CartServiceTest Documentation

Overview

This document details the unit tests implemented for the CartService class in our e-commerce application backend. The CartService manages the shopping cart functionality, allowing users to add products to their cart, update quantities, and clear the cart.

Test Class Structure

Dependencies

The test class uses:

- **JUnit 5**: For test execution and assertions
- **Mockito**: To mock dependencies and simulate interactions
- **Spring Boot Test**: For integration with the Spring testing framework

Mocked Components

- CartRepository: Database access for cart operations
- ProductRepository: Database access for product operations

Test Setup

Before each test, the following setup is performed:

1. Initialize mocks using MockitoAnnotations
2. Create a random UUID for the user

3. Prepare test data:
    - A test product with a stock count of 10 and price of $99.99
    - An empty test cart associated with the user

Test Cases

1. testAddToCart_NewItem_Successful

Purpose: Verify that a new product can be successfully added to an existing cart.

Test Scenario:

- **Arrange:**
    - Configure mocks to return the test product when findById is called
    - Configure mocks to return the test cart when findByUserId is called
    - Setup mock behavior for save operations
- **Act:**
    - Call cartService.addToCart with the userId and productId
- **Assert:**
    - Verify the response contains a success message with 200 status code
    - Verify the cart now contains 1 item with the correct productId
    - Verify the item quantity is set to 1
    - Verify the product stock is reduced by 1 (from 10 to 9)
    - Verify both the product and cart were saved to the database

Business Logic Verified:

- Products are correctly added to the cart
- Product stock is decreased appropriately
- The system returns appropriate success responses

2. testAddToCart_ExistingItem_IncreasesQuantity

Purpose: Verify that adding a product already in the cart increases its quantity instead of creating a duplicate entry.

Test Scenario:

- **Arrange:**
    - Pre-populate the cart with 1 unit of the test product
    - Configure mocks as in the previous test
- **Act:**
    - Call cartService.addToCart with the same userId and productId
- **Assert:**
    - Verify the success response
    - Verify the cart still has only 1 item (no duplicates)
    - Verify the quantity increased to 2
    - Verify stock was reduced again
    - Verify database save operations

Business Logic Verified:

- The system properly tracks quantities rather than creating duplicate entries
- Product stock reduction works consistently

3. testAddToCart_CreateNewCartIfNotExists

Purpose: Verify the system creates a new cart for a user who doesn't have one yet.

Test Scenario:

- **Arrange:**
    - Configure productRepository mock to return the test product

    - Configure cartRepository to return empty (indicating no cart exists)
    - Setup save mocks
- **Act:**
    - Call cartService.addToCart
- **Assert:**
    - Verify success response
    - Verify cart creation by confirming save was called with any Cart object
    - Verify product stock reduction and save

Business Logic Verified:

- New users automatically get a cart created when adding their first product
- No errors occur when a user without a cart attempts to add products

4. testDeleteProductsInCart_Successful

Purpose: Verify the cart can be cleared of all products.

Test Scenario:

- **Arrange:**
    - Pre-populate cart with a test item (quantity of 2)
    - Configure cartRepository mocks
- **Act:**
    - Call cartService.deleteProductsInCart for the user
- **Assert:**
    - Verify success response
    - Verify cart items list is now empty
    - Verify the cart was saved to persist the cleared state

Business Logic Verified:

- Users can empty their entire cart in one operation
- The system correctly maintains the empty cart state

Mocking Strategy

The tests use a consistent mocking strategy to isolate the CartService from its dependencies:

1. **Mock Responses:** Return prepared test objects when repository methods are called
2. **Behavior Verification:** Verify that the service calls repository methods as expected
3. **State Verification:** Check that objects are modified correctly before being saved

Test Coverage

These tests cover the core functionality of the CartService:

- Adding new products to a cart
- Increasing quantity of existing products
- Creating a new cart if needed
- Clearing a cart completely

Conclusion

The CartServiceTest thoroughly verifies the core functionality of the shopping cart system. The tests ensure that products can be added to carts, quantities are tracked correctly, and carts can be emptied. These tests help maintain the integrity of the shopping cart system as the application evolves.