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, email verification, and account management. It implements security features such as secure token verification for email confirmation.

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
- **EmailService**: Service for sending account verification emails
- **SecureTokenService**: Service for creating and validating secure tokens
- **Environment**: Spring environment for accessing application properties

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 and surname
    - "Customer" role
    - Location information (city, address)
    - Phone number

Test Cases

1. testRegisterUser_Successful

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

Test Scenario:

- **Arrange**:
    - Configure userRepository to return empty when checking if email exists
    - Configure userRepository save method to return the test user
    - Configure emailService to do nothing when sendMail is called
    - Configure environment to return a base URL
    - Configure secureTokenService to return a new token valid for 15 minutes
- **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 secureTokenService.createToken was called once

    - Verify emailService.sendMail was called once with an email context

Business Logic Verified:

- New users can be registered in the system
- A secure token is created for email verification
- A verification email is sent to the user
- Changes are 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. testVerifyUser_Successful

Purpose: Verify that a user can successfully verify their account with a valid token.

Test Scenario:

- **Arrange**:
    - Create a valid token string
    - Create a secure token object linked to the test user, valid for 10 more minutes
    - Configure secureTokenService to return the token when getToken is called
    - Configure userRepository save method to return the test user
- **Act**:
    - Call userService.verifyUser with the token string and assert it doesn't throw exceptions
- **Assert**:
    - Verify the test user's accountVerified flag is set to true
    - Verify userRepository.save was called once with the updated user

Business Logic Verified:

- Users can verify their accounts with valid tokens
- Account verification status is correctly updated
- Changes are persisted to the database

4. testVerifyUser_ExpiredToken

Purpose: Verify that the system rejects expired verification tokens.

Test Scenario:

- **Arrange**:
    - Create an expired token string
    - Create a secure token object linked to the test user, expired 10 minutes ago
    - Configure secureTokenService to return the token when getToken is called
- **Act & Assert**:

    - Call userService.verifyUser with the token string and assert it throws InvalidTokenException
    - Verify the test user's accountVerified flag remains false
    - Verify userRepository.save was never called

Business Logic Verified:

- The system rejects expired verification tokens
- Appropriate exceptions are thrown when tokens are invalid
- User accounts remain unverified when token validation fails
- No database operations occur when validation fails

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
4. **Void Method Mocking**: Configure void methods like emailService.sendMail to do nothing

Test Coverage

These tests cover the core functionality of the UserService:

- Registering new users
- Handling duplicate email registration attempts
- Verifying user accounts with tokens

- Handling expired verification tokens

Conclusion

The UserServiceTest thoroughly verifies the core functionality of the user account management system. The tests ensure that users can register accounts, receive verification emails, and verify their accounts through secure tokens.

These tests help maintain the integrity and security of the user management system as the application evolves. The verification process ensures that email addresses are valid and belong to the registering users, which is crucial for secure account management. Exception handling tests verify that the system gracefully handles error conditions such as duplicate registrations and expired tokens.