Test Plan - Unit Test

1. Introduction

1.1. Objective

This Test Plan outlines the strategy and scope for unit testing within the project. It is intended for software developers, QA engineers, and project stakeholders involved in the development and quality assurance processes. The document specifically addresses unit testing activities and does not serve as the main test plan for the entire project. Instead, it focuses on the early-stage verification of individual software components—such as functions, methods, or classes—ensuring that each unit performs as expected in isolation.

The primary objectives of this unit testing effort are to:

- Detect and resolve defects at an early development stage.
- Validate the correctness of each module independently of others.
- Establish a reliable foundation for subsequent testing phases, including integration and system testing.

Security and privacy considerations include ensuring that no sensitive data is hardcoded or exposed during unit test execution and that all test data used complies with relevant data protection standards.

1.2. Scope

The software product to be developed is **AIMS (An Internet Media Store)**, a digital platform that enables users to browse, purchase, and download a wide range of media content, including applications, music, videos, and e-books. AIMS serves as a centralized marketplace designed to deliver a seamless and secure media consumption experience across web and mobile platforms.

AIMS will allow users to create accounts, search and filter media content, manage their purchases, and access their media library anytime. The platform will also support content uploads and management for verified publishers. Its primary goals include enhancing media accessibility, streamlining digital content distribution, and supporting user-friendly interactions.

AIMS will not handle content creation or moderation beyond basic automated checks; these responsibilities lie with the content providers and external policies. This overview is consistent with the system’s Software Requirements Specification (SRS) and outlines the scope, benefits, and intended use of the application without detailing individual requirements.

1.3. Glossary

No	***Term***	***Explanation***	***Example***	***Note***
1	token	A piece of data created by server, and contains the user's information, as well as a special token code that user can pass to the server with every method that supports authentication, instead of passing a username and password directly.	JSON Web Token (JWT)	Compact, URL-safe and usable especially in web browser single sign-on (SSO) context.
2	API (Application Programming Interface)	A set of rules that allows different software applications to communicate with each order	VNPay API	Includes online shopping, digital payments, and logistics
3	E-Commerce (Electronic Commerce)	The buying and selling of goods and services over the internet	Amazon, Shopee	Includes online shopping, digital payment, and logistics
4	VAT (Value Added Tax	Value-added tax, a consumption tax added to goods and services.	10% VAT in Vietnam	Applied at each stage of production and distribution

5	Rush Order	An order that requires expedited processing and delivery.	Express shipping, Same-day delivery.	May involve additional fees for faster service.
6	Payment Gateway	A service that authorizes and processes online payments securely.	VNPay	Ensures secure transactions between customers and merchants.

1.4. Reference

Centers for Medicare & Medicaid Services. (n.d.). Test Case Specification. Retrieved from Centers for Medicare & Medicaid Services: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/TestCaseSpecification.docx

Centers for Medicare & Medicaid Services. (n.d.). Test Plan. Retrieved from Centers for Medicare & Medicaid Services: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/TestPlan.docx

guru99. (n.d.). Download Sample Test Case Template: Example Excel, Word Formats. Retrieved from guru99: https://www.guru99.com/download-sample-test-case-template-with-explanation-of-important-fields.html

2. Overall Description

2.1. General Overview

2.1.1. System Context and Design Philosophy

The design of AIMS (An Internet Media Store) centers around delivering a robust and scalable e-commerce platform for physical media products, tailored to meet both business requirements and user expectations. At its core, the system is built to be reliable, responsive, and easy to use — even under demanding conditions. Key design objectives include:

- **High Performance**: The system is expected to handle user requests swiftly, ensuring responsive interactions even during peak traffic periods.
- **Scalability**: AIMS aims to support up to 1,000 concurrent users, providing a seamless experience regardless of user volume.
- **Reliability**: The system should remain operational for at least 300 hours continuously and be able to recover within 1 hour following any downtime.
- **Security**: Strong security practices are integrated, including encrypted data storage and secure user authentication, to protect user privacy and transaction safety.

2.1.2. System and Software Architecture

- **Client Application**: A modern, web-based interface built using technologies like React, HTML, CSS, and JavaScript. It enables users to browse products, manage shopping carts, and complete purchases with ease.
- **Server Application**: The server-side logic is implemented using the SpringBoot framework, which handles business operations, database management, and integration with third-party services.
- **Database**: A relational database serves as the backbone for storing user accounts, product listings, order details, and transaction history.
- **External Integration**: The system integrates with **VNPay**, a third-party payment gateway, to securely process credit card transactions.

2.2. Assumptions, Constraints, and Risks

2.2.1. Assumptions

- **Platform**: Users will interact with AIMS via up-to-date web browsers such as Chrome, Firefox, Safari, or Edge.
- **User Capabilities**: It is assumed that users have basic computer literacy and stable internet access.
- **Third-party Services**: VNPay's APIs are assumed to be stable and consistently available.

- **Scope**: The system will exclusively handle physical media items; digital goods and subscription services are not included.

2.2.2. Constraints

- **Hardware**: The application must operate smoothly on standard web-enabled devices, without requiring high-performance hardware.

- **User Interface**: The UI should be intuitive enough for users with minimal technical background to navigate and complete tasks.

- **Resource Usage**: Efficient resource management is essential to ensure stable performance under varying loads.

- **Performance Metrics**: Response times must not exceed 2 seconds under normal load, and should remain under 5 seconds during peak usage.

- **Network Reliability**: Frontend and backend communications must be consistent and low-latency to ensure real-time feedback and reliability.

