ProductServiceTest Documentation

Overview

This document details the unit tests implemented for the ProductService class in our e-commerce application backend. The ProductService manages the product catalog, allowing for the addition of new products, updating product information, managing inventory levels, and searching for products.

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

- ProductRepository: Database access for product operations
- CategoryRepository: Database access for category operations

Test Setup

Before each test, the following setup is performed:

1. Initialize mocks using MockitoAnnotations
2. Create a test category with:

    - Random UUID as categoryId
    - "Electronics" as categoryName
    - Empty list of productIds
3. Create a test product with:
    - Random UUID as productId
    - "Test Product" as product name
    - "Test product description" as productInfo
    - Reference to the test category's ID
    - Stock count of 10
    - Price of 99.99
    - Serial number "SN12345"
    - Warranty status "1 year"
    - Distributor info "Test Distributor"

Test Cases

1. testAddProduct_Successful

Purpose: Verify that a new product can be successfully added to the catalog and associated with a category.

Test Scenario:

- **Arrange:**
    - Configure productRepository mock to return null when checking for existing product name
    - Configure categoryRepository mock to return the test category when searching by name

    - Configure productRepository and categoryRepository save methods to return the test objects
- **Act:**
    - Call productService.addProduct with all product details (product object, name, info, category name, stock count, serial number, warranty status, distributor info)
- **Assert:**
    - Verify the response message is "Product added successfully!"
    - Verify the response has 200 status code
    - Verify productRepository.save was called once
    - Verify categoryRepository.save was called once

Business Logic Verified:

- New products can be added to the catalog
- Products are correctly associated with existing categories
- Changes are persisted to both product and category repositories
- The system returns appropriate success messages

2. testUpdateStock_Successful

Purpose: Verify that product stock levels can be updated.

Test Scenario:

- **Arrange:**
    - Set a new stock count value of 20
    - Configure productRepository findById method to return the test product
    - Configure productRepository save method to return the updated product
- **Act:**
    - Call productService.updateStock with the product ID and new stock count

- **Assert:**
    - Verify the returned product has the updated stock count
    - Verify productRepository.save was called once with the updated product

Business Logic Verified:

- Product stock levels can be updated
- Changes to stock count are persisted to the database
- The updated product information is returned from the operation

3. testUpdateProduct_WithMultipleFields

Purpose: Verify that multiple product attributes can be updated simultaneously.

Test Scenario:

- **Arrange:**
    - Create a map of updates with new values for product name, price, and stock count
    - Configure productRepository findById method to return the test product
    - Configure productRepository save method to return the updated product
- **Act:**
    - Call productService.updateProduct with the product ID and updates map
- **Assert:**
    - Verify the returned product has the updated name, price, and stock count
    - Verify productRepository.save was called once with the updated product

Business Logic Verified:

- Multiple product attributes can be updated in a single operation
- Updates handle different data types (string and numeric values)
- Changes are persisted to the database

- The updated product with all changes is returned

4. testSearchProducts_ReturnsMatchingProducts

Purpose: Verify that products can be searched by name or description.

Test Scenario:

- **Arrange:**
    - Create a search query "Test"
    - Create an expected list containing the test product
    - Configure productRepository search method (findByProductNameContainingIgnoreCaseOrProductInfoContainingIgnoreCase) to return this list
- **Act:**
    - Call productService.searchProducts with the search query
- **Assert:**
    - Verify the returned list matches the expected products
    - Verify the repository search method was called once with the correct query parameters

Business Logic Verified:

- Products can be searched by name or description
- The search is case-insensitive
- The correct search results are returned to the user
- The search query is correctly passed to the repository layer

Mocking Strategy

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

- Adding new products to the catalog
- Updating product stock levels
- Updating multiple product attributes
- Searching for products by name or description

Conclusion

The ProductServiceTest thoroughly verifies the core functionality of the product management system. The tests ensure that products can be added to the catalog, their information can be updated, and they can be found through search.

These tests help maintain the integrity of the product catalog as the application evolves, ensuring that customers have access to accurate and up-to-date product information. The product system's integration with the category system is also verified, ensuring proper organization of products.