CategoryServiceTest Documentation

Overview

This document details the unit tests implemented for the CategoryService class in our e-commerce application backend. The CategoryService manages product categories, allowing administrators to create categories and assign products to them, which helps in organizing the product catalog.

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

- **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

Test Cases

1. testAddCategory_Successful

Purpose: Verify that a new category can be successfully added when the name is unique.

Test Scenario:

- **Arrange**:
    - Configure mock to return false when checking if a category with the name already exists
    - Configure mock to return the test category when save is called
- **Act**:
    - Call categoryService.addCategory with the test category and its name
- **Assert**:
    - Verify the response contains "Category added successfully!" message
    - Verify the response has 200 status code
    - Verify the category was saved to the database

Business Logic Verified:

- Categories with unique names can be added to the system
- The system returns appropriate success responses

2. testAddCategory_NameAlreadyExists

Purpose: Verify that adding a category with an existing name is properly rejected.

Test Scenario:

- **Arrange**:
  - Configure mock to return true when checking if a category with the name already exists
- **Act**:
  - Call categoryService.addCategory with the test category and its name
- **Assert**:
  - Verify the response contains "Category with this name already exists!" message
  - Verify the response has 400 status code
  - Verify no save operation was performed

Business Logic Verified:

- The system prevents duplicate category names
- Appropriate error messages and status codes are returned
- No database operation occurs when validation fails

3. testGetAllCategories_ReturnsAllCategories

Purpose: Verify that all categories can be retrieved from the system.

Test Scenario:

- **Arrange**:
  - Create two test categories with different names ("Electronics" and "Clothing")
  - Configure mock to return these categories when findAll is called
- **Act**:
  - Call categoryService.getAllCategories
- **Assert**:
  - Verify the result matches the expected categories list

    - Verify the repository's findAll method was called once

Business Logic Verified:

- The system correctly retrieves all categories
- The returned data structure matches the expected format

Mocking Strategy

The tests use a consistent mocking strategy to isolate the CategoryService 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**: Validate the responses from the service based on different repository states

Test Coverage

These tests cover the core functionality of the CategoryService:

- Adding new categories with validation
- Handling duplicate category names
- Retrieving all categories

Conclusion

The CategoryServiceTest thoroughly verifies the core functionality of the category management system. The tests ensure that categories can be created with proper validation, duplicate names are rejected, and all categories can be retrieved.

These tests help maintain the integrity of the category system as the application evolves, ensuring that product organization remains consistent and reliable for both administrators and end users.