Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN3251: Internet of Things

Course Project: IoT-Based Health Monitoring System

Group 16

Anjula M.K	210368V
Shamal G.B.A	210599E
Thennakoon T.M.K.R	210642G

Content

1.Introduction 3

2.Problem Statement 3

3.Our Solution 3

4.Functional Block Diagrams 4

5.Live Health Hub 4

6.Hardware Equipment 5

a.MAX30102 Oximeter Heart Rate Beat Pulse Sensor 5

b.Human Body Temperature Sensor MAX30205 MTA 5

c.Node MCU ESP-32 5

7.Product Outcome 6

8.Threshold values used in Dashboard 6

9.Challenges that can be Faced in the Project 7

10.Future Works 7

11.Code work 8

a. Temperature sensor Arduino code 8

b. Oximeter Heart Rate Beat Pulse Sensor Arduino code 9

c.16x2 Blue Backlight LCD Display Arduino code 12

d. MQTT Publisher Code 14

1.Introduction

In today's rapidly advancing world, the demand for continuous and efficient healthcare solutions has never been more critical. The ability to monitor vital signs like blood oxygen levels (SpO2), heart rate, and body temperature in real time is essential for providing optimal care, especially for individuals with chronic illnesses, the elderly, and those in critical conditions. Our IoT-based health monitoring system is designed to meet this pressing need by offering an accessible, reliable, and user-friendly platform. This innovative system enables patients to effortlessly track their health from the comfort of their homes, while healthcare professionals can remotely monitor and assess their condition. By reducing the frequency of hospital visits, this solution not only alleviates the strain on healthcare systems but also allows for proactive interventions, ensuring that patients receive timely, personalized care when it matters most. Our project aims to revolutionize the way healthcare is delivered, bridging the gap between patients and healthcare providers through seamless, real-time monitoring and communication.

2.Problem Statement

Healthcare systems often face challenges in providing continuous monitoring for patients, particularly those with chronic conditions or recovering from surgery. Some key issues include:

1. Lack of Real-Time Health Monitoring: Many patients, especially in rural or remote areas, do not have access to continuous monitoring of vital health parameters such as SpO2, heart rate, and body temperature, which can delay timely interventions.
2. Overburdened Healthcare Facilities: Hospitals and clinics are often overwhelmed, limiting their ability to provide constant oversight of all patients, particularly those who require long-term monitoring.

These challenges highlight the need for an innovative solution that allows healthcare providers to track patient health remotely, ensuring better management of chronic conditions and reducing hospital overcrowding.

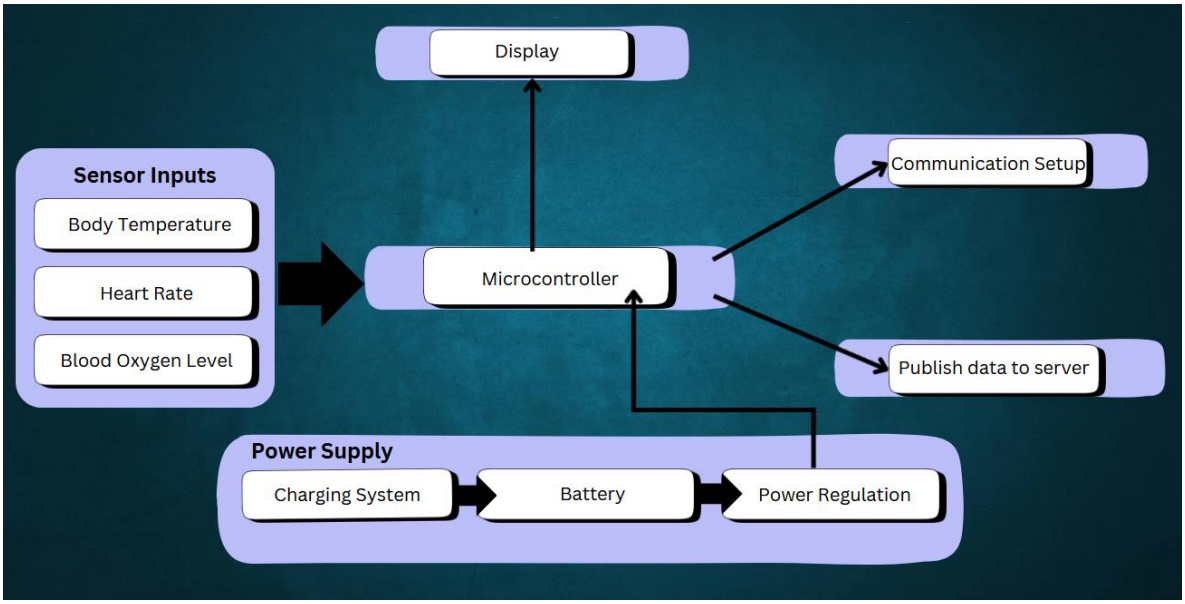
3.Our Solution

To address these healthcare challenges, we propose an IoT-based health monitoring system that continuously tracks vital signs and provides real-time data to both patients and healthcare providers. The system comprises the following components:

