**NATIONAL INSTITUTE OF BUSINESS MANAGEMENT**

**SCHOOL OF COMPUTING AND ENGINEERING**

**HIGHER NATIONAL DIPLOMA IN SOFTWARE ENGINEERING**

**KANDY 24.1F**

**INTERNET OF THINGS SECOND PROGRESS REPORT**

**ADVACE HEALTH CARE MONITORING MONITORRING SYSTEM**

**GROUP NO - 10**

**SUBMITTED BY:**

**M.M.M. AMRY KAHDSE241F - 023**

**A.DHANUSHANANDAN KAHDSE241F - 028**

**M.A.M.AMMAR KAHDSE241F - 026**

**M.Z.F.ZEENA KAHDSE241F**  - **023**

A blue and white logo

Description automatically generated

**JANUARY 2025**

**Table of Contents**

[MECHANICAL DESIGN 2](#_Toc188257692)

[Product 3D Model 2](#_Toc188257693)

[BLYNK mobile App dashboard 3](#_Toc188257694)

[BLYNK WEB dashboard 4](#_Toc188257695)

[Firebase dashboard 4](#_Toc188257696)

[BOM (BILL OF AMOUNT) 6](#_Toc188257697)

[CIRCUIT DIAGRAM 7](#_Toc188257698)

[FLOW CHART 8](#_Toc188257699)

[TIMELINE (GANTT CHART) 9](#_Toc188257700)

[PROGRAMMING CODE 10](#_Toc188257701)

[Arduino Code 10](#_Toc188257702)

[NodeMcu (ESP8266) Code 28](#_Toc188257703)

[PROJECT GITHUB LINK 34](#_Toc188257704)

[References 34](#_Toc188257705)

# MECHANICAL DESIGN

We used Tinker cad website to create the 3D Design of our final Project product.

Its Easy to wearable like a watch for patients.

## A circuit board with wires and wires Description automatically generatedProduct 3D Model

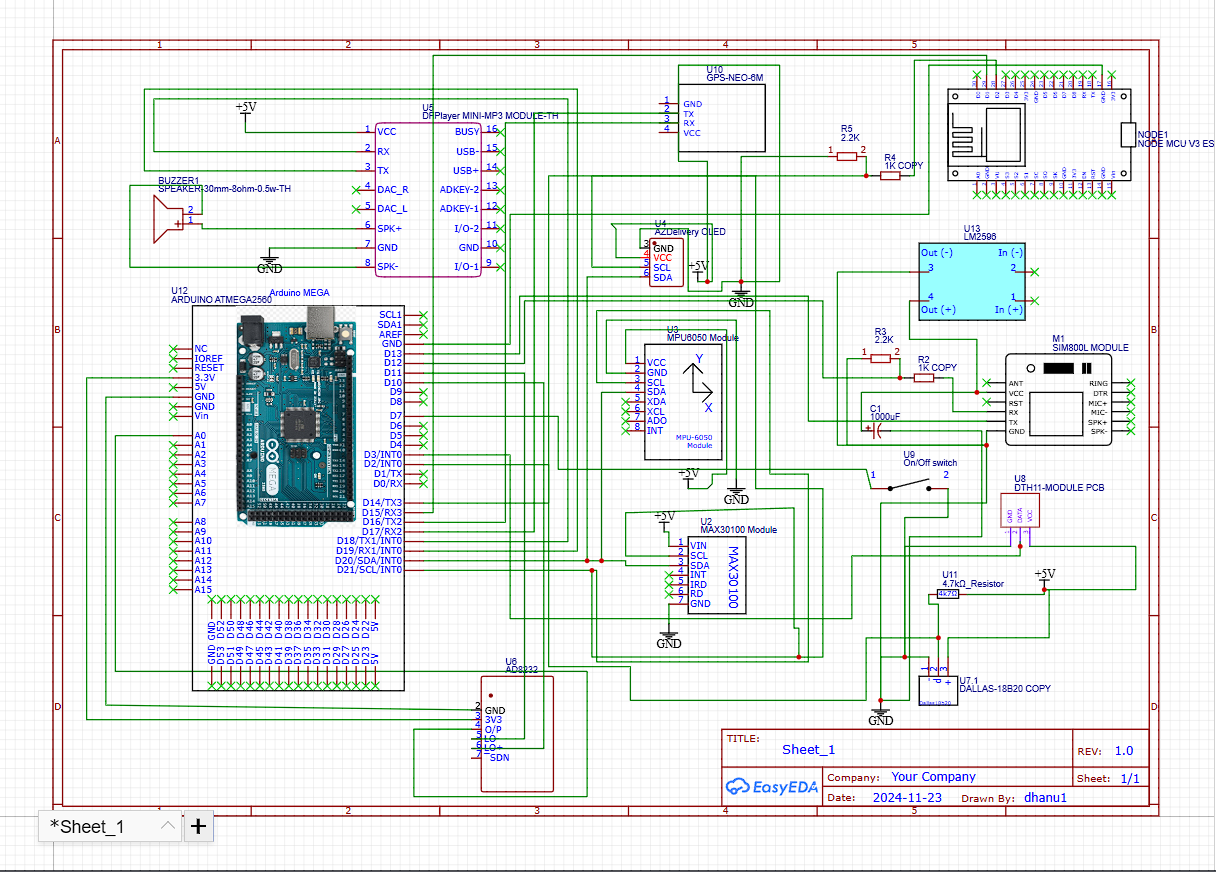
## BLYNK mobile App dashboardA screenshot of a device Description automatically generated

