CAB BOOKING MANAGEMENT SYSTEM

Problem Description

In the bustling city of Metroville, **ZoomRide**, a popular ride-hailing platform, aims to offer seamless transportation experiences for its users. To enhance the platform's efficiency, the **RideService** layer must be implemented with three key functionalities: booking a ride, retrieving recent completed rides, and ending a ride.

**a. model package**:

->Contains the Driver, RideDetails and User entity class with attributes, Constructors, getters and setters.

->The **Driver** entity should have the following attributes:

* **id** : The unique identifier for the driver (Long, Primary Key).
* **availability** : Indicates if the driver is currently available. It uses a BIT(1) type to represent true (available) or false (not available). (Boolean).
* **name** : The name of the driver, with a maximum length of 255 characters. It can be NULL if not specified. (String).
* **rating** : The rating of the driver, representing their performance. This field is mandatory and cannot be NULL.(Double)
* **vehicleDetails** : Contains details about the vehicle used by the driver, with a maximum length of 255 characters. This field can also be NULL.(String)

->The **User** entity should have the following attributes:

* **id** : The unique identifier for the user (Long, Primary Key).
* **email** : The email address of the user, with a maximum length of 255 characters. It can be NULL.(String)
* **name** : The name of the user, with a maximum length of 255 characters. This field can be NULL if not specified.(String)
* **paymentMethod** : Indicates the payment method used by the user (e.g., credit card, cash). This field has a maximum length of 255 characters and can be NULL.(String)
* **phone** : The contact number of the user, with a maximum length of 255 characters. It can also be NULL.(String)

->The **RideDetails** entity should have the following attributes:

* **id** : The unique identifier for the ride. (Long, Primary Key).
* **driverId** : The ID of the assigned driver for this ride (Foreign Key referencing the driver table).(Long)
* **endLocation** : The destination of the ride, with a maximum length of 255 characters. This field is mandatory and cannot be NULL.(String)
* **endTime** : The end time of the ride. This field can be NULL if the ride is ongoing.(LocalDateTime)
* **fare** : The fare for the ride, which is mandatory and cannot be NULL.(Double)
* **rideStatus** : The current status of the ride, which can be "Ongoing" or "Completed". This field is mandatory and cannot be NULL.(String)
* **startLocation** : The starting point of the ride, with a maximum length of 255 characters. This field is mandatory and cannot be NULL.(String)
* **startTime** : The start time of the ride. This field is mandatory and cannot be NULL.(LocalDateTime)
* **userId** : The ID of the user who booked the ride (Foreign Key referencing the user table).(Long)

**b. repository package**: Contains the interface DriverRepository, RideDetailsRepository and UserRepository extending JpaRepository.

**c.** **service package**: Contains RideService class with some method signature performing business logics

**d.** **controller package**: Contains RideController with skeleton CRUD API endpoints.

Task 1: Book a Ride

**Complete the method with logic as per the requirement.**

* **Function Name**: bookRide(Long userId, String startLocation, String endLocation, double distance):
* **Return Type**: RideDetails
* **Functionality**: If no drivers are available, throw an exception indicating "No available drivers". Otherwise, calculate the fare based on the distance (Rs.25 per KM) and return the ride details, including the first driver available and the fare calculated.

Task 2: Get Recent Completed Rides

**Complete the method with logic as per the requirement.**

* **Function Name**: getRecentCompletedRides(Long userId)
* **Return Type**: List
* **Functionality**: Retrieve completed rides for the user within the last 30 minutes. Return a list of RideDetails that match the criteria.

Task 3: End a Ride

**Complete the method with logic as per the requirement.**

* **Function Name**: endRide(Long rideId)
* **Return Type**: RideDetails
* **Functionality**: End the specified ride by updating its status and setting the end time. If the ride ID does not exist, throw an exception indicating**"Ride not found"**.

