SpringBoot :Delivery Application

**Description**

Implement REST APIs to facilitate managing delivery orders and customer details for a delivery application system. The APIs should enable users to update delivery order information based on specific criteria, calculate average delivery duration, and get the count of delivered orders for all customers.

Here is an example of a Customer JSON object:

{

"customerId": 1,

"name": "Charlie Adams",

"email": "charlie.adams@example.com",

"phoneNumber": "+1212345678"

}

Here is an example of a DeliveryPerson JSON object:

{

"deliveryPersonId": 1,

"name": "Alice Blue",

"vehicleType": "Bike",

"status": "Active"

}

Here is an example of a DeliveryOrder JSON object:

{

"orderId": 3,

"customerId": 5,

"deliveryPerson": {

"deliveryPersonId": 4,

"name": "Sarah Yellow",

"vehicleType": "Car",

"status": "Active"

},

"items": "Book,Notebook",

"pickupLocation": "456 Walnut St",

"deliveryLocation": "789 Pine Ln",

"status": "PLACED",

"orderTime": "2024-07-04T08:00:00",

"pickupTime": "2024-07-04T08:20:00",

"deliveryTime": "2024-07-04T09:00:00"

}

You are provided with the implementation of the models required for all the APIs. Your task is to implement a set of REST services that expose the endpoints, enabling users to update delivery order information based on specific criteria, calculate average delivery duration, and get the count of delivered orders for all customers.

| API Route | API Type | Success Response Code| Validation Error Code|

|---------------------------------------------------------|----------|--------------------- |----------------------|

| /api/delivery-orders/{orderId}/status | PUT | 200 | 400, 404, 500 |

| /api/delivery-orders/average-delivery-duration | GET | 200 | 404, 500 |

| /api/delivery-orders/delivered-orders-count-by-customer | GET | 200 | 404, 500 |

Task 1: Service Layer Implementation in DeliveryOrderService

Implement the logic in the service layer using DeliveryOrderRepository, CustomerRepository, and DeliveryPersonRepository:

**Method Details**:

* **updateOrderStatus()**: It should update the status of a delivery order using the specified orderId and status. If the orderId is found, save the the delivery order status. If the orderId is not found, a NoSuchElementException should be thrown with the message "Order not found".
* **calculateAverageDeliveryDuration()**: It should calculate the average delivery duration (in minutes) for all the orders that are delivered. The duration is the time between pickupTime and deliveryTime, rounded to two decimal places. If there are no delivered orders, return null. For delivered orders, return the average delivery duration rounded to two decimal places.
* **getDeliveredOrdersCountForAllCustomers()**: It should fetch all delivery orders from the repository, filter out the orders with a status of "**DELIVERED**" and retrieve the count of delivered("DELIVERED") orders for each customer. The result should return a map where the keys are customer names and the values are the counts of their delivered orders.

Task 2: REST API Endpoints in DeliveryOrderController

**PUT request to /api/delivery-orders/{orderId}/status**

Updates the status of a delivery order using the order ID and the specified status. The status must be either "DELIVERED" or "CANCELLED".

**Request Body**: String representing the new status ("DELIVERED" or "CANCELLED").

**Response Body**: String indicating the updated status of the delivery order

**HTTP Status Code**:

* 200 - If the status is updated successfully - "Delivery Status Updated Successfully".
* 400 - If the request body contains an invalid status - "Invalid status".
* 404 - If the given orderId is not found or there is not any details for the specific orderId.
* 500 - If there is an internal server error while updating the status- "An error occurred while updating the order status".

**GET request to /api/delivery-orders/average-delivery-duration**

Calculates and retrieves the average delivery duration in minutes up to 2 decimal places.

**Response Body**: The average delivery duration (in minutes) should be of a double data type.

**HTTP Status Code**:

* 200 - If you received the average delivery duration.
* 404 - If there are no delivered orders.
* 500 - Error in calculating average delivery duration.

**GET request to /api/delivery-orders/delivered-orders-count-by-customer**

Retrieves the count of delivered orders for each customer.

**Response Body**: JSON object representing customer names and their respective delivered order counts

**HTTP Status Code**:

* 200 - If you received the customer order counts.
* 404 - If there are no customers with delivered orders.
* 500 - Error in receiving the customer order counts.

Complete the given project so that it passes all the test cases when running the provided unit tests.

