ReviewServiceTest Documentation

Overview

This document details the unit tests implemented for the ReviewService class in our e-commerce application backend. The ReviewService manages product reviews, allowing users to submit feedback on products they've purchased, administrators to approve reviews, and customers to view verified reviews.

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

- ReviewRepository: Database access for review operations
- ProductRepository: Database access for product operations
- OrderRepository: Database access for order operations
- CartRepository: Database access for cart operations

Test Setup

Before each test, the following setup is performed:

1. Initialize mocks using MockitoAnnotations
2. Create a test product with:
    - Random UUID as productId
    - "Test Product" as product name
    - Empty list of reviewIds
3. Create a test review with:
    - Random UUID as reviewId
    - Random UUID as userId
    - Product ID linking to the test product
    - Rating of 4.5
    - Comment "Great product!"
    - Verification status set to false (unverified)

Test Cases

1. testPostReview_Successful

Purpose: Verify that a new review can be successfully posted and associated with a product.

Test Scenario:

- **Arrange:**
    - Configure productRepository findById method to return the test product
    - Configure reviewRepository save method to return the test review
    - Configure productRepository save method to return the updated product
    - Configure reviewRepository findByProductId to return a list with the test review
- **Act:**
    - Call reviewService.postReview with the test review

- **Assert:**
    - Verify the returned review is not marked as verified
    - Verify reviewRepository.save was called once with the test review
    - Verify productRepository.save was called twice (twice due to rating calculation)
    - Verify the product's reviewIds list now contains the review ID

Business Logic Verified:

- New reviews can be posted for products
- Reviews are initially unverified
- Reviews are correctly associated with their respective products
- Changes are persisted to both review and product repositories

2. testApproveReview_Successful

Purpose: Verify that administrators can approve previously submitted reviews.

Test Scenario:

- **Arrange:**
    - Configure reviewRepository findById method to return the test review
    - Configure reviewRepository save method to return the updated review
- **Act:**
    - Call reviewService.approveReview with the review ID
- **Assert:**
    - Verify the returned review is marked as verified
    - Verify reviewRepository.save was called once with the updated review

Business Logic Verified:

- Reviews can be marked as verified by administrators

- The verification status is correctly updated
- Changes are persisted to the database

3. testApproveReview_ReviewNotFound

Purpose: Verify that the system handles attempts to approve non-existent reviews.

Test Scenario:

- **Arrange:**
    - Create a random non-existent review ID
    - Configure reviewRepository findById method to return empty for this ID
- **Act & Assert:**
    - Call reviewService.approveReview with the non-existent ID and verify it throws an IllegalArgumentException
    - Verify reviewRepository.save was never called

Business Logic Verified:

- The system correctly handles attempts to approve non-existent reviews
- Appropriate exceptions are thrown when reviews cannot be found
- No database operations occur when validation fails

4. testGetVerifiedReviewsForProduct_ReturnsVerifiedReviews

Purpose: Verify that the system can retrieve only verified reviews for a specific product.

Test Scenario:

- **Arrange:**
    - Set the test review's verified status to true

    - Configure reviewRepository findByProductIdAndVerifiedTrue method to return a list containing the test review
- **Act:**
    - Call reviewService.getVerifiedReviewsForProduct with the product ID
- **Assert:**
    - Verify the returned list contains one review
    - Verify the review in the list is marked as verified
    - Verify reviewRepository.findByProductIdAndVerifiedTrue was called once with the correct product ID

Business Logic Verified:

- The system can filter reviews based on verification status and product ID
- Only verified reviews for the specified product are returned when requested
- The correct query method is used to retrieve verified reviews

Mocking Strategy

The tests use a consistent mocking strategy to isolate the ReviewService 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
4. **Exception Testing:** Verify proper exception handling for error conditions

Test Coverage

These tests cover the core functionality of the ReviewService:

- Posting new product reviews

- Approving reviews by administrators
- Handling invalid review approval attempts
- Retrieving only verified reviews for a specific product

Conclusion

The ReviewServiceTest thoroughly verifies the core functionality of the review management system. The tests ensure that reviews can be posted, approved by administrators, and filtered based on verification status and product ID.

These tests help maintain the integrity of the review system as the application evolves, ensuring that customers can provide feedback on products and view reliable, verified reviews from other customers. The verification process helps maintain quality and trustworthiness in the product review system.