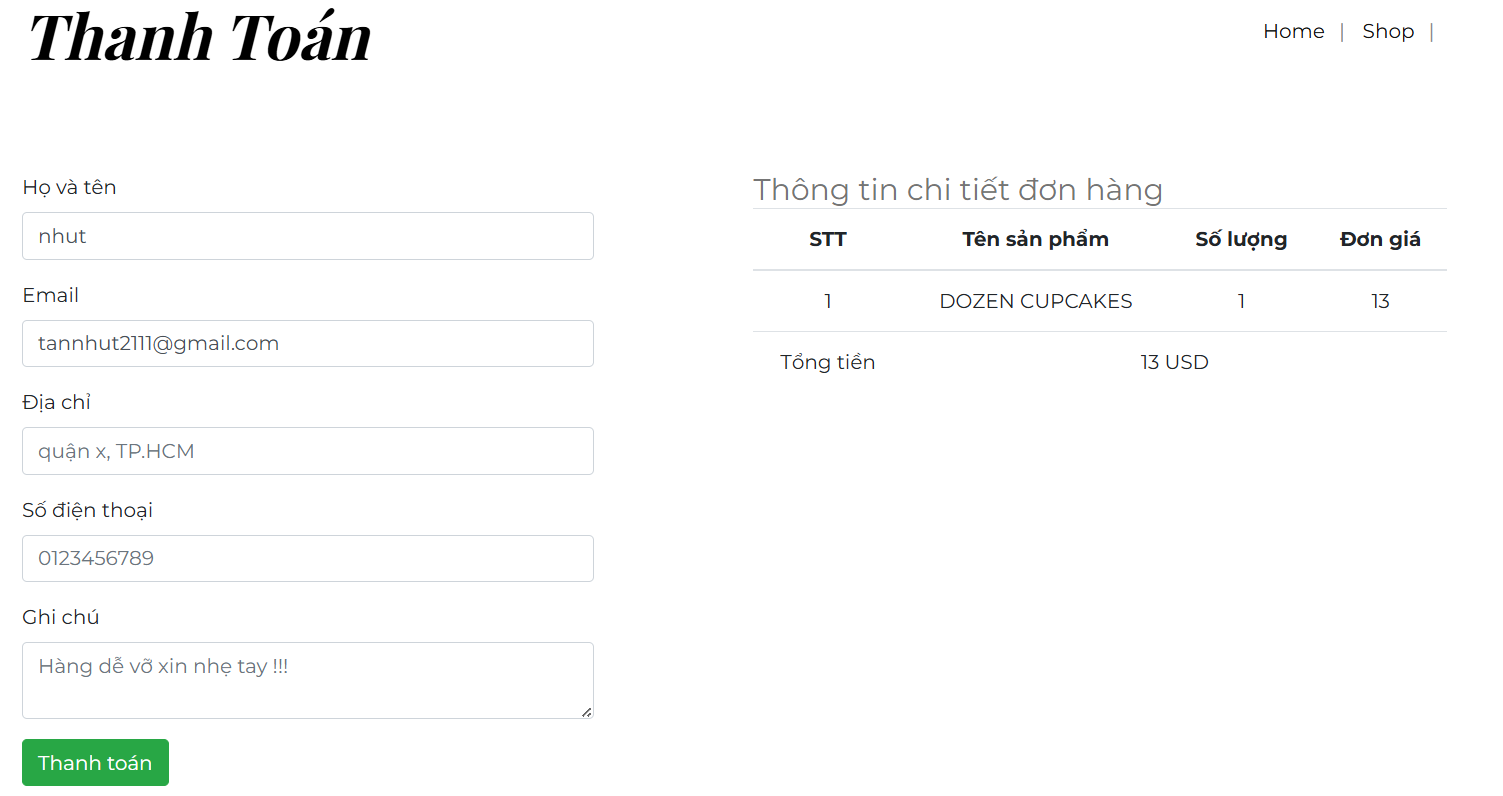


Figure 22. Shopping cart payment interface​

# CHAPTER 3: TEST PLAN

## 1. Introduction

### 1.1. Purpose

This test plan aims to:

Identify project information and software components to be tested.

List the test requirements (at high level).

Propose and describe a testing strategy.

Identify resources and estimate testing effort.

List the outputs of the testing project.

### 1.2. Background

The system developed is an online cake sales website, supporting users to choose and buy cakes with many different flavors and designs. The website provides services such as online ordering, online payment, shopping cart management, product search and filtering, as well as suggestions for featured cakes.

### 1.3. Scope

This test plan applies to all the requirements of the Cake-Sale-Website project defined in the Software Requirements Document. The testing scope includes all the main functions of the cake sales website, from product display, shopping cart management, ordering, to online payment and customer information management. In addition, the system must also be tested for the ability to search, filter products by category and support the suggestion of featured products. Factors related to user experience such as the interface , will also be tested. This plan also includes ensuring the website operates stably, testing normal use cases as well as exceptional cases, to ensure the system operates effectively and without errors during actual user use.

### 1.4. Project identification

The table below identifies the documents and resources used to develop the test plan, such as requirements documents, functional specification documents, and use-case statements.

The table below identifies the documents and availability used to develop the Test Plan:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Documentation (version / date)** | **Created or available** | **Received or reviewed** | **Author or Source** | **Note** |
| Request | Yes No | Yes No |  |  |
| Functional specification | Yes No | Yes No |  |  |
| Use-Case Report | Yes No | Yes No |  |  |
| Project plan | Yes No | Yes No |  |  |
| Design | Yes No | Yes No |  |  |
| Prototype | Yes No | Yes No |  |  |
| User manual | Yes No | Yes No |  |  |
| Business model or process | Yes No | Yes No |  |  |
| Data model or process | Yes No | Yes No |  |  |
| Business functions and rules | Yes No | Yes No |  |  |
| Project or business risk assessment | Yes No | Yes No |  |  |

### 1.5 References

|  |  |  |
| --- | --- | --- |
| **STT** | **Document** | **Describe** |
| 1 | ABC Project Software Requirements Document, Version 1.0.3.3, Monday, August 25, 2010 | The SRS document fully describes the external behavior of the application or subsystem being defined. It also describes non-functional requirements, design constraints, and other elements necessary to provide a complete and comprehensive description of the software requirements. |
| 2 | ABC Project Software Development Plan, Version 1.0.5, 09/05/2010 | The purpose of the Software Development Plan is to define the ABC project tasks (known as IMs or projects) to be developed. It focuses on the capabilities that stakeholders, especially the target users, are looking for and the reasons for this search. |

## 2. Features to test

### 2.1. Function (Customer)

2.1.1. Add products to cart

* Verify that the system allows customers to select products from the catalog and add them to the cart .
* Verify that each time a product is added, the system saves and updates the corresponding quantity and total in the shopping cart.

2.1.2. Remove products from cart

* Verify that the system provides functionality to remove unwanted products from the shopping cart.
* Verify that when you select delete, the product will be removed from the cart . .

2.1.3. Update products in cart

* Verify that the system allows changing the quantity of products in the shopping cart.
* Verify that when you press update, the system recalculates the product quantity and total price.

2.1.4. Shopping cart payment

* Verify that the system displays the selected products in the shopping cart and the total price.
* Verify that the system allows customers to checkout for the products selected in the shopping cart.

### 2.2. Function (Manager)

2.2.1. Add products to inventory

* Verify that the system allows the manager to add new products to inventory.
* Verify that product information including name, price, image, quantity, and other attributes are entered completely and stored accurately.

2.2.2. Remove product from inventory

* Verify that the system allows managers to remove out-of-stock products from inventory.
* Verify that the deleted product is no longer visible in the product catalog.

2.2.3. Update products in stock

* Verify that the system allows managers to update product information.
* Verify that information such as price, quantity, image, and description are updated correctly and display correctly after saving.

### 2.3. Availability

* Verify that the system supports displaying the pie image icon in the interface.
* Verify that the user interface is friendly, easy to use, and intuitive.

### 2.4. Design limitations

**2.4.1. Message length**

* Verify that the system does not allow messages or notifications to exceed the specified maximum length.

**2.4.2. Testing environment**

* Verify that the system works stably on popular browsers and on both mobile and desktop devices.

**2.4.3. Language**

* Verify that the system supports displaying Vietnamese correctly and without font errors.

**2.4.4. Multi-device**

* Verify that the system displays correctly and is compatible on both computers and phones with different screen sizes.

### 2.5. Interface

#### 2.5.1. User interface

* Verify that the main page interface provides prominent product information, including images, prices, and an “Add to Cart” button.
* Verify that the shopping cart interface displays the selected products, allows for updating quantities, deleting products, and checking out.
* Verify that the product details view displays complete product information including image, description, price, quantity, cake type, and brand name.

**2.5.2. Manager interface**

* Verify that the cake type management interface allows adding, modifying, and deleting cake types.

## 3. Features that do not need testing

### 3.1. Function (Customer)

**3.1 .1 . Contact the administrator**

* Verify that the system allows customers to submit contact information or inquiries about products or services.
* Verify that contact information/inquiries are sent to the correct manager for response.

**3.1 .2 . Search for products**

* Verify that customers can search for products by name via the search bar.
* Verify that search results are returned accurately based on the entered keyword.

**3.1 .3 . Product filter**

* Verify that customers can filter products by cake type.
* Verify that the product list displays correctly according to the filter criteria.

**3.1 .4 . View purchase history**

* Verify that the system displays purchase history including product details, price, delivery status, and related information.
* Verify that customers can only view their own purchase history.

**3.1 .5 . Track order status**

* Verify that customers can track their order status as the manager updates.
* Verify that order statuses are displayed accurately and clearly.

**3.1 .6 . View products in cart**

* Verify that customers can see the list of products added to the cart.
* Verify that the system displays complete product information in the shopping cart (name, quantity, price, total).

**3.1 .7 . View product details**

* Verify that the system displays detailed information for each product, including product name, description, image, price, stock quantity, cake type, and brand.
* Verify that product details are fully updated and accurate.

### 3.2. Function (Manager)

**3.2 . 1. Create a user account**

* Verify that the system provides user account creation functionality for the bakery service.
* Verify that the administrator can create multiple user accounts.
* Verify that the system supports the "Remember password" option for each user account.

**3.2 . 2. Delete user account**

* Verify that the system allows deletion of user accounts from the system.
* Verify that the deleted account will not be able to access or use the service.

**3. 2 . 3. Create an administrator account**

* Verify that the system provides the ability to create manager accounts to manage stores and products.

**3.2 .4 . Featured product suggestions**

* Verify that the system automatically recommends featured products based on customer views.
* Verify that the featured product list changes dynamically based on real-world data.

**3.2 .5 . Filter products in stock**

* Verify that the system allows filtering of products in inventory by criteria such as cake type, brand, or stock status.

**3.2.6. Add cake type**

* Verify that the administrator can add new cakes to the system.
* Verify that the cake type is correctly categorized to display on the website.

**3.2.7. Delete cake type**

* Verify that the system allows deletion of out-of-sale cakes.
* Verify that the deleted cake type is no longer visible in the catalog.

**3.2.8. Update cake type**

* Verify that the manager can update the cake type information, including the cake type name.
* Verify that the updated cake type information is displayed correctly.

**3.2.9. Add brand**

* Verify that the system allows adding new brands, including brand names and images.

**3.2.10. Remove brand**

* Verify that the brand removal support system is no longer in business or partnership.
* Verify that the removed brand will no longer appear in the catalog.

**3. 2.11. Brand Update**

* Verify that the manager can edit brand information, including brand name and image.

**3.2.12. View user information**

* Verify that managers can view customer account information, including purchase history and contact information.
* Verify that only administrators have access to this information.

**3.2.13. View user contact information**

* Verify that the system allows managers to view and manage customer contact information.
* Verify that contact information is displayed completely and accurately.

**3.2.14. Search for products in stock**

* Verify that managers can search for products in inventory based on product name, product code, or other criteria.

**3.2.15. View detailed order information**

* Verify that the manager can view full details of each order, including buyer information and products purchased.

**3.2.16. Update order status**

* Verify that the system allows order status updates (pending, processing, processed, shipped).
* Verify that the order status displayed matches the actual situation.

**3.2.17. Delete order**

* Verify that the system supports deleting invalid or canceled orders.
* Verify that deleted orders are no longer visible in the system.

## 4. Testing strategy

### 4.1. Types of testing

|  |  |
| --- | --- |
| **Test Objective** | Ensure that the target functionality under test works properly, including navigation, input, processing, and retrieval. |
| **Technique** | Execute each use case, use case flow, or function, using valid and invalid data, to verify the following:  - Expected results occur when valid data is used. - Appropriate error or warning messages are displayed when invalid data is used. |
| **Completion criteria** | - All tests have been performed.  - All identified bugs have been fixed. |
| **Special Considerations** | Identify or describe the items or issues (internal or external) that affect the implementation and execution of functional testing. |

### 4.2. Tools

Testing tools include manual testing and automated testing

|  |  |  |
| --- | --- | --- |
| **Tools** | **Supplier/Internal** | **Version** |
| Manual Testing | Internal |  |
| Auto Testing | Internal​ |  |

## 5. Resources

The table below shows the personnel required for the project.

|  |  |
| --- | --- |
| **Human resources** | **Describe specific responsibilities/jobs** |
| **Test Developer** | Identify, prioritize and execute test cases. |
|  | **Responsibility:**  - Create test plan  - Evaluate the effectiveness of testing efforts |
| **Tester** | Perform the tests. |
|  | **Responsibility:**  - Develop test case specifications  - Develop test cases |
|  | - Perform testing  - Record the results  - Monitor bug fixes  - Request to change documents |

## 6. Project milestones

The project consists of three development phases with full testing cycles from planning, design, development, implementation and evaluation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Milestone Task** | **Effort** | **Start Date** | **End Date** |
| **Construction 1** |  |  |  |
| Test Plan | 123.7 | 11/01/2024 | 11/01/2024 |
| Test Design | 300 .5 | 11/10/2024 | 11/15/2024 |
| Test Execution | 623 .3 | 11/20/2024 | 11/30/2024​ |
| Test Evaluation | 216 .1 | 01/12/2024​ | 02/12/2024​ |

## 7. Output products

### 7.1 Testing model

The test model defines all the test cases, references, test procedures and test scenarios associated with each test case.

### 7.2 Test log

Use Microsoft Word to record and report test results, ensuring detailed and clear tracking of test activities and results.

# CHAPTER 4: TESTING METHODS

## 1. Unit Testing

**Unit testing** is a software testing method in which the smallest units of source code (usually functions or methods) are tested independently to ensure that they function as expected. The goal of unit testing is to detect errors early in the basic components of the system while they are still simple, preventing errors from spreading into more complex parts of the program.

**Purpose of Unit Testing:**

* **Verify accuracy** : Ensure that the smallest units of the system operate as expected.
* **Early Error Detection** : Errors are detected early in the development process, reducing the cost of fixing them later.
* **Increased maintainability** : Unit testing makes it easier to maintain source code, as changes can be tested immediately.
* **Improve software quality** : Increase software reliability and quality by ensuring that code units function correctly.
* **Support for code refactoring** : With unit tests, programmers can refactor source code without fear of disrupting existing functionality.

