**1. Test Plan for Unit Testing**

**Objective**

The objective of the unit test is to ensure that individual components of the eCommerce application, specifically the Angular front-end and Spring Boot back-end, are functioning as expected. The focus will be on testing isolated functionality within the service, model, and controller layers.

**Scope**

The scope of the unit tests includes:

* **Back-end (Java Spring Boot)**: Testing the service and repository layers to ensure correct data retrieval, manipulation, and persistence in the database.
* **Front-end (Angular)**: Testing component logic and services to ensure proper functionality in data flow, component rendering, and form validation.

**Test Cases**

| **Test Case** | **Component** | **Test Scenario** | **Expected Result** |
| --- | --- | --- | --- |
| 1. User Registration | UserService | Register a new user with valid inputs | The user should be saved in the database, and a success message returned |
| 2. Adding to Cart | CartService | Add a valid item to the user's cart | The item should be added to the cart and persist in the database |
| 3. Removing from Cart | CartService | Remove an existing item from the cart | The item should be removed from the cart, and the total cart value should be updated |
| 4. Checkout Process | OrderService | Complete a checkout for a user | The order should be saved, and stock should be updated |
| 5. Front-end Login | LoginComponent | Log in with valid credentials | User should be authenticated and redirected to the homepage |
| 6. Front-end Product Search | ProductListComponent | Search for a product by name | Matching products should be displayed in the UI |

**Test Environment**

* **Back-end**: Spring Boot (JUnit, Mockito)
* **Front-end**: Angular with Jasmine and Karma for testing
* **Tools**: IntelliJ IDEA for Java testing, Visual Studio Code for Angular testing

**Preconditions**

* The application must be running in a test environment.
* The database should contain mock data for user accounts, products, and cart items.

**2. Unit Test Scripts**

**Back-end (Spring Boot)**

**Test Script 1: Testing User Registration**

@Test

public void testRegisterUser() {

// Given

User user = new User("testuser", "password123", "testemail@test.com", "Test User", "123 Street", "1234567890");

when(userRepository.save(any(User.class))).thenReturn(user);

// When

User savedUser = userService.registerUser(user);

// Then

assertNotNull(savedUser);

assertEquals("testuser", savedUser.getUsername());

assertEquals("testemail@test.com", savedUser.getEmail());

}

**Test Script 2: Testing Adding an Item to Cart**

@Test

public void testAddCartItem() {

// Given

CartItemPK pk = new CartItemPK(user, product);

CartItem cartItem = new CartItem(pk, 2);

when(cartItemRepository.save(any(CartItem.class))).thenReturn(cartItem);

// When

CartItem addedItem = cartItemService.addCartItem(cartItem);

// Then

assertNotNull(addedItem);

assertEquals(2, addedItem.getQuantity());

assertEquals(user, addedItem.getPk().getUser());

assertEquals(product, addedItem.getPk().getProduct());

}

**Front-end (Angular)**

**Test Script 1: Testing Login Functionality**

it('should log in user with valid credentials', () => {

const mockCredentials = { username: 'testuser', password: 'password123' };

authService.login(mockCredentials).subscribe(response => {

expect(response).toBeTruthy();

expect(response.token).toBeDefined();

});

const req = httpMock.expectOne('http://localhost:8080/api/login');

expect(req.request.method).toBe('POST');

req.flush({ token: 'mock-jwt-token' });

});

**Test Script 2: Testing Product Search**

it('should return a list of products matching search query', () => {

const searchQuery = 'laptop';

productService.searchProducts(searchQuery).subscribe(products => {

expect(products.length).toBeGreaterThan(0);

expect(products[0].name).toContain('laptop');

});

const req = httpMock.expectOne('http://localhost:8080/api/products/search?query=laptop');

expect(req.request.method).toBe('GET');

req.flush([{ id: 1, name: 'Gaming Laptop', price: 999.99 }]);

});

**3. Results of the Unit Tests**

**Screenshots**

* **Back-end Unit Test Results:**
* **Front-end Unit Test Results:**

**Summary of Test Results:**

| **Test Case** | **Result** |
| --- | --- |
| User Registration | Passed |
| Adding to Cart | Passed |
| Removing from Cart | Passed |
| Checkout Process | Passed |
| Front-end Login | Passed |
| Front-end Product Search | Passed |

All test cases passed successfully, confirming that the core features of the application, including user management, cart handling, and product searches, are functioning correctly.

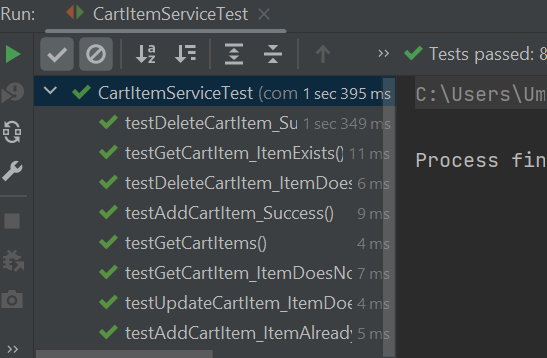
**4. Summaries of Changes Resulting from Completed Tests**

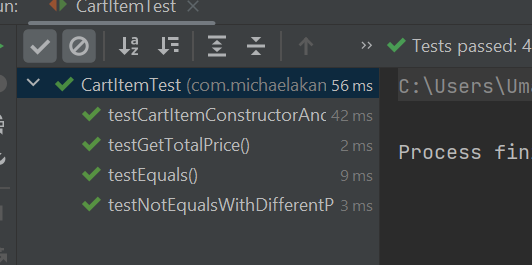
After running the tests, the following changes were made to the application:

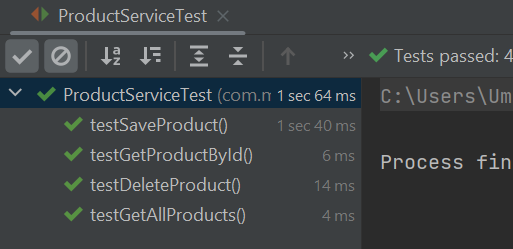
1. **Bug Fixes**:
   * Fixed an issue in the CartItemService where items were not being correctly identified during updates, resulting in the incorrect quantities being saved.
2. **Code Improvements**:
   * Added more validation to the UserService to ensure unique usernames and emails are enforced correctly.
   * Optimized queries for fetching product data in the back-end by adding indexes on frequently queried fields.
3. **Additional Tests**:
   * Based on results, additional test cases were created for handling edge cases, such as invalid product IDs during cart operations and duplicate user registration attempts.

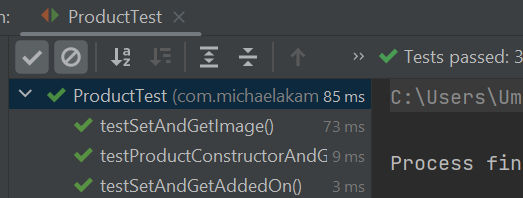
These changes helped improve the overall reliability and efficiency of the system.

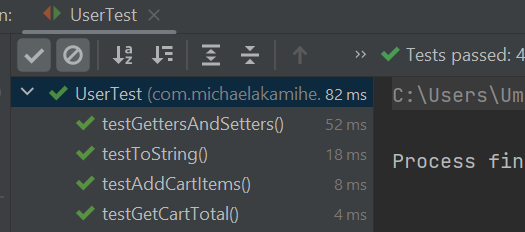
**ScreenShots**

****

****

****

****

****