2.2.3. Risks

Risk	Description	Mitigation Strategy
Downtime	System outages may affect the shopping experience.	Employ redundancy and automatic failover systems.
Security Vulnerabilities	Risks of data breaches and unauthorized access.	Perform regular security audits, updates, and enforce best practices.
Scalability Limitations	Inability to support increasing traffic.	Optimize queries and backend logic, and implement load balancing.
Payment Integration Issues	Failures in VNPay integration could block payments.	Conduct continuous integration testing and maintain communication with VNPay support.
Performance Bottlenecks	Slow responses under heavy usage.	Conduct load testing, enable caching, and optimize backend processes.
Data Loss	Potential data corruption or loss due to bugs or hardware failure.	Schedule regular backups and implement integrity checks.

3. Testing Approach/Strategy

3.1. Overview

To ensure high reliability and maintainability of the AIMS software, we adopt a comprehensive unit testing strategy grounded in modern software engineering practices:

- **Test-Driven Development (TDD):**
  Developers are expected to write unit test cases *before* implementing functionalities. This helps clarify the expected behavior and guides clean, modular design.
- **Behavior-Driven Testing (BDT):**
  Each unit is tested according to its *expected behavior* as described in the AIMS requirement document. For example, when a product manager updates a price, the system must validate that it stays within 30–150% of the product's value.
- **Automation First:**
  All unit tests are to be integrated into an automated test suite, enabling continuous execution through CI/CD pipelines.
- **Mocking and Isolation:**
  To isolate each unit under test:
  - External systems like **VNPay**, databases, and email servers will be replaced with **mocks** or **stubs**.
  - This allows testing of units like payment validation or order confirmation without real transactions.

Example:
For a `login()` function:

- Test cases should validate:
  a) correct credentials → success
  b) incorrect credentials → failure
  c) locked accounts → proper error message
  → **Database access must be mocked** so that the test behavior is predictable and isolated.

3.2. Scope of Unit Testing

In-Scope

- Core business logic:
  - Product price constraints (30%–150%)
  - Product type-specific data validation (books, CDs, DVDs, etc.)
  - Delivery fee calculation (standard vs. rush order)
  - VAT and order total calculations
  - Cart management and update operations
  - Inventory checks and insufficient stock warnings

- Utility functions:
    - Email format validation
    - Weight-based shipping fee calculation
    - VAT application and rounding
- Error Handling and Edge Cases:
    - Null/invalid inputs
    - Maximum allowed product operations (e.g., delete ≤ 30 products)

Out-of-Scope

- Integration with:
    - **VNPay Sandbox** (use mock endpoints)
    - **Actual databases** (use stubs/fake repositories at the moment)
    - **Email servers**
    - **User Interface**

3.3. Strategy for Designing Unit Tests

Approach

We use a hybrid of **Black-Box Testing** and **White-Box Testing**, tailored to the type of function or module.

A. Black-Box Testing

Used for testing behavior against expected outputs (especially for business logic modules).

Technique	Use Case	Example Unit
Equivalence Partitioning	Input classification	Book, DVD, CD
Boundary Value Analysis	Edge conditions	Price limits at 30% and 150% of product value
Decision Table Testing	Complex rules	Delivery fee with standard vs. rush orders, VAT rules, free shipping conditions

B. White-Box Testing

Used for internal structure-based testing of algorithms and control flow-heavy units.

Technique	Use Case	Example Unit
Statement Coverage	Ensure all lines are executed	Email validation, VAT application
Branch Coverage	All logical branches tested	Conditional pricing and validation logic
Path Coverage	Full control path traversal	Delivery flow with/without rush delivery

3.4. Tools and Frameworks

To facilitate effective and automated unit testing, the following tools will be employed depending on the language and platform:

Tool	Description	Use
JUnit	Java unit testing framework	Main testing tool for Java-based AIMS implementation
Mockito	Mocking framework for Java	Mocking database access, VNPay integration

4. Unit Testing Summary

4.1. Traceability from Test Cases to Use Cases

USE CASE ID			UC 001	UC 002	UC003	UC 004	UC 005	UC 006	UC 007	UC 008	UC 009	UC 010	UC 011
Test Case ID	Test Case Title	Totals	17	5	5	4	6		24		7	9	
UT 001	testCreateProductSuccess (3 cases: Book, CD,	1	x										

	DVD)												
UT 002	testCreateProductInvalidCategory	1	x										
UT 003	testCreateProductStrategyThrowsException	1	x										
UT 004	testCreateProductStrategyThrowsException	1	x										
UT 005	testCreateBookWithMissingRequiredFields	1	x										
UT 006	testCreateProductWithDuplicateId	1	x										
UT 007	testUpdateProductSuccess (3 cases: Book, CD, DVD)	1	x										
UT 008	testUpdateProductNotFound	1	x										
UT 009	testUpdateProductInvalidCategory	1	x										
UT 010	testUpdateProductStrategyError	1	x										
UT 011	testUpdateProductWithNullInput	1	x										
UT 012	testUpdateBookWithMissingRequiredFields	1	x										

UT 013	testUpdateProductWithDifferentCategory	1	x										
UT 014	testUpdateProductWithDifferentId	1	x										
UT 015	testDeleteProductSuccess (3 cases: Book, CD, DVD)	1	x										
UT 016	testDeleteProductNotFound	1	x										
UT 017	testDeleteProductInUse	1	x										
UT 018	Pay Order- Successful get Payment URL	1		x									
UT 019	PayOrder - getPaymentURL Error Order Not Found Exception	1		x									
UT 020	PayOrder - Unsuccessful Payment, Order Not in "PENDING" State	1		x									
UT 021	PayOrder - ProcessPayment - Success Case	1		X									
UT 022	PayOrder - ProcessPayment - Declined Case	1		x									
UT 023	ProcessPayment - Error	1		x									
UT	PayOrder	1		x									

024	ProcessPayment - Order Not Found												
UT 025	GetPaymentHistory - Success	1		x									
UT 026	GetOrderInfo_Success												
UT 027	Get Order Products - Success												
UT 025	Rush Order - SuccessfulRushOrder (both address and product support)	1			x								
UT 024	Rush Order - Some Products Eligible	1			x								
UT 025	Rush Order - Address Not Supported	1			x								
UT 026	Rush Order - No Eligible Products	1			x								
UT 027	Get Product - Not Found	1				x							
UT 028	Add Product to Cart - Success	1					x						
UT 029	Add Product to Cart - Invalid Quantity	1					x						