### 1.1. TestPlan

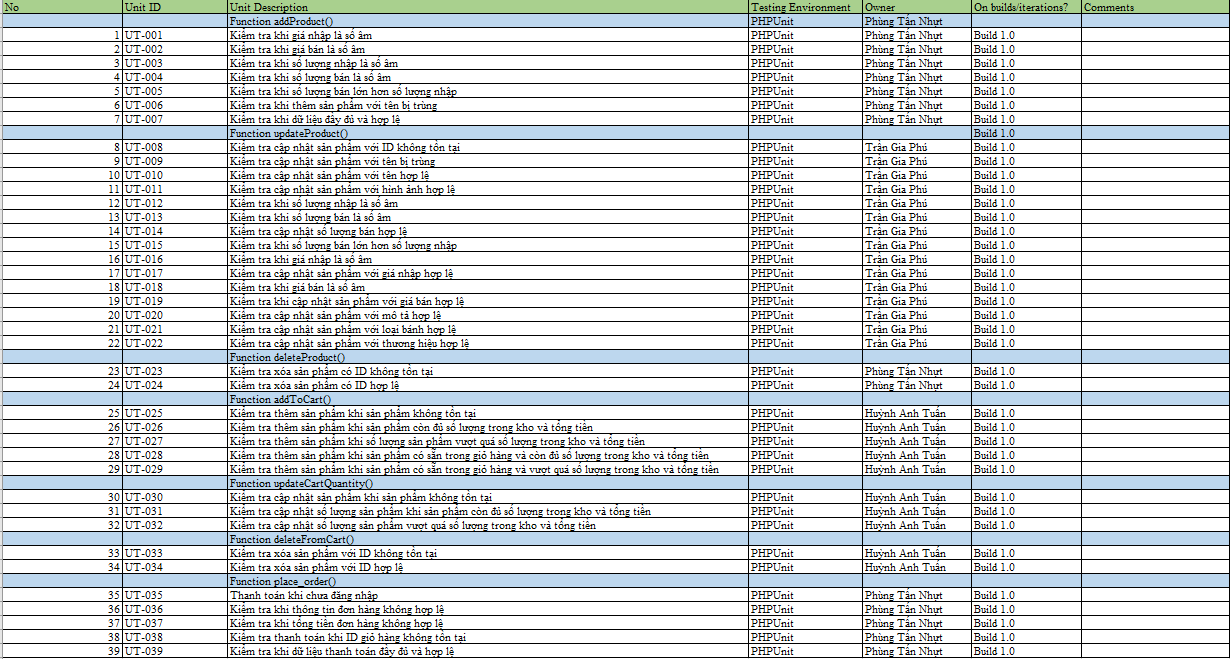


Figure 23. TestPlan Test Plan Table

**Function addProduct() :**

* Check if the cases when adding new products are valid.
  + Check if the input price is negative (Test when the input price is negative).
  + Test when the selling price is a negative value.
  + Check if the input quantity is negative (Test when the input quantity is negative).
  + Test when sales quantity is negative.
  + Test when sales quantity exceeds input quantity.
  + Check when product name is duplicated (Test when adding product with name duplicated with another product).
  + Check if input data is complete and valid.

**Function updateProduct ( ) :**

* Check if product update cases are valid or not
  + Update product with non-existing ID (Test when product ID does not exist in database).
  + Update product with duplicate name (Test when updating product with existing name).
  + Test when the input quantity is negative (Test when updating the input quantity is negative).
  + Test when sales quantity is negative (Test when sales quantity update is negative).
  + Update valid sales quantity (Test when sales quantity is valid).
  + Update product name , image, import quantity, sale quantity, import price, sale price and description (Test when product name , image, import quantity, sale quantity, import price, sale price and description are valid).

**Function deleteProduct() (Delete product):**

* Check the cases when deleting products:
  + Delete product with non-existing ID (Test when deleting product with ID not in database).
  + Delete product with valid ID (Test when deleting product with valid ID).

**Function addToCart() (Add product to cart):**

* Check the cases of adding products to cart:
  + Add product when product does not exist (Test when adding product when product does not exist in system).
  + Add product when stock is sufficient and total price (Test when adding product to cart with stock is sufficient and total price ).
  + Add product when quantity exceeds stock quantity and total price (Test when adding product when quantity in cart exceeds stock quantity and total price ).
  + Add product when product is available in cart with sufficient quantity and total price (Test when product is already in cart and still has sufficient quantity in stock and total price ).
  + Add products when available products exceed stock quantity and total price (Test when the product is already in the cart exceed the quantity in stock and still have enough quantity in stock and total amount ).

**Function updateCartQuantity() (Update the quantity of products in the cart):**

* Check the cases of updating the cart product quantity:
  + Update product quantity when product does not exist (Test when updating product quantity when product does not exist in shopping cart).
  + Update product quantity when there is enough stock and total price (Test when updating product quantity in cart when there is enough stock and total price ).
  + Update product quantity when quantity exceeds stock quantity (Test when updating product where quantity in cart exceeds available quantity and total price ).

**Function deleteFormCart() (Delete product from cart):**

* Check the cases of removing products from the cart:
  + Delete product with non-existent ID (Test when deleting product with ID not in cart).
  + Delete product with existing ID (Test when deleting product with ID in cart).

**Function place \_order () (Payment):**

* Checkout in cart:
  + Checkout while not logged in (Test when user is not logged in and tries to pay).
  + Checkout when cart ID does not exist (Test when cart ID does not exist in the system).
  + Check payment when order information is invalid (Test when order information is invalid when paying ).
  + Check payment when total order amount is invalid (Test when total order amount is invalid when paying ).
  + Pay when payment data is complete and valid (Test when payment data is complete and valid).

### 1.2. Import process

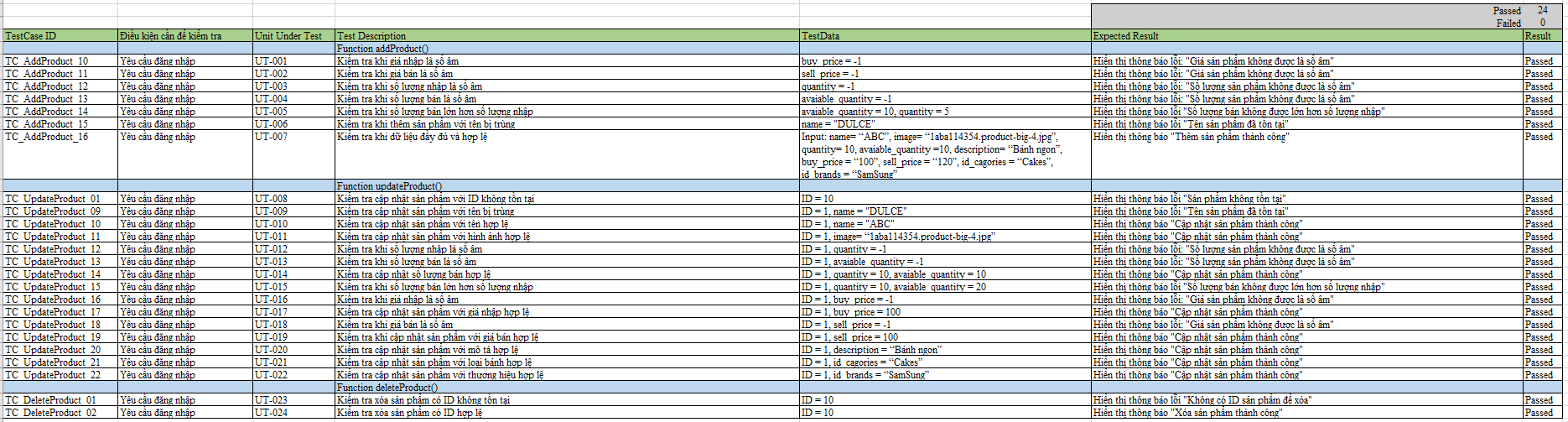


Figure 24. Unit Test Importing goods

**Passed 24 / Failed 0** :

* All tests in this table passed, none failed.

**Function addProduct() :**

**TC\_AddProduct\_10:** Test when the input price is negative (buy\_price = -1) – Test passed with error message: "Product price cannot be negative".

**TC\_AddProduct\_11:** Test when the selling price is negative (sell\_price = -1) – Test passed with error message: "Product price cannot be negative".

**TC\_AddProduct\_12:** Test when quantity entered is negative (quantity = -1) – Test passed with error message: "Quantity cannot be negative".

**TC\_AddProduct\_13:** Test when quantity sold is negative (avaiable\_quantity = -1) – Test passed with error message: "Quantity cannot be negative".

**TC\_AddProduct\_14:** Test when sales quantity is greater than import quantity (avaiable\_quantity = 10, quantity = 5) – Test passed with error message: "Sales quantity cannot be greater than import quantity".

**TC\_AddProduct\_15:** Test when adding product with duplicate name (name = "DULCE") – Test passed with error message: "Product name already exists".

**TC\_AddProduct\_16:** Check when data is complete and valid – Test passed with message: "Added product successfully".

**Function updateProduct() :**

**TC\_UpdateProduct\_01:** Test to update product with non-existent ID (ID = 10) – Test passed with error message: "Product does not exist".

**TC\_UpdateProduct\_09:** Test to update product with duplicate name (name = "DULCE") – Test passed with error message: "Product name already exists".

**TC\_UpdateProduct\_10:** Test to update product with valid name (name = "ABC") – Test passed with message: "Product update successful".

**TC\_UpdateProduct\_11:** Test product update with valid image (image = “1aba114354.product-big-4.jpg”) – Test passed with message: "Product update successful".

**TC\_UpdateProduct\_12:** Test when quantity entered is negative (quantity = -1) – Test passed with error message: "Quantity of product cannot be negative".

**TC\_UpdateProduct\_13:** Test when quantity sold is negative (avaiable\_quantity = -1) – Test passed with error message: "Product quantity cannot be negative".

**TC\_UpdateProduct\_14:** Check for valid sales quantity update (quantity = 10, available\_quantity = 10) – Test passed with message: "Product update successful".

**TC\_UpdateProduct\_15:** Test when sales quantity is greater than input quantity (quantity = 10, available\_quantity = 20) – Test passed with error message: "Sale quantity cannot be greater than input quantity".

**TC\_UpdateProduct\_16:** Test when the input price is negative (buy\_price = -1) – Test passed with error message: "Product price cannot be negative".

**TC\_UpdateProduct\_17:** Test product update with valid input price (buy\_price = 100) – Test passed with message: "Product update successful".

**TC\_UpdateProduct\_18:** Test when the selling price is negative (sell\_price = -1) – Test passed with error message: "Product price cannot be negative".

**TC\_UpdateProduct\_19:** Test when updating product with valid selling price (sell\_price = 100) – Test passed with message: "Product updated successfully".

**TC\_UpdateProduct\_20:** Test product update with valid description (description = “Banh ngon”) – Test passed with message: "Product update successful".

**TC\_UpdateProduct\_21:** Test to update product with valid cake type (id\_cagories = “Cakes”) – Test passed with message: "Product updated successfully".

**TC\_UpdateProduct\_22:** Test for product update with valid brand (id\_brands = “SamSung”) – Test passed with message: "Product update successful".

**Function deleteProduct() :**

**TC\_DeleteProduct\_01:** Test to delete a product with a non-existent ID (ID = 10) – Test passed with error message: "No product ID to delete".

**TC\_DeleteProduct\_02:** Test to delete a product with a valid ID (ID = 10) – Test passed with message: "Product deleted successfully".

### 1.3. **Purchasing process**

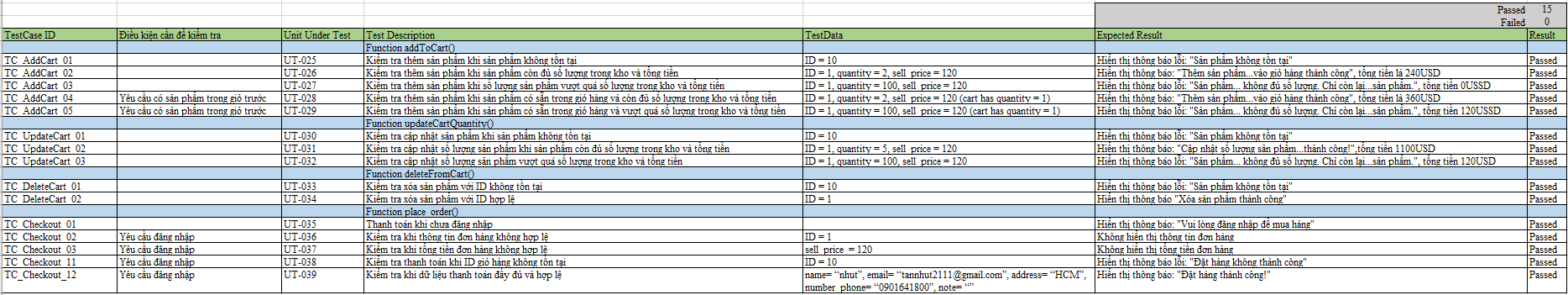
****

Figure 25. Purchase Unit Test

**Passed 15 / Failed 0** :

* All tests in this table passed, none failed.

**Function addToCart() (Add product to cart):**

* **TC\_AddCart\_01** : Test when adding a non-existent product to the cart. The test passed with the error message : "Product does not exist".
* **TC\_AddCart\_02** : Test when adding product to cart and product has enough quantity in stock. Test passed with message: "Added product to cart successfully".
* **TC\_AddCart\_03** : Test when the quantity of products to be added to the cart exceeds the quantity in stock. The test passed with the error message : "Product quantity is insufficient. Only ... products left".
* **TC\_AddCart\_04** : Test when the product is already in the cart and still has enough quantity in stock. The test passed with the message: "Product added to cart successfully".
* **TC\_AddCart\_05** : Test when the product is already in the cart but the quantity to be added exceeds the quantity in stock. The test passed with the error message : "Product quantity is insufficient. Only ... products left".
* **Function updateCartQuantity() (Update the quantity of products in the cart):**
* **TC\_UpdateCart\_01 : Test when updating quantity of product in cart with non-existent ID. Test passed with** error message : "Product does not exist".
* **TC\_UpdateCart\_02** : Check when updating product quantity and product has enough quantity in stock. The test passed with the message: "Product quantity updated successfully!".
* **TC\_UpdateCart\_03** : Test when the quantity of products to be updated exceeds the quantity in stock . The test passed with the error message : "Product quantity is insufficient. Only ... products left".

**Function deleteFormCart() (Delete product from cart):**

* **TC\_DeleteCart\_01** : Test when deleting a product with a non-existent ID. The test passed with the error message : "Product does not exist".
* **TC\_DeleteCart\_02** : Test when deleting a product with a valid ID. The test passed with the message: "Product deleted successfully".

**Function place \_order () (Payment):**

* **TC\_Checkout\_01** : Test when not logged in and try to checkout. Test passed with message: "Please login to purchase".
* **TC\_Checkout\_09** : Checkout when cart ID does not exist. Test passed with error message : "Order failed".
* **TC\_Checkout\_10** : Checkout when the required information (such as name, email, address, phone number) is complete and valid. The test passed with the message: "Order successful!".
* **TC\_Checkout\_11** : Check when order information is invalid. Test passed and order information is not displayed .
* **TC\_Checkout\_12** : Check when total amount is invalid. Test passed and total amount is not displayed .

## 2. Integration Testing

**Integration** Testing is a phase in the software testing process that focuses on testing the interactions between modules or components in a system. The goal is to ensure that the integrated modules function as expected when they work together.

### 2.1. Test Design

**Test design** is the process of creating scenarios, test cases, and test methods to test a software or system to ensure that the system operates correctly and meets technical and functional requirements. The goal of test design is to detect bugs and ensure software quality.

**Purpose of Test Design:**

* **Software quality assurance** : Test design helps identify the parts of the software that need to be tested to ensure they function as required.
* **Error and Problem Detection** : Test design aims to detect errors or problems in the software before it is released.
* **Risk Reduction** : Adequate testing helps to reduce quality risks when the software is deployed.
* **Ensuring user requirements are met** : Test design helps verify that the software works as per the requirements, thereby improving end user satisfaction.
* **Improve software performance** : Performance testing helps to check the system's ability to handle load and handling under extreme conditions.

