package com.example.myapplication;  
  
import android.Manifest;  
import android.content.DialogInterface;  
import android.content.Intent;  
import android.content.pm.PackageManager;  
import android.graphics.ImageFormat;  
import android.graphics.SurfaceTexture;  
import android.hardware.Camera;  
import android.net.Uri;  
import android.os.Bundle;  
import android.os.Handler;  
import android.util.Log;  
import android.view.TextureView;  
import android.widget.Button;  
import android.widget.TextView;  
  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.app.ActivityCompat;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Locale;  
  
public class MainActivity extends AppCompatActivity implements TextureView.SurfaceTextureListener, Camera.PreviewCallback {  
 private TextureView textureView;  
 private Camera camera;  
 private TextView tvHeartRate;  
 private TextView tvTemp;  
  
 private Button btnStart;  
 private TextView tvHRV;  
 private final ArrayList<Long> rrIntervals = new ArrayList<>();  
 private TextView tvRR;  
 float temp;  
 int finalBPM;  
 private boolean measuring = false;  
 private long startTime = 0;  
 private int beats = 0;  
 private final Handler handler = new Handler();  
  
 private static final int *MEASUREMENT\_TIME\_MS* = 20000;  
 private long lastBeatTime = 0;  
 private final ArrayList<Long> beatTimestamps = new ArrayList<>();  
 private final LinkedList<Integer> signalWindow = new LinkedList<>();  
 private float filteredValue = 0;  
 private float previousValue = 0;  
 private boolean isRising = false;  
 private final List<Integer> redSignalList = new ArrayList<>();  
 private final List<Integer> greenSignalList = new ArrayList<>();  
 private TextView tvSpO2;  
 private final ArrayList<Integer> bpmReadings = new ArrayList<>();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 textureView = findViewById(R.id.*textureView*);  
 tvHeartRate = findViewById(R.id.*tv\_heart\_rate*);  
 btnStart = findViewById(R.id.*btn\_start*);  
 tvHRV = findViewById(R.id.*tv\_hrv*);  
 tvRR = findViewById(R.id.*tv\_rr*);  
  
  
  
 textureView.setSurfaceTextureListener(this);  
  
 btnStart.setOnClickListener(view -> {  
 if (camera != null) {  
 beats = 0;  
 filteredValue = 0;  
 signalWindow.clear();  
 bpmReadings.clear();  
 measuring = true;  
 startTime = System.*currentTimeMillis*();  
 camera.setPreviewCallback(this);  
 camera.startPreview();  
 startMeasurement();  
 }  
 });  
  
 ActivityCompat.*requestPermissions*(this,  
 new String[]{Manifest.permission.*CAMERA*},  
 100);  
 }  
  
 private void startMeasurement() {  
 handler.postDelayed(() -> {  
 measuring = false;  
 camera.setPreviewCallback(null);  
  
 long elapsedTime = System.*currentTimeMillis*() - startTime;  
 int bpm = (int) ((beats \* 60000.0) / elapsedTime);  
  
 if (bpm >= 0 && bpm <= 200) {  
 bpmReadings.add(bpm);  
 }  
 //calculateRespiratoryRate();  
 /\* if (!bpmReadings.isEmpty()) {  
 int avgBPM = 0;  
 for (int b : bpmReadings) avgBPM += b;  
 avgBPM /= bpmReadings.size();  
 tvHeartRate.setText("Heart Rate: " + (avgBPM+14) + " bpm");  
 } else {  
 tvHeartRate.setText("Try again: Unstable signal");  
 }\*/if(finalBPM<40&&*MEASUREMENT\_TIME\_MS*==20000)  
 {  
 AlertDialog.Builder builder=new AlertDialog.Builder(this);  
 builder.setTitle("Heart Rate too Low .");  
 builder.setMessage("Immediate Medical Attention needed");  
 builder.setPositiveButton("Open Map", (dialog, which) -> {  
 // Example location: New Delhi  
 String geoUri = "geo:0,0?q=hospitals";  
 Intent intent = new Intent(Intent.*ACTION\_VIEW*, Uri.*parse*(geoUri));  
 intent.setPackage("com.google.android.apps.maps"); // Optional: ensure Maps app opens  
 startActivity(intent);  
 });  
 builder.setNegativeButton("Cancel", (dialog, which) -> dialog.dismiss());  
 AlertDialog alertDialog=builder.create();  
  
 alertDialog.show();  
 }  
 if(temp<35.0f||temp>38.0f&&*MEASUREMENT\_TIME\_MS*==20000){  
 AlertDialog.Builder builder=new AlertDialog.Builder(this);  
 builder.setTitle("Fever is detected .");  
 builder.setMessage("Hospital Visit is recommended");  
 builder.setPositiveButton("Open Map", (dialog, which) -> {  
 // Example location: New Delhi  
 String geoUri = "geo:0,0?q=hospitals";  
 Intent intent = new Intent(Intent.*ACTION\_VIEW*, Uri.*parse*(geoUri));  
 intent.setPackage("com.google.android.apps.maps"); // Optional: ensure Maps app opens  
 startActivity(intent);  
 });  
 builder.setNegativeButton("Cancel", (dialog, which) -> dialog.dismiss());  
 AlertDialog alertDialog=builder.create();  
  
 alertDialog.show();}  
 }, *MEASUREMENT\_TIME\_MS*);  
  
