HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Test Plan

AIMS – An Internet Media Store

Subject: ITSS Software Development

Group 13

|  |  |
| --- | --- |
| Trương Công Thái Đức | 20225815 |
| Vũ Ngọc Lâm | 20225645 |
| Phan Sỹ Hùng | 20225631 |
| Nguyễn Mạnh Tuấn | 20225679 |

[1. Introduction 1](#_Toc498384509)

[1.1. Objective 1](#_Toc1530843107)

[1.2. Scope 1](#_Toc1025013138)

[1.3. Glossary 1](#_Toc372293196)

[1.4. References 1](#_Toc2091535339)

[2. Overall Description 1](#_Toc1308793064)

[2.1. General Overview 1](#_Toc77809743)

[2.2. Assumptions/Constraints/Risks 1](#_Toc1124140047)

[2.2.1. Assumptions 1](#_Toc968091616)

[2.2.2. Constraints 1](#_Toc1840676993)

[2.2.3. Risks 1](#_Toc378012467)

[3. Testing Approach/Strategy 1](#_Toc259187113)

[3.1. Overview 1](#_Toc1290874448)

[3.2. Scope of Unit Testing 1](#_Toc1851504458)

[3.3. Strategy for Designing Unit Tests 1](#_Toc380419633)

[3.4. Tools and Frameworks 1](#_Toc1257910631)

[4. Unit Testing Summary 1](#_Toc1954716077)

[4.1. Traceability from Test Cases to Use Cases 1](#_Toc728231621)

[4.2. Test Suite 1](#_Toc756952945)

[4.2.1. UC001 - “Place Order” 1](#_Toc838263058)

[4.2.2. UC002 1](#_Toc1199052903)

[4.2.3. UC003 1](#_Toc1435112501)

[4.2.4. UC004 1](#_Toc607939830)

[4.2.5. UC005 1](#_Toc21948850)

[4.2.6. UC006 1](#_Toc1827081445)

[4.2.7. UC007 1](#_Toc1027360904)

[4.2.8. UC008 1](#_Toc470332855)

[4.2.9. UC009 1](#_Toc1512608167)

[4.2.10. UC010 1](#_Toc1163933539)

[4.2.11. UC011 1](#_Toc1632950245)

[5. Test case Details 1](#_Toc1787534881)

# **Introduction**

<Identify the purpose of this document, its intended audience, and expected evolution of the document. Identify if this Test Plan covers all test functions for the project (i.e., main test plan for the project) or if it is only specific to a particular test phase(s) (e.g., Development Testing) or a specific test function(s). Also describe any security or privacy considerations associated with use of the Test Plan>

<Unit testing focuses on verifying the correctness of individual software components, such as functions, methods, or classes. The primary objectives include:

- Detecting and fixing defects at an early stage.

- Ensuring each module works as intended, independently of others.

- Providing a foundation for higher levels of testing (integration, system)>

## ***Objective***

The primary purpose of this document is to establish a common understanding of the software requirements among all stakeholders. It defines the functionalities, features, and performance expectations of the software system to be developed. By documenting these requirements in detail, this document serves as a reference point throughout the software development lifecycle, guiding the design, implementation, testing, and deployment phases. The intended audience includes project stakeholders, development teams, system architects, UI/UX designers, quality assurance testers, end-user representatives, and regulatory officers, all of whom rely on this document to ensure the successful realization of AIMS.

## ***Scope***

This AIMS – ‘An Internet Media Store’ software is developed to be a desktop platform e-commerce software, which helps users to order media products on the Internet, and the store managers, at the same time, are easier to manage their store as well as the orders.

This software can serve up to 1,000 customers simultaneously without significantly reducing performance and can operate continuously for 300 hours without failure. Additionally, the software can resume normal operation within a maximum of 1 hour after an incident. The maximum response time of the software is 2 seconds under normal conditions or 5 seconds during peak hours.

In AIMS, customers can not only search for products, but also sort products as they desire, they can place order or rush order for necessary cases. AIMS is supported by VNPay transactions; thus, customers can easily pay for their order. Moreover, customers can review their order and modify any information during the processing order stage. While shopkeepers can many their store by managing products directly in the system. They, meanwhile, can process the orders of the customers. For administrators, they are capable of managing users and privileges problems of users.