**Import process**

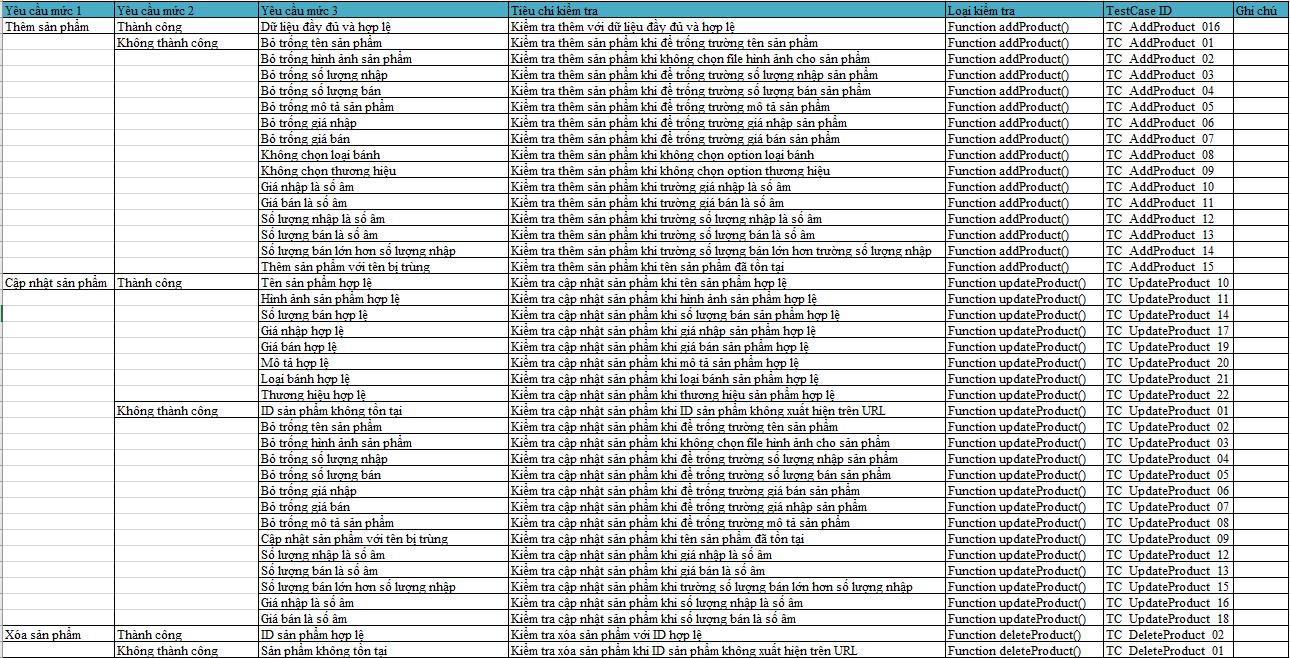


Figure 26. Test Design Import

**Add product**

**Success** : Test the add product functionality with valid data, including name, image, quantity, price, description, cake type, and brand.

**Failed** : Check for invalid cases like:

* Leave important fields blank (product name, image, quantity, price, description, cake type, brand).
* Negative values for quantity or price.
* Sales quantity is greater than import quantity.
* Duplicate product name.

**Product Updates**

**Success** : Test the product update functionality with valid fields like name, image, quantity, import price, sale price, description, cake type, and brand.

**Failed** : Check for invalid cases when updating:

* Product ID does not exist.
* Important fields are left blank (name, image, quantity, price, description).
* Negative values for quantity and price.
* Sales quantity is greater than import quantity.
* Duplicate product name.

**Delete product**

**Success** : Check the product deletion with valid ID.

**Failed** : Test the case of deleting a product when the product ID does not exist.

**Purchasing process**



Figure 27. Test Design Purchase

**Add product to cart**

**Success** :

* Check for more products while supplies last
* Check more products available in cart when quantity is sufficient

**Failed :**

* Check for more products when product does not exist
* Check for additional products exceeding the quantity in stock
* Check more products available in cart exceeds the number of products in stock

**Update product quantity in stock**

**Success :** Check the updated quantity of products in the cart

**Unsuccessful:**

* Check update product in cart when product does not exist
* Check when updating quantity exceeds quantity in stock

**Remove product from cart**

**Success :** Check to delete products in cart that do not have an ID

**Failed** : Check to delete product in cart delete product with valid ID

**Checkout cart**

**Success** : Checkout cart with complete and valid data

**Failed :**

* Check must login before payment
* Leave important fields blank (full name , email, address , phone number ).
* Check payment when first and last name field is number
* Check payment when email field is not in correct format
* Check payment when phone number is less than and greater than 10 digits
* Checkout when cart ID does not exist
* Check payment when order information is invalid
* Check payment when order total is invalid

### 2.2. White box testing​

**White-box Testing Checklist** is a list of items to be checked to ensure the quality of the source code and internal logic of an application. This checklist is used by testers or developers to thoroughly check the technical aspects during white-box testing.

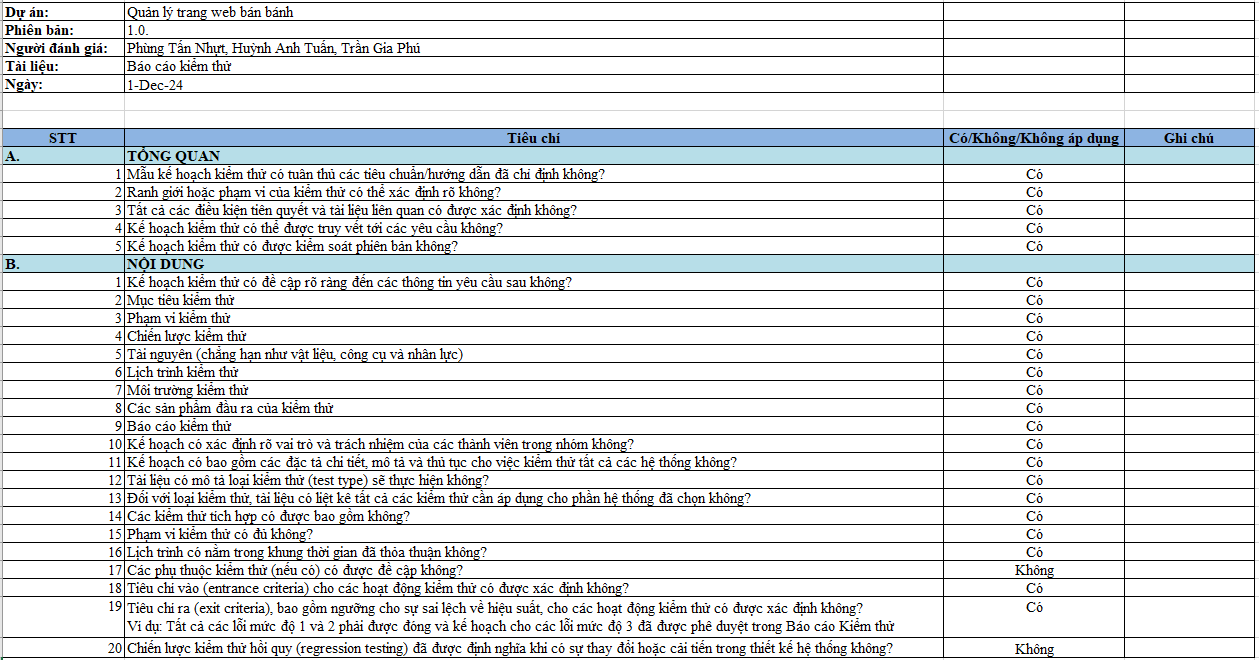


Figure 28. White Box CheckList

**White box testing** (White Box Testing) is a software testing method that focuses on testing the internal logic and source code structure of an application to ensure that the functions are implemented as expected. This method is used to:

-Verify interface event handling: Ensure that events triggered from the interface (such as button presses or input) are correctly mapped to internal handler functions, and that processing steps (such as saving data, updating state) are performed logically.

-Test interface state transition flow: Ensure that interface state transition logic, based on user actions, is properly implemented, covering all possible states without creating error states.

**Techniques used in the project:**

- **Control Flow Testing** is a white box testing method that focuses on the logical execution flow of the application. It ensures that:

* All possible branches or paths in the program are tested.
* The conditions and decision points are verified.

- **Branch coverage testing:** Test decision branches like "Add successful" or "Add failed."

- **Path coverage testing:** Tests the various logical flows from start to finish, including error messages, validations, and database saves.

#### 2.2.1. Import process

**Add product**

**Control flow graph diagram**

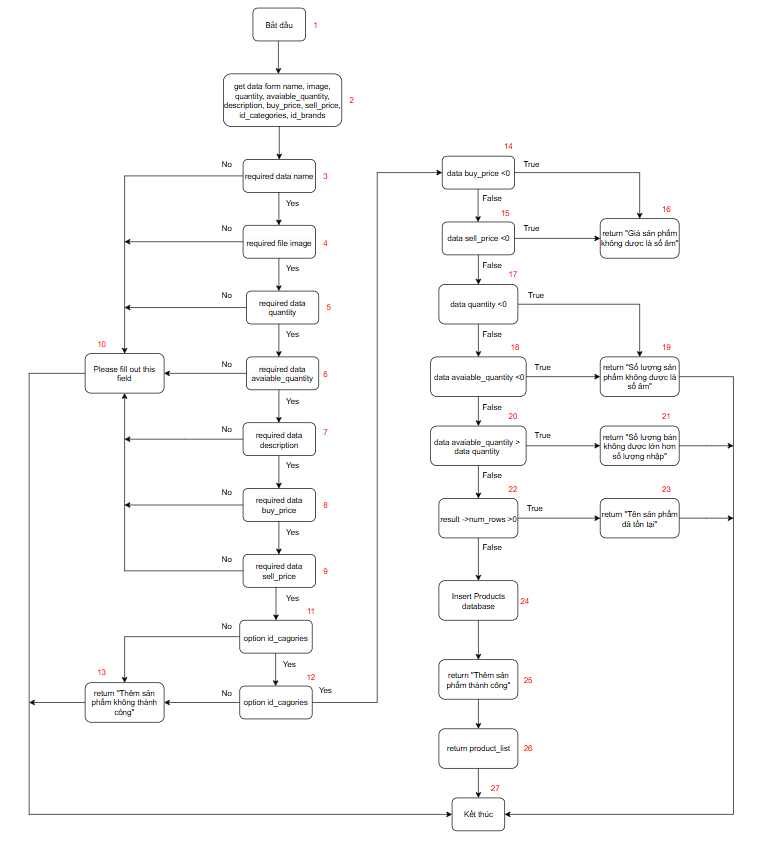
****

Figure 29. Control flow diagram Add product Import goods

**Complexity​**

E = number of sides = 4 1

N = number of buttons = 2 7

P = number of connected components = 1

M = E – N + 2P = 4 1 – 2 7 + 2\*1 = 16

**Independent road**

Independent line 1: 1-2 -3 -10-2 7

Independent line 2: 1-2-3 -4 -10-2 7

Independent line 3: 1-2-3-4 -5 -10-2 7

Independent line 4: 1-2-3-4-5 -6 -10-2 7

Independent line 5: 1-2-3-4-5-6 -7 -10-2 7

Independent line 6: 1-2-3-4-5-6-7 -8 -10-2 7

Independent line 7: 1-2-3-4-5-6-7-8 -9 -10-2 7

Independent line 8: 1-2-3-4-5-6-7-8-9-11-13-2 7

Independent line 9: 1-2-3-4-5-6-7-8-9 -11 -12-13-2 7

Independent line 10: 1-2-3-4-5-6-7-8-9- 11-12- 14-16-2 7

Independent line 11: 1-2-3-4-5-6-7-8-9-1 1-12-1 4-15-16-2 7

Independent line 12: 1-2-3-4-5-6-7-8-9- 11-12- 14-15-17-19-2 7

Independent line 13: 1-2-3-4-5-6-7-8-9- 11-12- 14-15-17-18-19-2 7

Independent line 14: 1-2-3-4-5-6-7-8-9- 11-12- 14-15-17-18-20-21-2 7

Independent line 15: 1-2-3-4-5-6-7-8-9- 11-12- 14-15-17-18-20-22-23-2 7

Independent line 16: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-18-20-22-2 5 -2 6 -2 7

**Test Suite**

**TC\_AddProduct\_01: 1-2 -3 -10-27**

Input: name = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_02: 1-2-3 -4 -10-27**

Input: image = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_03: 1-2-3-4 -5 -10-27**

Input: quantity = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_04: 1-2-3-4-5 -6 -10-27**

Input : available\_quantity = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_05: 1-2-3-4-5-6 -7 -10-27**

Input: description = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_06: 1-2-3-4-5-6-7 -8 -10-27**

Input: buy\_price = “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_07: 1-2-3-4-5-6-7-8 -9 -10-27**

Input: sell \_price= “ ”

Expected result: Please fill out this field

**TC\_AddProduct\_08** : **1-2-3-4-5-6-7-8-9-11-13-27**

Input: id\_cagories = “Cake Types”

Expected result: "Add product failed"

**TC\_AddProduct\_09: 1-2-3-4-5-6-7-8-9- 11- 12-13-27**

Input: id\_brands = “Brand Type”

Expected result: "Add product failed"

**TC\_AddProduct\_10:** **1-2-3-4-5-6-7-8-9 -11-12 -14-16-27**

Input: buy\_price = -1

Expected result: "Product price cannot be negative"

**TC\_AddProduct\_11: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-16-27**

Input: sell\_price = - 1

Expected result: "Product price cannot be negative"

**TC\_AddProduct\_12: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-19-27**

Input: quantity = -1

Expected result: "The number of products cannot be negative"

**TC\_AddProduct\_13: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-18-19-27**

Input: available\_quantity =-1

Expected result: "The number of products cannot be negative"

**TC\_AddProduct\_14: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-18-20-21-27**

Input: available\_quantity = 10, quantity = 5

Expected result: "The quantity sold must not be greater than the quantity imported"

**TC\_AddProduct\_15: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-18-20-22-23-27**

Input: name = “ DULCE ”

Expected result: “Product name already exists”

**TC\_AddProduct\_16: 1-2-3-4-5-6-7-8-9 -11-12 -14-15-17-18-20-22-25-26-27**

Input: name = “ABC”, image = “1aba114354.product-big-4.jpg”, quantity = 10, available\_quantity = 10, description = “ Delicious cakes”, buy\_price = “100”, sell\_price = “120”, id\_cagories = “Cakes”, id\_brands = “SamSung”

Expected result: “Added product successfully”

**Product Updates**

**Control flow graph diagram**

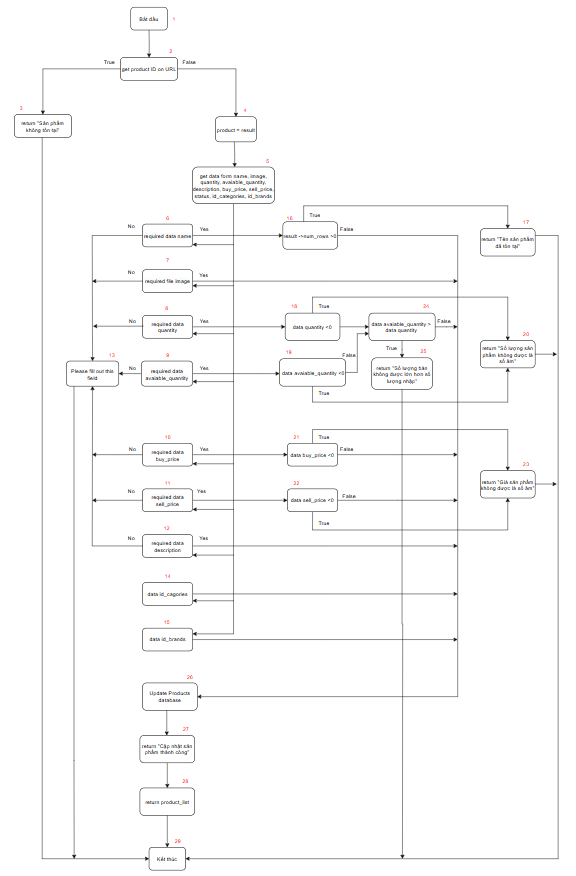
****

Figure 30. Product Update Control Flow Diagram

**Complexity​**

E = number of sides = 49

N = number of buttons = 29

P = number of connected components = 1

M = E – N + 2P = 49 – 29 + 2\*1 = 22

**Independent road**

Independent line 1: 1-2-3-29

Independent line 2: 1-2-4-5-6-13-29

Independent line 3: 1-2-4-5-7-13-29

Independent line 4: 1-2-4-5-8-13-29

Independent Line 5: 1-2-4-5-9-13-29

Independent line 6: 1-2-4-5-10-13-29

Independent line 7: 1-2-4-5-11-13-29

Independent line 8: 1-2-4-5-12-13-29

Independent line 9: 1-2-4-5-6-16-17-29

Independent Line 10: 1-2-4-5-6-16-27-28-29

Independent line 11: 1-2-4-5-7-27-28-29

Independent line 12: 1-2-3-4-5-8-18-20-29

Independent Line 13: 1-2-3-4-5-9-19-20-29

Independent Line 14: 1-2-3-4-5-9-19-24-27-28-29

Independent line 15: 1-2-3-4-5-9-19-24-25-29

Independent line 16: 1-2-3-4-5-10-21-23-29

Independent Line 17: 1-2-3-4-5-10-21-27-28-29

Independent Line 18: 1-2-3-4-5-11-22-23-29

Independent Line 19: 1-2-3-4-5-11-22-27-28-29

Independent line 20: 1-2-3-4-5-12-27-28-29

Independent Line 21: 1-2-3-4-5-14-27-28-29

Independent line 22: 1-2-3-4-5-15-27-28-29

**Test Suite**

**TC\_UpdateProduct\_01: 1-2-3-29**

Input: id = 10

Expected result: “Product does not exist”