UT 030	Update Cart Item - Success	1					x						
UT 031	Update Cart Item - Insufficient Stock	1					x						
UT 032	Remove Cart Item - Success	1					x						
UT 033	Remove Cart Item - Not Found	1					x						
UT 034	Place Order - Null Request DTO	1						x					
UT 035	Place Order - Null Delivery Info	1						x					
UT 036	Place Order - Null Cart Items	1						x					
UT 037	Place Order - Successful Order Creation	1						x					
UT 038	Place Order - Product Not Found	1						x					
UT 039	Place Order - Database Error on Save	1						x					
UT 040	Rush Order Check Request - NULL data	1			x								
UT 041	Rush Order Check	1			x								

	Request - valid data												
UT 042	testFindAll _Success	1							x				
UT 043	testFindAll _WithDesc Sort	1							x				
UT 044	testFindAll _WithInvali dSort	1							x				
UT 045	testFindById_Success	1							x				
UT 046	testFindById_UserNot Found	1							x				
UT 047	testSave_S uccess	1							x				
UT 048	testSave_ WithNullSt atus	1							x				
UT 049	testUpdate _Success	1							x				
UT 050	testUpdate _UserNotF ound	1							x				
UT 051	testUpdate _EmailServ iceExceptio n	1							x				
UT 052	testChange Password_ Success	1							x				
UT 053	testChange Password_ Password Mismatch	1							x				
UT 054	testChange Password_ UserNotFo und	1							x				
UT 055	testChange Password_ EmailServi ceExceptio n	1							x				
UT 056	testBlock_ Success	1							x				
UT 057	testBlock_ UserNotFo und	1							x				
UT 058	testDelete_ Success	1							x				

UT 059	testDelete_UserNotFound	1							x				
UT 060	testDelete_EmailServiceException	1							x				
UT 061	testFindAll_EmptyResult	1							x				
UT 062	void testFindAll_WithNullKeyword								x				
UT 063	testFindAll_WithNullSort								x				
UT 064	testFindById_WithNullUserStatus								x				
UT 065	testFindById_WithNullUserType								x				
UT 066	testSearchProducts_ReturnsMatchingProducts	1									x		
UT 067	testSearchProducts_NoMatchingProducts_ReturnsEmptyList	1									x		
UT 068	testSearchProducts_WithPagination_ReturnsPagedResults	1									x		
UT 069	testSearchProducts_MixedProductTypes_ReturnsCorrectDTOs	1									x		
UT 070	testSearchProducts_CaseInsensitiveSearch	1									x		
UT 071	testGetProductsByCategory_Book_ReturnsOnlyBooks	1									x		
UT 072	testGetProductsByCat	1									x		

	egory_InvalidCategory_ThrowsException												
UT 073	Cancel Order - Success	1								x			
UT 074	Cancel Order - Order Not Found	1								x			
UT0 75	Cancel Order - Already Cancelled	1								x			
UT0 76	Cancel Order - Approved Order	1								x			
UT0 77	Cancel Order - Rejected Order	1								x			
UT0 78	Cancel Order - Transaction Not Found	1								X			
UT0 79	Cancel Order - Refund Fails	1								x			
UT0 80	Cancel Order - Invalid Status	1								x			
UT 073	testGetBookFound	1				x							
UT 074	testGetCDFound	1				x							
UT 075	testGetDVDFound	1				x							

UT076	approveOrder_ValidPendingOrder_ShouldSucceed	1										x	
UT077	approveOrder_MissingDeliveryInfo_ShouldFail	1										x	
UT078	approveOrder_NonPendingOrder_ShouldFail	1										x	
UT079	approveOrder_NonPendingOrder_ShouldFail	1										x	
UT080	approveOrder_OrderNotFound_ShouldFail	1										x	
UT081	rejectOrder_ValidPendingOrder_ShouldSucceed	1										x	
UT082	rejectOrder_MissingReason_ShouldFail	1										x	
UT083	rejectOrder_NonPendingOrder_ShouldFail	1										x	
UT084	rejectOrder_OrderNotFound_ShouldFail	1										x	
UT093	Authenticate User - Success	1											x
UT094	Authenticate User - Failure (Bad Credentials)	1											x

4.2. Test Suite for UC001-"AddUpdateProductToStore"

Test Suite ID	Test Suite Title	Description	Test Cases
US001	Test Add Product To Store	Test the createProduct() method to ensure the right type of product (book, CD, DVD) can be added to the warehouse, validate product information, and handle errors when the product data is invalid.	UT001, UT002, UT003, UT004, UT005, UT006
US002	Test Update Product To Store	Test the updateProduct() method to ensure the correct update of existing product information by type, validate updated data, and handle errors when the update request is invalid.	UT007, UT008, UT009, UT0010, UT011, UT012, UT013, UT014
US003	Test Delete Product To Store	Test the deleteProduct() method to ensure products can be safely removed from the warehouse, check product existence before deletion, and handle errors when deletion conditions are not met.	UT015, UT016, UT017

4.3. Test Suite for UC002-"Pay Order"

Test Suite ID	Test Suite Title	Description	Test Cases
US004	Test Processing Payment	Test the ability processing a PaymentTransaction Object, and throw exception when needed.	UT021, UT022, UT023, UT024
US005	Test method get the PaymentTransaction by OrderID	Test getPaymentTransactionByOrderId method in PayOrderService	UT025
US006	Test method get payment url	Test getPaymentURL method in PayOrderService	UT018, UT019, UT020
US007	Test method Get Order Info	Test GetOrderInfo method in PayOrderService	UT026
US008	Test method Get Order Product	Test GetOrderProduct method in PayOrderService	UT027

