// lib/services/realtime\_service.dart

import 'dart:async';

import 'package:flutter/material.dart';

import 'package:geolocator/geolocator.dart';

import 'package:firebase\_database/firebase\_database.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:flutter\_local\_notifications/flutter\_local\_notifications.dart';

class RealTimeService {

// Singleton pattern

static final RealTimeService \_instance = RealTimeService.\_internal();

factory RealTimeService() => \_instance;

RealTimeService.\_internal();

// Firebase instances

final FirebaseDatabase \_database = FirebaseDatabase.instance;

final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

final FirebaseAuth \_auth = FirebaseAuth.instance;

// Location tracking

final GeolocatorPlatform \_geolocator = GeolocatorPlatform.instance;

StreamSubscription<Position>? \_positionStream;

// Notifications

final FlutterLocalNotificationsPlugin \_notificationsPlugin = FlutterLocalNotificationsPlugin();

// Stream controllers

final \_statusUpdateController = StreamController<Map<String, dynamic>>.broadcast();

final \_emergencyRequestController = StreamController<Map<String, dynamic>>.broadcast();

// Getters for streams

Stream<Map<String, dynamic>> get statusUpdates => \_statusUpdateController.stream;

Stream<Map<String, dynamic>> get emergencyRequests => \_emergencyRequestController.stream;

// Initialize the service

Future<void> initialize() async {

await \_initNotifications();

await \_checkLocationPermission();

\_listenForStatusUpdates();

}

// Set up notifications

Future<void> \_initNotifications() async {

const AndroidInitializationSettings androidSettings =

AndroidInitializationSettings('@mipmap/ic\_launcher');

const DarwinInitializationSettings iosSettings =

DarwinInitializationSettings(

requestAlertPermission: true,

requestBadgePermission: true,

requestSoundPermission: true,

);

const InitializationSettings initSettings = InitializationSettings(

android: androidSettings,

iOS: iosSettings,

);

await \_notificationsPlugin.initialize(

initSettings,

onDidReceiveNotificationResponse: (NotificationResponse response) {

// Handle notification taps

},

);

}

// Check and request location permissions

Future<bool> \_checkLocationPermission() async {

bool serviceEnabled;

LocationPermission permission;

serviceEnabled = await \_geolocator.isLocationServiceEnabled();

if (!serviceEnabled) {

return false;

}

permission = await \_geolocator.checkPermission();

if (permission == LocationPermission.denied) {

permission = await \_geolocator.requestPermission();

if (permission == LocationPermission.denied) {

return false;

}

}

if (permission == LocationPermission.deniedForever) {

return false;

}

return true;

}

// 1. Live Location Tracking

Future<void> startLocationTracking() async {

// Cancel any existing subscription

await stopLocationTracking();

// Get current user ID

final String? userId = \_auth.currentUser?.uid;

if (userId == null) return;

// Start tracking location

\_positionStream = \_geolocator.getPositionStream(

locationSettings: const LocationSettings(

accuracy: LocationAccuracy.high,

distanceFilter: 10,

),

).listen((Position position) {

\_updateLocationInFirebase(userId, position);

});

}

Future<void> stopLocationTracking() async {

await \_positionStream?.cancel();

\_positionStream = null;

}

Future<void> \_updateLocationInFirebase(String userId, Position position) async {

await \_database.ref('locations/$userId').set({

'latitude': position.latitude,

'longitude': position.longitude,

'heading': position.heading,

'speed': position.speed,

'timestamp': ServerValue.timestamp,

});

}

// Get worker's current location

Future<Position?> getCurrentLocation() async {

try {

return await \_geolocator.getCurrentPosition();

} catch (e) {

debugPrint('Error getting current location: $e');

return null;

}

}

// Get a specific worker's location

Stream<Map<String, dynamic>> getWorkerLocation(String workerId) {

return \_database.ref('locations/$workerId').onValue.map((event) {

return Map<String, dynamic>.from(event.snapshot.value as Map);