**Example Responses:**

**PUT request to /api/delivery-orders/1/status:**

Request Body (text):

**DELIVERED**

The response code is 200 and the response body should return a message as follows: plaintext

**Request**: PUT - /api/delivery-orders/1/status

**Delivery** Status Updated Successfully

**GET request to /api/delivery-orders/average-delivery-duration:**

The response code is 200 and the response body, when converted to JSON, is as follows:

36.67

**GET request to /api/delivery-orders/delivered-orders-count-by-customer:**

The response code is 200 and the response body, when converted to JSON, is as follows:

**Request**: GET - /api/delivery-orders/delivered-orders-count-by-customer

{

"Eve Black": 2,

"David Wilson": 1,

"Frank White": 2,

"Charlie Adams": 1,

"Grace Green": 1

}

DeliveryOrderController.java

package com.example.delivery.controller;

import com.example.delivery.entity.Customer;

import com.example.delivery.entity.DeliveryOrder;

import com.example.delivery.entity.DeliveryPerson;

import com.example.delivery.service.DeliveryOrderService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Map;

@RestController

@RequestMapping("/api/delivery-orders")

public class DeliveryOrderController {

    @Autowired

    private DeliveryOrderService deliveryOrderService;

    @GetMapping

    public List<DeliveryOrder> getAllDeliveryOrders() {

        return deliveryOrderService.getAllDeliveryOrders();

    }

    @GetMapping("customers")

    public List<Customer> getAllCustomers() {

        return deliveryOrderService.getAllCustomers();

    }

    @GetMapping("delivery-persons")

    public List<DeliveryPerson> getAllDeliveryPersons() {

        return deliveryOrderService.getAllDeliveryPersons();

    }

    @PutMapping("/{orderId}/status")

    public ResponseEntity<String> updateOrderStatus(@PathVariable Long orderId, @RequestBody String status) {

        // Code here ...

        return new ResponseEntity<>(null);

    }

    @GetMapping("/average-delivery-duration")

    public ResponseEntity<Double> calculateAverageDeliveryDuration() {

        // Code here ...

        return new ResponseEntity<>(null);

    }

    @GetMapping("/delivered-orders-count-by-customer")

    public ResponseEntity<Map<String, Long>> getDeliveredOrdersCountForAllCustomers() {

        // Code here ...

        return new ResponseEntity<>(null);

    }

}

Customer.java

package com.example.delivery.entity;

import javax.persistence.\*;

import javax.validation.constraints.NotNull;

@Entity

public class Customer {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column

    @NotNull

    private Long customerId;

    @Column

    @NotNull

    private String name;

    @Column

    @NotNull

    private String email;

    @Column

    @NotNull

    private String phoneNumber;

    // Getters and setters

    public Long getCustomerId() {

        return customerId;

    }

    public void setCustomerId(Long customerId) {

        this.customerId = customerId;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getEmail() {

        return email;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    public String getPhoneNumber() {

        return phoneNumber;

    }

    public void setPhoneNumber(String phoneNumber) {

        this.phoneNumber = phoneNumber;

    }

}

DeliveryOrder.java

package com.example.delivery.entity;

import javax.persistence.\*;

import javax.validation.constraints.NotNull;

import java.time.LocalDateTime;

import java.util.List;

@Entity

public class DeliveryOrder {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column

    @NotNull

    private Long orderId;

    @Column

    @NotNull

    private Long customerId;

    @ManyToOne

    @JoinColumn(name = "deliveryPersonId")

    private DeliveryPerson deliveryPerson;

    @Column

    @NotNull

    private String items;

    @Column

    @NotNull

    private String pickupLocation;

    @Column

    @NotNull

    private String deliveryLocation;

    @Column

    @NotNull

    private String status;

    @Column

    @NotNull

    private LocalDateTime orderTime;

    @Column

    private LocalDateTime pickupTime;

    @Column

    private LocalDateTime deliveryTime;

    // Getters and setters

    public Long getOrderId() {

        return orderId;

    }

    public void setOrderId(Long orderId) {

        this.orderId = orderId;

    }

    public Long getCustomerId() {

        return customerId;

    }

    public void setCustomerId(Long customerId) {

        this.customerId = customerId;

    }

    public DeliveryPerson getDeliveryPerson() {

        return deliveryPerson;

    }

    public void setDeliveryPerson(DeliveryPerson deliveryPerson) {

        this.deliveryPerson = deliveryPerson;

    }

    public String getItems() {

        return items;

    }

    public void setItems(String items) {

        this.items = items;

    }

    public String getPickupLocation() {

        return pickupLocation;

    }

    public void setPickupLocation(String pickupLocation) {

        this.pickupLocation = pickupLocation;