4.4. Test Suite for UC003-"Place Rush Order"

Test Suite ID	Test Suite Title	Description	Test Cases
US007	PlaceRushOrderServiceTest	Verify rush order logic including eligibility, delivery area support in PlaceRushOrderService	UT024, UT025, UT026, UT040, UT041

4.5. Test Suite for UC004-"View Product Details"

Test Suite ID	Test Suite Title	Description	Test Cases
US008	ViewProductDetailsTest	Tests retrieving corresponding product details by ID and handling the case when the product is not found.	UT028, UT073, UT074, UT075

4.6. Test Suite for UC005-"Manage Cart"

Test Suite ID	Test Suite Title	Description	Test Cases
US009	Test Add Product to Cart	Tests adding a product to the cart, including normal cases and invalid quantities.	UT028, UT029
US010	Test Update Cart Item	Tests updating the quantity of a product in the cart and checks for stock constraints.	UT030, UT031
US0011	Test Remove Cart Item	Tests removing a product from the cart and handling non-existent cart items.	UT032, UT033

4.7. Test Suite for UC006-"Place Order"

Test Suite ID	Test Suite Title	Description	Test Cases
US0012	Test Placing an Order	Tests the entire order placement process, including successful creation, validation of input DTOs, and	UT034, UT035, UT036, UT037, UT038, UT039

		handling of exceptions like database errors or non-existent products.	

4.8. Test Suite for UC007-"CRUD_User"

Test Suite ID	Test Suite Title	Description	Test Cases
US013	Test get user list	Verifies the retrieval of paginated user lists, covering both populated and empty results.	From UT042 to UT044
US014	Test find user	Validates finding user by ID, including valid ID, invalid ID.	From UT045 to UT046
US015	Test save user	Tests user creation with valid input, missing fields, and duplicate usernames.	From UT047 to UT048
US016	Test update user	Checks user update logic with correct data, invalid user ID, and incorrect formats.	From UT049 to UT051
US017	Test change password	Ensures password change works correctly, and handles mismatches and missing users.	From UT052 to UT055
US018	Test block user	Confirms users can be soft-deleted, and handles cases like non-existent user ID.	From UT056 to UT057
US019	Test delete user	Validates hard deletion of a user and handles errors such as user not found.	From UT058 to UT060
US020	Test find user with null value input	Tests behavior when null or invalid values are used in user lookup operations.	From UT061 to UT065

4.9. Test Suite for UC008-"Cancel Order"

Test Suite ID	Test Suite Title	Description	Test Cases
US021	Test Cancel Order		From UT073 to UT080

4.10.Test Suite for UC009-”Search products”

Test Suite ID	Test Suite Title	Description	Test Cases
US022	Test Product Search Functionality	Test to ensure it returns correct products based on keywords, category, supports pagination, handles case insensitivity, and returns proper DTOs.	From UT066 to UT072

4.11. Test Suite for UC010-”Approve/Reject Order”

Test Suite ID	Test Suite Title	Description	Test Cases
US023	Test Approve Order	Test the approveOrder() method to ensure pending orders can be approved by managers, validate order status and delivery information, and handle errors when approval conditions are not met.	UT076, UT077, UT078, UT079, UT080
US024	Test Reject Order	Test the rejectOrder() method to ensure pending orders can be rejected with valid reasons, validate rejection requirements, and handle errors when rejection conditions are not satisfied.	UT081, UT082, UT083, UT084

4.12. Test Suite for UC011-”Login”

Test Suite ID	Test Suite Title	Description	Test Cases
US025	Test User Authentication	Verifies the user authentication logic, including successful login with valid credentials and failure handling for invalid credentials.	UT085, UT086

5. Test Case Details

5.1. Test Case Details - UC001 “AddUpdateProductToStore”

+ Add product to store:

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT001	Add Book Product Successfully	Verify product is added successfully when valid book data is provided	ProductService.createProduct	Book DTO with `productID="P001", title="Java Basics", value=120000d, price=150000d, quantity=5, authors="John Doe", publisher="Tech Books", genre="Programming"`	DTO with `productID="P001", category="book", title="Java Basics"`	DTO with `productID="P001", category="book", title="Java Basics"`	Pass	Valid book creation
UT002	Add CD Product Successfully	Verify product is added successfully when valid CD data is provided	ProductService.createProduct	CD DTO with `productID="CD001", title="Greatest Hits", value=80000d, price=100000d, quantity=5, artist="Artist Name", musicType="Rock"`	DTO with `productID="CD001", category="cd", title="Greatest Hits"`	Match with expected	Pass	Valid CD creation
UT003	Add DVD Product Successfully	Verify product is added successfully when valid DVD data is provided	ProductService.createProduct	DVD DTO with `productID="DVD001", title="Movie Title", value=150000d,`	DTO with `productID="DVD001", category="dvd", title="Mo`	Match with expected	Pass	Valid DVD creation

				price=200000d, quantity=3, director="Director Name"	vie Title"			
UT004	Add Product - Invalid Category	Verify error when adding product with invalid category	ProductService.createProduct	Product DTO with category="invalid_type"	Throws BadRequestException: "Unsupported product type"	Throws BadRequestException	Pass	Negative test case
UT005	Add Product - Missing Fields	Verify error when adding product with missing required fields	ProductService.createProduct	Product DTO missing title and price	Throws BadRequestException: "Required fields missing"	Throws BadRequestException	Pass	Validation test
UT006	Add Product - Duplicate ID	Verify error when adding product with existing ID	ProductService.createProduct	Product DTO with existing ID="P001"	Throws RuntimeException: "Product ID already exists"	Throws RuntimeException	Pass	Duplicate check