- SpO2 and Heart Rate Monitoring Device: This sensor measures blood oxygen levels and heart rate, transmitting the data to a central system for real-time analysis.
- Body Temperature Sensor: A reliable body temperature sensor will monitor and send temperature readings to ensure timely detection of fever or abnormal temperature fluctuations.
- Remote Monitoring and Alerts: The data collected by these sensors will be transmitted to a cloud-based platform where doctors can monitor patients remotely. If any parameter exceeds pre-defined limits, an automatic alert will be sent to both the patient and the healthcare provider, enabling timely medical intervention.

With this system, we aim to improve patient care, reduce hospital workloads, and enable proactive healthcare management through seamless integration of IoT technology.

4.Functional Block Diagrams



5.Live Health Hub

The heart of our IoT-based Health Monitoring System is the Node-RED Dashboard, a powerful and user-friendly interface designed to monitor vital health parameters in real time. This dashboard acts as the central hub where doctors, caregivers, and users can access and visualize live data from various health sensors.

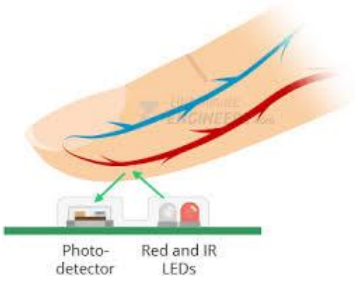
With the Node-RED Dashboard, users can track critical health metrics such as blood oxygen levels (SpO2), heart rate, and body temperature, all displayed in an intuitive format. The system enables users to monitor their health conditions with ease, and doctors can remotely access patient data for timely interventions. The collected sensor data is transmitted to the dashboard through devices connected to a Wi-Fi network, which communicate via an MQTT Broker.

In case any health parameters fall outside the normal range, the system will automatically trigger an alert and notify doctors or caregivers via a message. Additionally, all health data is securely stored in a database for future analysis, providing historical records that allow for trend tracking and more informed medical decisions.

This system not only empowers patients to monitor their health but also provides healthcare professionals with the tools they need for remote care and proactive treatment.

6.Hardware Equipment

a.MAX30102 Oximeter Heart Rate Beat Pulse Sensor



The MAX30102 is an integrated sensor for pulse oximetry and heart rate monitoring. It uses red and infrared LEDs to measure blood absorbance through a photodetector at the fingertip. Configurable via software, it features a 16-deep FIFO for data storage and communicates using an I2C interface.

With ambient light cancellation and a 16-bit ADC, the MAX30102 operates efficiently on 1.8V to 3.3V power supplies. Its ultra-low power consumption makes it suitable for wearable and medical devices, often referred to as the Heart Rate Sensor Module or Pulse Oximeter Module.

b.Human Body Temperature Sensor MAX30205 MTA

The MAX30205 is a high-accuracy digital temperature sensor optimized for human body temperature measurement. It has a range of 0°C to +50°C and offers an impressive accuracy of ±0.1°C, making it ideal for health mon itoring applications.

This sensor operates on a supply voltage of 2.7V to 3.3V, with a typical active current of 600 µA and a shutdown current of only 0.1 µA, ensuring energy efficiency. It communicates via I2C with a default address of 0x48, simplifying integration into various systems.

Additional features include an overtemperature alarm and one-shot mode for reduced power consumption. The MAX30205 is perfect for wearable health devices, providing reliable and efficient body temperature measurements.

c.Node MCU ESP-32

The ESP32-WROOM-32D serves as the main component for collecting data from sensors and processing the required information in IoT applications. With its 240 MHz dual-core processor, 520 KB SRAM, and 16 MB flash, it can efficiently handle multiple sensor inputs and perform real-time data processing. Its integrated Wi-Fi and Bluetooth capabilities allow for seamless data transmission to cloud services or mobile devices, making it ideal for advanced health monitoring systems and other IoT-based projects.