RideController.java

package com.example.cabBooking.controller;

import com.example.cabBooking.entity.Driver;

import com.example.cabBooking.entity.RideDetails;

import com.example.cabBooking.entity.User;

import com.example.cabBooking.service.RideService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@CrossOrigin

@RestController

@RequestMapping("/api/rides")

public class RideController {

    @Autowired

    private RideService rideService;

    // Endpoint to book a ride

    @PostMapping("/book")

    public ResponseEntity<RideDetails> bookRide(

            @RequestParam Long userId,

            @RequestParam String startLocation,

            @RequestParam String endLocation,

            @RequestParam double distance) {

        RideDetails rideDetails = rideService.bookRide(userId, startLocation, endLocation, distance);

        return new ResponseEntity<>(rideDetails, HttpStatus.CREATED); // 201 Created

    }

    // Endpoint to get recent completed rides for a user

    @GetMapping("/completed")

    public ResponseEntity<List<RideDetails>> getRecentCompletedRides(@RequestParam Long userId) {

        List<RideDetails> completedRides = rideService.getRecentCompletedRides(userId);

        if (completedRides.isEmpty()) {

            return new ResponseEntity<>(HttpStatus.NO\_CONTENT); // 204 No Content

        }

        return new ResponseEntity<>(completedRides, HttpStatus.OK); // 200 OK

    }

    // Endpoint to end a ride

    @PostMapping("/end/{rideId}")

    public ResponseEntity<RideDetails> endRide(@PathVariable Long rideId) {

        RideDetails rideDetails = rideService.endRide(rideId);

        return new ResponseEntity<>(rideDetails, HttpStatus.OK); // 200 OK

    }

    // Endpoint to get all users

    @GetMapping("/users")

    public ResponseEntity<List<User>> getAllUsers() {

        List<User> users = rideService.getAllUsers();

        if (users.isEmpty()) {

            return new ResponseEntity<>(HttpStatus.NO\_CONTENT); // 204 No Content

        }

        return new ResponseEntity<>(users, HttpStatus.OK); // 200 OK

    }

    // Endpoint to get all drivers

    @GetMapping("/drivers")

    public ResponseEntity<List<Driver>> getAllDrivers() {

        List<Driver> drivers = rideService.getAllDrivers();

        if (drivers.isEmpty()) {

            return new ResponseEntity<>(HttpStatus.NO\_CONTENT); // 204 No Content

        }

        return new ResponseEntity<>(drivers, HttpStatus.OK); // 200 OK

    }

    // Endpoint to get all ride details

    @GetMapping("/ride-details")

    public ResponseEntity<List<RideDetails>> getAllRideDetails() {

        List<RideDetails> rideDetails = rideService.getAllRideDetails();

        if (rideDetails.isEmpty()) {

            return new ResponseEntity<>(HttpStatus.NO\_CONTENT); // 204 No Content

        }

        return new ResponseEntity<>(rideDetails, HttpStatus.OK); // 200 OK

    }

}

Driver.java

package com.example.cabBooking.entity;

import javax.persistence.\*;

@Table(name = "driver")

@Entity

public class Driver {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String name;

    private String vehicleDetails;

    private double rating;

    private boolean availability;

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getVehicleDetails() {

        return vehicleDetails;

    }

    public void setVehicleDetails(String vehicleDetails) {

        this.vehicleDetails = vehicleDetails;

    }

    public double getRating() {

        return rating;

    }

    public void setRating(double rating) {

        this.rating = rating;

    }

    public boolean isAvailability() {

        return availability;

    }

    public void setAvailability(boolean availability) {

        this.availability = availability;

    }

}

RideDetails.java

package com.example.cabBooking.entity;

import javax.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "ride\_details")

public class RideDetails {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "user\_id", nullable = false)

    private Long userId;

    @Column(name = "driver\_id", nullable = false)

    private Long driverId;

    @Column(name = "start\_location", nullable = false)

    private String startLocation;

    @Column(name = "end\_location", nullable = false)

    private String endLocation;

    @Column(name = "fare", nullable = false)

    private double fare;

    @Column(name = "ride\_status", nullable = false)

    private String rideStatus;

    @Column(name = "start\_time", nullable = false)

    private LocalDateTime startTime;