## BLYNK WEB dashboard

A screenshot of a computer

Description automatically generated

## A screenshot of a computer Description automatically generatedFirebase dashboard

Product Schematic Diagram

# BOM (BILL OF AMOUNT)

|  |  |  |
| --- | --- | --- |
| **Item Name** | **Qty** | **Price(LKR)** |
| Arduino mega | 1 | 5390 |
| SIM800L | 1 | 1255 |
| OLED **0.96-inch** Display | 1 | 780 |
| MAX30100 | 1 | 590 |
| ECG(AD8232) | 1 | 1750 |
| DF-Mini player | 1 | 525 |
| Speaker | 1 | 360 |
| Data cable | 1 | 290 |
| GPS-Module(NEO 6VM) | 1 | 1350 |
| DS18B20(waterproof temp) | 1 | 450 |
| MPU6050 | 1 | 595 |
| Switch | 1 | 75 |
| LM2596 | 1 | 280 |
| Clopper Clad board(FR4  Type) | 1 | 1350 |
| Ferric Chloride(FeCl3) | 1 | 170 |
| Sandpaper(100gsm) | 1 | 100 |
| Photo sheet | 1 | 200 |
| ESP8266 | 1 | 1570 |
| DTH11 | 1 | 590 |
| Other Items |  | 3500 |

# CIRCUIT DIAGRAM

A circuit board with many wires

Description automatically generated

# FLOW CHART

A diagram of a flowchart

Description automatically generated

# TIMELINE (GANTT CHART)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **WEEK**  **01** | **WEEK**  **02** | **WEEK**  **03** | **WEEK**  **04** | **WEEK**  **05** | **WEEK**  **06** | **WEEK**  **07** |
| **PLANNING** |  |  |  |  |  |  |  |
| Discuss the topic |  |  |  |  |  |  |  |
| **ANALYZING** |  |  |  |  |  |  |  |
| Identify components and gathering |  |  |  |  |  |  |  |
| **DESIGN** |  |  |  |  |  |  |  |
| Designing the prototype |  |  |  |  |  |  |  |
| **DEVELOPMENT** |  |  |  |  |  |  |  |
| Start to build the project |  |  |  |  |  |  |  |
| **IMPLEMENTATION** |  |  |  |  |  |  |  |
| Develop the project features |  |  |  |  |  |  |  |
| **SUBMIT THE**  **PROJECT REPORT** |  |  |  |  |  |  |  |

# PROGRAMMING CODE

## Arduino Code

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include "MAX30100\_PulseOximeter.h"

#include <MPU6050.h>

#include "DFRobotDFPlayerMini.h"

#include <OneWire.h>

#include <DallasTemperature.h>

#include <DHT.h>

#include <SoftwareSerial.h>

#include <TinyGPS++.h>

// OLED Display settings

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

#define OLED\_RESET -1

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);

// MAX30100 sensor settings

PulseOximeter pox;

#define REPORTING\_PERIOD\_MS 1000 // Report every 1 second

uint32\_t tsLastReport = 0;

// Blood pressure calibration constants

#define BASELINE\_SBP 120

#define BASELINE\_DBP 80

#define CALIBRATION\_CONSTANT\_SBP 0.3

#define CALIBRATION\_CONSTANT\_DBP 0.1

#define MAX\_READINGS 20

int heartRateBuffer[MAX\_READINGS] = { 0 };

int bufferIndex = 0;

int bufferCount = 0;

// MPU6050 Fall Detection settings

MPU6050 mpu;

#define FALL\_THRESHOLD 200

#define STABILIZATION\_THRESHOLD 20

#define STABILIZATION\_TIME 1000

bool fallDetected = false;

unsigned long stabilizationStartTime = 0;

bool fingerDetected = false;

#define FPSerial Serial1

DFRobotDFPlayerMini myDFPlayer;

// Data wire is plugged into port 2 on the Arduino

#define ONE\_WIRE\_BUS 2

#define TEMPERATURE\_PRECISION 9 // Lower resolution

// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)

OneWire oneWire(ONE\_WIRE\_BUS);

// Pass our oneWire reference to Dallas Temperature.

DallasTemperature sensors(&oneWire);

int numberOfDevices; // Number of temperature devices found

DeviceAddress tempDeviceAddress; // We'll use this variable to store a found device address

// Button switch input pin

#define BUTTON\_PIN 7 // Define the button pin

// DHT11 sensor settings

#define DHTPIN 3 // Pin connected to the DHT11 sensor data pin

#define DHTTYPE DHT11 // Define the type of DHT sensor

DHT dht(DHTPIN, DHTTYPE);

// Define the threshold values for abnormal readings (adjust as needed)