7.Product Outcome

- Real-Time Monitoring
  The health sensors will continuously publish data to the Node-RED dashboard, allowing users to track vital health parameters in real-time.
- Data Visualization
  The dashboard will display critical health metrics, including blood oxygen levels, heart rate, and body temperature, alongside the date and time the data was collected.
- Alerts for Abnormal Readings
  If any health parameter falls outside the normal range, the system will automatically trigger an alert, turning a designated LED red to indicate a violation. Additionally, notifications will be sent to doctors or caregivers via SMS.

8.Threshold values used in Dashboard

1. Heart Rate (BPM)

- Normal Range: 60 to 100 beats per minute (BPM) for adults.
- Threshold Values:
  - Low Alert: Below 60 BPM (bradycardia)
  - High Alert: Above 100 BPM (tachycardia)

2. Blood Oxygen Level (SpO2)

- Normal Range: 95% to 100% saturation.
- Threshold Values:
  - Low Alert: Below 90% (hypoxemia)
  - Critical Alert: Below 85% (severe hypoxemia)

3. Body Temperature (°C)

- Normal Range: Approximately 36.1°C to 37.2°C (97°F to 99°F).
- Threshold Values:
  - Low Alert: Below 36.1°C (hypothermia)
  - High Alert: Above 37.5°C (low-grade fever)
  - Critical Alert: Above 38.0°C (fever)

9.Challenges that can be Faced in the Project

1. Sensor Calibration and Accuracy: Ensuring the accuracy of the MAX30102 and MAX30205 sensors was a challenge. Calibration was required to account for variations in readings due to external factors like lighting conditions and sensor placement, especially for the oximeter and heart rate sensor. Precision was critical in detecting vital health parameters such as SpO2 and body temperature.
2. Real-Time Data Transmission: Transmitting real-time health data to the cloud and ensuring minimal latency was another challenge. The ESP32-WROOM-32D module was instrumental in achieving this, but ensuring reliable and continuous Wi-Fi connectivity, especially in areas with weak signals, posed difficulties.
3. Power Management: Since the health monitoring system needs to run continuously, optimizing power consumption was important. The challenge was to minimize the power usage of both the sensors and the ESP32 module while maintaining accuracy in measurements.
4. User Interface Design: Developing a user-friendly and intuitive interface on the Node-RED dashboard required significant effort to ensure that both patients and healthcare providers could easily interpret the displayed health metrics and respond promptly to alerts.

10.Future Works

For future improvements, the system could be enhanced by integrating machine vision to provide additional insights into the patient's condition. One proposed feature is an eye-tracking system using machine vision to monitor a patient’s eye movements. This could enable the system to detect when the patient's eyes are open or closed, providing valuable data for healthcare providers. For instance, in critical care scenarios, it could inform doctors when a patient regains consciousness by sending an alert as soon as the system detects that the patient's eyes are open. This could be achieved using cameras and image processing algorithms, integrated with the existing monitoring system to offer a comprehensive solution for patient monitoring.

11.Code work

a. Temperature sensor Arduino code

```
#include "Protocentral_MAX30205.h"
#include <Wire.h>
#include <LiquidCrystal.h> // Include the LCD library

// Define MAX30205 object for temperature sensor
MAX30205 tempSensor;

// Show the temperature in Fahrenheit
const bool fahrenheitTemp = true; // Change to false for Celsius

// Initialize the LCD (RS, EN, D4, D5, D6, D7) Adjust GPIO pins for ESP32
LiquidCrystal lcd(21, 22, 19, 18, 5, 17); // Adjust these GPIO pins based on your ESP32 wiring

void setup() {
  // Set up the serial port at 115200 baud rate for ESP32
  Serial.begin(115200);
  delay(1000);

  // Initialize the LCD
  lcd.begin(16, 2); // Set the LCD size
  lcd.print("Temp: "); // Print static text on the first row

  // Initialize I2C with ESP32's default SDA and SCL pins
  Wire.begin(21, 22); // SDA on GPIO 21, SCL on GPIO 22 for ESP32

  // Start MAX30205 temperature sensor in continuous mode
  tempSensor.begin();
}

void loop() {
  // Read temperature from MAX30205
  float temp = tempSensor.getTemperature();

  // Convert to Fahrenheit if needed
  if (fahrenheitTemp) {
  temp = (temp * 1.8) + 32; // Convert to Fahrenheit
 }



// Clear the second row of the LCD and display temperature
```

```
  lcd.setCursor(0, 1); // Move to the second row
  lcd.print("  "); // Clear previous data
  lcd.setCursor(0, 1); // Move back to the start of the second row

  // Display temperature on the LCD
  lcd.print(temp, 2); // Print temperature with 2 decimal places
  lcd.print(fahrenheitTemp ? " °F" : " °C"); // Add appropriate unit

  // Print temperature to the Serial Monitor for debugging
  Serial.print(temp, 2);
  Serial.print(fahrenheitTemp ? " °F" : " °C");
  Serial.println();

  // Wait for a second before the next update
  delay(1000);
}
```