+ Update Product to Store

-	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT007	Update Book Successfully	Verify book update with valid changes	ProductService.updateProduct	Updated Book DTO for P001 with new price	Updated DTO with new values	Match with expected	Pass	Valid update
UT008	Update CD Successfully	Verify CD update with valid changes	ProductService.updateProduct	Updated CD DTO for CD001 with new artist	Updated DTO with new values	Match with expected	Pass	Valid update

UT009	Update DVD Successfully	Verify DVD update with valid changes	ProductService.updateProduct	Updated DVD DTO for `DVD001` with new price	Updated DTO with new values	Match with expected	Pass	Valid update
UT010	Update – Product Not Found	Verify error for non-existent product	ProductService.updateProduct	Update request for invalid ID	Throws `ResourceNotFoundException`	Throws `ResourceNotFoundException`	Pass	Not found case
UT011	Update – Invalid Category	Verify error when changing category	ProductService.updateProduct	Book DTO with category changed to CD	Throws `BadRequestException`	Throws `BadRequestException`	Pass	Category change
UT012	Update – Missing Fields	Verify error with missing required fields	ProductService.updateProduct	Update DTO missing required fields	Throws `BadRequestException`	Throws `BadRequestException`	Pass	Validation test
UT013	Update – Invalid Price	Verify error with negative price	ProductService.updateProduct	Update DTO with `price=-100`	Throws `BadRequestException`	Throws `BadRequestException`	Pass	Price validation
UT014	Update – Invalid Quantity	Verify error with negative quantity	ProductService.updateProduct	Update DTO with `quantity=-5`	Throws `BadRequestException`	Throws `BadRequestException`	Pass	Quantity check

+ Delete Product to Store

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail
UT015	testDeleteProductSuccess (Book, CD, DVD)	Verify successful deletion of Book, CD, and DVD	ProductService.deleteProduct	Valid product ID for Book, CD, DVD	No exception, correct strategy called	Match with expected	Pass

UT016	testDeleteProductNotFound	Verify deletion with non-existent product	ProductService.deleteProduct	Non-existent product ID	Exception "Product not found" thrown	Match with expected	Pass
UT017	testDeleteProductInUse	Verify deletion when product is currently in use	ProductService.deleteProduct	Valid product ID, product is in use	Exception "cannot be deleted" thrown	Match with expected	Pass

5.2. Test Case Details - UC002 “Pay Order”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT018	Pay Order-Successful get Payment URL	This scenario tests the get URL payment when everything goes smoothly	getPaymentURL(String orderId, String paymentType) is called	status = PENDINGorderId =”ORD001”	–paymentUrl: http://pay.url	TBD	TBD	Valid input case
UT019	PayOrder - getPaymentURL Error Order Not Found Exception	This scenario tests behavior when the order ID is not found in database	getPaymentURL(String orderId, String paymentType)	orderId = "ORD404" orderRepository returns empty	Exception: `IllegalArgumentException` with message containing "Order not found"	TBD	TBD	Invalid orderId case
UT020	PayOrder - getPaymentURL Error Order Status Not Pending	This scenario tests behavior when the order is not in `PENDING` status	getPaymentURL(String orderId, String paymentType)	orderId = "ORD002" status = APPROVED order found in repository	Exception: `IllegalStateException` with message containing `"Order is not in PENDING"`	TBD	TBD	Invalid order status case
UT021	PayOrder - ProcessPayment - Success Case	Tests if successful payment updates	processPayment(Map params, String	params = {vnp_ResponseCode =	Redirect URL contains "payment-success" and	TBD	TBD	Valid success scenario

		transaction, saves order, and returns success redirect	type)	"00", vnp_TxnRef = "ORD001"} paymentType = "vnpay" order exists with status = PENDING transaction saved with ID = "TXN001"	"orderId=ORD001"			
UT022	PayOrder - ProcessPayment - Declined Case	Tests if the service returns decline URL when responseCode = 24	processPayment(Map params, String type)	params = {vnp_ResponseCode = "24"} paymentType = "vnpay""	Redirect URL contains "payment-decline"	TBD	TBD	Payment was cancelled
UT023	ProcessPayment - Error	Tests error mapping and URL redirection when unexpected error happens	processPayment(Map params, String type)	params = {vnp_ResponseCode = "99"} paymentType = "vnpay" Mock throws PaymentException	Redirect URL contains `"payment-error"`	TBD	TBD	Error scenario handling
UT024	ProcessPayment - Order Not Found	Tests if exception is thrown when order does not exist	processPayment(Map params, String type)	`params = {vnp_ResponseCode = "00", vnp_TxnRef = "ORD404"}` order not found in repository	Throws `IllegalArgumentException`	TBD	TBD	Invalid order reference

UT025	GetPayment History - Success	Gets a valid transaction and maps it to TransactionResponseDTO	getPaymentHistory(String orderId)	`orderId = "TXN001"` `transaction exists`	Returns TransactionResponseDTO with: transactionId = TXN001 amount = 100.0 paymentType = "vnpay" ...	TBD	TBD	Transaction found
UT026	GetOrderInfo _Success	Verify that the method returns correct order information when a valid order ID is provided.	PayOrderService.getOrderInfo(String orderId)	`orderId = "ORD001"`	`OrderInfoDTO` object with `orderID = "ORD001"`	TBD	TBD	`orderRepository.findByOrderID` and `orderMapper.toOrderInfoDTO` are mocked to simulate repository and mapper behavior.
UT027	Get Order Products - Success	Verify that the method returns a correct list of ordered products when a valid order ID is provided.	PayOrderService.getOrderProduct(String orderId)	`orderId = "ORD001"` `- Contains 1 product: Book(BOOK001, Java, 100.0, quantity=2)`	A list of 1 `OrderItemDTO` with: • `productID = "BOOK001"` • `title = "Java"` • `price = 100.0` • `quantity = 2`	A list of 1 `OrderItemDTO` with same expected values	Pass	Mocks used: `orderRepository`, `orderMapper`, and `orderItemRepository` to simulate data retrieval and mapping.