#define ABNORMAL\_HEART\_RATE\_LOW 40

#define ABNORMAL\_HEART\_RATE\_HIGH 200

#define ABNORMAL\_SPO2\_LOW 70

#define ABNORMAL\_SPO2\_HIGH 100

#define ABNORMAL\_TEMP\_LOW 33.0

#define ABNORMAL\_TEMP\_HIGH 38.5

#define ABNORMAL\_ECG\_THRESHOLD 600 // Threshold for abnormal ECG reading (adjust as needed)

#define ABNORMAL\_ECG\_THRESHOLD\_LOW 300

#define ABNORMAL\_SystolicBP\_HIGH 180

#define ABNORMAL\_SystolicBP\_LOW 100

#define ABNORMAL\_DiastolicBP\_HIGH 120

#define ABNORMAL\_DiastolicBP\_LOW 70

// Define the Serial Pins for SIM800L

#define SIM800\_TX 12

#define SIM800\_RX 13

// Initialize the SIM800L module

SoftwareSerial sim800Serial(SIM800\_RX, SIM800\_TX);

// Emergency phone number

#define EMERGENCY\_PHONE "+94752051204"

// Emergency phone number hospital

#define EMERGENCY\_PHONE2 "+94712051203"

const int ecgPin = A0; // Connect to the OUT pin of AD8232

const int loPlusPin = 10; // Connect to LO+ pin of AD8232 (optional)

const int loMinusPin = 11; // Connect to LO- pin of AD8232 (optional)

static const int RXPin = 22, TXPin = 23; // RX and TX pins for GPS

static const uint32\_t GPSBaud = 9600; // Change to 9600 for better compatibility with SoftwareSerial

static const unsigned long timeout = 5000; // 30 seconds timeout for GPS

TinyGPSPlus gps; // Create an instance of the TinyGPSPlus object

// SoftwareSerial ss(17, 16); // Set up SoftwareSerial on pins 22 (RX) and 23 (TX)

// Callback for MAX30100 on beat detection

void onBeatDetected() {

fingerDetected = true;

}

// Function to reset for a new blood pressure reading

void resetForNextReading() {

fingerDetected = false;

bufferIndex = 0;

bufferCount = 0;

memset(heartRateBuffer, 0, sizeof(heartRateBuffer));

if (!pox.begin()) {

Serial.println("FAILED to reinitialize MAX30100 sensor");

return;

}

pox.setIRLedCurrent(MAX30100\_LED\_CURR\_24MA);

display.clearDisplay();

display.setCursor(0, 0);

display.println("Place your finger");

display.println(" & other sensors");

display.println("to start reading.");

display.display();

Serial.println("Place your finger & other sensors to start reading.");

}

// Function to calculate and display average blood pressure

void calculateAndDisplayAverage() {

if (bufferCount == 0) {

Serial.println("No valid readings captured.");

display.clearDisplay();

display.setCursor(0, 0);

display.println("No valid readings.");

display.println("Try again.");

display.display();

return;

}

int totalHeartRate = 0;

for (int i = 0; i < bufferCount; i++) {

totalHeartRate += heartRateBuffer[i];

}

int averageHeartRate = totalHeartRate / bufferCount;

float averageSystolicBP = BASELINE\_SBP + CALIBRATION\_CONSTANT\_SBP \* averageHeartRate;

float averageDiastolicBP = BASELINE\_DBP + CALIBRATION\_CONSTANT\_DBP \* averageHeartRate;

float averageSpO2 = pox.getSpO2();

display.clearDisplay();

display.setCursor(0, 0);

display.print("Avg Heart Rate: ");

display.print(averageHeartRate);

display.println(" bpm");

display.print("Avg SpO2: ");

display.print(averageSpO2, 1);

display.println(" %");

display.print("Avg Sys BP: ");

display.print(averageSystolicBP, 1);

display.println(" mmHg");

display.print("Avg Dia BP: ");

display.print(averageDiastolicBP, 1);

display.println(" mmHg");

display.display();

Serial.println("Final Average Readings:");

Serial.print("Heart Rate: ");

Serial.print(averageHeartRate);

Serial.println(" bpm");

Serial.print("SpO2: ");

Serial.print(averageSpO2, 1);

Serial.println(" %");

Serial.print("Systolic BP: ");

Serial.print(averageSystolicBP, 1);

Serial.println(" mmHg");

Serial.print("Diastolic BP: ");

Serial.print(averageDiastolicBP, 1);

Serial.println(" mmHg");

delay(2000);

// Post-reading instruction

display.clearDisplay();

display.setCursor(0, 0);

display.println("Remove your finger.");

display.println("Place it again for");

display.println("a new reading.");

display.display();

Serial.println("Remove your finger. Place it again for a new reading");

printTemperature(averageHeartRate, averageSpO2, averageSystolicBP, averageDiastolicBP);

}

// Initialize MPU6050

void initFallDetection() {

Serial.println("Initialize MPU6050");

while (!mpu.begin(MPU6050\_SCALE\_2000DPS, MPU6050\_RANGE\_2G)) {

Serial.println("Could not find a valid MPU6050 sensor, check wiring!");

delay(500);

}

mpu.calibrateGyro();

mpu.setThreshold(3);

Serial.println("MPU6050 initialized successfully");

}

// Function to check for falls

void checkForFalls() {

Vector normGyro = mpu.readNormalizeGyro();

float totalGyro = sqrt(normGyro.XAxis \* normGyro.XAxis + normGyro.YAxis \* normGyro.YAxis + normGyro.ZAxis \* normGyro.ZAxis);