    @Column(name = "end\_time")

    private LocalDateTime endTime;

    // Getters and Setters

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public Long getUserId() {

        return userId;

    }

    public void setUserId(Long userId) {

        this.userId = userId;

    }

    public Long getDriverId() {

        return driverId;

    }

    public void setDriverId(Long driverId) {

        this.driverId = driverId;

    }

    public String getStartLocation() {

        return startLocation;

    }

    public void setStartLocation(String startLocation) {

        this.startLocation = startLocation;

    }

    public String getEndLocation() {

        return endLocation;

    }

    public void setEndLocation(String endLocation) {

        this.endLocation = endLocation;

    }

    public double getFare() {

        return fare;

    }

    public void setFare(double fare) {

        this.fare = fare;

    }

    public String getRideStatus() {

        return rideStatus;

    }

    public void setRideStatus(String rideStatus) {

        this.rideStatus = rideStatus;

    }

    public LocalDateTime getStartTime() {

        return startTime;

    }

    public void setStartTime(LocalDateTime startTime) {

        this.startTime = startTime;

    }

    public LocalDateTime getEndTime() {

        return endTime;

    }

    public void setEndTime(LocalDateTime endTime) {

        this.endTime = endTime;

    }

}

User.java

package com.example.cabBooking.entity;

import javax.persistence.\*;

@Table(name = "user")

@Entity

public class User {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String name;

    private String phone;

    private String email;

    private String paymentMethod;

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getPhone() {

        return phone;

    }

    public void setPhone(String phone) {

        this.phone = phone;

    }

    public String getEmail() {

        return email;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    public String getPaymentMethod() {

        return paymentMethod;

    }

    public void setPaymentMethod(String paymentMethod) {

        this.paymentMethod = paymentMethod;

    }

}

RideService.java

package com.example.cabBooking.service;

import com.example.cabBooking.entity.Driver;

import com.example.cabBooking.entity.RideDetails;

import com.example.cabBooking.entity.User;

import com.example.cabBooking.repository.DriverRepository;

import com.example.cabBooking.repository.RideDetailsRepository;

import com.example.cabBooking.repository.UserRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class RideService {

    private static final double FARE\_PER\_KM = 25.0; // Fare in INR

    @Autowired

    private DriverRepository driverRepository;

    @Autowired

    private RideDetailsRepository rideDetailsRepository;

    @Autowired

    private UserRepository userRepository;

    // Book a ride and return ride details with fare calculation

    public RideDetails bookRide(Long userId, String startLocation, String endLocation, double distance) {

        //code here

        return null;

    }

    // Method to calculate fare based on distance

    private double calculateFare(double distance) {

        return distance \* FARE\_PER\_KM; // Fare calculation

    }

    // Get completed rides for a user within the last 30 minutes

    public List<RideDetails> getRecentCompletedRides(Long userId) {

        //code here

        return null;

    }

    // End a ride and calculate the total fare

    public RideDetails endRide(Long rideId) {

        //code here

        return null;

    }

    // Get all users

    public List<User> getAllUsers() {

        return userRepository.findAll();

    }

    // Get all drivers

    public List<Driver> getAllDrivers() {

        return driverRepository.findAll();

    }

    // Get all ride details

    public List<RideDetails> getAllRideDetails() {

        return rideDetailsRepository.findAll();

    }

}

AppTest.java

package com.example.cabBooking;

import com.example.cabBooking.entity.Driver;

import com.example.cabBooking.entity.RideDetails;

import com.example.cabBooking.repository.DriverRepository;

import com.example.cabBooking.repository.RideDetailsRepository;

import com.example.cabBooking.service.RideService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.MethodOrderer;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.test.annotation.DirtiesContext;

import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDateTime;

import java.util.Collections;

import java.util.List;

import java.util.Optional;

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

import static org.mockito.Mockito.any;

import static org.mockito.Mockito.when;

@DataJpaTest

@EnableAutoConfiguration

@ComponentScan(basePackages = "com.example.cabBooking")

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE\_EACH\_TEST\_METHOD)