5.3. Test Case Details - UC003 “Place Rush Order”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass /Fail	Notes
UT024	Rush Order - Successful Order	Verify rush order when all products are eligible and address is	evaluateRushOrder(info, products)	province = "Ba Đình", product1.eligible = true,	isSupported = true, 1 rush product, 1 regular	TBD	TBD	Valid full rush case

		supported		product2.eligible = false	product			
UT025	Rush Order - Address Not Supported	Address outside inner city → rush order not supported	evaluateRushOrder(info, products)	province = "Ho Chi Minh", district = "District 1", product1.eligible = true	isSupported = false, 0 rush products, 1 regular product, prompt message shown	TBD	TBD	Address edge case
UT026	Rush Order - No Eligible Products	No products eligible → rush not supported even if address valid	evaluateRushOrder(info, products)	province = "Ba Đình", product1.eligible = false, product2.eligible = false	isSupported = false, 0 rush products, 2 regular products, prompt message shown	TBD	TBD	Product filter case
UT040	Rush Order Check Request - NULL data	Ensure DTO handles null delivery info and products list gracefully	RushOrderCheckRequest	deliveryInfo = null, products = null	deliveryInfo == null, products == null	TBD	TBD	DTO null handling check
UT041	Rush Order Check Request - Valid Data	Ensure DTO stores and returns valid delivery info and product list	RushOrderCheckRequest	deliveryInfo.city = "Hà Nội", product.productID = "B001", product.eligible = true	deliveryInfo and product list match input	TBD	TBD	DTO input/output check

5.4 Test Case Details - UC004 "View Products Details"

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/ Fail	Notes
UT028	Get Product - Not Found	Verify behavior when the product ID is missing	ProductService.getProductById()	productId = "nonexistent"	Throws `RuntimeException` with appropriate message	RuntimeException: Product not found with id: nonexistent	Pass	Not found case
UT073	testGetBookFound	Verify the correct data	ProductService.getProductById()	productId = 'BK-001"	Returns `BookDTO` containing the correct product info	`BookDTO` with productId = 'BK-001"	Pass	
UT074	testGetCDFound	Verify the correct data	ProductService.getProductById()	productId = 'CD-001"	Returns `CdDTO` containing the correct product info	`CdDTO` with productId = 'CD-001"	Pass	

UT075	testGetDVDFound	Verify the correct data	ProductService.getProductById()	productId = 'DVD-001"	Returns DvdDTO containing the correct product info	DvdDTO with productId = 'DVD-001"	Pass	

5.5 Test Case Details - UC005 "Manage Cart"

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass /Fail	Notes
UT028	Add Product to Cart - Success	Add a new product to the cart successfully	CartService.addToCart	customerId='customer123', productId='product456', quantity=2	CartItemDTO with correct details	CartItemDTO with 'product456', quantity=2	Pass	Valid input
UT029	Add Product to Cart - Invalid Quantity	Add product with quantity = 0 should throw BadRequestException	CartService.addToCart	customerId='customer123', productId='product456', quantity=0	Throws BadRequestException	BadRequestException with message 'Quantity must be greater than zero'	Pass	Invalid quantity
UT030	Update Cart Item - Success	Update cart item with valid quantity and product stock	CartService.updateCartItem	customerId='customer123', productId='product456', quantity=3	CartItemDTO updated with new quantity	CartItemDTO with quantity=3	Pass	Stock available
UT031	Update Cart Item - Insufficient Stock	Updating cart item with quantity exceeding stock throws exception	CartService.updateCartItem	customerId='customer123', productId='product456', quantity=15	Throws BadRequestException	BadRequestException with message 'Not enough stock available. Available: 10'	Pass	Exceeds stock
UT032	Remove Cart Item - Success	Remove an existing cart item	CartService.removeFromCart	customerId='customer123', productId='product456'	Cart item removed	Repository.deleteById called	Pass	Item exists

UT033	Remove Cart Item - Not Found	Try to remove a non-existing cart item should throw exception	CartService.removeFromCart	customerId='customer123', productId='product456'	Throws ResourceNotFoundException	ResourceNotFoundException with message 'Cart item not found'	Pass	Item not found

5.6 Test Case Details - UC006 “Place Order”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT034	Place Order - Null Request	This tests that the service correctly handles a completely null request.	placeOrder(OrderRequestDTO request)	request is null.	NullPointerException is thrown.	TBD	TBD	Null input validation.
UT035	Place Order - Null Delivery Info	This tests that the service requires delivery information to be present in the request.	placeOrder(OrderRequestDTO request)	request object where deliveryInfo is null.	NullPointerException is thrown.	TBD	TBD	Partial null input validation.
UT036	Place Order - Null Cart Items	This tests that the service requires a list of cart items to be present in the request.	placeOrder(OrderRequestDTO request)	request object where cartItems list is null.	NullPointerException is thrown.	TBD	TBD	Partial null input validation.

UT037	Place Order - Success Case	This scenario tests the successful creation of an order when all inputs are valid.	placeOrder(OrderRequestDTO request)	- Valid OrderRequestDTO with cartItems and deliveryInfo. - Mocks for repositories and mappers are configured for a successful flow.	A valid OrderDTO is returned with the correct ID and calculated total price.	TBD	TBD	"Happy path" scenario.
UT038	Place Order - Product Not Found	This tests the system's response when an item in the cart does not correspond to an existing product.	placeOrder(OrderRequestDTO request)	OrderRequestDTO containing a cartItem with a productID that is not found by the productRepository.	A RuntimeException (or a specific custom exception) is thrown, indicating the product was not found.	TBD	TBD	Business rule validation.
UT039	Place Order - Database Error on Save	This scenario tests the service's behavior when the database fails during an orderRepository.save() operation.	placeOrder(OrderRequestDTO request)	orderRepository.save() is mocked to throw a RuntimeException.	The RuntimeException from the repository is propagated.	TBD	TBD	Exception handling for system failures.