// Only check for falls if the button is not pressed (if the switch is off).

if (!fallDetected && totalGyro > FALL\_THRESHOLD) {

fallDetected = true;

Serial.println("ALERT: Fall detected!");

display.clearDisplay();

display.setCursor(0, 0);

display.println("Fall Detected!");

display.display();

myDFPlayer.play(1);

// Play the audio for 1 second if the button is pressed

if (digitalRead(BUTTON\_PIN) == HIGH) {

while (digitalRead(BUTTON\_PIN) == LOW) {

myDFPlayer.play(1); // Play the first track (0001.mp3)

delay(1000); // Play audio for 1 second

}

}

sendSMS(EMERGENCY\_PHONE, "Emergency Alert: Fall Detected!");

stabilizationStartTime = millis();

}

// If a fall is detected, check if the user has stabilized.

if (fallDetected) {

if (totalGyro < STABILIZATION\_THRESHOLD) {

if (millis() - stabilizationStartTime >= STABILIZATION\_TIME) {

fallDetected = false;

Serial.println("Stabilization complete. Fall state cleared.");

display.clearDisplay();

display.setCursor(0, 0);

display.println("Stabilized.");

display.println("Place your finger");

display.println("to start reading.");

display.display();

myDFPlayer.stop(); // Stop the music only after stabilization

resetForNextReading();

}

} else {

stabilizationStartTime = millis();

}

}

}

// int ECGcalculation() {

// int ecgValue = analogRead(ecgPin); // Read the ECG signal from A0

// // Lead-off detection

// int loPlusStatus = digitalRead(loPlusPin);

// int loMinusStatus = digitalRead(loMinusPin);

// if (loPlusStatus == 1 || loMinusStatus == 1) {

// Serial.println("Lead off detected!");

// return 0;

// } else {

// // Output the ECG value

// Serial.println(ecgValue);

// // Send values to NodeMCU using Serial3

// Serial3.print("ECG: ");

// Serial3.print(ecgValue);

// return ecgValue;

// }

// delay(10); // Small delay for smoother serial output

// }

int ECGcalculation() {

int ecgValue = analogRead(ecgPin); // Read the ECG signal from A0

// Lead-off detection

int loPlusStatus = digitalRead(loPlusPin);

int loMinusStatus = digitalRead(loMinusPin);

if (loPlusStatus == HIGH || loMinusStatus == HIGH) {

Serial.println("Lead off detected!");

// Send lead-off status to NodeMCU

// Serial3.println("ECG: Lead off detected!");

return -1; // Return -1 to indicate lead-off

} else {

// Output the ECG value

Serial.println(ecgValue);

// Send values to NodeMCU using Serial3

// Serial3.print("ECG: ");

// Serial3.println(ecgValue);

return ecgValue;

}

}

// function to print the temperature for a device

void printTemperature(int averageHeartRate, float averageSpO2, float averageSystolicBP, float averageDiastolicBP) {

// Request temperatures from DallasTemperature sensor

Serial.print("Requesting temperatures Hold Sensor...");

display.clearDisplay();

display.setCursor(0, 0);

display.print("Requesting temperatures");

display.print(" Hold Sensor...");

display.display();

delay(10000);

sensors.requestTemperatures(); // Send the command to get temperatures

Serial.println("DONE");

float tempC = sensors.getTempC(tempDeviceAddress);

if (tempC == DEVICE\_DISCONNECTED\_C) {

Serial.println("Error: Could not read temperature data");

return;

}

Serial.print("Temp C: ");

Serial.print(tempC);

Serial.print(" Temp F: ");

Serial.println(DallasTemperature::toFahrenheit(tempC)); // Converts tempC to Fahrenheit

// Send values to NodeMCU using Serial3

Serial3.print("Body Temp: ");

Serial3.print(tempC);

// Serial3.print(", Room Temp: ");

// Read room temperature and humidity from DHT11 sensor

float temp = dht.readTemperature(); // Celsius temperature

float hum = dht.readHumidity(); // Humidity

// Check if the readings are valid

if (isnan(temp) || isnan(hum)) {

Serial.println("Failed to read from DHT sensor!");

} else {

Serial.print("Room Temperature: ");

Serial.print(temp);

Serial.println(" °C");

Serial.print("Humidity: ");

Serial.print(hum);

Serial.println(" %");

// Send data to NodeMCU over Serial3

Serial3.print(",Room Temp: ");

Serial3.print(temp);

Serial3.print(", Humidity: ");

Serial3.println(hum);

}

// Send more sensor data (e.g., averageHeartRate, averageSpO2, averageSystolicBP, averageDiastolicBP)

Serial3.print("HeartRate: ");

Serial3.print(averageHeartRate);

Serial3.print(",SpO2: ");

Serial3.print(averageSpO2);

Serial3.print(",SystolicBP: ");

Serial3.print(averageSystolicBP);

Serial3.print(",DiastolicBP: ");

Serial3.println(averageDiastolicBP);

// Check for abnormal readings

checkForAbnormalReadings(averageHeartRate, averageSpO2, tempC, averageSystolicBP, averageDiastolicBP);

delay(5000);