**TC\_UpdateProduct\_02: 1-2-4-5-6-13-29**

Input: id = 1, name = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_03: 1-2-4-5-7-13-29**

Input: id = 1, image = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_04: 1-2-4-5-8-13-29**

Input: id = 1, quantity = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_05: 1-2-4-5-9-13-29**

Input: id = 1, available \_quantity = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_06: 1-2-4-5-10-13-29**

Input: id = 1, buy \_price = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_07: 1-2-4-5-11-13-29**

Input: id = 1, sell \_price = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_08: 1-2-4-5-12-13-29**

Input: id = 1, description = “ ”

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_09: 1-2-4-5-6-16-17-29**

Input: id = 1, name = “ DULCE ”

Expected result: “Product name already exists”

**TC\_UpdateProduct\_10: 1-2-4-5-6-16-27-28-29**

Input: id = 1, name = “ ABC ”

Expected result: “Product update successful”

**TC\_UpdateProduct\_11: 1-2-4-5-7-27-28-29**

Input : id = 1, image = “1aba114354.product-big-4.jpg”

Expected result: “Product update successful”

**TC\_UpdateProduct\_12: 1-2-3-4-5-8-18-20-29**

Input: id = 1, quantity = -1

Expected result: "The number of products cannot be negative"

**TC\_UpdateProduct\_13: 1-2-3-4-5-9-19-20-29**

Input: id = 1, available\_quantity =-1

Expected result: "The number of products cannot be negative"

**TC\_UpdateProduct\_14: 1-2-3-4-5-9-19-24-27-28-29**

Input: id = 1, quantity = 10, available\_quantity = 10

Expected result: “Product update successful”

**TC\_UpdateProduct\_15: 1-2-3-4-5-9-19-24-25-29**

Input: id = 1, quantity = 10, available\_quantity = 20

Expected result: “The quantity sold must not be greater than the quantity imported”

**TC\_UpdateProduct\_16: 1-2-3-4-5-10-21-23-29**

Input: id = 1, buy\_price = -1

Expected result: "Product price cannot be negative"

**TC\_UpdateProduct\_17: 1-2-3-4-5-10-21-27-28-29**

Input: id = 1, buy\_price = 100

Expected result: “Product update successful”

**TC\_UpdateProduct\_18: 1-2-3-4-5-11-22-23-29**

Input: id = 1, sell\_price = - 1

Expected result: "Product price cannot be negative"

**TC\_UpdateProduct\_19: 1-2-3-4-5-11-22-27-28-29**

Input: id = 1, sell\_price = 100

Expected result: “Product update successful”

**TC\_UpdateProduct\_20: 1-2-3-4-5-12-27-28-29**

Input: id = 1, description = “Delicious cake”

Expected result: “Product update successful”

**TC\_UpdateProduct\_21: 1-2-3-4-5-14-27-28-29**

Input: id = 1, id\_cagories = “Cakes”

Expected result: “Product update successful”

**TC\_UpdateProduct\_22: 1-2-3-4-5-15-27-28-29**

Input: id = 1¸id\_brands = “SamSung”

Expected result: “Product update successful”

**Delete product**

**Control flow graph diagram**

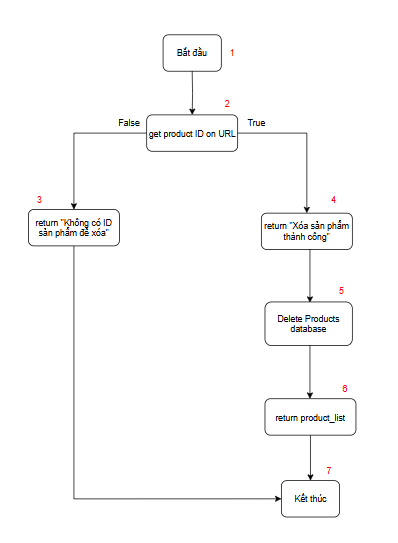


Figure 31. Control flow diagram Delete product Import goods

**Complexity​**

E = number of sides = 7

N = number of buttons = 7

P = number of connected components = 1

M = E – N + 2P = 7-7 + 2\*1 = 2

**Independent road**

Independent line 1: 1-2-3-7

Independent line 2: 1-2-4-5-6-7

**Test Suite**

**TC\_DeleteProduct\_01: 1-2-3-7**

Input: ID = 10

Expected result: “No product ID to delete”

**TC\_DeleteProduct\_02: 1-2-4-5-6-7**

Input: ID = 1

Expected result: “Product deleted successfully”

#### 2.2.2. Purchasing process

**Add product to cart**

**Control flow graph diagram**

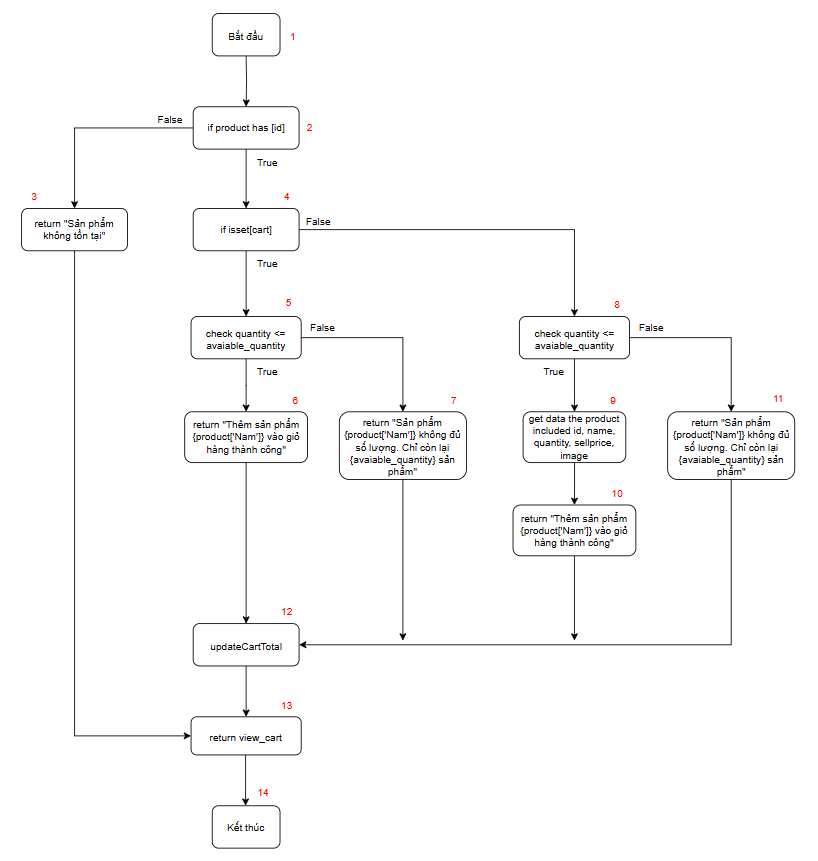
****

Figure 32. Control flow diagram Add product Shopping cart

**Complexity​**

E = number of sides = 17

N = number of buttons = 14

P = number of connected components = 1

M = E – N + 2P = 18-15 + 2\*1 = 5

**Independent road**

Independent line 1: 1-2-3-14-15

Independent line 2: 1-2-4-5-6-12-13-14

Independent line 3: 1-2-4-5-7-12-13-14

Independent line 4: 1-2-4-8-9-10-12-13-14

Independent Line 5: 1-2-4-8-11-12-13-14

**Test Suite**

**TC\_AddCart\_01: 1-2-3-13-14**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_AddCart\_02: 1-2-4-5-6-12-13-14**

Input: ID = 1, quantity = 2, sell\_price = 120

Expected result: “ Successfully added product {product['Male']} to cart ”, total amount 240USD

**TC\_AddCart\_03: 1-2-4-5-7-12-13-14**

Input: ID = 1, quantity = 100, sell\_price = 120

Expected result: “ Product {product['Male']} is out of stock. Only {avaiable\_quantity} products left ”, total price 0USD

**TC\_AddCart\_04: 1-2-4-8-9-10-12-13-14**

Input: : ID = 1, quantity = 2, sell\_price = 120, condition: cart has quatity = 1

Expected result: “ Successfully added product {product['Male']} to cart ”, total amount 360USD

**TC\_AddCart\_05: 1-2-4-8-11-12-13-14**

Input: ID = 1, quantity = 100, sell\_price = 120, condition: cart has quatity = 1

Expected result: “ Product {product['Male']} is out of stock. Only {avaiable\_quantity} products left ”, total price 120USD

**Update quantity of products in cart**

**Control flow graph diagram**

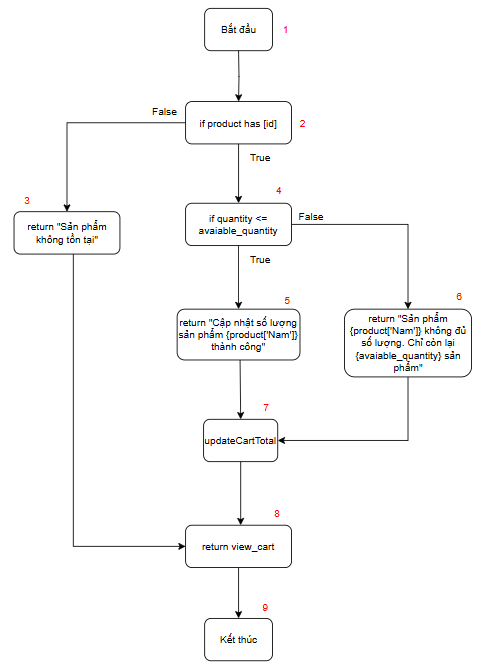
****

Figure 33. Control flow diagram of Update Shopping Cart

**Complexity​**

E = number of sides = 10

N = number of buttons = 9

P = number of connected components = 1

M = E – N + 2P = 10-19 + 2\*1 = 3

**Independent road**

Independent line 1: 1-2-3-8-9

Independent line 2: 1-2-4-5-7-8-9

Independent line 3: 1-2-4-6-7-8-9

**Test Suite**

**TC\_UpdateCart\_01: 1-2-3-8-9**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_UpdateCart\_02: 1-2-4-5-7-8-9**

Input: ID = 1, quantity = 5, sell\_price = 120, cart has quatity = 1

Expected result: “ Update product quantity {product['Male']} success ", total 1100USD

**TC\_UpdateCart\_03: 1-2-4-6-7-8-9**

Input: ID = 1, quantity = 100, sell\_price = 120, cart has quatity = 1

Expected result: “ Product {product['Male']} is out of stock. Only {avaiable\_quantity} products left ”, total price 120USD

**Remove product from cart**

**Control flow graph diagram**

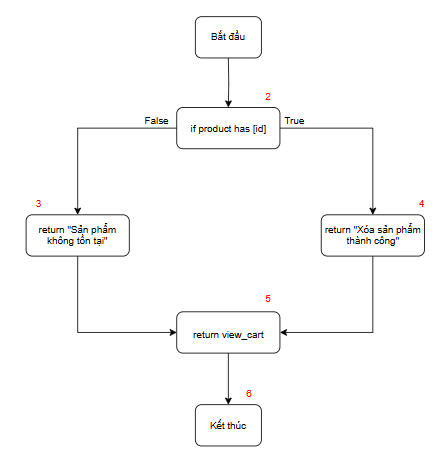


Figure 34. Control flow diagram Delete product Shopping cart

**Complexity​**

E = number of sides = 6

N = number of buttons = 6

P = number of connected components = 1

M = E – N + 2P = 6-6 + 2\*1 = 2

**Independent road**

Independent line 1: 1-2-3-5-6

Independent line 2: 1-2-4-5-6

**Test Suite e**

**TC\_DeleteCart\_01: 1-2-3-6**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_DeleteCart\_02: 1-2-4-5-6**

Input: ID = 1

Expected result: “Product deleted successfully”

**Checkout cart**

**Control flow graph diagram**

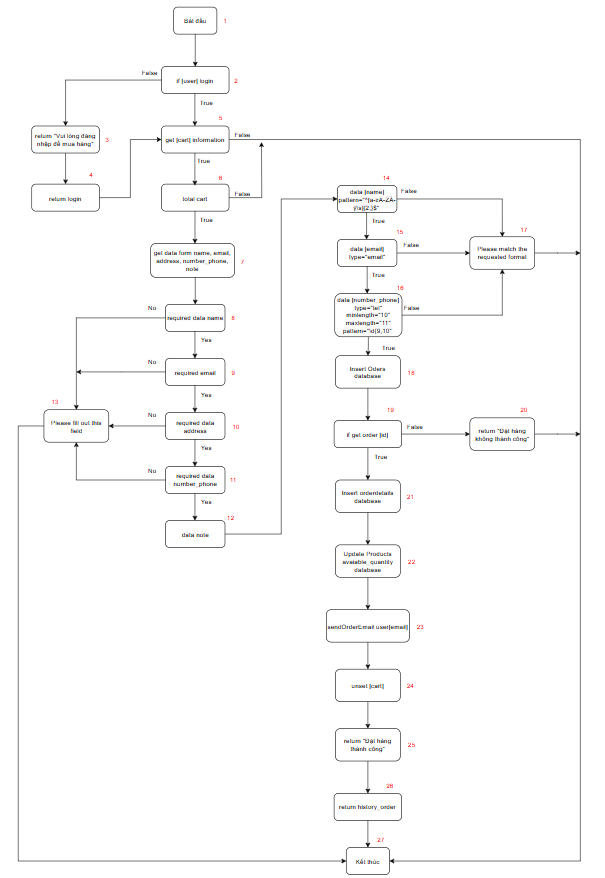


Figure 35. Control flow diagram of Checkout Shopping Cart

**Complexity​**

E = number of sides = 37

N = number of buttons = 27

P = number of connected components = 1

M = E – N + 2P = 37-27 + 2\*1 = 12

**Independent road**

Independent line 1: 1-2-3-4-5...

