**Original code**

package com.example.projectcampusride.controllers;

import android.util.Log;

import com.example.projectcampusride.NotificationUtils;

import com.example.projectcampusride.models.Ride;

import com.example.projectcampusride.models.RideStatus;

import com.example.projectcampusride.models.User;

import com.example.projectcampusride.repositories.PassengerRepository;

import com.example.projectcampusride.repositories.RideRepository;

import com.google.android.gms.tasks.Task;

import com.google.firebase.firestore.DocumentSnapshot;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QueryDocumentSnapshot;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

public class PassengerController {

    private static final String COLLECTION\_RIDES = "rides";

    private static final String TAG = "PassengerController";

    private final FirebaseFirestore db;

    private final PassengerRepository passengerRepository;

    public PassengerController() {

        this.db = FirebaseFirestore.getInstance();

        this.passengerRepository = PassengerRepository.getInstance();

    }

    public interface SearchCallback {

        void onSuccess(List<Ride> rides);

        void onFailure(String errorMessage);

    }

    public void searchRides(String startLocation, String endLocation, SearchCallback callback) {

        if (startLocation.isEmpty() || endLocation.isEmpty()) {

            callback.onFailure("Please enter both locations");

            return;

        }

        db.collection(COLLECTION\_RIDES)

                .whereEqualTo("startLocation", startLocation)

                .whereEqualTo("endLocation", endLocation)

                .get()

                .addOnSuccessListener(queryDocumentSnapshots -> {

                    Set<Map<String, Object>> matchingRides = new HashSet<>();

                    for (QueryDocumentSnapshot document : queryDocumentSnapshots) {

                        matchingRides.add(document.getData());

                    }

                    callback.onSuccess(convertToRideList(matchingRides));

                })

                .addOnFailureListener(e -> {

                    callback.onFailure("Error: " + e.getMessage());

                    Log.e(TAG, "Error searching rides", e);

                });

    }

    private List<Ride> convertToRideList(Set<Map<String, Object>> rides) {

        List<Ride> rideList = new ArrayList<>();

        for (Map<String, Object> rideData : rides) {

            String rideId = (String) rideData.get("rideId");

            String driverId = (String) rideData.get("driverId");

            String driverName = (String) rideData.get("driverName");

            String startLocation = (String) rideData.get("startLocation");

            String endLocation = (String) rideData.get("endLocation");

            Long seats = (Long) rideData.get("availableSeats");

            Object priceObj = rideData.get("price");

            double price = 0.0;

            if (priceObj instanceof Long) {

                price = ((Long) priceObj).doubleValue();

            } else if (priceObj instanceof Double) {

                price = (Double) priceObj;

            }

            String rideDate = (String) rideData.get("rideDate");

            String rideTime = (String) rideData.get("rideTime");

            if (startLocation != null && endLocation != null && seats != null && priceObj != null) {

                Ride ride = new Ride(rideId, driverId, driverName,startLocation, endLocation, rideDate, rideTime, seats.intValue(), price, RideStatus.ACTIVE);

                rideList.add(ride);

            }

        }

        return rideList;

    }

    public Task<String> requestToJoinRide(String rideId, User passenger, String driverId, String driverName, String date, String time, String startLocation, String endLocation) {

        return passengerRepository.checkExistingRequest(rideId, passenger.getUserId())

                .continueWithTask(task -> {

                    if (task.isSuccessful() && task.getResult()) {

                        throw new Exception("Request already exists for this ride.");

                    }

                return passengerRepository.requestToJoinRide(rideId, passenger,driverId,driverName,date,time);

            })

                .addOnSuccessListener(requestId -> {

                    // שלח התראה לנהג

                    String detailedMessage = passenger.getName() + " ביקש להצטרף לנסיעה מס' " + rideId

                            + " בתאריך " + date + " בשעה " + time

                            + " מ-" + startLocation + " ל-" + endLocation;

                    NotificationUtils.addNotification(driverId, detailedMessage, "decision", passenger.getUserId(), passenger.getName(),requestId, rideId);

                    Log.d("RideRequest", "Notification sent to driver: " + driverId);

                })

                .addOnFailureListener(e -> {

                    Log.e("RideRequest", "Failed to create ride request: " + e.getMessage());

                });

    }

    public void getPassengerRides(String passengerId, PassengerController.PassengerRidesCallback callback) {

        passengerRepository.getPassengerRides(passengerId)

                .addOnSuccessListener(querySnapshot -> {

                    if (querySnapshot != null && !querySnapshot.isEmpty()) {

                        List<Ride> rides = new ArrayList<>();

                        // גישה לכל מסמך ב-querySnapshot

                        for (Ride document : querySnapshot) {

                            try {

                                Ride ride = document;

                                if (ride != null) {

                                    rides.add(ride);

                                }

                            } catch (Exception e) {

                                Log.e("PassengerRides", "Error parsing ride: " + e.getMessage(), e);

                            }

                        }

                        if (!rides.isEmpty()) {

                            callback.onSuccess(rides);

                        } else {

                            callback.onFailure("No rides found for the passenger.");

                        }

                    } else {

                        callback.onFailure("No requests found for the passenger.");

                    }

                })

                .addOnFailureListener(e -> {

                    callback.onFailure("Failed to fetch rides: " + e.getMessage());

                });

    }

    public interface PassengerRidesCallback {

        void onSuccess(List<Ride> rides);

        void onFailure(String errorMessage);

    }

}