**Title: Day17Session3Que3: orders Management REST API Using JPA**

**Problem Statement**

You are tasked with creating a Spring Boot REST API to manage orders in an e-commerce system. The API should provide the following operations:

1. **Retrieve all orders (GET):** Fetch all orders stored in the database.
2. **Update order details by ID (PUT):** Modify the customer name, items, status, or total amount of an order.
3. **Delete an order by ID (DELETE):** Remove an order from the database.

Each order in the system must include:

* **ID:** Automatically generated using JPA's @GeneratedValue with IDENTITY strategy.
* **Customer Name:** The name of the customer.
* **Items:** List of items purchased (comma-separated string).
* **Status:** Current status of the order (e.g., "Pending", "Shipped", "Delivered").
* **Total Amount:** Total monetary value of the order.

The objective is to implement a RESTful API following best practices and maintaining a clean separation of concerns using DAO, Service, and Controller layers.

**Examples**

**Example 1: Retrieve All Orders**

**Input:**

http

GET /api/orders

**Description:** Fetches all orders stored in the database.

**Output:**

json

[

{ "id": 1, "customerName": "John Doe", "items": "Laptop, Mouse", "status": "Pending", "totalAmount": 1500.50 },

{ "id": 2, "customerName": "Jane Smith", "items": "Phone, Charger", "status": "Shipped", "totalAmount": 1000.00 }

]

**Example 2: Update an Existing Order**

**Input:**

http

PUT /api/orders/{id}

**Body:**

json

{ "customerName": "Alice Johnson", "items": "Tablet, Headphones", "status": "Delivered", "totalAmount": 2000.00 }

**Description:** Updates the customer name, items, status, or total amount of the order with the specified ID.

**Output:**

json

{ "id": 1, "customerName": "Alice Johnson", "items": "Tablet, Headphones", "status": "Delivered", "totalAmount": 2000.00 }

**Example 3: Delete an Order**

**Input:**

http

DELETE /api/orders/{id}

**Description:** Deletes the order with the specified ID.

**Output:**

json

{ "message": "Order deleted successfully" }

**Java Source Code**

**Entity Layer**

java

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "orders\_table") // Renamed table to avoid conflicts with "order" keyword

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String customerName;

private String items;

private String status;

private Double totalAmount;

// Default Constructor

public Order() {}

// Parameterized Constructor

public Order(String customerName, String items, String status, Double totalAmount) {

this.customerName = customerName;

this.items = items;

this.status = status;

this.totalAmount = totalAmount;

}

// Getters and Setters

// (Include them in your code for all fields)

}

**DAO Layer**

java

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {}

**Service Layer**

java

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class OrderService {

private final OrderRepository orderRepository;

// Constructor Dependency Injection

public OrderService(OrderRepository orderRepository) {

this.orderRepository = orderRepository;

}

public List<Order> getAllOrders() {

return orderRepository.findAll();

}

public Order updateOrder(Long id, Order orderDetails) {

Order existingOrder = orderRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Order not found with ID: " + id));

existingOrder.setCustomerName(orderDetails.getCustomerName());

existingOrder.setItems(orderDetails.getItems());

existingOrder.setStatus(orderDetails.getStatus());

existingOrder.setTotalAmount(orderDetails.getTotalAmount());

return orderRepository.save(existingOrder);

}

public void deleteOrder(Long id) {

if (!orderRepository.existsById(id)) {

throw new RuntimeException("Order not found with ID: " + id);

}

orderRepository.deleteById(id);

}

}

**Controller Layer**

java

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/orders")

public class OrderController {

private final OrderService orderService;

// Constructor Dependency Injection

public OrderController(OrderService orderService) {

this.orderService = orderService;

}

@GetMapping

public List<Order> getAllOrders() {

return orderService.getAllOrders();

}

@PutMapping("/{id}")

public Order updateOrder(@PathVariable Long id, @RequestBody Order orderDetails) {

return orderService.updateOrder(id, orderDetails);

}

@DeleteMapping("/{id}")

public void deleteOrder(@PathVariable Long id) {

orderService.deleteOrder(id);

}

}

**Test Class**

java

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

public class OrderServiceTest {

@Mock

private OrderRepository orderRepository;

@InjectMocks

private OrderService orderService;

@BeforeEach

public void setUp() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testUpdateOrder() {

Order existingOrder = new Order("John Doe", "Laptop", "Pending", 1500.00);

existingOrder.setId(1L);

Order updatedOrderDetails = new Order("Alice Johnson", "Tablet", "Delivered", 2000.00);

when(orderRepository.findById(1L)).thenReturn(Optional.of(existingOrder));

when(orderRepository.save(existingOrder)).thenReturn(existingOrder);

Order updatedOrder = orderService.updateOrder(1L, updatedOrderDetails);

assertNotNull(updatedOrder);

assertEquals("Alice Johnson", updatedOrder.getCustomerName());

assertEquals("Tablet", updatedOrder.getItems());

verify(orderRepository, times(1)).findById(1L);

verify(orderRepository, times(1)).save(existingOrder);

}

@Test

public void testDeleteOrder() {

when(orderRepository.existsById(1L)).thenReturn(true);

doNothing().when(orderRepository).deleteById(1L);

orderService.deleteOrder(1L);

verify(orderRepository, times(1)).existsById(1L);

verify(orderRepository, times(1)).deleteById(1L);

}

}

**Execution Flow**

1. **Entity Layer (Order):** Represents the Order table in the database.
2. **DAO Layer (OrderRepository):** Handles all database operations using Spring Data JPA.
3. **Service Layer (OrderService):** Implements the business logic for retrieving, updating, and deleting orders.
4. **Controller Layer (OrderController):** Defines RESTful endpoints for API interactions.
5. **Test Class (OrderServiceTest):** Validates that the service layer interacts correctly with the DAO and performs the expected operations.