Independent Line 2: 1-2-5-27

Independent line 3: 1-2-5-6-27

Independent line 4: 1-2-5-6-7-8-13-27

Independent line 5: 1-2-5-6-7-8-9-13-27

Independent line 6: 1-2-5-6-7-8-9-10-13-27

Independent line 7: 1-2-5-6-7-8-9-10-11-13-27

Independent line 8: 1-2-5-6-7-8-9-10-11-12-14-17-27

Independent line 9: 1-2-5-6-7-8-9-10-11-12-14-15-17-27

Independent line 10: 1-2-5-6-7-8-9-10-11-12-14-15-16-17-27

Independent line 11: 1-2-5-6-7-8-9-10-11-12-14-15-16-18-19-20-27

Independent line 12: 1-2-5-6-7-8-9-10-11-12-14-15-16-18-19-21-22-23-24-25-26-27

**Test Suite**

**TC\_Checkout\_01: 1-2-3-4-5...**

Input: user no login

Expected result: “Please login to purchase”

**TC\_Checkout\_ 02: 1-2-5-27**

Input: ID = 1

Expected result: No order information displayed

**TC\_Checkout\_03: 1-2-5-6-27**

Input: sell\_price = 120

Expected result: Order total not displayed

**TC\_Checkout\_04: 1-2-5-6-7-8-13-27**

Input: name = “ ”

Expected result: Please fill out this field

**TC\_Checkout\_05: 1-2-5-6-7-8-9-13-27**

Input: email = “ ”

Expected result: Please fill out this field

**TC\_Checkout\_06: 1-2-5-6-7-8-9-10-13-27**

Input: address = “ ”

Expected result: Please fill out this field

**TC\_Checkout\_07: 1-2-5-6-7-8-9-10-11-13-27**

Input: number \_ phone = “ ”

Expected result: Please fill out this field

**TC\_Checkout\_08: 1-2-5-6-7-8-9-10-11-12-14-17-27**

Input: name = 123

Expected result: Please match the requested format

**TC\_Checkout\_09: 1-2-5-6-7-8-9-10-11-12-14-15-17-27**

Input: email = “ tannhut2111”

Expected result: Please match the requested format

**TC\_Checkout\_10: 1-2-5-6-7-8-9-10-11-12-14-15-16-17-27**

Input: number \_ phone = “090164180000"”

Expected result: Please match the requested format

**TC\_Checkout\_11: 1-2-5-6-7-8-9-10-11-12-14-15-16-18-19-20-27**

Input: ID = 10

Expected result: “Order failed”

**TC\_Checkout\_12:1-2-5-6-7-8-9-10-11-12-14-15-16-18-19-21-22-23-24-25-26-27**

Input: name = “nhut”, email = “ [tannhut2111@gmail.com](mailto:tannhut2111@gmail.com) ”, address = “HCM”, number\_phone = “0901641800”, note = “”

Expected result: “Order successfully”

### 2.3. Black box testing

Black-box Testing **Checklist** is a list of criteria or items to be tested to ensure that the software functions as per the requirements and expectations without regard to the source code structure or internal logic. This checklist focuses on testing the functionality and user interface of the application based on the inputs and outputs.

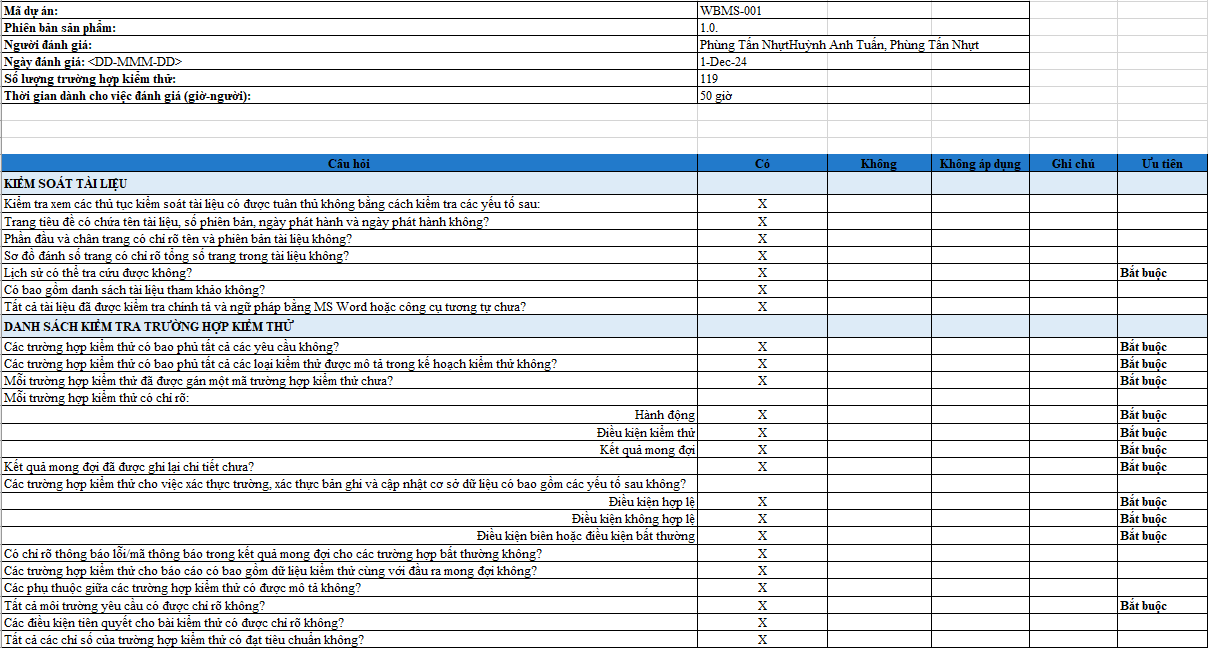


Figure 36. Black Box CheckList

**Black Box Testing** is a software testing method that focuses on testing the functionality of a system without regard to its internal structure. The main purpose of black box testing is to ensure that the software operates according to its specified requirements and handles all input scenarios correctly.

- Ensure software meets requirements:

* Check that the output matches expectations based on the inputs and functional requirements.
* Verify that business and technical requirements are met.

- Error detected in function:

* Incorrect input data handling.
* Miscalculation.
* Missing functionality or not handling as required.

- Focus on the external behavior of the system:

* No need to know the internal workings of the system, just care about the results.
* Suitable for testers without programming experience to perform.

- Ensure interoperability:

* Ensure that the system interacts properly with other systems or with users.

**Techniques used in the project:**

- **The Decision Table method** is a black box testing method used to test the combination of input conditions and output actions. This method is very effective when the system has complex conditions and needs to test all of their possibilities. Each input condition is combined with the corresponding output actions to form a rule table, which helps to determine the necessary test scenarios. It consists of 3 main parts:

**Input conditions (Conditions):** These are the factors that the system checks. Each condition can have a value of True or False.

**Actions:** Actions to be performed based on the values of input conditions. These actions can include error messages, success messages, or other system operations.

**Rules:** Each rule in a decision table describes a specific combination of conditions and actions. A decision table can have multiple rules, each rule testing a different scenario.

#### 2.3.1. Import process

**Add product**

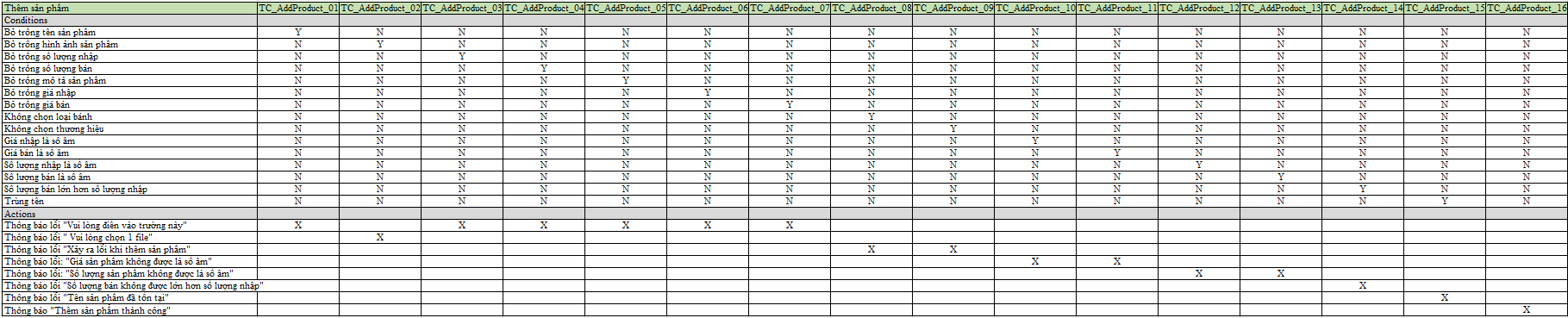
****

Figure 37. Decision table Add product

**Test Suite e**

**TC\_AddProduct\_01**

Input: Product name = ""

Expected result: Please fill out this field

**TC\_AddProduct\_02**

Input: Image = ""

Expected result: Please fill out this field

**TC\_AddProduct\_03**

Input: Quantity entered = ""

Expected result: Please fill out this field

**TC\_AddProduct\_04**

Input : Quantity sold = ""

Expected result: Please fill out this field

**TC\_AddProduct\_05**

Input: Description = ""

Expected result: Please fill out this field

**TC\_AddProduct\_06**

Input: Input price = ""

Expected result: Please fill out this field

**TC\_AddProduct\_07**

Input: Selling price = ""

Expected result: Please fill out this field

**TC\_AddProduct\_08**

Input: Cake type = ""

Expected result: "Add product failed"

**TC\_AddProduct\_09**

Input: Brand = ""

Expected result: "Add product failed"

**TC\_AddProduct\_10**

Input: Input price = -1

Expected result: "Product price cannot be negative"

**TC\_AddProduct\_11**

Input: Selling price = -1

Expected result: "Product price cannot be negative"

**TC\_AddProduct\_12**

Input: Quantity entered = -1

Expected result: "The number of products cannot be negative"

**TC\_AddProduct\_13**

Input: Quantity sold = -1

Expected result: "The number of products cannot be negative"

**TC\_AddProduct\_14**

Input: Quantity imported = 100, Quantity sold = 120

Expected result: "The quantity sold must not be greater than the quantity imported"

**TC\_AddProduct\_15**

Input: Product name = "DULCE"

Expected result: “Product name already exists”

**TC\_AddProduct\_16**

Input: Product name = “ABC”, Image = “1aba114354.product-big-4.jpg”, Import quantity = 10, Sales quantity = 10, Description = “Delicious cakes”, Import price = “100”, Sales price = “120”, Cake type = “Cakes”, Brand = “SamSung”

Expected result: “Added product successfully”

**Product Updates**

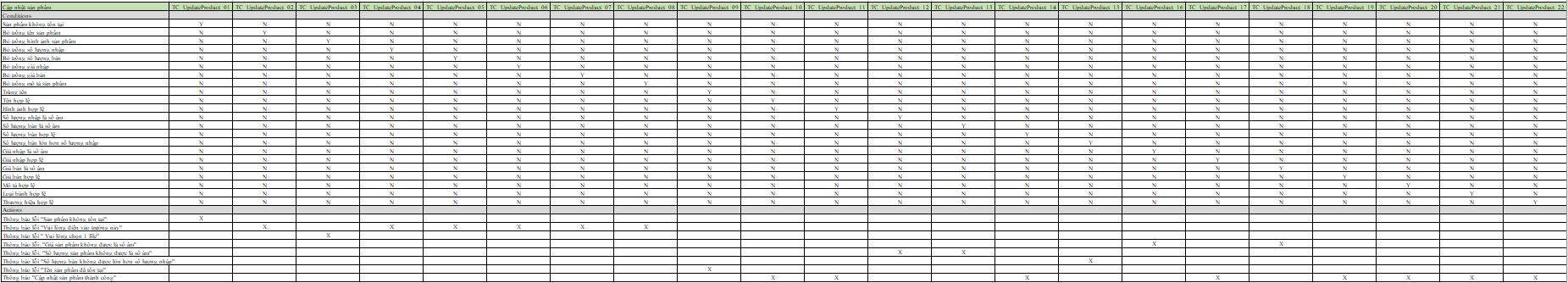


Figure 38. Product Update Decision Table

**Test Suite**

**TC\_UpdateProduct\_01**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_UpdateProduct\_02**

Input: ID = 1, Product name = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_03**

Input: ID = 1, Image = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_04**

Input: ID = 1, Quantity entered = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_05**

Input: ID = 1, Quantity sold = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_06**

Input: ID = 1, Input price = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_07**

Input: ID = 1, Price = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_08**

Input: ID = 1, Description = ""

Expected result: “Please fill out this field”

**TC\_UpdateProduct\_09**

Input: ID = 1, Product name = "DULCE"

Expected result: “Product name already exists”

**TC\_UpdateProduct\_10**

Input: ID 1, Product Name = "ABC"

Expected result: “Product update successful”

**TC\_UpdateProduct\_11**

Input : ID = 1, Image = “1aba114354.product-big-4.jpg”

Expected result: “Product update successful”

**TC\_UpdateProduct\_12**

Input: ID = 1 , Quantity entered = -1

Expected result: "The number of products cannot be negative"

**TC\_UpdateProduct\_13**

Input: ID = 1, Quantity sold = -1

Expected result: "The number of products cannot be negative"

**TC\_UpdateProduct\_14**

Input: ID = 1, Quantity entered = 100, Quantity sold = 80

Expected result: “Product update successful”

**TC\_UpdateProduct\_15**

Input: ID = 1, Quantity entered = 100, Quantity sold = 120

Expected result: “The quantity sold must not be greater than the quantity imported”

**TC\_UpdateProduct\_16**

Input: ID = 1, Input price = -1

Expected result: "Product price cannot be negative"

**TC\_UpdateProduct\_17**

Input: ID = 1, Input price = 100

Expected result: “Product update successful”

**TC\_UpdateProduct\_18**

Input: ID = 1, Selling price = -1

Expected result: "Product price cannot be negative"

**TC\_UpdateProduct\_19**

Input: ID = 1, Price = 100

Expected result: “Product update successful”

**TC\_UpdateProduct\_20**

Input: ID = 1 , Description = “Delicious cake”

Expected result: “Product update successful”

**TC\_UpdateProduct\_21**

Input: ID = 1, Cake type = “Cakes”

Expected result: “Product update successful”

**TC\_UpdateProduct\_22**

Input: ID = 1, Brand = “SamSung”

Expected result: “Product update successful”

**Delete product**

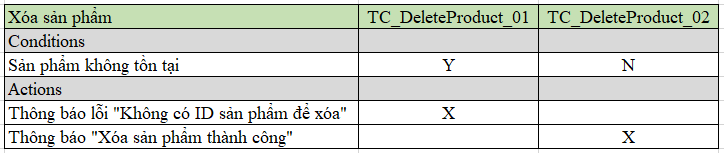


Figure 39. Product Delete Decision Table

**Test Suite e**

**TC\_DeleteCart\_01**

Input: ID = 10

Expected result: “ No product ID to delete”

**TC\_DeleteCart\_02**

Input: ID = 1

Expected result: “Product deleted successfully”

#### 2.3.2. Purchasing process

**Add product to cart**



Figure 40. Decision table Add product to Cart

**Test Suite e**

**TC\_AddCart\_01**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_AddCart\_02**

Input: ID =1, Quantity = 2, Price = 120USD

Expected result: “ Successfully added product ... to cart ”, total amount is 120USD

**TC\_AddCart\_03**

Input: ID =1, Quantity = 100, Price = 120USD

Expected result: “ Product ... quantity is not enough. Only ... products left ”, total amount 0USD

**TC\_AddCart\_04**

Input: ID = 1, Quantity = 2, Price = 120USD, Condition: Cart quantity = 1

Expected result: “ Successfully added product ... to cart ”, total amount 360USD

**TC\_AddCart\_05**

Input: ID = 1, Quantity = 100, Price = 120USD, Condition: Cart quantity = 1

Expected result: “ Product ... quantity is not enough. Only ... product left ”, total price 120USD

**Update products in cart**

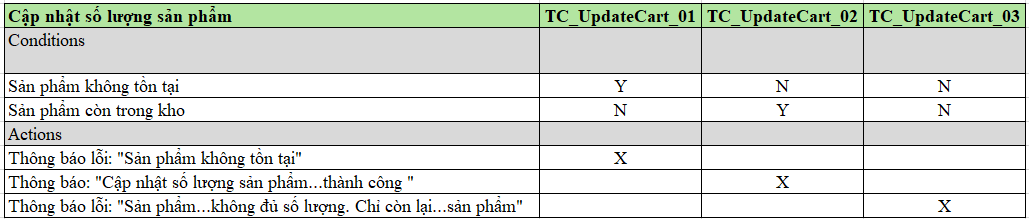


Figure 41. Decision table Update quantity in Shopping Cart

**Test Suite**

**TC\_UpdateCart\_01**

Input: ID = 10

Expected result: “Product does not exist”