Additionally, for a desktop website, the graphical user interface (GUI) is carefully considered to meet end-user requirements and enhance the overall user experience. Throughout the development process, all documentation is systematically recorded to facilitate future maintenance and upgrades. We maintain a strong focus on each stage, ensuring adherence to the client’s timeline and the delivery of high-quality software. If any modifications are required, our team swiftly adapts to revise and refine our work accordingly.

## ***Glossary***

## ***References***

# **Overall Description**

**AIMS Project** is a desktop e-commerce software that operates 24/7, allowing new users to easily familiarize themselves. The software can serve up to 1,000 customers simultaneously without a significant decrease in performance and can operate continuously for 300 hours without failure. In the event of an incident, the software can resume normal operation within a maximum of 1 hour. The maximum response time of the software is 2 seconds under normal conditions and 5 seconds during peak hours.

AIMS software supports transactions only for physical media products (books, CDs, LP records, and DVDs), and requires information such as barcode, product description, quantity, warehouse entry date, product dimensions, and weight. The software allows product managers to add, view, edit, or delete products and stores a history of product addition, editing, and deletion operations. It will notify the product manager if any operation is invalid.

## ***General Overview***

**Software Overview**:

AIMS is a desktop e-commerce software that supports the management of media products like books, CDs, LP records, and DVDs. It allows product managers to add, edit, and delete products from the database. Customers are able to view products, order products and pay orders through VNPays.

**Actors**:

* **Product Manager**: Has the rights to add, edit, and delete products in the system.
* **Administrator**: Manages user accounts, can create, edit, delete users, and assign roles to users.
* **Customer**: Searches for products, views product details, adds products to the shopping cart, and proceeds with the checkout process.
* **VNPay:** facilitates online transactions

## ***Assumptions/Constraints/Risks***

## ***Assumptions***

## ***Constraints***

## ***Risks***

# **Testing Approach/Strategy**

## ***Overview***

## ***Scope of Unit Testing***

## ***Strategy for Designing Unit Tests***

**Mocking the Service Layer**: Since the controllers depend on services (like **ProductService**, **CartService**, **OrderService**, etc.), we will mock these services using **Mockito**. This ensures that we test the controller in isolation without worrying about the actual service logic.

## ***Tools and Frameworks***

* **JUnit 5**: A popular framework for unit testing in Java. It will be used to organize and run the tests.
* **Mockito**: A framework for mocking service dependencies. It will be used to simulate the service layer’s behavior without needing actual implementation.
* **Spring Test**: Used to simulate HTTP requests in Spring Boot applications for testing controllers.
* **Maven**: Used to run tests and manage project dependencies.

#### **Test Setup :**

1. **Mocks**: Mock service layer dependencies using **Mockito**.
2. **Controller Injection**: Inject mocked services into the controllers using **@InjectMocks**.
3. **Test Initialization**: Use **@BeforeEach** to set up mock data or configure services before each test.

# **Unit Testing Summary**

## ***Traceability from Test Cases to Use Cases***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| USE CASE ID | | | *UC001* | *UC002* | *UC003* | *UC004* | *UC005* | *…* |
| *Test Case ID* | *Test CaseTitle* | *Totals* | *2* | *1* | *2* | *0* | *1* | *1* |
| *UT001* |  | *1* | *x* |  |  |  |  |  |
| *UT002* |  | *3* | *x* | *x* | *x* |  |  |  |
| *UT006* |  | *1* |  |  |  |  | *x* |  |
| *UT007* |  | *1* |  |  |  |  |  | *x* |
| *…* |  | *1* |  |  | *x* |  |  |  |

## ***Test Suite***

## ***UC001 - “Place Order”***

## ***UC002***

## ***UC003***

## ***UC004***

## ***UC005***

## ***UC006***

## ***UC007***

## ***UC008***

## ***UC009***

## ***UC010***

## ***UC011***

# **Test case Details**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| UT001 | Validate Addition Function | Verify correct output for adding two positive integers | Matherator.add(a, b) | a = 3, b = 5 | 8 | TBD | TBD | Simple valid case |
| UT002 | Test Addition with Zero | Check addition when one operand is zero | Matherator.add(a, b) | a = 0, b = 7 | 7 | TBD | TBD | Edge case |

