RefundTest Documentation

Overview

This document details the unit tests implemented for refund-related endpoints in the e-commerce backend. The tests exercise the `OrderController` methods responsible for requesting a refund, listing active refund requests, approving refunds, and rejecting refunds.

Test Class Details

- **Class name:** `RefundTest`
- **Package:** `com.cs308.backend.services`
- **Controller under test:** `OrderController`
- **Test frameworks:** JUnit 5, Mockito
- **MockMvc setup:** Standalone via `MockMvcBuilders.standaloneSetup(orderController)`

Mocked Dependencies

Field	Role
`@Mock UserService`	(Not used by refund endpoints directly)
`@Mock CartService`	Cart-related operations
`@Mock PaymentService`	Payment logic (unused here)
`@Mock OrderService`	Contains refund business logic
`@Mock OrderHistoryService`	Order history retrieval
`@InjectMocks OrderController`	REST controller handling refund routes

Test Cases

1. **requestRefund_ShouldReturnOkAndMessage**
    - **Arrange:** Stub `orderService.requestRefundSingle(orderId, userId, productId, quantity)` to return `ResponseEntity.ok("Refund requested")`.
    - **Act:** `PUT /api/order/requestRefund/{orderId}` with query params `userId`, `productId`, `quantity`.
    - **Assert:** HTTP **200 OK**, response body equals **"Refund requested"**.
2. **getActiveRefundRequests_ShouldReturnListAsJson**

- **Arrange:** Stub `orderService.getRefundRequestsByProcessed(false)` to return a list containing one `RefundRequest` with a single `RefundItem`.
- **Act:** `GET /api/order/refundRequests/active`.
- **Assert:** HTTP **200 OK**, JSON array with element 0 having:
  - `requestId` = `req123`
  - `orderId` = `order123`
  - `items[0].productId` = `prod1`
  - `items[0].quantity` = `1`

3. **approveRefund_ShouldReturnOkAndApprovalMessage**
   - **Arrange:** Stub `orderService.approveRefund(requestId)` to return `ResponseEntity.ok("Approved")`.
   - **Act:** `PUT /api/order/refund/approve/{requestId}`.
   - **Assert:** HTTP **200 OK**, response body equals **"Approved"**.
4. **rejectRefund_ShouldReturnOkAndRejectionMessage**
   - **Arrange:** Stub `orderService.rejectRefund(requestId)` to return `ResponseEntity.ok("Rejected")`.
   - **Act:** `DELETE /api/order/refund/reject/{requestId}`.
   - **Assert:** HTTP **200 OK**, response body equals **"Rejected"**.
5. **requestRefund_MissingParams_ShouldReturnBadRequest**
   - **Arrange:** No stubbing (missing query params).
   - **Act:** `PUT /api/order/requestRefund/{orderId}` without params.
   - **Assert:** HTTP **400 Bad Request**.
6. **requestRefund_ServiceReturnsBadRequest_ShouldReturnBadRequest**
   - **Arrange:** Stub `orderService.requestRefundSingle(...)` to return `ResponseEntity.badRequest().body("Invalid refund")`.
   - **Act:** `PUT /api/order/requestRefund/{orderId}` with valid params.
   - **Assert:** HTTP **400 Bad Request**, body equals **"Invalid refund"**.
7. **requestRefund_ServiceThrowsException_ShouldReturnServerError**
   - **Arrange:** Stub `orderService.requestRefundSingle(...)` to throw `RuntimeException("boom")`.
   - **Act:** `PUT /api/order/requestRefund/{orderId}` with valid params.
   - **Assert:** HTTP **500 Internal Server Error**.
8. **getActiveRefundRequests_NoRequests_ShouldReturnEmptyList**
   - **Arrange:** Stub `orderService.getRefundRequestsByProcessed(false)` to return `Collections.emptyList()`.

- **Act:** `GET /api/order/refundRequests/active`.
- **Assert:** HTTP **200 OK**, response body equals `[]`.

9. **approveRefund_NonexistentId_ShouldReturnNotFound**
    - **Arrange:** Stub `orderService.approveRefund("nope")` to return `ResponseEntity.notFound().build()`.
    - **Act:** `PUT /api/order/refund/approve/nope`.
    - **Assert:** HTTP **404 Not Found**.
10. **rejectRefund_NonexistentId_ShouldReturnNotFound**

- **Arrange:** Stub `orderService.rejectRefund("nope")` to return `ResponseEntity.notFound().build()`.
- **Act:** `DELETE /api/order/refund/reject/nope`.
- **Assert:** HTTP **404 Not Found**.

Mocking Strategy

- Use Mockito's `@Mock` for all service dependencies and `@InjectMocks` for the controller.
- Configure a standalone `MockMvc` to isolate controller behavior.
- Employ `when(...).thenReturn(...)` and `thenThrow(...)` for stubbing.
- Validate with `status()`, `content()`, and `jsonPath()` assertions.

Test Coverage and Conclusion

These 10 unit tests comprehensively cover both successful and failure scenarios for the refund endpoints, validating correct HTTP status codes and response bodies under various conditions. Together, they provide confidence that the refund functionality in OrderController is robust, reliable, and handles edge cases and invalid inputs gracefully.