    }

    public String getDeliveryLocation() {

        return deliveryLocation;

    }

    public void setDeliveryLocation(String deliveryLocation) {

        this.deliveryLocation = deliveryLocation;

    }

    public String getStatus() {

        return status;

    }

    public void setStatus(String status) {

        this.status = status;

    }

    public LocalDateTime getOrderTime() {

        return orderTime;

    }

    public void setOrderTime(LocalDateTime orderTime) {

        this.orderTime = orderTime;

    }

    public LocalDateTime getPickupTime() {

        return pickupTime;

    }

    public void setPickupTime(LocalDateTime pickupTime) {

        this.pickupTime = pickupTime;

    }

    public LocalDateTime getDeliveryTime() {

        return deliveryTime;

    }

    public void setDeliveryTime(LocalDateTime deliveryTime) {

        this.deliveryTime = deliveryTime;

    }

}

DeliveryPerson.java

package com.example.delivery.entity;

import javax.persistence.\*;

import javax.validation.constraints.NotNull;

@Entity

public class DeliveryPerson {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column

    @NotNull

    private Long deliveryPersonId;

    @Column

    @NotNull

    private String name;

    @Column

    @NotNull

    private String vehicleType;

    @Column

    @NotNull

    private String status;

    // Getters and setters

    public Long getDeliveryPersonId() {

        return deliveryPersonId;

    }

    public void setDeliveryPersonId(Long deliveryPersonId) {

        this.deliveryPersonId = deliveryPersonId;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getVehicleType() {

        return vehicleType;

    }

    public void setVehicleType(String vehicleType) {

        this.vehicleType = vehicleType;

    }

    public String getStatus() {

        return status;

    }

    public void setStatus(String status) {

        this.status = status;

    }

}

CustomerRepository.java

package com.example.delivery.repository;

import com.example.delivery.entity.Customer;

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

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

DeliveryOrderRepository.java

package com.example.delivery.repository;

import com.example.delivery.entity.DeliveryOrder;

import com.example.delivery.entity.DeliveryPerson;

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

import java.util.List;

public interface DeliveryOrderRepository extends JpaRepository<DeliveryOrder, Long> {

    List<DeliveryOrder> findByStatus(String status);

}

DeliveryPersonRepository.java

package com.example.delivery.repository;

import com.example.delivery.entity.DeliveryPerson;

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

import java.util.List;

public interface DeliveryPersonRepository extends JpaRepository<DeliveryPerson, Long> {

}

DeliveryOrderService.java

package com.example.delivery.service;

import com.example.delivery.entity.Customer;

import com.example.delivery.entity.DeliveryOrder;

import com.example.delivery.entity.DeliveryPerson;

import com.example.delivery.repository.CustomerRepository;

import com.example.delivery.repository.DeliveryOrderRepository;

import com.example.delivery.repository.DeliveryPersonRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Map;

@Service

public class DeliveryOrderService {

    @Autowired

    private DeliveryOrderRepository deliveryOrderRepository;

    @Autowired

    private CustomerRepository customerRepository;

    @Autowired

    private DeliveryPersonRepository deliveryPersonRepository;

    public List<DeliveryOrder> getAllDeliveryOrders() {

        return deliveryOrderRepository.findAll();

    }

    public List<Customer> getAllCustomers() {

        return customerRepository.findAll();

    }

    public List<DeliveryPerson> getAllDeliveryPersons() {

        return deliveryPersonRepository.findAll();

    }

    public DeliveryOrder updateOrderStatus(Long orderId, String status) {

        // Code here ...

        return null;

    }

    public Double calculateAverageDeliveryDuration() {

        // Code here ...

        return null;

    }

    public Map<String, Long> getDeliveredOrdersCountForAllCustomers() {

        // Code here ...

        return null;

    }

}

DeliveryOrderController.java

package com.example.delivery;

import com.example.delivery.controller.DeliveryOrderController;

import com.example.delivery.entity.DeliveryOrder;

import com.example.delivery.service.DeliveryOrderService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.NoSuchElementException;

@WebMvcTest(DeliveryOrderController.class)

public class DeliveryOrderControllerTest {

    @Autowired

    private MockMvc mockMvc;

    @MockBean

    private DeliveryOrderService deliveryOrderService;

    @Test

    public void testUpdateOrderStatus\_Success() throws Exception {

        Long orderId = 1L;

        String status = "DELIVERED";

        DeliveryOrder updatedOrder = new DeliveryOrder();

        updatedOrder.setStatus(status);

