OrderServiceTest Documentation

Overview

This document details the unit tests implemented for the OrderService class in our e-commerce application backend. The OrderService manages the creation and status updates of orders, handling the transition from shopping cart to completed order and tracking the order through its delivery lifecycle.

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

- OrderRepository: Database access for order operations
- CartRepository: Database access for cart operations
- UserRepository: Database access for user operations
- ProductRepository: Database access for product operations
- OrderHistoryRepository: Database access for order history operations

Test Setup

Before each test, the following setup is performed:

1. Initialize mocks using MockitoAnnotations
2. Generate a random UUID for userId
3. Create test objects:
    - A test cart with multiple items (two cart items with different quantities)
    - A test order in "Processing" status that references the test cart

Test Cases

1. testCreateOrderFromCart_Successful

Purpose: Verify that an order can be successfully created from a user's cart.

Test Scenario:

- **Arrange:**
    - Configure cartRepository mock to return the test cart when findById is called
    - Configure orderRepository save method to return the test order
- **Act:**
    - Call orderService.createOrderFromCart with the cartId and userId
- **Assert:**
    - Verify the returned orderId is not null
    - Verify the orderRepository save method was called once

Business Logic Verified:

- Orders are created correctly from user carts
- Order creation process preserves necessary information (userId, cartId)
- The system returns a valid order ID reference

2. testMarkAsShipped_Successful

Purpose: Verify that an order can be marked as shipped, updating its status appropriately.

Test Scenario:

- **Arrange:**
    - Configure orderRepository findById method to return the test order
    - Configure orderRepository save method to return the updated order
- **Act:**
    - Call orderService.markAsShipped with the order ID
- **Assert:**
    - Verify the response message is "Order marked as shipped."
    - Verify the response has 200 status code
    - Verify the order's shipped flag is set to true
    - Verify the order's status is updated to "Delivered"
    - Verify the orderRepository save method was called once with the updated order

Business Logic Verified:

- Orders can be marked as shipped
- Order status and shipped flag are updated correctly
- The system returns appropriate success messages
- Changes are persisted to the database

3. testMarkAsInTransit_Successful

Purpose: Verify that an order can be marked as in-transit, updating its status appropriately.

Test Scenario:

- **Arrange:**

    - Configure orderRepository findById method to return the test order
    - Configure orderRepository save method to return the updated order
- **Act:**
    - Call orderService.markAsInTransit with the order ID
- **Assert:**
    - Verify the response message is "Order marked as in-transit."
    - Verify the response has 200 status code
    - Verify the order's status is updated to "In-Transit"
    - Verify the orderRepository save method was called once with the updated order

Business Logic Verified:

- Orders can be marked as in-transit
- Order status is updated correctly
- The system returns appropriate success messages
- Changes are persisted to the database

4. testGetOrdersByUser_ReturnsUserOrders

Purpose: Verify that all orders for a specific user can be retrieved.

Test Scenario:

- **Arrange:**
    - Create a list containing the test order
    - Configure orderRepository findByUserId method to return this list
- **Act:**
    - Call orderService.getOrdersByUser with the userId
- **Assert:**
    - Verify the returned list matches the expected orders

    - Verify the orderRepository findByUserId method was called once with the correct userId

Business Logic Verified:

- The system can retrieve all orders for a specific user
- The returned data structure matches the expected format
- Query operations use the correct parameters

Mocking Strategy

The tests use a consistent mocking strategy to isolate the OrderService 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 OrderService:

- Creating new orders from user carts
- Updating order status to in-transit
- Marking orders as shipped
- Retrieving orders for a specific user

Conclusion

The OrderServiceTest thoroughly verifies the core functionality of the order management system. The tests ensure that orders can be created from carts, their statuses can be updated throughout the shipping process, and users can retrieve their order information.

These tests help maintain the integrity of the order processing system as the application evolves, ensuring reliable order management for customers and administrators.