UserServiceTest Documentation

Overview

This document details the unit tests implemented for the UserService class in our e-commerce application backend. The UserService manages user accounts, including registration, authentication, and wishlist management, providing a secure and personalized experience for users.

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

- UserRepository: Database access for user operations
- ProductRepository: Database access for product operations
- EmailService: Service for sending account verification emails
- SecureTokenService: Service for creating and validating secure tokens
- Environment: Spring environment for accessing application properties
- PasswordEncoder: Service for encoding and verifying passwords

Test Setup

Before each test, the following setup is performed:

1. Initialize mocks using MockitoAnnotations
2. Create a test user with:
    - Random UUID as userId
    - Email address "test@gmail.com"
    - Password "Password1!"
    - Name "Test" and surname "User"
    - "Customer" role
    - Location information (city "ISTANBUL", address)
    - Phone number "05551234567"
    - Empty wishlist
3. Create a test product ID using UUID

Test Cases

1. testRegisterUser_Successful

Purpose: Verify that a new user can be successfully registered.

Test Scenario:

- **Arrange:**
    - Configure userRepository to return empty when checking if email exists
    - Configure passwordEncoder to return "encodedPassword" when encoding any string
    - Configure userRepository save method to return the test user
- **Act & Assert:**
    - Call userService.registerUser with the test user and assert it doesn't throw exceptions
    - Verify userRepository.save was called once with the test user
    - Verify passwordEncoder.encode was called once with any string

Business Logic Verified:

- New users can be registered in the system
- Passwords are properly encoded before storing
- User data is persisted to the database

2. testRegisterUser_EmailAlreadyExists

Purpose: Verify that the system prevents registration with an existing email address.

Test Scenario:

- **Arrange:**
  - Configure userRepository to return the test user when checking if email exists
- **Act & Assert:**
  - Call userService.registerUser with the test user and assert it throws UserAlreadyExistsException
  - Verify the exception message contains the test user's email
  - Verify userRepository.save was never called

Business Logic Verified:

- The system prevents duplicate email addresses
- Appropriate exceptions are thrown with informative messages
- No database operations occur when validation fails

3. testAuthenticate_Successful

Purpose: Verify that a user can successfully authenticate with correct credentials.

Test Scenario:

- **Arrange:**

    - Set a raw password "Password1!" and encoded password "encodedPassword"
    - Update test user's password to the encoded version
    - Configure userRepository to return the test user when findByEmail is called
    - Configure passwordEncoder.matches to return true when comparing the raw and encoded passwords
- **Act:**
    - Call userService.authenticate with the email and raw password
- **Assert:**
    - Verify the returned user is not null
    - Verify the returned user matches the test user

Business Logic Verified:

- Users can authenticate with correct credentials
- Password verification is properly handled
- The correct user object is returned upon successful authentication

4. testAuthenticate_InvalidCredentials

Purpose: Verify that authentication fails with incorrect credentials.

Test Scenario:

- **Arrange:**
    - Set a wrong raw password "WrongPassword" and encoded password "encodedPassword"
    - Update test user's password to the encoded version
    - Configure userRepository to return the test user when findByEmail is called
    - Configure passwordEncoder.matches to return false when comparing the passwords
- **Act:**

    - Call userService.authenticate with the email and wrong password
- **Assert:**
    - Verify the returned result is null

Business Logic Verified:

- Authentication fails with incorrect password
- No user object is returned when authentication fails

5. testAddToWishlist_Successful

Purpose: Verify that a product can be successfully added to a user's wishlist.

Test Scenario:

- **Arrange:**
    - Configure userRepository to return the test user when findById is called
    - Configure userRepository save method to return the updated user
- **Act:**
    - Call userService.addToWishlist with the userId and productId
- **Assert:**
    - Verify the response message is "Product added to wishlist."
    - Verify the response has 200 status code
    - Verify the product ID was added to the user's wishlist
    - Verify userRepository.save was called once with the updated user

Business Logic Verified:

- Users can add products to their wishlist
- The wishlist is properly updated and persisted
- The system returns appropriate success messages

6. testRemoveFromWishlist_Successful

Purpose: Verify that a product can be successfully removed from a user's wishlist.

Test Scenario:

- **Arrange:**
    - Add the test product ID to the user's wishlist
    - Configure userRepository to return the test user when findById is called
    - Configure userRepository save method to return the updated user
- **Act:**
    - Call userService.removeFromWishlist with the userId and productId
- **Assert:**
    - Verify the response message is "Product removed from wishlist."
    - Verify the response has 200 status code
    - Verify the product ID was removed from the user's wishlist
    - Verify userRepository.save was called once with the updated user

Business Logic Verified:

- Users can remove products from their wishlist
- The wishlist is properly updated and persisted
- The system returns appropriate success messages

7. testGetWishlist_Successful

Purpose: Verify that a user's wishlist can be successfully retrieved.

Test Scenario:

- **Arrange:**
    - Add the test product ID to the user's wishlist

    - Create a test product with the test product ID
    - Configure userRepository to return the test user when findById is called
    - Configure productRepository to return a list containing the test product when findAllById is called
- **Act:**
    - Call userService.getWishlist with the userId
- **Assert:**
    - Verify the response has 200 status code
    - Verify the response body contains a list with one product
    - Verify the product in the list has the correct product ID

Business Logic Verified:

- Users can retrieve their wishlist products
- The system correctly fetches the product details for wishlist items
- The system returns the appropriate data structure

Mocking Strategy

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

1. **Mock Responses:** Return prepared test objects when repository and service methods are called
2. **Behavior Verification:** Verify that the service calls other services and repositories as expected
3. **Exception Testing:** Verify proper exception handling for error conditions

Test Coverage

These tests cover the core functionality of the UserService:

- Registering new users
- Handling duplicate email registration attempts
- Authenticating users with credentials
- Managing user wishlist (add, remove, retrieve)

Conclusion

The UserServiceTest thoroughly verifies the core functionality of the user account management system. The tests ensure that users can register accounts, authenticate with their credentials, and manage their wishlist of products.

These tests help maintain the integrity and security of the user management system as the application evolves. The authentication process ensures secure access to user accounts, while the wishlist functionality provides a personalized shopping experience. Exception handling tests verify that the system gracefully handles error conditions such as duplicate registrations.