        Mockito.when(deliveryOrderService.updateOrderStatus(orderId, status)).thenReturn(updatedOrder);

        mockMvc.perform(MockMvcRequestBuilders.put("/api/delivery-orders/{orderId}/status", orderId)

                        .contentType(MediaType.APPLICATION\_JSON)

                        .content(status))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().string("Delivery Status Updated Successfully"));

    }

    @Test

    public void testUpdateOrderStatus\_InvalidStatus() throws Exception {

        Long orderId = 1L;

        String status = "INVALID";

        Mockito.doThrow(new IllegalArgumentException("Invalid status")).when(deliveryOrderService).updateOrderStatus(orderId, status);

        mockMvc.perform(MockMvcRequestBuilders.put("/api/delivery-orders/{orderId}/status", orderId)

                        .contentType(MediaType.APPLICATION\_JSON)

                        .content(status))

                .andExpect(MockMvcResultMatchers.status().isBadRequest())

                .andExpect(MockMvcResultMatchers.content().string("Invalid status"));

    }

    @Test

    public void testUpdateOrderStatus\_OrderNotFound() throws Exception {

        Long orderId = 1L;

        String status = "DELIVERED";

        Mockito.doThrow(new NoSuchElementException("Order not found")).when(deliveryOrderService).updateOrderStatus(orderId, status);

        mockMvc.perform(MockMvcRequestBuilders.put("/api/delivery-orders/{orderId}/status", orderId)

                        .contentType(MediaType.APPLICATION\_JSON)

                        .content(status))

                .andExpect(MockMvcResultMatchers.status().isNotFound());

    }

    @Test

    public void testCalculateAverageDeliveryDuration\_Success() throws Exception {

        Double averageDuration = 30.0;

        Mockito.when(deliveryOrderService.calculateAverageDeliveryDuration()).thenReturn(averageDuration);

        mockMvc.perform(MockMvcRequestBuilders.get("/api/delivery-orders/average-delivery-duration"))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().contentType(MediaType.APPLICATION\_JSON))

                .andExpect(MockMvcResultMatchers.content().json("30.0"));

    }

    @Test

    public void testCalculateAverageDeliveryDuration\_NoContent() throws Exception {

        Mockito.when(deliveryOrderService.calculateAverageDeliveryDuration()).thenReturn(null);

        mockMvc.perform(MockMvcRequestBuilders.get("/api/delivery-orders/average-delivery-duration"))

                .andExpect(MockMvcResultMatchers.status().isNotFound());

    }

    @Test

    public void testGetDeliveredOrdersCountForAllCustomers\_Success() throws Exception {

        Map<String, Long> result = new HashMap<>();

        result.put("Customer 1", 2L);

        Mockito.when(deliveryOrderService.getDeliveredOrdersCountForAllCustomers()).thenReturn(result);

        mockMvc.perform(MockMvcRequestBuilders.get("/api/delivery-orders/delivered-orders-count-by-customer"))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().contentType(MediaType.APPLICATION\_JSON))

                .andExpect(MockMvcResultMatchers.jsonPath("$.size()").value(1))

                .andExpect(MockMvcResultMatchers.jsonPath("$.['Customer 1']").value(2));

    }

    @Test

    public void testGetDeliveredOrdersCountForAllCustomers\_NoContent() throws Exception {

        Mockito.when(deliveryOrderService.getDeliveredOrdersCountForAllCustomers()).thenReturn(Collections.emptyMap());

        mockMvc.perform(MockMvcRequestBuilders.get("/api/delivery-orders/delivered-orders-count-by-customer"))

                .andExpect(MockMvcResultMatchers.status().isNotFound());

    }

}

DeliveryOrderServiceTest.java

package com.example.delivery;

import com.example.delivery.entity.Customer;

import com.example.delivery.entity.DeliveryOrder;

import com.example.delivery.repository.CustomerRepository;

import com.example.delivery.repository.DeliveryOrderRepository;

import com.example.delivery.service.DeliveryOrderService;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.Mockito;

import org.mockito.junit.jupiter.MockitoExtension;

import java.time.LocalDateTime;

import java.util.\*;

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

@ExtendWith(MockitoExtension.class)

public class DeliveryOrderServiceTest {

    @Mock

    private DeliveryOrderRepository deliveryOrderRepository;

    @Mock

    private CustomerRepository customerRepository;

    @InjectMocks

    private DeliveryOrderService deliveryOrderService;

    @Test

    public void testUpdateOrderStatus\_Success() {

        // Mock data

        Long orderId = 1L;