}

// function to print a device address

void printAddress(DeviceAddress deviceAddress) {

for (uint8\_t i = 0; i < 8; i++) {

if (deviceAddress[i] < 16) Serial.print("0");

Serial.print(deviceAddress[i], HEX);

}

}

// Function to handle the button press and play sound continuously while pressed

void handleButtonPress() {

static bool lastButtonState = LOW;

bool currentButtonState = digitalRead(BUTTON\_PIN); // Read the state of the button

bool gpsx = false;

if (lastButtonState == LOW && currentButtonState == HIGH) { // Button pressed

Serial.println("Emergency button pressed. Sending SMS...");

myDFPlayer.play(1);

sendSMS(EMERGENCY\_PHONE, "Emergency Alert: Immediate assistance needed!");

delay(1000);

sendSMS(EMERGENCY\_PHONE2, "Emergency Alert: Immediate assistance needed!");

delay(1000);

// Start time tracking to check for 60 seconds timeout

unsigned long startMillis = millis();

// Check GPS data for 60 seconds

while (millis() - startMillis < timeout) {

if (Serial2.available() > 0) {

char incomingByte = Serial2.read();

Serial.print("Received byte: ");

Serial.println(incomingByte, DEC); // Print the raw byte received from GPS

if (gps.encode(incomingByte)) // Decode the GPS data

{

if (gps.location.isValid()) // Check if location is valid

{

Serial.println("Sending Location SMS...");

sendLocationViaSMS(); // Send valid GPS location via SMS

// return; // Exit after sending valid location

gpsx = true;

}

}

} else {

Serial.println("Waiting for GPS data..."); // Added to check if the GPS is sending anything

}

}

// If no valid location is found after 30 seconds, send the default location via SMS

// sendSMS2("Location not found. Sending default location.");

if (gpsx == false) {

sendDefaultLocationViaSMS();

}

myDFPlayer.stop();

display.clearDisplay();

display.setCursor(0, 0);

display.println("Stabilized.");

display.println("Place your finger");

display.println("to start reading.");

display.display();

myDFPlayer.stop(); // Stop the music only after stabilization

resetForNextReading();

} else if (lastButtonState == HIGH && currentButtonState == LOW) { // Button released

myDFPlayer.stop(); // Stop the music

}

lastButtonState = currentButtonState; // Update the last state

}

void sendLocationViaSMS() {

String location = "Live Location: " + String(gps.location.lat(), 6) + ", " + String(gps.location.lng(), 6);

String message = location + " Date: " + String(gps.date.month()) + "/" + String(gps.date.day()) + "/" + String(gps.date.year());

sendSMS2(message);

displayInfo();

}

void sendDefaultLocationViaSMS() {

// Sending the default location via SMS if no GPS data is found

String defaultLocation = "Location: 7.299093, 80.634076 ";

sendSMS2(defaultLocation);

displayDefaultLocation();

}

void sendSMS2(String message) {

Serial.println("Sending Location SMS...");

sim800Serial.println("AT"); // Test the connection

delay(1000);

sim800Serial.println("AT+CMGF=1"); // Set SMS text mode

delay(1000);

sim800Serial.println("AT+CMGS=\"+94712051203\""); // Recipient phone number

delay(1000);

sim800Serial.println(message); // The message to send

delay(1000);

sim800Serial.write(26); // ASCII code for Ctrl+Z (End of message)

delay(5000); // Give some time for SMS to send

Serial.println("Sending SMS Completed");

}

void displayInfo() {

Serial.print(F("Location: "));

Serial.print(gps.location.lat(), 6); // Latitude with 6 decimal places

Serial.print(F(", "));

Serial.print(gps.location.lng(), 6); // Longitude with 6 decimal places

Serial.print(F(" Date: "));

if (gps.date.isValid()) {

Serial.print(gps.date.month());

Serial.print(F("/"));

Serial.print(gps.date.day());

Serial.print(F("/"));

Serial.print(gps.date.year());

} else {

Serial.print(F("INVALID"));

}

Serial.println(); // New line

}

void displayDefaultLocation() {

// Display the default location if no GPS signal is found

Serial.print(F("Location: 7.299093, 80.634076 Date: INVALID"));

Serial.println(); // New line

}

// Function to check if the readings are abnormal

void checkForAbnormalReadings(float heartRate, float spo2, float tempC, float averageSystolicBP, float averageDiastolicBP) {

bool abnormal = false;

// Check if heart rate is abnormal

if (heartRate < ABNORMAL\_HEART\_RATE\_LOW || heartRate > ABNORMAL\_HEART\_RATE\_HIGH) {

abnormal = true;

Serial.println("Abnormal Heart Rate!");

}

// Check if SpO2 is abnormal

if (spo2 < ABNORMAL\_SPO2\_LOW || spo2 > ABNORMAL\_SPO2\_HIGH) {

abnormal = true;

Serial.println("Abnormal SpO2!");

}

// Check if body temperature is abnormal

if (tempC < ABNORMAL\_TEMP\_LOW || tempC > ABNORMAL\_TEMP\_HIGH) {

abnormal = true;

Serial.println("Abnormal Body Temperature!");

}

int ecgvalue = ECGcalculation();