**5.1 Test case Details for “Guest Order"**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| UT001 | Validate Stock with valid Data | Test the **validateStock** method in **GuestOrderController** when there is enough stock for the product. The goal is to ensure the API returns the correct result when the product is available in sufficient quantity. | validateStock method in GuestOrderController | “ProductIDs"=101,”quantities”=4 | statusCode: 200 - isValid: true - errors: [] - availableStock: {101: 5} | - HTTP status: 200 - isValid: true - errors: [] - availableStock: {101: 5} | Pass | Simple valid case |
| UT002 | Validate Stock with Insufficient Stock | Test the **validateStock** method in **GuestOrderController** when there is not enough stock for the product. | validateStock method in GuestOrderController | - productIds: [101] - quantities: [4] | - statusCode: 200 - isValid: false - errors: ["Product 101 is out of stock"] - availableStock: {101: 3} | - HTTP status: 200 - isValid: false - errors: ["Product 101 is out of stock"] - availableStock: {101: 3} | Pass | None |

**5.2 Test case Details for “Order"**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| UT001 | Get All Orders | Test the GET /order endpoint to ensure it returns a list of all orders correctly. | getAllOrders method in OrderController | None | - statusCode: 200 - orders: List of orders | - HTTP status: 200 - List of orders. | Pass | Simple valid case |
| UT002 | Get Orders by User ID | Test the GET /order/{id} endpoint to check if it correctly retrieves orders for a specific user ID. | getOrdersByUserId method in OrderController | Id=’1’ (userid) | - statusCode: 200 | - HTTP status: 200 | Pass | None |
| UT003 | Change Order Status | Test the PUT /order/status/{id} endpoint to ensure that it correctly updates the status of an order. |  | Input: id=’1’(orderid) | - statusCode: 200 | - HTTP status: 400 | Fail |  |

**5.3 Test case Details for “Cart"**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| TC001 | Get All Cart Items | Test the GET /api/cart endpoint to ensure it returns a list of all cart items. | getAllOrderItems method in CartController | **Input**: None | **Expected Output**: - statusCode: 200 - orderItems: A list of order items in the cart (non-empty list). | **Actual Output**: - HTTP status: 200 - List of order items in the cart. | Pass |  |
| TC002 | Get Cart Item by ID (Found) | Test the GET /api/cart/{id} endpoint to ensure it returns the correct order item for a given ID. | getOrderItemById method in CartController | **Input**: id = 1 (Order Item ID) | **Expected Output**: - statusCode: 200 - orderItem: The order item with ID 1. | **Actual Output**: - HTTP status: 200 - Order item with ID 1. | Pass |  |
| TC003 | Get Cart Item by ID (Not Found) | Test the GET /api/cart/{id} endpoint to ensure it returns a 404 status code when the order item is not found. | getOrderItemById method in CartController | **Input**: id = 999 (Non-existing Order Item ID) | **Expected Output**: - statusCode: 404 - orderItem: null. | **Actual Output**: - HTTP status: 404 - No order item found for ID 999. | Pass |  |
| TC004 | Add Cart Item | Test the POST /api/cart endpoint to ensure it successfully adds an order item to the cart. | addOrderItem method in CartController | **Input**: - orderProduct: An order item to add. | **Expected Output**: - statusCode: 200 - orderItem: The order item added to the cart. | **Actual Output**: - HTTP status: 200 - The order item added to the cart. | Pass |  |
| TC005 | Add Multiple Cart Items | Test the POST /api/cart/add endpoint to ensure it adds multiple order items to the cart. | addOrderItems method in CartController | **Input**: - orderProducts: A list of order items. | **Expected Output**: - statusCode: 400 - | **Actual Output**: - HTTP status: 200 - A list of order items added to the cart. | Fail |  |
| TC006 | Update Cart Item | Test the PUT /api/cart/{id} endpoint to ensure it updates the order item in the cart. | updateOrderItem method in CartController | **Input**: - id = 1 (Order Item ID) - orderProduct: Updated order item details. | **Expected Output**: - statusCode: 200 - updatedOrderItem: The updated order item details. | **Actual Output**: - HTTP status: 200 - Updated order item with ID 1. | Pass |  |