**TC\_UpdateCart\_02**

Input: Quantity = 5, Cart has quantity = 1

Expected result: “ Update product quantity... successfully ”, total amount 1100USD

**TC\_UpdateCart\_03**

Input: Quantity = 100, Cart has quantity = 1

Expected result: “ Product ... quantity is not enough. Only ... product left ”, total price 120USD

**Remove product from cart**

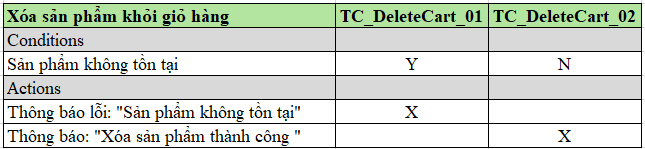


Figure 42. Decision table to Remove product from shopping cart

**Test Suite**

**TC\_DeleteCart\_01**

Input: ID = 10

Condition: Product does not exist (N)

Expected result: “Product does not exist”

**TC\_DeleteCart\_02**

Input: ID = 1

Condition: Product does not exist (Y)

Expected result: “Product deleted successfully”

**Checkout cart**

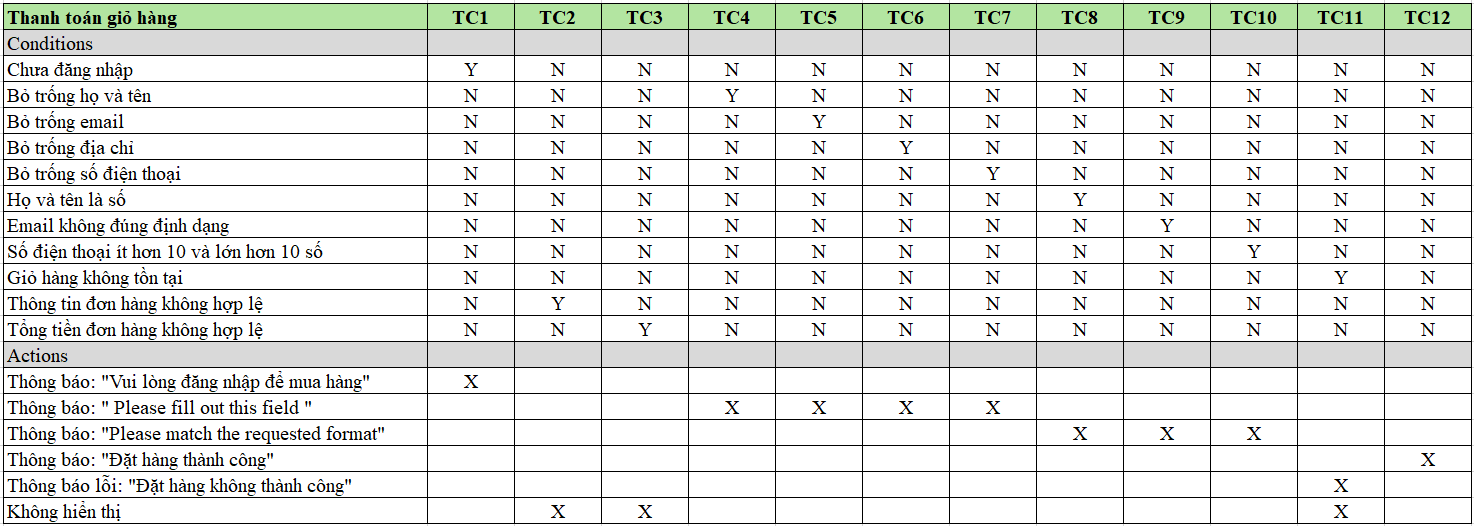


Figure 43. Cart Checkout Decision Table

**Test Suite**

**TC\_Checkout\_01**

Input: user no login

Expected result: “Please login to purchase”

**TC\_Checkout\_02**

Input: ID = 1

Expected result: No order information displayed

**TC\_Checkout\_03**

Input: Price = 120USD

Expected result: Order total not displayed

**TC\_Checkout\_04**

Input: Full name = " "

Expected result: Please fill out this field

**TC\_Checkout\_05**

Input: Email = " "

Expected result: Please fill out this field

**TC\_Checkout\_06**

Input: Address = " "

Expected result: Please fill out this field

**TC\_Checkout\_07**

Input: Phone number = " "

Expected result: Please fill out this field

**TC\_Checkout\_08**

Input: Full name = "1"

Expected result: Please match the requested format

**TC\_Checkout\_09**

Input : Email = "tannhut2111"

Expected result: Please match the requested format

**TC\_Checkout\_10**

Input: Phone number = "090164180"

Expected result: Please match the requested format

**TC\_Checkout\_11**

Input: ID = 10

Expected result: “Order failed”

**TC\_Checkout\_12**

Input: Full name = “nhut”, Email = "tannhut2111@gmail.com", Address = “HCM”, Phone number = “0901641800”, Note = “”

Expected result: “Order failed”

## 3. System Testing

**Test Case** is a set of conditions, inputs, actions to be taken, and expected results designed to test a specific functionality of a software or system. Each test case describes in detail how to test and determine whether the software meets the requirements or test criteria.

**Purpose of Test Case**

**Software quality assurance** : Test cases help to check that the software works as required.

**Error detected** : Identify errors or defects in the system for repair.

**Testing instructions** : Test cases provide detailed instructions to testers to perform testing consistently.

**Reuse** : Test cases can be reused in subsequent tests or similar projects.

**Documentation support** : Test case is an important reference document in software development and maintenance process.

**Verify request** : Test cases help ensure that the software meets the specified technical and functional requirements.

### 3.1. TestCase List

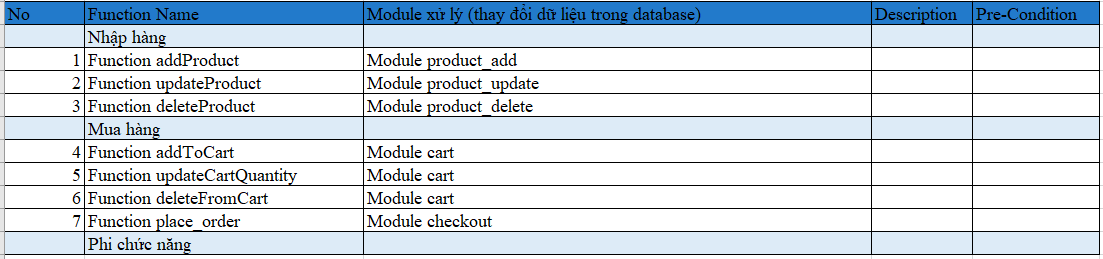


Figure 44. List of test cases

**Import goods**

* **Function addProduct**
  + **Processing module** : product\_add
  + **Description** : This function allows the user to add a new product to the database. The information to be entered includes product name, image, quantity, price, description, cake type, and brand.
  + **Pre-Condition** : Access to the import module and valid product data is required.
* **Function updateProduct**
  + **Processing module** : product\_update
  + **Description** : Function to update product information already in the system, including fields such as name, image, quantity, import price, selling price, description, cake type, and brand.
  + **Pre-Condition** : The product needs to be identified and have a valid ID before the update can be performed.
* **Function deleteProduct**
  + **Processing module** : product\_delete
  + **Description** : Function to delete a product from the system using the product ID. After deletion, the product will no longer exist in the database.
  + **Pre-Condition** : The product must exist in the system and have a valid ID to be deleted.

**Purchase**

* **Function addToCart**
  + **Processing module** : cart
  + **Description** : Function to add products to the user's shopping cart. Users can add multiple products to the cart, and the system will store that product information in the cart.
  + **Pre-Condition** : User must be logged in and have permission to add products to cart. Product must be in stock.
* **Function updateCartQuantity**
  + **Processing module** : cart
  + **Description** : Function to update the quantity of products in the user's shopping cart. The user can change the quantity of products already in the cart.
  + **Pre-Condition** : The cart already contains a product whose quantity needs to be changed. The product must have sufficient quantity in stock.
* **Function deleteFromCart**
  + **Processing module** : cart
  + **Description** : Function to delete products from the shopping cart. Users can delete any product in their shopping cart.
  + **Pre-Condition** : The shopping cart must contain products to delete and the user must have permission to manipulate his/her shopping cart.
* **Function place\_order**
  + **Processing module** : checkout
  + **Description** : Shopping cart payment function. The user will make payment when completing the shopping cart, and the system will process the order, deduct money and save transaction information.
  + **Pre-Condition** : User needs to be logged in, shopping cart cannot be empty, and payment information must be complete and valid.

**Non-functional**

Handle large number of users simultaneously

2-factor authentication when logging in

Protection against SQL Injection

Encrypt connections with HTTPS

### 3.2. Import process

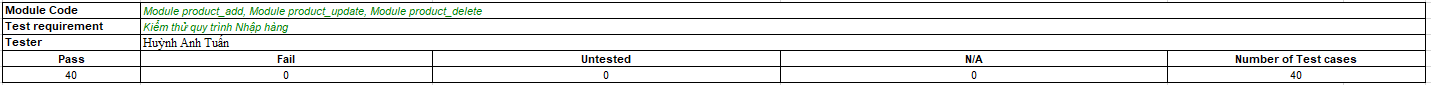


Figure 45. Module and test results for the import process

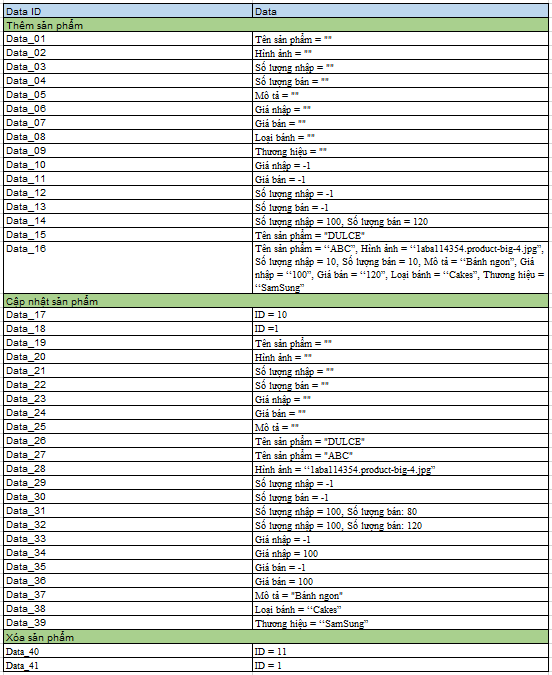


Figure 46. Data Test for the import process

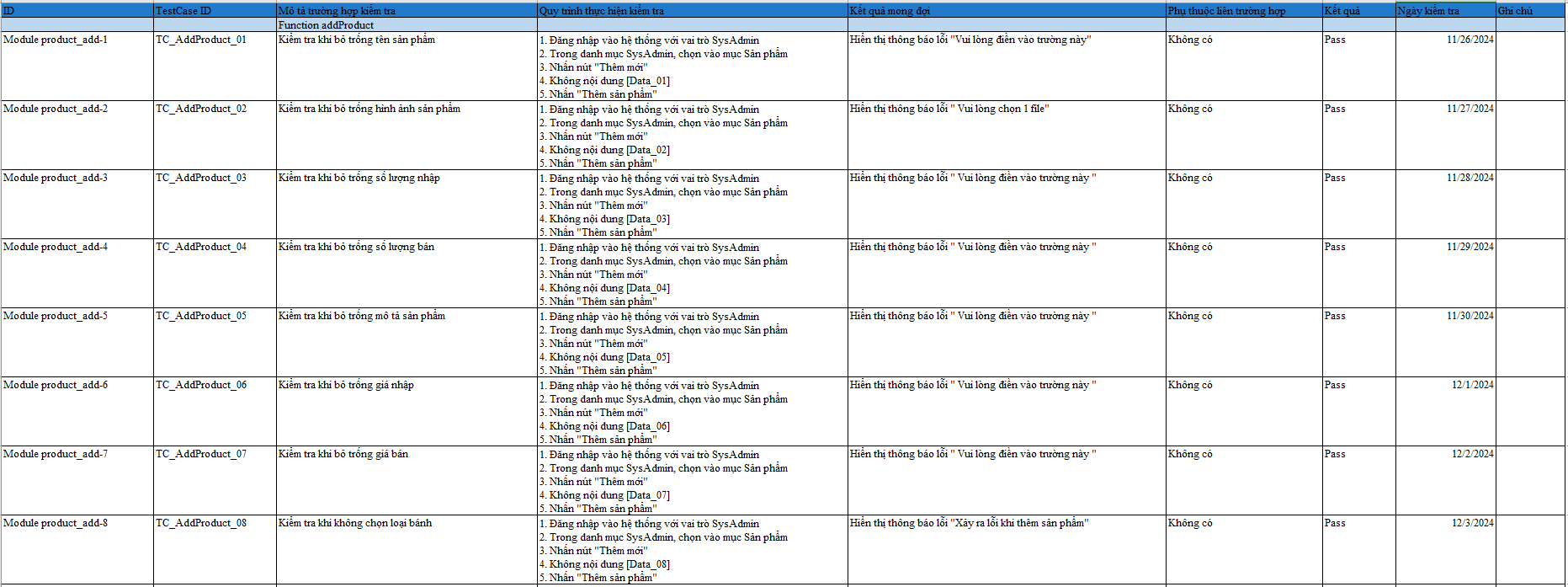
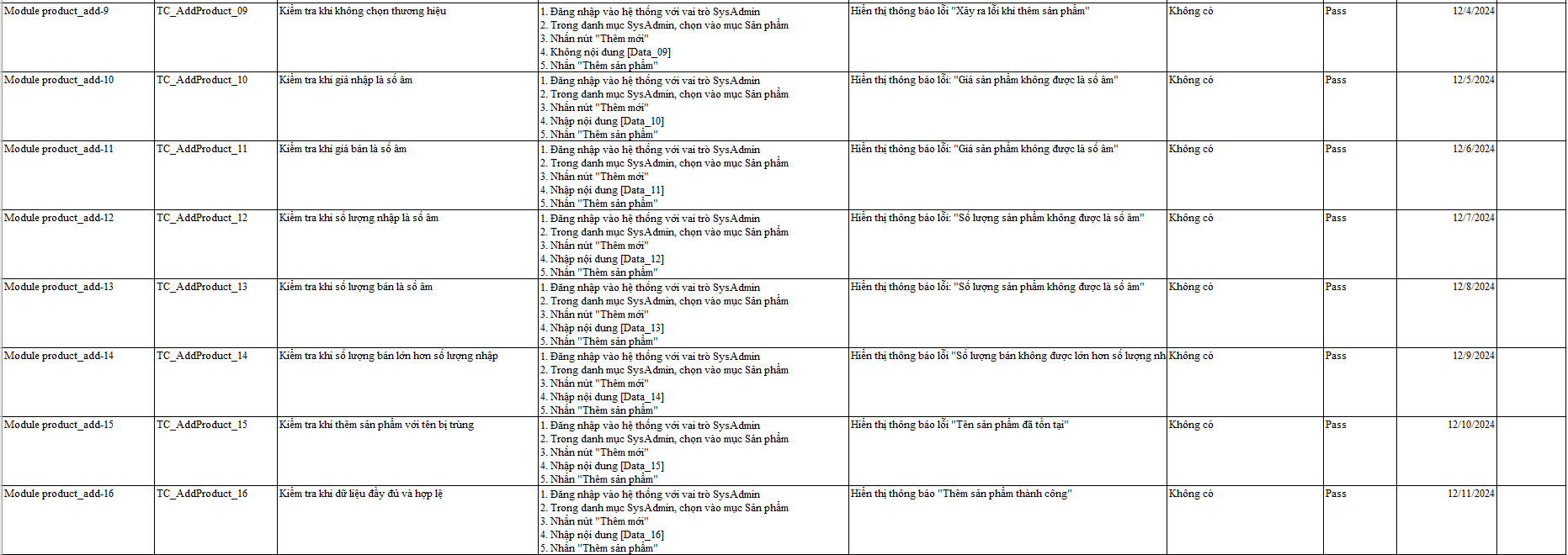
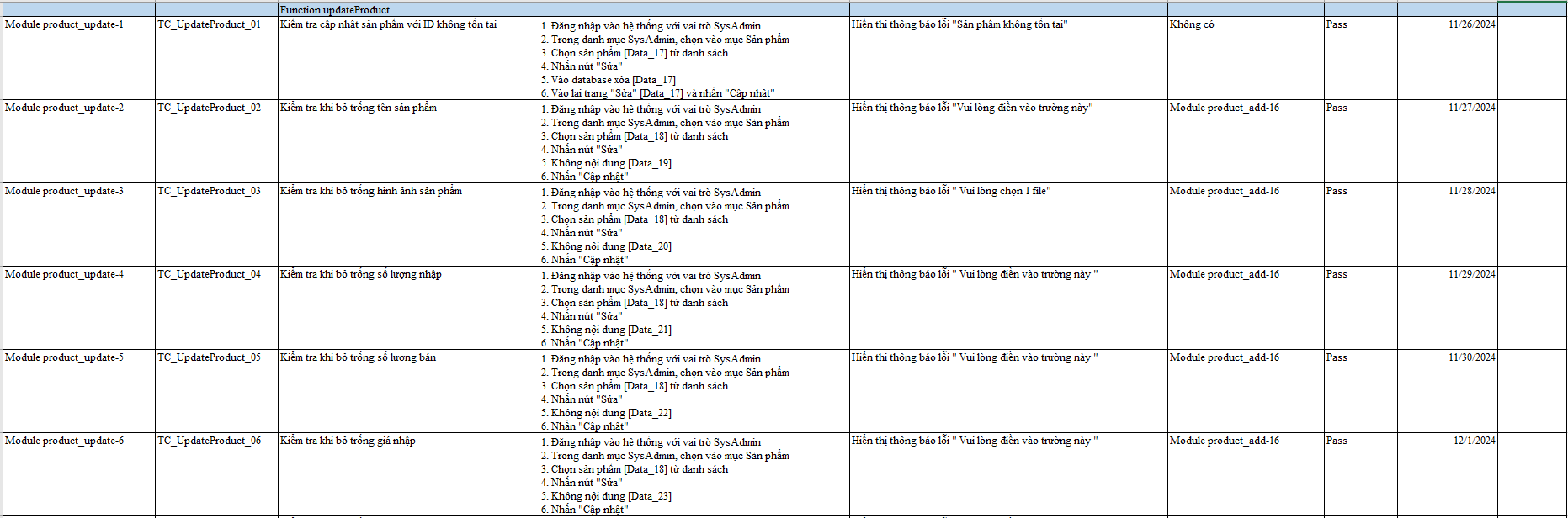
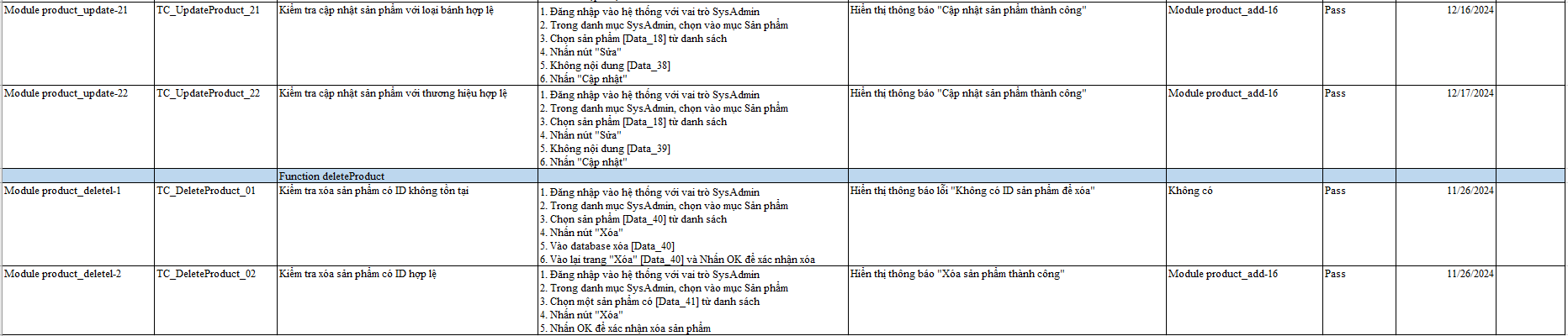
     