5.7 Test Case Details - UC007 “CRUD_User”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass /Fail	Notes
UT042	testFindAll_Success	Search users by keyword with ascending username sort	UserService.findAll()	`keyword = "test", sort = "username:asc"`	Returns list of users sorted by username (asc)	Users sorted ascending	Pass	
UT053	testFindAll_WithDescSort	Search all users with descending sort	UserService.findAll()	sort = "username:desc"	Returns all users sorted by username (desc)	Users sorted descending	Pass	
UT044	testFindAll_InvalidSortParam	Provide invalid sort string	UserService.findAll()	sort = "invalid_sort"	Returns unsorted list or default sort applied	Default sort	Pass	
UT045	testFindById_Success	Find user by valid ID	UserService.findById()	id = 1	User object with id=1 returned	Correct user objec	Pass	
UT046	testFindById_UserNotFound	Find user with non-existent ID	UserService.findById()	id = 999	Exception: "User not found	Exception thrown	Pass	
UT047	testSaveUser_Success	Save user with valid input	UserService.save()	`UserCreationRequest` with all valid fields	New user created, returns ID	New user ID returned	Pass	
UT048	testSaveUser_NullStatus	Save user with status = null	UserService.save()	UserCreationRequest.status = null	Status set to default (`NONE`)	Status = NONE	Pass	
UT049	testUpdate_Success	Verifies that a user is successfully updated and a notification email is sent.	UserService.update()	A valid `UserUpdateRequest` + user exists (`findById()` returns `Optional`	`save()` is called with updated user, `emailService.send()` is triggered	As expected	Pass	

				`.of(testUser)` + `save()` and `send()` mocked				
UT050	testUpdateUser_NotFound	Try updating non-existent user	UserService.update()	id = 404	Exception: "User not found"	Exception thrown	Pass	
UT051	testUpdateUser_EmailFailure	Email service fails after update	UserService.update()	Valid update request, email service throws error	Update succeeds, email failure ignored	Update success, log error	Pass	
UT052	testChangePassword_Success	Change password with correct confirm	UserService.changePassword()	`password = "1234", confirmPassword = "1234"`	Password changed, email sent	Password updated	Pass	
UT053	testChangePassword_PasswordMismatch	Verifies that the method throws an error when password and confirmPassword do not match.	UserService.changePassword()	UserPasswordRequest{id=1, password="abc", confirmPassword="differentpassword"}	Throws `RuntimeException` with message `"Password not match"`	As expected	Pass	
UT054	testChangePassword_UserNotFound	Ensures an error is thrown when the user with the given ID does not exist.	UserService.changePassword()	`UserPasswordRequest{id=999, password="abc", confirmPassword="abc"}` with mock: `userRepository.findById()` returns `Optional.empty()`	Throws `RuntimeException` with message `"User not found"`	As expected	Pass	

UT055	testChangePassword_EmailServiceException	Verifies that an exception thrown by the email service is caught and logged, not propagated.	UserService.changePassword()	Valid `UserPasswordRequest`, existing user returned by `userRepository`, `emailService.send()` throws `RuntimeException("Email service error")`	Password is updated successfully; exception from email is logged only	As expected	Pass	
UT056	testBlockUser_Success	Block an existing user	UserService.block()	id = 2	User status set to `BLOCKED`	Status = BLOCKED	Pass	
UT057	testBlockUser_NotFound	Block user that doesn't exist	UserService.block()	id = 100	Exception: "User not found"	Exception thrown	Pass	
UT058	testDeleteUser_Success	Delete a valid user	UserService.delete()	id = 3	User deleted, email sent	User deleted, email sent	Pass	
UT059	testDeleteUser_NotFound	Delete a non-existent user	UserService.delete()	id = 500	Exception: "User not found"	Exception thrown	Pass	
UT060	testDeleteUser_EmailFailure	Email fails during delete process	UserService.delete()	Valid delete, email service fails	User deleted, logs email error	Deleted, email failed	Pass	
UT061	testFindAll_EmptyResult	Verifies that the method returns an empty list when no users match the criteria.	UserService.findAll()	`keyword = "nonexistent", sort = "username:asc", page = 0,`	An empty `List<UserResponse>` is returned	As expected	Pass	

				`size = 20` + mock `userRepository.findAll()` returns empty `PageImpl`				
UT062	testFindAll_NullKeyword	Search with null keyword (should return all users)	UserService.findAll()	keyword = null	All users returned	Full user list	Pass	
UT063	testFindAll_NullSort	Search with null sort	UserService.findAll()	sort = null	Default sort applied	Default order	Pass	
UT064	testFindById_NullStatus	User found with null status	UserService.findById()	User object with `status = null`	Returns user with `status = null`	Null status returned	Pass	
UT065	testFindById_NullType	User found with null type	UserService.findById()	User object with `type = null`	Returns user with `type = null`	Null type returned	Pass	

5.8. Test Case Details - UC008 “Cancel Order”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass /Fail	Notes
UT073	Cancel Order - Success	Cancels a PENDING order and processes refund successfully	cancelOrder(String orderId, String txnId, String pt)	`order.status = PENDING`, `valid transaction`, `refund returns string`	"Refund processed successfully", order.status = CANCELLED	TBD	TBD	Valid scenario with successful refund