**5.4 Test case Details for “DeliveryInfo"**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| TC001 | Get All Delivery Info | Test the GET /api/delivery-info endpoint to ensure that it returns all delivery information records. | getAllDeliveryInfos in DeliveryInfoController | **Input**: None | **Expected Output**: - statusCode: 200 - deliveryInfos: List of all delivery information records. | **Actual Output**: - HTTP status: 200 - List of delivery information records. | Pass |  |
| TC002 | Get Delivery Info by ID (Found) | Test the GET /api/delivery-info/{id} endpoint to ensure that it returns the correct delivery information by ID. | getDeliveryInfoById in DeliveryInfoController | **Input**: id = 1 (Valid delivery info ID) | **Expected Output**: - statusCode: 200 - deliveryInfo: Delivery information with ID 1. | **Actual Output**: - HTTP status: 200 - Delivery information with ID 1. | Pass |  |
| TC003 | Get Delivery Info by ID (Not Found) | Test the GET /api/delivery-info/{id} endpoint to ensure it returns 404 if the delivery information with the given ID does not exist. | getDeliveryInfoById in DeliveryInfoController | **Input**: id = 999 (Non-existing delivery info ID) | **Expected Output**: - statusCode: 404 - deliveryInfo: null (Not Found). | **Actual Output**: - HTTP status: 404 - No delivery information found for ID 999. | Pass |  |
| TC004 | Add Delivery Info | Test the POST /api/delivery-info endpoint to ensure that a new delivery information is added successfully. | addDeliveryInfo in DeliveryInfoController | **Input**: A valid DeliveryInformation object with necessary details. | **Expected Output**: - statusCode: 200 - deliveryInfo: The newly added delivery information. | **Actual Output**: - HTTP status: 200 - Newly added delivery information. | Pass |  |
| TC005 | Update Delivery Info | Test the PUT /api/delivery-info/{id} endpoint to ensure that it updates the delivery information successfully. | updateDeliveryInfo in DeliveryInfoController | **Input**: id = 1 (Delivery information ID to update) - Updated DeliveryInformation object. | **Expected Output**: - statusCode: 200 - deliveryInfo: The updated delivery information. | **Actual Output**: - HTTP status: 400 | Fail |  |
| TC006 | Update Delivery Info (Not Found) | Test the PUT /api/delivery-info/{id} endpoint to ensure it returns 404 if the delivery information with the given ID does not exist. | updateDeliveryInfo in DeliveryInfoController | **Input**: id = 999 (Non-existing delivery information ID) - Updated DeliveryInformation object. | **Expected Output**: - statusCode: 404 - deliveryInfo: null (Not Found). | **Actual Output**: - HTTP status: 404 - No delivery information found for ID 999. | Pass |  |

**5.5 Test case Details for “Product"**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Unit Under Test** | **Input Data** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Notes** |
| TC001 | Add Product | Test the POST /api/products/add-product endpoint to ensure that it creates a new product successfully. | createProduct in ProductController | **Input**: Product object with valid data | **Expected Output**: - statusCode: 200 - product: Newly created product. | **Actual Output**: - HTTP status: 200 - Newly created product | Pass |  |
| TC002 | Get All Products | Test the GET /api/products endpoint to ensure that it returns a list of all products. | getAllProducts in ProductController | **Input**: None | **Expected Output**: - statusCode: 200 - products: List of all products. | **Actual Output**: - HTTP status: 200 - List of all products | Pass |  |
| TC003 | Get Product by ID | Test the GET /api/products/{id} endpoint to ensure it returns the correct product for a given ID. | getProduct in ProductController | **Input**: id = 1 (Product ID) | **Expected Output**: - statusCode: 200 - product: Product with the given ID. | **Actual Output**: - HTTP status: 200 - Product with ID 1. | Pass |  |
| TC004 | Get Product by Title | Test the GET /api/products/search/{title} endpoint to ensure it returns the products that contain the given title. | getProductByTitleContaining in ProductController | **Input**: title = "Product Title" (Search for products containing this title) | **Expected Output**: - statusCode: 200 - products: List of products containing the title. | **Actual Output**: - HTTP status: 200 - List of products containing the title. | Pass |  |
| TC005 | Get Products by Category | Test the GET /api/products/category/{category} endpoint to ensure it returns products for a specific category. | getProductsByCategory in ProductController | **Input**: category = "Book" (Product category) | **Expected Output**: - statusCode: 200 - products: List of products in the category. | **Actual Output**: - HTTP status: 200 - List of products in category "Book". | Pass |  |
| TC006 | Get All Categories | Test the GET /api/products/categories endpoint to ensure it returns all available product categories. | getAllCategories in ProductController | **Input**: None | **Expected Output**: - statusCode: 200 - categories: List of all categories. | **Actual Output**: - HTTP status: 200 - List of all product categories. | Pass |  |