Figure 47. TestCase of import process

**ID** : Identifier for the test case.

**TestCase ID** : The ID of the test case, used for tracking and reference.

**Test case description** : A description of the purpose of the test case.

**Test Execution Process** : The detailed steps that the tester will perform to test the feature.

**Expected result** : The system will display an error message asking the user to fill in the product name field completely.

**Inter-case Dependencies** : If this test case depends on another test case, this information will be recorded.

**Output** : The actual result of the test case, which determines whether the system behaves as expected.

**Test date** : Date the test was performed.

**Notes** : Additional notes about the test, such as any issues that arose or points of attention.

### 3.3. Purchasing process

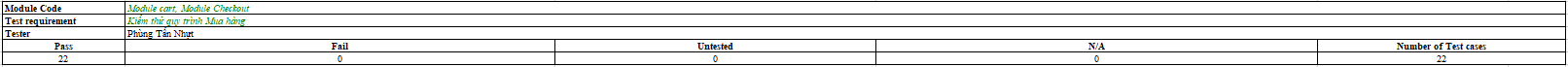


Figure 48. Module and test results for the purchasing process

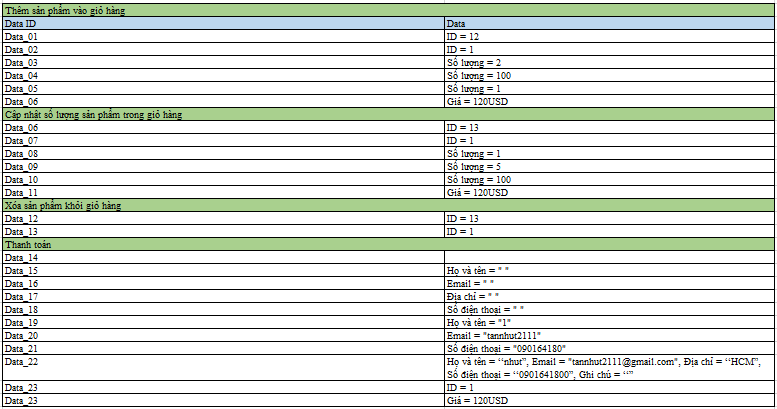


Figure 49. Test Data for the purchasing process

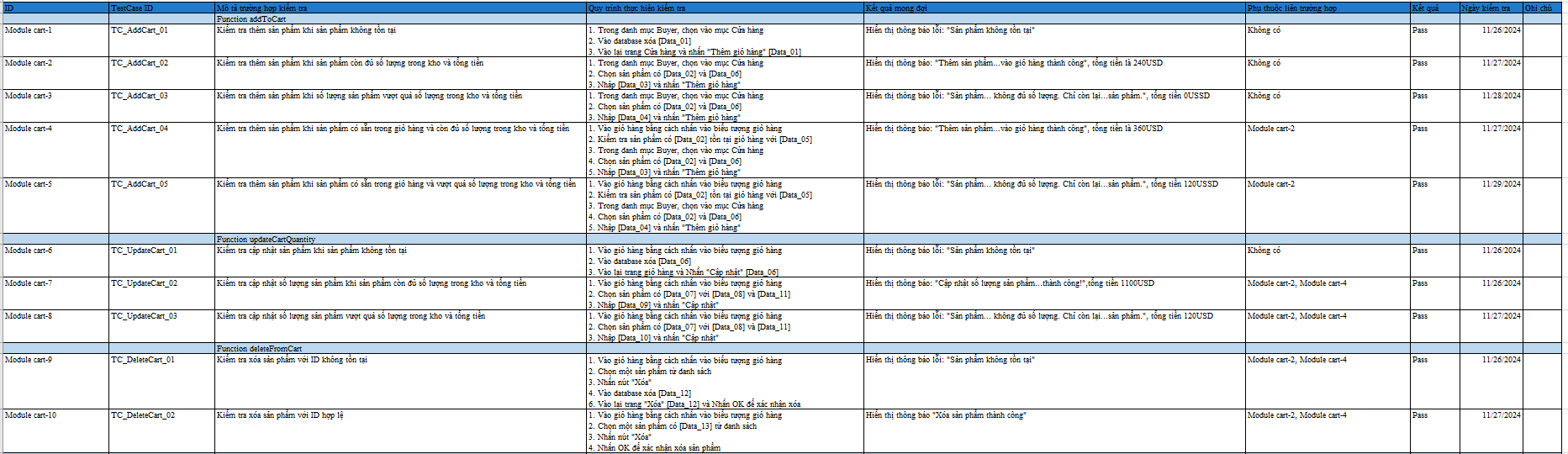
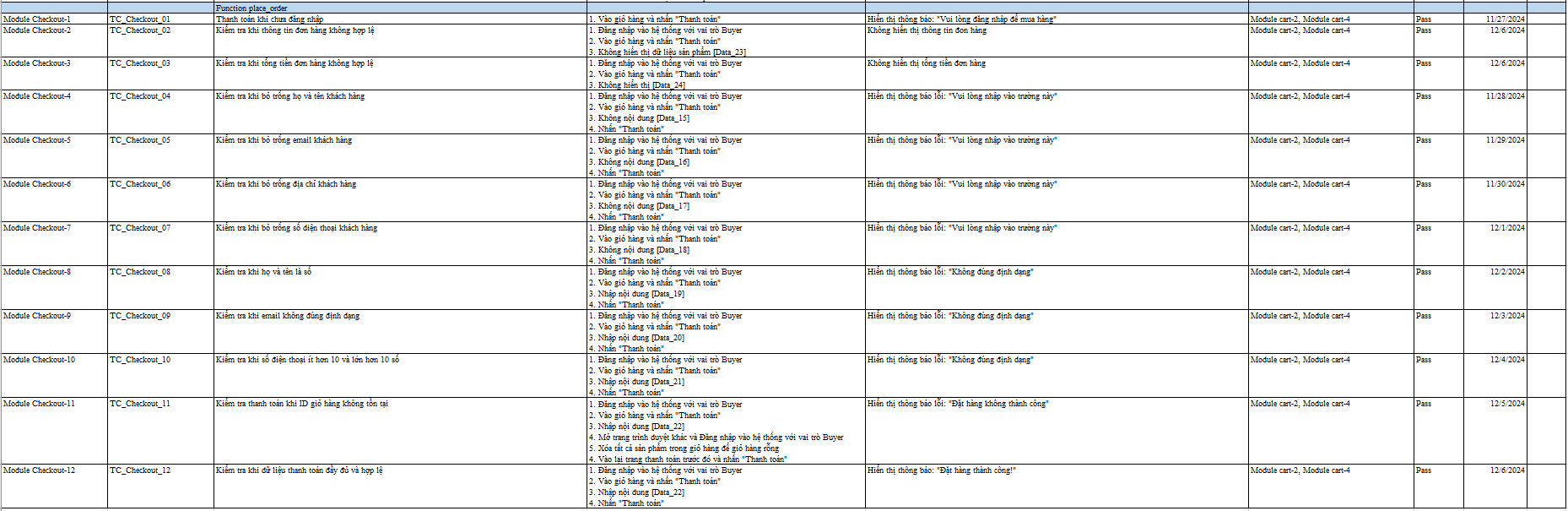
 

Figure 50. Purchase Process TestCae

**ID** : Identifier for the test case.

**TestCase ID** : The ID of the test case, used for tracking and reference.

**Test case description** : A description of the purpose of the test case.

**Test Execution Process** : The detailed steps that the tester will perform to test the feature.

**Expected result** : The system will display an error message asking the user to fill in the product name field completely.

**Inter-case Dependencies** : If this test case depends on another test case, this information will be recorded.

**Output** : The actual result of the test case, which determines whether the system behaves as expected.

**Test date** : Date the test was performed.

**Notes** : Additional notes about the test, such as any issues that arose or points of attention.

### 3.4 . Non-functional

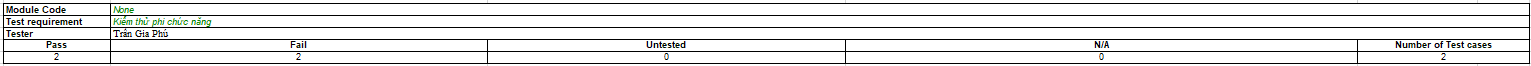


Figure 51. Test results for non-functionality

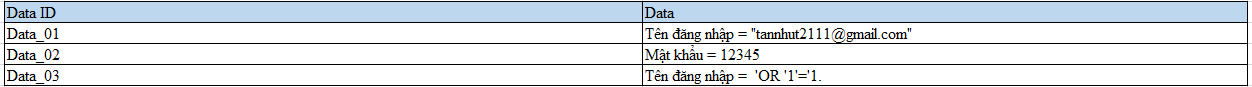


Figure 52. Test Data for Non-Functional



Figure 53. Non-functional TestCase

**ID** : Identifier for the test case.

**TestCase ID** : The ID of the test case, used for tracking and reference.

**Test case description** : A description of the purpose of the test case.

**Test Execution Process** : The detailed steps that the tester will perform to test the feature.

**Expected result** : The system will display an error message asking the user to fill in the product name field completely.

**Inter-case Dependencies** : If this test case depends on another test case, this information will be recorded.

**Output** : The actual result of the test case, which determines whether the system behaves as expected.

**Test date** : Date the test was performed.

**Notes** : Additional notes about the test, such as any issues that arose or points of attention.

## 4. Acceptance Testing

**Acceptance Testing** is the final testing phase before the software is deployed or released. Its main purpose is to ensure that the software meets all the business requirements and can function the way the customer or end user expects.

**Objective of Acceptance Testing**

* Verify that the software meets business requirements:
* Check that the software fulfills the functions and features that the customer expects.
* Ensuring Go-Live readiness:
* Ensure that the software is free of major bugs and can operate stably in a real environment.

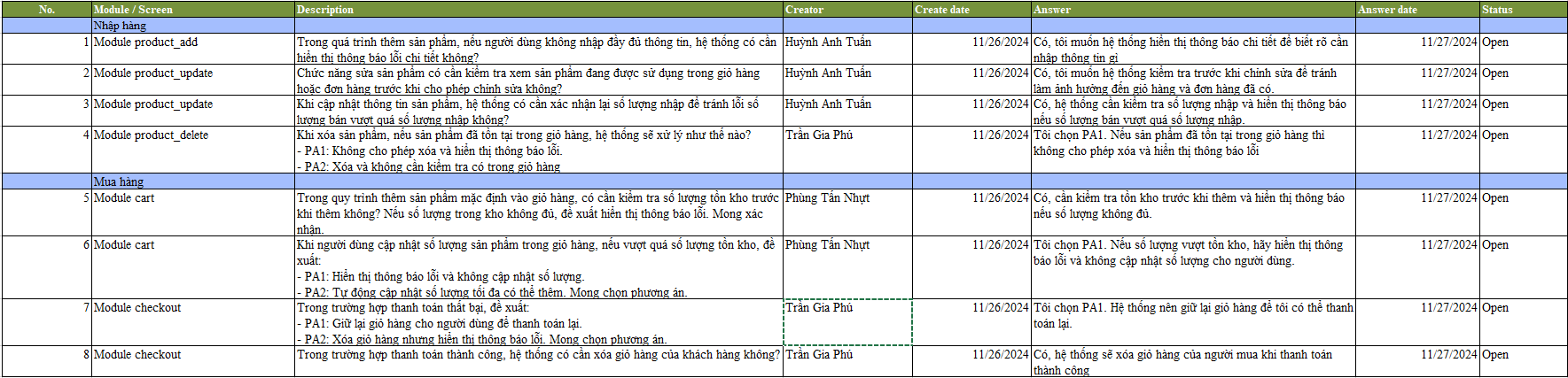


Figure 54. Q&A

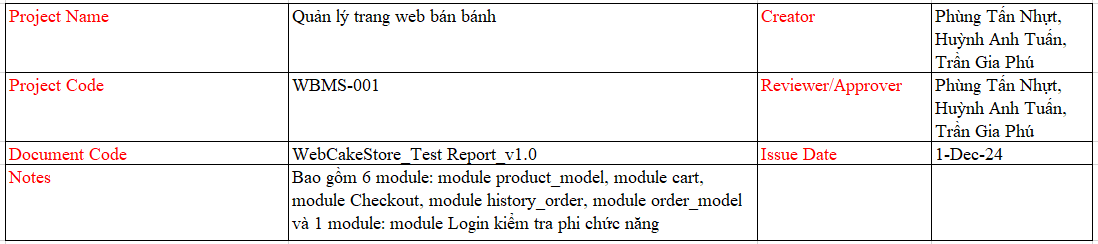
# CHAPTER 5: TEST RESULTS

## 1. Test Report

**Test** Report is a document that summarizes the results and findings from a software testing process. It is created to provide detailed information about the testing activities performed, the errors or issues found, and an overall assessment of the quality of the software.