  
 }  
  
  
  
 @Override  
 public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int height) {  
 try {  
 camera = Camera.*open*();  
 Camera.Parameters params = camera.getParameters();  
 params.setFlashMode(Camera.Parameters.*FLASH\_MODE\_TORCH*);  
 params.setPreviewFormat(ImageFormat.*NV21*);  
 Camera.Size previewSize = params.getSupportedPreviewSizes().get(0);  
 params.setPreviewSize(previewSize.width, previewSize.height);  
 camera.setParameters(params);  
 camera.setPreviewTexture(surface);  
 camera.setDisplayOrientation(90);  
 } catch (Exception e) {  
 Log.*e*("CAMERA", "Error setting up camera", e);  
 }  
 }  
  
 public int decodeRedFromYUV(byte[] data, int width, int height) {  
 int frameSize = width \* height;  
 long sumRed = 0;  
 int count = 0;  
  
 int startX = width / 2 - 5;  
 int startY = height / 2 - 5;  
  
 for (int y = startY; y < startY + 10; y++) {  
 for (int x = startX; x < startX + 10; x++) {  
 int yIndex = y \* width + x;  
  
 int yVal = data[yIndex] & 0xFF;  
  
 int uvIndex = frameSize + (y >> 1) \* width + (x & ~1);  
 int v = data[uvIndex] & 0xFF;  
 int u = data[uvIndex + 1] & 0xFF;  
  
 int c = yVal - 16;  
 int d = u - 128;  
 int e = v - 128;  
  
 int r = (int)(1.164 \* c + 1.596 \* e);  
 r = Math.*max*(0, Math.*min*(255, r));  
  
 sumRed += r;  
 count++;  
 }  
 }  
  
 return (int)(sumRed / count);  
 }  
  
 @Override  
 public void onPreviewFrame(byte[] data, Camera camera) {  
 if (!measuring) return;  
  
 Camera.Size size = camera.getParameters().getPreviewSize();  
 int width = size.width;  
 int height = size.height;  
  
 int redAvg = decodeRedFromYUV(data, width, height);  
 if (redAvg < 50) {  
 tvHeartRate.setText("Place finger properly");  
 return;  
 }  
  
  
 // Luminance brightness for PPG signal  
 long sum = 0;  
 for (int i = 0; i < width \* height; i++) {  
 sum += (data[i] & 0xFF);  
 }  
 int brightness = (int)(sum / (width \* height));  
  
 // Low-pass filter  
 filteredValue = (filteredValue == 0) ? brightness : (0.8f \* filteredValue + 0.2f \* brightness);  
  
 signalWindow.add((int) filteredValue);  
 if (signalWindow.size() > 300) signalWindow.removeFirst();  
  
 long currentTime = System.*currentTimeMillis*();  
 float temp=0f;  
 // Peak detection logic (slope-based)  
 if (filteredValue > previousValue && !isRising) {  
 isRising = true;  
 } else if (filteredValue < previousValue && isRising) {  
 isRising = false;  
 if (currentTime - lastBeatTime > 400) {  
 beats++;  
 if (lastBeatTime > 0) {  
  
 long interval = currentTime - lastBeatTime;  
 int currentBPM = (int)(60000.0 / interval);  
 if (currentBPM >= 40 && currentBPM <= 200) {  
 tvHeartRate.setText("Heart Rate: " + (int)(0.95\*currentBPM-2 ) + " bpm");  
 long rr = currentTime - lastBeatTime;  
 rrIntervals.add(rr);  
 temp = estimateTemperature((int)(0.95\*currentBPM-2));  
 TextView tvTemp = findViewById(R.id.*tv\_temperature*);  
 tvTemp.setText("Body Temp: " + String.*format*(Locale.*US*, "%.1f°C", temp));  
 tvRR.setText("RR: " + (int)(0.90\*((0.85\*currentBPM+5.3)/4)+1) + " bpm");  
 finalBPM=(int)(0.95\*currentBPM-2);  
  
  
 if (redSignalList.size() > 100) redSignalList.remove(0);  
 if (greenSignalList.size() > 100) greenSignalList.remove(0);  
 int[] avgRGB = decodeRGBfromYUV(data, width, height);  
 int redAvg1 = avgRGB[0];  
 int greenAvg1 = avgRGB[1];  
 redSignalList.add(redAvg1);  
 greenSignalList.add(greenAvg1);  
  
 if (redSignalList.size() > 200) {  
 int spO2 = estimateSpO2(redSignalList, greenSignalList);  
 TextView tvSpO2 = findViewById(R.id.*tv\_spo2*);  
 tvSpO2.setText("SpO₂: " + spO2 + "%");  
  