});

}

// 2. Emergency Service

Future<bool> sendEmergencyRequest({

required String jobId,

required String description,

required Map<String, dynamic> location,

}) async {

try {

final String? userId = \_auth.currentUser?.uid;

if (userId == null) return false;

// Add emergency request to Firestore

await \_firestore.collection('emergencyRequests').add({

'userId': userId,

'jobId': jobId,

'description': description,

'location': location,

'status': 'pending',

'timestamp': FieldValue.serverTimestamp(),

});

// Also update real-time database for immediate notification

await \_database.ref('emergencies/$jobId').set({

'userId': userId,

'description': description,

'location': location,

'status': 'pending',

'timestamp': ServerValue.timestamp,

});

return true;

} catch (e) {

debugPrint('Error sending emergency request: $e');

return false;

}

}

// Listen for emergency requests (for admin/manager app)

void listenForEmergencyRequests() {

\_database.ref('emergencies').onChildAdded.listen((event) {

final data = Map<String, dynamic>.from(event.snapshot.value as Map);

\_emergencyRequestController.add({

'id': event.snapshot.key,

...data,

});

// Show notification

\_showNotification(

id: 1,

title: 'Emergency Request',

body: 'New emergency request received for job ${data['jobId']}',

);

});

}

// 3. Status Updates

void \_listenForStatusUpdates() {

final String? userId = \_auth.currentUser?.uid;

if (userId == null) return;

// Listen for job status updates

\_database.ref('jobUpdates/$userId').onChildChanged.listen((event) {

final data = Map<String, dynamic>.from(event.snapshot.value as Map);

\_statusUpdateController.add({

'jobId': event.snapshot.key,

...data,

});

// Show notification for status update

\_showNotification(

id: 2,

title: 'Job Status Update',

body: 'Job ${event.snapshot.key}: ${data['status']}',

);

});

}

// Update job status

Future<bool> updateJobStatus({

required String jobId,

required String status,

String? notes,

}) async {

try {

// Get job information to find associated users

final jobSnapshot = await \_firestore.collection('jobs').doc(jobId).get();

if (!jobSnapshot.exists) return false;

final jobData = jobSnapshot.data() as Map<String, dynamic>;

final clientId = jobData['clientId'] as String;

final assignedWorkerId = jobData['workerId'] as String;

// Update in Firestore

await \_firestore.collection('jobs').doc(jobId).update({

'status': status,

'lastUpdated': FieldValue.serverTimestamp(),

if (notes != null) 'notes': notes,

});

// Update in realtime database for client notification

await \_database.ref('jobUpdates/$clientId/$jobId').update({

'status': status,

'timestamp': ServerValue.timestamp,

if (notes != null) 'notes': notes,

});

// Update for admin portal

await \_database.ref('allJobUpdates/$jobId').update({

'status': status,

'timestamp': ServerValue.timestamp,

'workerId': assignedWorkerId,

'clientId': clientId,

if (notes != null) 'notes': notes,

});

return true;

} catch (e) {

debugPrint('Error updating job status: $e');

return false;

}

}

// Show a notification

Future<void> \_showNotification({

required int id,

required String title,

required String body,

}) async {

const AndroidNotificationDetails androidDetails = AndroidNotificationDetails(

'real\_time\_channel',

'Real-time Updates',

channelDescription: 'Notifications for real-time updates',

importance: Importance.high,

priority: Priority.high,

);

const DarwinNotificationDetails iosDetails = DarwinNotificationDetails(

presentAlert: true,

presentBadge: true,

presentSound: true,

);

const NotificationDetails details = NotificationDetails(

android: androidDetails,

iOS: iosDetails,

);

await \_notificationsPlugin.show(id, title, body, details);

}

// Clean up resources

void dispose() {

\_positionStream?.cancel();

\_statusUpdateController.close();

\_emergencyRequestController.close();

}

}