**Purpose of Test Report:**

* Provide information about the state of the software: Reporting helps stakeholders (such as developers, managers, customers) understand the current state of the software, remaining bugs, and the level of meeting requirements.
* Release Decision Support: Test reports provide a basis for deciding whether the software is ready to be released.
* Documentation: Test reports help to keep a history of testing, which helps in future reference and comparison when needed.



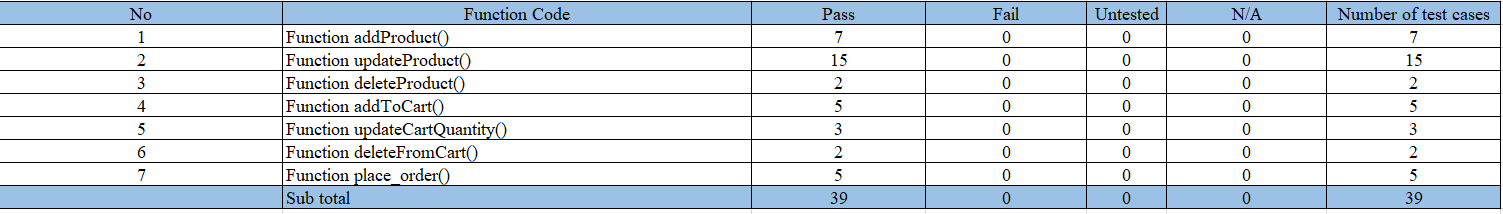


Figure 55. TestReport of TestCases (UnitTest

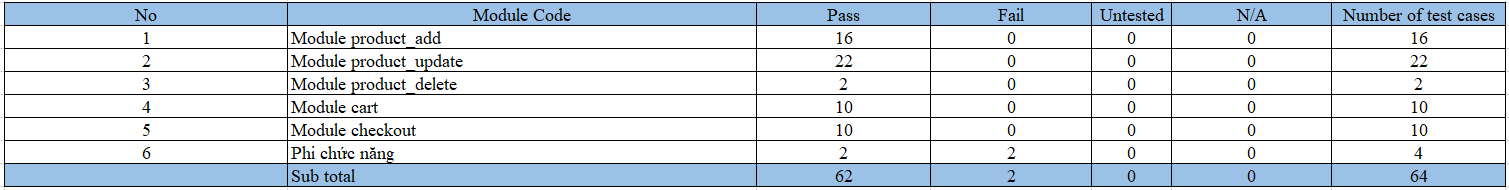
 

Figure 56. TestReport of TestCases (white box and black box)

## 2. Bug Report

**Error Report 1: Missing OTP Authentication**

**Title:** Missing OTP authentication when logging in

**Description:**   
When logging into the system, the OTP authentication step is missing. Users can log in without entering the OTP code, which reduces the security of the system.

**Steps to follow :**

1. Go to the system login page.
2. Enter a valid login name in the "Login Name" field.
3. Enter a valid password in the "Password" field.
4. Click the "Login" button.

**Expected result:**   
The system should prompt the user to enter the OTP code to complete the login. Only if the OTP code is valid, the user can log in successfully.

**Actual result:**   
The system does not display the OTP authentication step. The user can log in without entering the OTP code.

**Environment:**

* Test date: 11/26/2024
* Browser: [Google Chrome, Microsoft Edge]
* Operating System: [Windows]

**Renewable:** Always Happens

**Severity:** Severe

**Priority:** 1 (High)

**Solution:** New

**Error Report 2: Missing HTTPS encryption**

**Title:** HTTPS encryption not required by the system

**Description:**   
The system does not require the use of HTTPS connections, allowing users to access through unencrypted HTTP connections. This poses a great risk as sensitive information can be stolen while being transmitted over the network.

**Steps to follow :**

1. Open your browser and enter the system URL
2. Observe the protocol displayed in the browser's address bar.
3. Try accessing the system using HTTP instead of HTTPS.

**Expected result:**   
All connections to the system must be secured using HTTPS. If users access via HTTP, they must be automatically redirected to HTTPS.

**Actual result:**   
The system allows access over HTTP without automatically redirecting to HTTPS, leaving the connection unencrypted.

**Environment:**

* Test date: 11/28/2024
* Browser: [Google Chrome, Microsoft Edge]]
* Operating System: [Windows ]

**Renewable:** Always Happens

**Severity:** High

**Priority Level:** 2 ( Medium )

**Solution:** New

## 3. DefectList

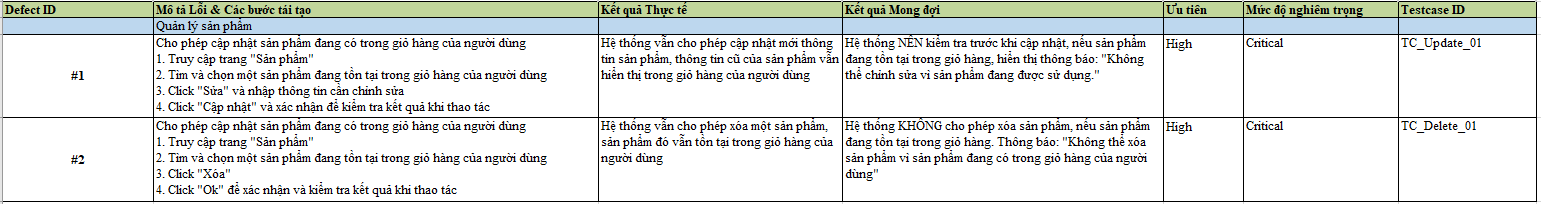


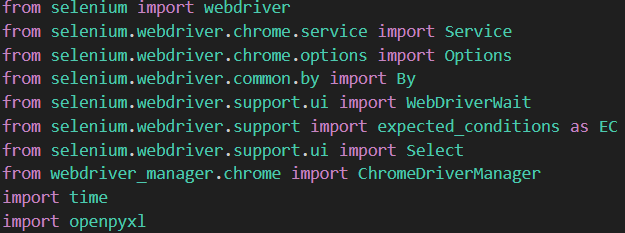
Figure 57. DefectList

## 4. Auto Test

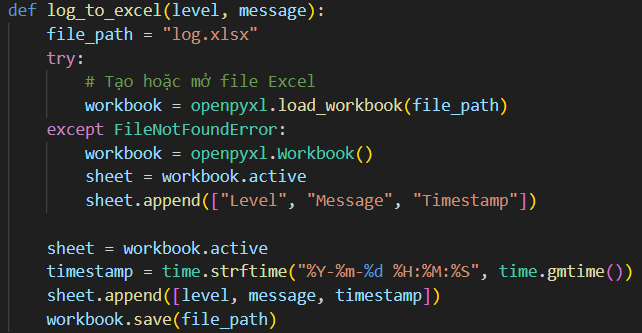
Automated testing is the process of using tools or scripts to automate software testing to ensure product quality and performance. Instead of manual testing, automated testing helps to execute test cases quickly, accurately, and repeatably, thereby saving time and reducing human errors.

Automated testing is commonly used in software development phases, especially with testing types such as functional, regression, and performance. A popular tool used is Selenium.

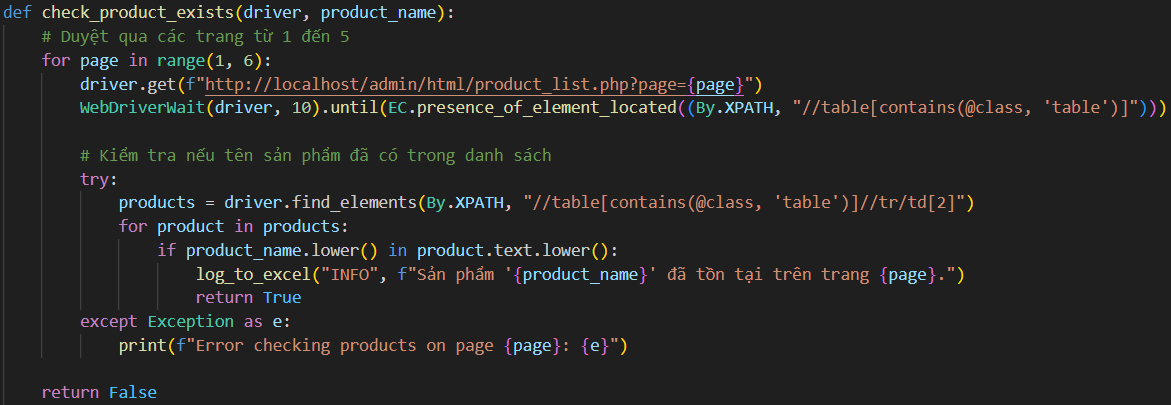
Steps to perform automated testing using Selenium for product functionality

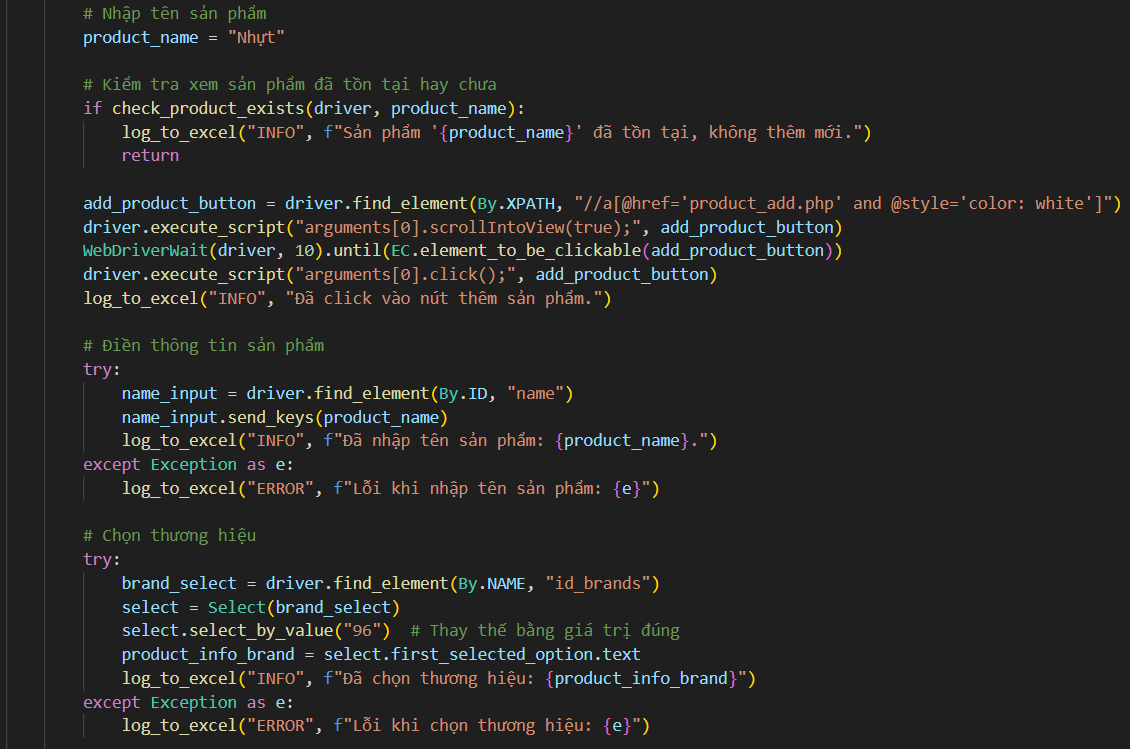
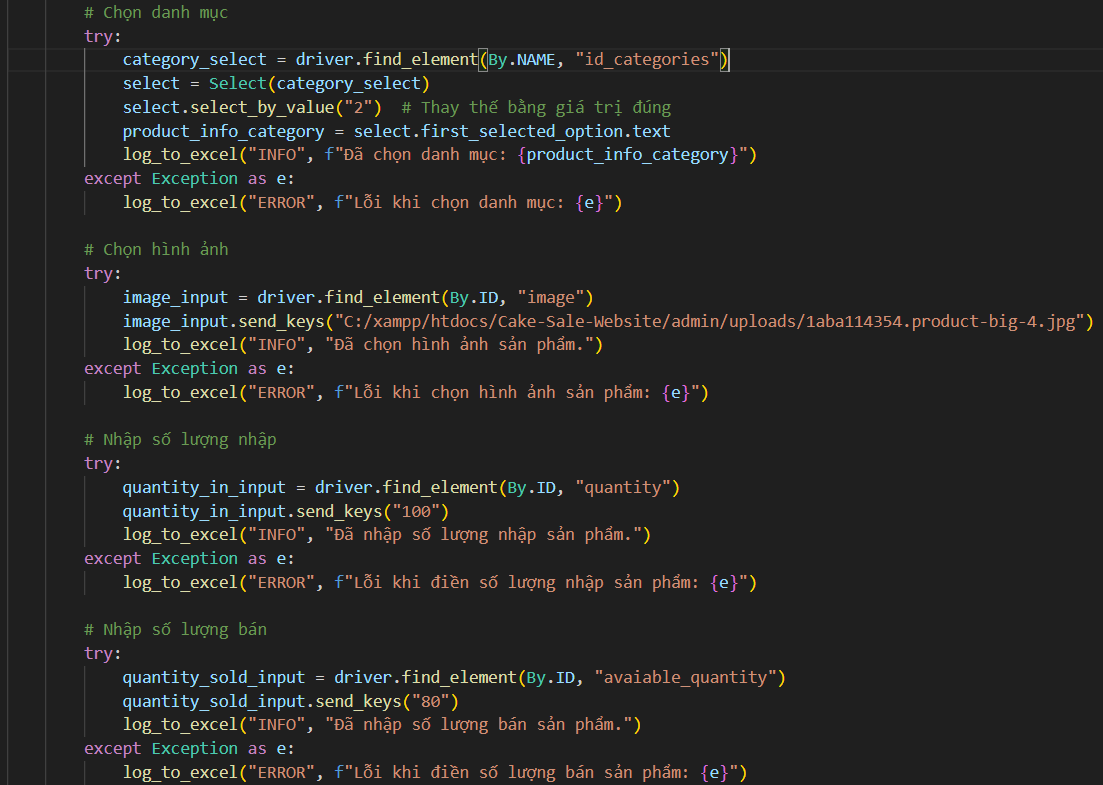
Step 1: Libraries used 

Step 2: Create a function to write logs to excel



Step 3: Create a function to check if the product exists or not (browse the pages)



Step 4: Create a function to add products   

Step 5: Execute



# CHAPTER 6. CONCLUSION

Software testing plays an extremely important role in ensuring the quality and stability of the system. When users make requests for a bakery website, the system designer needs to carefully analyze the requirements to build a suitable software architecture. The testing process will begin with writing test cases for each part of the system, including unit testing, integration testing, and system testing. Each of these types of testing helps detect errors at different levels, from the level of detail of each small part of the source code to the entire system. The results from this testing process will provide a clear view of the performance, security, and stability of the website, while ensuring that the system's functions meet user requirements, thereby improving the quality and reliability of the final product.