// Check if ECG reading is abnormal

if (ecgvalue > ABNORMAL\_ECG\_THRESHOLD || ecgvalue == ABNORMAL\_ECG\_THRESHOLD\_LOW) {

abnormal = true;

Serial.println("Abnormal ECG reading!");

}

if (averageSystolicBP < ABNORMAL\_SystolicBP\_LOW || averageSystolicBP > ABNORMAL\_SystolicBP\_HIGH) {

abnormal = true;

Serial.println("Abnormal Body ABNORMAL\_SystolicBP!");

}

if (averageDiastolicBP < ABNORMAL\_DiastolicBP\_LOW || averageDiastolicBP > ABNORMAL\_DiastolicBP\_HIGH) {

abnormal = true;

Serial.println("Abnormal Body ABNORMAL\_DiastolicBP!");

}

// If any reading is abnormal, play the sound

if (abnormal) {

myDFPlayer.play(1); // Play a specific sound (e.g., alert sound)

display.clearDisplay();

display.setCursor(0, 0);

display.println("Abnormal readings detected!");

display.display();

sendSMS(EMERGENCY\_PHONE, "Emergency Alert: Abnormal readings detected!");

delay(2000);

myDFPlayer.stop();

}

}

// Function to send SMS using the SIM800L module

void sendSMS(const char\* phoneNumber, const char\* message) {

sim800Serial.println("AT"); // Test if the SIM800L is responding

delay(1000);

sim800Serial.println("AT+CMGF=1"); // Set SMS mode to text

delay(1000);

sim800Serial.print("AT+CMGS=\""); // Command to send SMS

sim800Serial.print(phoneNumber); // Phone number

sim800Serial.println("\"");

delay(1000);

sim800Serial.println(message); // Message content

delay(1000);

sim800Serial.write(26); // ASCII code for Ctrl+Z to send the message

delay(5000); // Wait for the message to be sent

Serial.println("SMS sent successfully!");

}

// Function to check if SIM800L is initialized

bool checkSIM800L() {

sim800Serial.println("AT"); // Test if the SIM800L is responding

delay(1000);

// Check for the "OK" response

if (sim800Serial.available()) {

String response = sim800Serial.readString();

if (response.indexOf("OK") != -1) {

return true; // SIM800L is initialized and responding

}

}

return false; // SIM800L did not respond correctly

}

void setup() {

FPSerial.begin(9600); // Initialize the serial communication with DFPlayer Mini

Serial.begin(115200);

pinMode(loPlusPin, INPUT); // Configure LO+ as input

pinMode(loMinusPin, INPUT); // Configure LO- as input

// Initialize button pin

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {

Serial.println("SSD1306 allocation failed");

for (;;)

;

}

display.clearDisplay();

display.setTextSize(1);

display.setTextColor(SSD1306\_WHITE);

if (!pox.begin()) {

Serial.println("FAILED to initialize MAX30100 sensor");

display.println("FAILED to initialize MAX30100 sensor");

display.display();

for (;;)

;

}

pox.setIRLedCurrent(MAX30100\_LED\_CURR\_24MA);

pox.setOnBeatDetectedCallback(onBeatDetected);

// Initialize DFPlayer Mini

if (!myDFPlayer.begin(FPSerial, /\*isACK = \*/ true, /\*doReset = \*/ true)) {

Serial.println(F("Unable to begin! Please check the connection and SD card."));

display.println("Unable to begin! Please ");

display.println("check the connection and SD card.");

display.display();

while (true) { delay(0); } // Halt the program if DFPlayer is not detected

}

Serial.println(F("DFPlayer Mini initialized."));

myDFPlayer.volume(10); // Set the volume (0 to 30)

// Start up the library

sensors.begin();

// Grab a count of devices on the wire

numberOfDevices = sensors.getDeviceCount();

// locate devices on the bus

Serial.print("Locating devices...");

Serial.print("Found ");

Serial.print(numberOfDevices, DEC);

Serial.println(" devices.");

// report parasite power requirements

Serial.print("Parasite power is: ");

if (sensors.isParasitePowerMode()) Serial.println("ON");

else Serial.println("OFF");

// Loop through each device, print out address

for (int i = 0; i < numberOfDevices; i++) {

// Search the wire for address

if (sensors.getAddress(tempDeviceAddress, i)) {

Serial.print("Found device ");

Serial.print(i, DEC);

Serial.print(" with address: ");

printAddress(tempDeviceAddress);

Serial.println();

Serial.print("Setting resolution to ");

Serial.println(TEMPERATURE\_PRECISION, DEC);

// set the resolution to TEMPERATURE\_PRECISION bit (Each Dallas/Maxim device is capable of several different resolutions)

sensors.setResolution(tempDeviceAddress, TEMPERATURE\_PRECISION);

Serial.print("Resolution actually set to: ");

Serial.print(sensors.getResolution(tempDeviceAddress), DEC);

Serial.println();

} else {

Serial.print("Found ghost device at ");

Serial.print(i, DEC);

Serial.print(" but could not detect address. Check power and cabling");

}

}

dht.begin(); // Initialize DHT sensor

Serial.println("DHT11 Sensor Initialized");

sim800Serial.begin(9600); // Start communication with SIM800L module

delay(10000);