@Transactional

public class AppTest {

    @InjectMocks

    private RideService rideService;

    @Mock

    private DriverRepository driverRepository;

    @Mock

    private RideDetailsRepository rideDetailsRepository;

    private Driver driver;

    private RideDetails rideDetails;

    @BeforeEach

    void setUp() {

        MockitoAnnotations.openMocks(this);

        driver = new Driver();

        driver.setId(1L);

        driver.setName("John Doe");

        driver.setAvailability(true);

        rideDetails = new RideDetails();

        rideDetails.setId(1L);

        rideDetails.setUserId(1L);

        rideDetails.setDriverId(driver.getId());

        rideDetails.setStartLocation("Start Location");

        rideDetails.setEndLocation("End Location");

        rideDetails.setFare(100.0);

        rideDetails.setRideStatus("Ongoing");

        rideDetails.setStartTime(LocalDateTime.now());

    }

    @Test

    void testBookRide\_Success\_WithFareCalculation() {

        // Arrange

        when(driverRepository.findByAvailabilityTrue()).thenReturn(Collections.singletonList(driver));

        // Remove hardcoding of fare in rideDetails

        when(rideDetailsRepository.save(any(RideDetails.class))).thenAnswer(invocation -> invocation.getArgument(0));

        // Act

        double distance = 10.0; // Mock the distance

        double expectedFare = distance \* 25.0; // Fare per km is 25.0 as per the service

        RideDetails result = rideService.bookRide(1L, "Start Location", "End Location", distance);

        // Assert

        assertNotNull(result);

        assertEquals("Start Location", result.getStartLocation());

        assertEquals("End Location", result.getEndLocation());

        assertEquals(expectedFare, result.getFare()); // Now this should be 250.0

        assertEquals("Ongoing", result.getRideStatus());

        assertEquals(driver.getId(), result.getDriverId());

    }

    @Test

    void testBookRide\_NoAvailableDrivers() {

        // Arrange

        when(driverRepository.findByAvailabilityTrue()).thenReturn(Collections.emptyList());

        // Act & Assert

        Exception exception = assertThrows(RuntimeException.class, () ->

                rideService.bookRide(1L, "Start Location", "End Location", 10.0));

        assertEquals("No available drivers", exception.getMessage());

    }

    @Test

    void testGetRecentCompletedRides\_Success() {

        // Arrange

        rideDetails.setEndTime(LocalDateTime.now().minusMinutes(10));

        when(rideDetailsRepository.findByUserId(1L)).thenReturn(Collections.singletonList(rideDetails));

        // Act

        List<RideDetails> result = rideService.getRecentCompletedRides(1L);

        // Assert

        assertNotNull(result);

        assertEquals(1, result.size());

        assertEquals(rideDetails, result.get(0));

    }

    @Test

    void testGetRecentCompletedRides\_NoCompletedRides() {

        // Arrange

        rideDetails.setEndTime(null);

        when(rideDetailsRepository.findByUserId(1L)).thenReturn(Collections.singletonList(rideDetails));

        // Act

        List<RideDetails> result = rideService.getRecentCompletedRides(1L);

        // Assert

        assertNotNull(result);

        assertTrue(result.isEmpty());

    }

    @Test

    void testEndRide\_Success() {

        // Arrange

        rideDetails.setEndTime(null);

        when(rideDetailsRepository.findById(1L)).thenReturn(Optional.of(rideDetails));

        when(rideDetailsRepository.save(any(RideDetails.class))).thenReturn(rideDetails);

        // Act

        RideDetails result = rideService.endRide(1L);

        // Assert

        assertNotNull(result);

        assertEquals("Completed", result.getRideStatus());

        assertNotNull(result.getEndTime());

    }

    @Test

    void testEndRide\_RideNotFound() {

        // Arrange

        when(rideDetailsRepository.findById(1L)).thenReturn(Optional.empty());

        // Act & Assert

        Exception exception = assertThrows(RuntimeException.class, () ->

                rideService.endRide(1L));

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

    }

}