        String status = "DELIVERED";

        DeliveryOrder deliveryOrder = new DeliveryOrder();

        deliveryOrder.setOrderId(orderId);

        // Mock repository behavior

        Mockito.when(deliveryOrderRepository.findById(orderId)).thenReturn(Optional.of(deliveryOrder));

        Mockito.when(deliveryOrderRepository.save(deliveryOrder)).thenReturn(deliveryOrder);

        // Test service method

        DeliveryOrder updatedOrder = deliveryOrderService.updateOrderStatus(orderId, status);

        // Assertions

        assertEquals(status, updatedOrder.getStatus());

        Mockito.verify(deliveryOrderRepository, Mockito.times(1)).findById(orderId);

        Mockito.verify(deliveryOrderRepository, Mockito.times(1)).save(deliveryOrder);

    }

    @Test

    public void testUpdateOrderStatus\_OrderNotFound() {

        Long orderId = 1L;

        String status = "DELIVERED";

        Mockito.when(deliveryOrderRepository.findById(orderId)).thenReturn(Optional.empty());

        NoSuchElementException exception = assertThrows(NoSuchElementException.class,

                () -> deliveryOrderService.updateOrderStatus(orderId, status));

        assertEquals("Order not found", exception.getMessage());

    }

    @Test

    public void testCalculateAverageDeliveryDuration() {

        // Mock data

        DeliveryOrder order1 = new DeliveryOrder();

        order1.setPickupTime(LocalDateTime.now().minusMinutes(30));

        order1.setDeliveryTime(LocalDateTime.now());

        DeliveryOrder order2 = new DeliveryOrder();

        order2.setPickupTime(LocalDateTime.now().minusMinutes(45));

        order2.setDeliveryTime(LocalDateTime.now().minusMinutes(15));

        List<DeliveryOrder> deliveredOrders = Arrays.asList(order1, order2);

        // Mock repository behavior

        Mockito.when(deliveryOrderRepository.findByStatus("DELIVERED")).thenReturn(deliveredOrders);

        // Test service method

        Double averageDuration = deliveryOrderService.calculateAverageDeliveryDuration();

        // Assertion

        assertEquals(Double.valueOf(30.0), averageDuration); // Assuming exact calculations for simplicity

    }

    @Test

    public void testCalculateAverageDeliveryDuration\_NoDeliveredOrders() {

        // Mock repository behavior

        Mockito.when(deliveryOrderRepository.findByStatus("DELIVERED")).thenReturn(Collections.emptyList());

        // Test service method

        assertNull(deliveryOrderService.calculateAverageDeliveryDuration());

    }

    @Test

    public void testGetDeliveredOrdersCountForAllCustomers() {

        // Mock data

        DeliveryOrder order1 = new DeliveryOrder();

        order1.setCustomerId(1L);

        order1.setStatus("DELIVERED");

        DeliveryOrder order2 = new DeliveryOrder();

        order2.setCustomerId(2L);

        order2.setStatus("DELIVERED");

        DeliveryOrder order3 = new DeliveryOrder();

        order3.setCustomerId(1L);

        order3.setStatus("DELIVERED");

        List<DeliveryOrder> allOrders = Arrays.asList(order1, order2, order3);

        // Mock repository behavior

        Mockito.when(deliveryOrderRepository.findAll()).thenReturn(allOrders);

        // Mock customer repository behavior

        Customer customer1 = new Customer();

        customer1.setCustomerId(1L);

        customer1.setName("Customer 1");

        Customer customer2 = new Customer();

        customer2.setCustomerId(2L);

        customer2.setName("Customer 2");

        Mockito.when(customerRepository.findById(1L)).thenReturn(Optional.of(customer1));

        Mockito.when(customerRepository.findById(2L)).thenReturn(Optional.of(customer2));

        // Test service method

        Map<String, Long> result = deliveryOrderService.getDeliveredOrdersCountForAllCustomers();

        // Assertions

        assertEquals(2, result.size());

        assertEquals(Long.valueOf(2), result.get("Customer 1"));

        assertEquals(Long.valueOf(1), result.get("Customer 2"));

    }

    @Test

    public void testGetDeliveredOrdersCountForAllCustomers\_NoDeliveredOrders() {

        // Mock repository behavior

        Mockito.when(deliveryOrderRepository.findAll()).thenReturn(Collections.emptyList());

        // Test service method

        Map<String, Long> result = deliveryOrderService.getDeliveredOrdersCountForAllCustomers();

        // Assertions

        assertTrue(result.isEmpty());

    }

}