// Test the SIM800L with a simple AT command

sim800Serial.println("AT"); // Send AT command

delay(1000); // Wait for a response

if (sim800Serial.available()) {

String response = sim800Serial.readString();

Serial.println("SIM800L Response: " + response);

} else {

Serial.println("SIM800L did not respond.");

}

// Check if the SIM800L module is responding

if (checkSIM800L()) {

Serial.println("SIM800L initialized successfully.");

} else {

Serial.println("SIM800L initialization failed.");

}

Serial2.begin(GPSBaud); // Start communication with GPS at the defined baud rate

Serial.println("GPS initialization.");

// while (FPSerial.available()) {

// char c = FPSerial.read();

// Serial.print(c); // Print raw NMEA data to Serial Monitor

// }

// Wait for GPS to send valid data

// if (Serial2.available() > 0) {

// Serial.println("\nNeo-6M GPS Module initialized successfully!");

// } else {

// Serial.println("\nNeo-6M GPS Module initialization failed!");

// }

// Serial3.begin(115200); // For communication with NodeMCU (Serial3 uses pins 14 and 15)

// // Start time tracking to check for 60 seconds timeout

// unsigned long startMillis = millis();

// // Check GPS data for 60 seconds

// while (millis() - startMillis < 50000) {

// if (Serial3.available() > 0) {

// Serial3.println("Hello from Arduino Mega!");

// Serial.println("Hello from Arduino Mega!");

// } else {

// Serial.println("Hello");

// }

// }

Serial3.begin(9600); // Serial3 test

Serial.println("Starting Serial3 Loopback Test");

Serial3.println("Hello from Mega!"); // Send test data

Serial.println("Sent to NodeMCU: Hello from Mega!");

delay(1000); // Delay to slow down communication for debugging

if (Serial3.available()) {

String response = Serial3.readStringUntil('\n');

Serial.println("Received from NodeMCU: " + response);

} else {

Serial.println("No response from NodeMCU");

}

delay(1000); // Slow down for debugging

initFallDetection();

resetForNextReading();

}

void loop() {

pox.update();

checkForFalls();

handleButtonPress(); // Continuously check for button presses

static unsigned long startTime = 0;

static bool waitingForReadings = false;

if (fingerDetected && !waitingForReadings) {

waitingForReadings = true;

startTime = millis();

display.clearDisplay();

display.setCursor(0, 0);

display.println("Collecting readings...");

display.println("Keep steady.");

display.display();

Serial.println("Collecting Readings.....Keep steady.");

}

if (waitingForReadings) {

if (millis() - startTime <= 10000) { // Collect readings for 10 seconds

float heartRate = pox.getHeartRate();

float spo2 = pox.getSpO2();

float tempC = sensors.getTempC(tempDeviceAddress); // Get body temperature

int ecgReading = ECGcalculation(); // Get the ECG reading

if (heartRate > 40 && heartRate < 200 && spo2 > 70 && spo2 < 100) {

heartRateBuffer[bufferIndex] = round(heartRate);

bufferIndex = (bufferIndex + 1) % MAX\_READINGS;

if (bufferCount < MAX\_READINGS) bufferCount++;

}

} else {

waitingForReadings = false;

calculateAndDisplayAverage();

resetForNextReading();

}

}

// ECGcalculation();

}

## NodeMcu (ESP8266) Code

#define BLYNK\_TEMPLATE\_ID "TMPL6BcIXK908"

#define BLYNK\_TEMPLATE\_NAME "IoT Project"

#define BLYNK\_AUTH\_TOKEN "3PLpTKpeBSAiO7\_7SZXcuOiCGAlKKetZ"

#include <ESP8266WiFi.h>

#include <BlynkSimpleEsp8266.h>

#include <SoftwareSerial.h>

#include <Firebase\_ESP\_Client.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#include <TimeLib.h>

#include "addons/TokenHelper.h"

#include "addons/RTDBHelper.h"

// Wi-Fi credentials

char ssid[] = "DARK PHOENIX";

char pass[] = "123asd07a";

// Firebase setup

#define API\_KEY "AIzaSyASOLXA-khPFKOGeMqyfR\_c8moY\_PAhcnY"

#define DATABASE\_URL "iot-project-1639f-default-rtdb.asia-southeast1.firebasedatabase.app/"

FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;

FirebaseJson dataJson;

bool signupOK = false;

// Virtual pins for Blynk

#define V1 1 // Heart Rate

#define V2 2 // SpO2

#define V3 3 // Systolic BP

#define V4 4 // Diastolic BP

#define V5 5 // Body Temp

#define V6 6 // Room Temp

#define V7 7 // Humidity

#define V8 8 // ECG

// SoftwareSerial for Mega communication

#define NODEMCU\_TX D1

#define NODEMCU\_RX D2

SoftwareSerial MegaSerial(NODEMCU\_RX, NODEMCU\_TX);

// NTP setup

WiFiUDP udp;

NTPClient timeClient(udp, "pool.ntp.org", 0, 3600000);

unsigned long lastReceivedTime = 0;

const unsigned long noDataInterval = 5000;

void setup() {

// Serial communication for debugging

Serial.begin(9600);

MegaSerial.begin(9600);