b. Oximeter Heart Rate Beat Pulse Sensor Arduino code

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "spo2_algorithm.h"
#include <LiquidCrystal.h> // Include the LCD library

// Initialize the LCD (RS, EN, D4, D5, D6, D7) Adjust GPIO pins for ESP32
LiquidCrystal lcd(21, 22, 19, 18, 5, 17); // Adjust GPIO pins according to your
                                          wiring
#define MAX_BRIGHTNESS 255
// Data buffers
uint32_t irBuffer[100]; // infrared LED sensor data
uint32_t redBuffer[100]; // red LED sensor data

const byte RATE_SIZE = 4; // Increase for more averaging
byte rates[RATE_SIZE]; // Array of heart rates
byte rateSpot = 0;

long lastBeat = 0; // Time at which the last beat occurred
float beatsPerMinute;
int beatAvg;
int32_t spo2; // SPO2 value
int8_t validSPO2; // Indicator to show if SPO2 calculation is valid

int32_t heartRate; // Heart rate value
```

```
int8_t validHeartRate; // Indicator to show if heart rate calculation is valid
int32_t bufferLength; // Data length

void setup() {
  Serial.begin(115200);
  Serial.println("Initializing...");
  // Initialize I2C for ESP32 (SDA = GPIO 21, SCL GPIO 22)
  Wire.begin(21, 22);
  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 was not found. Please check wiring/power.");
    while (1);
  }
  Serial.println("Place your index finger on the sensor with steady pressure.");

  // Set up the LCD
  lcd.begin(16, 2);
  lcd.print("Initializing...");

  particleSensor.setup(); // Configure sensor with default settings
  particleSensor.setPulseAmplitudeRed(0x0A); // Turn Red LED on
  particleSensor.setPulseAmplitudeGreen(0); // Turn off Green LED

  delay(1000);
  lcd.clear();
}
void loop() {
  // Read the first 100 samples for SpO2 calculation
  bufferLength = 100; // Buffer length of 100 samples
  for (byte i = 0; i < bufferLength; i++) {
    while (particleSensor.available() == false) // Do we have new data?
      particleSensor.check(); // Check the sensor for new data

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); // We're finished with this sample


    // Check for heart beat
    if (checkForBeat(irBuffer[i]) == true) {
      long delta = millis() - lastBeat;
      lastBeat = millis();
      beatsPerMinute = 60 / (delta / 1000.0);

      if (beatsPerMinute < 255 && beatsPerMinute > 20) {
```

```
            rates[rateSpot++] = (byte)beatsPerMinute; // Store this reading
            rateSpot %= RATE_SIZE; // Wrap variable

            // Take average of readings
            beatAvg = 0;
            for (byte x = 0; x < RATE_SIZE; x++)
               beatAvg += rates[x];
            beatAvg /= RATE_SIZE;
         }
      }
   }

   // Calculate heart rate and SpO2
   maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2,
                                          &validSPO2, &heartRate, &validHeartRate);
   // Display on LCD
   lcd.setCursor(0, 0);
   lcd.print("HR: ");
   lcd.print(heartRate);
   lcd.print(validHeartRate ? " bpm" : " N/A");

   lcd.setCursor(0, 1);
   lcd.print("O2: ");
   lcd.print(spo2);
   lcd.print(validSPO2 ? "%" : " N/A");