 // Clear for next batch  
 redSignalList.clear();  
 greenSignalList.clear();  
 }  
  
  
 }  
  
 } beatTimestamps.add(currentTime);  
 lastBeatTime = currentTime;  
 Log.*d*("PPG", "Beat detected at " + currentTime);  
  
  
  
  
 }  
 }  
  
 previousValue = filteredValue;  
  
 Log.*d*("PPG", "Brightness=" + brightness + ", Filtered=" + filteredValue);  
 if (!rrIntervals.isEmpty()) {  
 double sum1 = 0;  
 double mean = 0;  
 for (long rr : rrIntervals) mean += rr;  
 mean /= rrIntervals.size();  
  
 for (long rr : rrIntervals) {  
 double diff = rr - mean;  
 sum1 += diff \* diff;  
 }  
 double sdnn = Math.*sqrt*(sum1 / rrIntervals.size()); // HRV in ms  
  
 tvHRV.setText("HRV: " + (int) sdnn + " ms");  
 } else {  
 tvHRV.setText("HRV: -- ms");  
 }  
  
  
  
  
 }  
 private int estimateSpO2(List<Integer> redSignal, List<Integer> greenSignal) {  
 if (redSignal.size() < 100 || greenSignal.size() < 100) {  
 return 0; // Not enough data  
 }  
  
 // Calculate AC and DC components for red and green signals  
 double redAC = calculateACComponent(redSignal);  
 double redDC = calculateDCComponent(redSignal);  
 double greenAC = calculateACComponent(greenSignal);  
 double greenDC = calculateDCComponent(greenSignal);  
  
 if (redDC == 0 || greenDC == 0) {  
 return 0; // Prevent division by zero  
 }  
  
 // Ratio of ratios method  
 double ratio = (redAC / redDC) / (greenAC / greenDC);  
  
 // Estimate SpO2 using a commonly used empirical formula  
 int spO2 = (int) (110 - 25 \* ratio);  
  
 // Clamp result to realistic physiological range  
 spO2 = Math.*max*(70, Math.*min*(100, spO2));  
  
 return spO2;  
 }  
 private double calculateACComponent(List<Integer> signal) {  
 double mean = calculateDCComponent(signal);  
 double sum = 0;  
 for (int value : signal) {  
 sum += Math.*pow*(value - mean, 2);  
 }  
 return Math.*sqrt*(sum / signal.size()); // Standard deviation as AC  
 }  
 private double calculateDCComponent(List<Integer> signal) {  
 double sum = 0;  
 for (int value : signal) {  
 sum += value;  
 }  
 return sum / signal.size();  
 }  
  
 private int[] decodeRGBfromYUV(byte[] data, int width, int height) {  
 int frameSize = width \* height;  
 long sumRed = 0, sumGreen = 0;  
 int count = 0;  
  
 int startX = width / 2 - 5;  
 int startY = height / 2 - 5;  
  
 for (int y = startY; y < startY + 10; y++) {  
 for (int x = startX; x < startX + 10; x++) {  
 int yIndex = y \* width + x;  
  
 int yVal = data[yIndex] & 0xFF;  
 int uvIndex = frameSize + (y >> 1) \* width + (x & ~1);  
  
 int v = data[uvIndex] & 0xFF;  
 int u = data[uvIndex + 1] & 0xFF;  
  
 int c = yVal - 16;  
 int d = u - 128;  
 int e = v - 128;  
  
 int r = (int)(1.164 \* c + 1.596 \* e);  
 int g = (int)(1.164 \* c - 0.813 \* e - 0.391 \* d);  
  
 r = Math.*max*(0, Math.*min*(255, r));  
 g = Math.*max*(0, Math.*min*(255, g));  
  
 sumRed += r;  
 sumGreen += g;  
 count++;  
 }  
 }  
  
 return new int[]{(int)(sumRed / count), (int)(sumGreen / count)};  
 }  
 private float estimateTemperature(int bpm) {  
 float baseTemp = 36.5f; // average resting temperature  
 if (bpm < 90) return baseTemp;  
 if (bpm < 110) return baseTemp + 0.3f;  
 if (bpm < 130) return baseTemp + 0.6f;  
 return baseTemp + 1.0f;  
 }  
  
  
  
 @Override  
 public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {  
 if (camera != null) {  
 camera.setPreviewCallback(null);  
 camera.stopPreview();  
 camera.release();  
 }  
 return true;  
 }  
  
 @Override  
 public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width, int height) {}  
  
 @Override  
 public void onSurfaceTextureUpdated(SurfaceTexture surface) {}  
}