// Blynk and Wi-Fi initialization

Blynk.begin(BLYNK\_AUTH\_TOKEN, ssid, pass);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println("\nConnected to Wi-Fi");

// Firebase initialization

config.api\_key = API\_KEY;

config.database\_url = DATABASE\_URL;

if (Firebase.signUp(&config, &auth, "", "")) {

signupOK = true;

Serial.println("Firebase signup successful");

} else {

Serial.printf("Firebase signup failed: %s\n", config.signer.signupError.message.c\_str());

}

config.token\_status\_callback = tokenStatusCallback;

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

// NTP initialization

timeClient.begin();

timeClient.setTimeOffset(19800); // Adjust for UTC+5:30

}

void loop() {

// Sync time

timeClient.update();

setTime(timeClient.getEpochTime());

String currentTime = String(hour()) + ":" + String(minute()) + ":" + String(second());

String currentDate = String(day()) + "/" + String(month()) + "/" + String(year());

if (MegaSerial.available()) {

String receivedData = MegaSerial.readStringUntil('\n');

Serial.println("Received from Mega: " + receivedData);

lastReceivedTime = millis();

// Parse and handle data

if (receivedData.indexOf("HeartRate:") != -1) {

String heartRate = receivedData.substring(receivedData.indexOf("HeartRate:") + 10, receivedData.indexOf(",SpO2"));

int heartRateValue = heartRate.toInt();

Blynk.virtualWrite(V1, heartRateValue);

dataJson.add("heartRate", heartRateValue);

}

if (receivedData.indexOf("SpO2:") != -1) {

String spo2 = receivedData.substring(receivedData.indexOf("SpO2:") + 5, receivedData.indexOf(",SystolicBP"));

double spo2Value = spo2.toFloat();

Blynk.virtualWrite(V2, spo2Value);

dataJson.add("spo2", spo2Value);

}

if (receivedData.indexOf("SystolicBP:") != -1) {

String systolicBP = receivedData.substring(receivedData.indexOf("SystolicBP:") + 11, receivedData.indexOf(",DiastolicBP"));

double systolicBPValue = systolicBP.toFloat();

Blynk.virtualWrite(V3, systolicBPValue);

dataJson.add("systolicBP", systolicBPValue);

}

if (receivedData.indexOf("DiastolicBP:") != -1) {

String diastolicBP = receivedData.substring(receivedData.indexOf("DiastolicBP:") + 12);

double diastolicBPValue = diastolicBP.toFloat();

Blynk.virtualWrite(V4, diastolicBPValue);

dataJson.add("diastolicBP", diastolicBPValue);

}

if (receivedData.indexOf("Body Temp:") != -1) {

String bodyTemp = receivedData.substring(receivedData.indexOf("Body Temp:") + 10, receivedData.indexOf(","));

double bodyTempValue = bodyTemp.toFloat();

Blynk.virtualWrite(V5, bodyTempValue);

dataJson.add("bodyTemp", bodyTempValue);

}

if (receivedData.indexOf("Room Temp:") != -1) {

String roomTemp = receivedData.substring(receivedData.indexOf("Room Temp:") + 10, receivedData.indexOf(", Humidity"));

double roomTempValue = roomTemp.toFloat();

Blynk.virtualWrite(V6, roomTempValue);

dataJson.add("roomTemp", roomTempValue);

}

if (receivedData.indexOf("Humidity:") != -1) {

String humidity = receivedData.substring(receivedData.indexOf("Humidity:") + 9);

double humidityValue = humidity.toFloat();

Blynk.virtualWrite(V7, humidityValue);

dataJson.add("humidity", humidityValue);

}

// Add time and date to Firebase JSON

dataJson.add("time", currentTime);

dataJson.add("date", currentDate);

// Push data to Firebase

if (Firebase.RTDB.pushJSON(&fbdo, "SensorData", &dataJson)) {

Serial.println("Data sent to Firebase");

} else {

Serial.println("Failed to send data to Firebase: " + fbdo.errorReason());

}

dataJson.clear(); // Clear the JSON object for the next iteration

}

if (millis() - lastReceivedTime > noDataInterval) {

Serial.println("No data received from Arduino Mega in the last 10 seconds");

lastReceivedTime = millis();

}

Blynk.run();

}

# PROJECT GITHUB LINK

<https://github.com/Dhanushanandan/IOT-AHMS.git>

# References

*BLYNK*. (n.d.). Retrieved from https://blynk.cloud/dashboard/642144/global/devices/1/organization/642144/devices/2910482/dashboard

firebase. (n.d.). Retrieved from https://console.firebase.google.com/u/1/project/iot-project-1639f/database/iot-project-1639f-default-rtdb/data

geeksforgeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/

*github*. (n.d.). Retrieved from https://github.com/Dhanushanandan/IOT-AHMS

*stackoverflow*. (n.d.). Retrieved from https://stackoverflow.com/

*youtube*. (n.d.). Retrieved from https://youtu.be/98Ph6cvq6U8?feature=shared

*youtube*. (n.d.). Retrieved from https://youtu.be/BfUtpScdQ9Y?feature=shared

*Youtube*. (n.d.). Retrieved from https://youtu.be/lIpgGru2Wv0?feature=shared