UT074	Cancel Order - Order Not Found	Throws exception when the order doesn't exist	cancelOrder(String orderId, String txnId, String pt)	order not found	RuntimeException("Order not found")	TBD	TBD	Invalid order reference
UT075	Cancel Order - Already Cancelled	Returns early when the order is already CANCELLED	cancelOrder(String orderId, String txnId, String pt)	order.status = CANCELLED	"Order is already cancelled"	TBD	TBD	Should not trigger refund or update
UT076	Cancel Order - Approved Order	Cannot cancel an APPROVED order	cancelOrder(String orderId, String txnId, String pt)	order.status = APPROVED	"Order cannot be cancelled after approval or rejection"	TBD	TBD	Cancellation after approval disallowed
UT077	Cancel Order - Rejected Order	Cannot cancel a REJECTED order	cancelOrder(String orderId, String txnId, String pt)	order.status = REJECTED	"Order cannot be cancelled after approval or rejection"	TBD	TBD	Cancellation after rejection disallowed
UT78	Cancel Order - Transaction Not Found	Throws exception if the payment transaction cannot be found	cancelOrder(String orderId, String txnId, String pt)	valid order.status = PENDING, but transaction not found	RuntimeException("Payment transaction not found")	TBD	TBD	Missing transaction error
UT079	Cancel Order - Refund Fails	Returns error if refund info from gateway is null	cancelOrder(String orderId, String txnId, String pt)	valid order and transaction, but `getRefundInfo(...)` returns `null`	"Failed to process refund", order.status = CANCELLED	TBD	TBD	Refund failure from gateway
UT080	Cancel Order - Invalid Status	Handles case where order status is invalid (null or unexpected)	cancelOrder(String orderId, String txnId, String pt)	order.status = null	"Order cannot be cancelled at this stage"	TBD	TBD	Defensive programming for bad status

5.9. Test Case Details - UC009 “Search products”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass /Fail	Notes
UT066	testSearchProducts_ReturnsMatchingProducts	Search products by keyword, expect matching products	ProductServiceImpl.searchProducts()	keyword = "test"	List of products with "test" in title/desc	List of matching products	Pass	Covers Book & CD
UT067	testSearchProducts_NoMatchingProducts_ReturnsEmptyList	Search products with a keyword that matches nothing	ProductServiceImpl.searchProducts()	keyword = "nonexistent"	Empty list	Empty list	Pass	
UT068	testSearchProducts_WithPagination_ReturnsPagedResults	Search products with pagination	ProductServiceImpl.searchProducts()	keyword = "test", page = 0, size = 10	Page object with products, correct page info	Page with 2 products	Pass	Tests pagination logic
UT069	testSearchProducts_MixedProductTypes_ReturnsCorrectDTOs	Search products, expect multiple types (Book, DVD, etc.)	ProductServiceImpl.searchProducts()	keyword = "programming"	List of ProductDTOs of different types	BookDTO, DvdDTO	Pass	Checks type mapping
UT070	testSearchProducts_CaseInsensitiveSearch	Search is case-insensitive	ProductServiceImpl.searchProducts()	keyword = "TEST"	List of products matching "test" (case-insensitive)	List with "Test Book"	Pass	
UT071	testGetProductsByCategory_Book_ReturnsOnlyBooks	Get products by category "book"	ProductServiceImpl.getProductsByCategory()	category = "book"	List of BookDTOs only	List of BookDTOs	Pass	

UT072	testGetProducts ByCategory_InvalidCategory_ThrowsException	Invalid category should throw exception	ProductServiceImpl.getProductsByCategory()	category = "invalid"	Exception thrown with message "Unsupported product type"	Exception thrown	Pass	

5.10. Test Case Detail - UC010 “Approve/Reject Order”

+ Approve Order:

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT076	Approve Order Successfully	Verify successful order approval	OrderService.approveOrder	Valid order ID and approver	Success response, `status=APPROVED`	Match with expected	Pass	Valid approval
UT077	Approve – Missing Delivery	Verify error without delivery info	OrderService.approveOrder	Order without delivery info	Error response: `"Missing delivery info"`	Match with expected	Pass	Validation test
UT078	Approve – Wrong Status	Verify error for non-pending order	OrderService.approveOrder	Order in `APPROVED` status	Error: `"Invalid status"`	Match with expected	Pass	Status check
UT079	Approve – Order Not Found	Verify error for invalid order	OrderService.approveOrder	Non-existent order ID	Error: `"Order not found"`	Match with expected	Pass	Not found case
UT080	Approve – Invalid Approver	Verify error for invalid approver	OrderService.approveOrder	Invalid approver details	Error: `"Invalid approver"`	Match with expected	Pass	Approver check

+ Reject Order

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT081	Reject Order Successfully	Verify successful order rejection	OrderService.rejectOrder	Valid order ID and reason	Success response, `status=REJECTED`	Match with expected	Pass	Valid rejection
UT082	Reject – Missing Reason	Verify error without reason	OrderService.rejectOrder	Rejection without reason	Error: `"Reason required"`	Match with expected	Pass	Validation test
UT083	Reject – Wrong Status	Verify error for non-pending order	OrderService.rejectOrder	Order in `REJECTED` status	Error: `"Invalid status"`	Match with expected	Pass	Status check
UT084	Reject – Order Not Found	Verify error for invalid order	OrderService.rejectOrder	Non-existent order ID	Error: `"Order not found"`	Match with expected	Pass	Not found case

5.1. Test Case Details - UC011 “Place Order”

Test Case ID	Test Case Name	Description	Unit Under Test	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
UT093	Authenticate User - Success	This scenario tests the "happy path" where a user provides valid credentials and receives a JWT token.	authenticateUser(LoginRequestDTO)	-LoginRequestDTO with valid username and passwor. -AuthenticationManager is mocked to return a valid Authentication object. - JwtUtils is mocked to return a	A valid JwtResponseDTO is returned, containing the user's ID, username, email, roles, and the JWT token.	TBD	TBD	Valid credentials case.

				fake JWT string.				
UT094	Authenticate User - Failure (Bad Credentials)	This scenario tests the system's response when the AuthenticationManager rejects the user's credentials.	authenticateUser(LoginRequestDTO)	-LoginRequestDTO with an invalid username/password.-AuthenticationManager is mocked to throw a RuntimeException.	A RuntimeException is thrown, indicating authentication failure.	TBD	TBD	Invalid credentials case.