   // Print to Serial Monitor for debugging
   Serial.print("HR=");
   Serial.print(heartRate);
   Serial.print(", SPO2=");
   Serial.print(spo2);
   Serial.println();

   delay(1000); // Update every second
}
```

c.16x2 Blue Backlight LCD Display Arduino code

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "spo2_algorithm.h"
#include <LiquidCrystal.h> // Include the LCD library

MAX30105 particleSensor;

const byte RATE_SIZE = 4; // Increase this for more averaging
byte rates[RATE_SIZE]; // Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; // Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

#define MAX_BRIGHTNESS 255
uint32_t irBuffer[100]; // Infrared LED sensor data
uint32_t redBuffer[100]; // Red LED sensor data
int32_t bufferLength; // Data length

int32_t spo2; // SpO2 value
int8_t validSPO2; // Indicator to show if the SpO2 calculation is valid
int32_t heartRate; // Heart rate value
int8_t validHeartRate; // Indicator to show if the heart rate calculation is valid

// Initialize the LCD with pins (RS, EN, D4, D5, D6, D7) Adjust GPIO pins for ESP32
LiquidCrystal lcd(21, 22, 19, 18, 5, 17); // Adjust to match ESP32's GPIO

void setup() {
  Serial.begin(115200);

  delay(1000);

  // Initialize the LCD
  lcd.begin(16, 2); // Set the LCD size
  lcd.print("Initializing..."); // Print static text on the first row


  // Initialize I2C for ESP32 (SDA = GPIO 21, SCL = GPIO 22)
  Wire.begin(21, 22);

  // Initialize sensor
```

```
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    lcd.clear();
    lcd.print("MAX30102 not found!");
    while (1);
  }

  // Configure the sensor with default settings
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0A); // Red LED
  particleSensor.setPulseAmplitudeGreen(0); // Green LED off

  // Clear the LCD
  lcd.clear();
  lcd.print("Place finger...");
}
void loop() {
  // Heart rate measurement
  long irValue = particleSensor.getIR();
  if (checkForBeat(irValue) == true) {
    long delta = millis() - lastBeat;
    lastBeat = millis();
    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20) {
      rates[rateSpot++] = (byte)beatsPerMinute; // Store this reading in the
                                                array
      rateSpot %= RATE_SIZE; // Wrap variable
      // Take average of readings
      beatAvg = 0;
      for (byte x = 0; x < RATE_SIZE; x++) {
        beatAvg += rates[x];
      }
      beatAvg /= RATE_SIZE;
    }
  }   // SpO2 measurement
  bufferLength = 100; // buffer length of 100 samples

  for (byte i = 0; i < bufferLength; i++) {
    while (particleSensor.available() == false) { // Do we have new data?
      particleSensor.check(); // Check the sensor for new data
    }
    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); // Move to next sample
  }
```

```
// Calculate heart rate and SpO2
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate,
&validHeartRate);

  // Display data on the LCD
  lcd.clear();
  lcd.setCursor(0, 0); // First row
  lcd.print("HR:");
  lcd.print(heartRate);
  lcd.print(" BPM");

  lcd.setCursor(0, 1); // Second row
  lcd.print("SpO2:");
  lcd.print(spo2);
  lcd.print("%");

  // Print to Serial Monitor for debugging
  Serial.print("HR=");
  Serial.print(heartRate);
  Serial.print(", SpO2=");
  Serial.println(spo2);

  delay(1000); // Update every second
}
```

d. MQTT Publisher Code

```
#include <WiFi.h>
#include <PubSubClient.h>  // Include the PubSubClient library for MQTT

// Define the hardware serial for communication
#define TXD2 17 // GPIO17 for TX (change if needed)
#define RXD2 16 // GPIO16 for RX (change if needed)

// Initialize hardware serial 2 for ESP32
HardwareSerial SUART(2); // Use UART2 (TX2, RX2)

// Wi-Fi and MQTT broker information
const char *WIFI_SSID = "Group16";
const char *WIFI_PASSWORD = "Zxcvb1010";

const char *MQTT_BROKER = "broker.hivemq.com";
const int MQTT_PORT = 1883;
const char *MQTT_CLIENT_ID = "fsfsdfsdfsfrdrd"; // Change this ID if needed
const char *MQTT_TOPIC = "HealthMonitor";
```

```
// Create Wi-Fi and MQTT clients
WiFiClient wifiClient; // Use WiFiClient from the WiFi library
PubSubClient client(wifiClient);

void setup() {
  // Start serial communication
  SUART.begin(9600, SERIAL_8N1, RXD2, TXD2); // Initialize UART2 with baud rate
                                             9600
  Serial.begin(115200);
  // Connect to Wi-Fi
  Serial.print("Connecting to WiFi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  delay(500);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
  }


  Serial.println("Connected to Wi-Fi network:");
  Serial.println(WIFI_SSID);
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  // Connect to MQTT broker
  client.setServer(MQTT_BROKER, MQTT_PORT);

  while (!client.connected()) {
    Serial.print("Connecting to MQTT broker...");

    if (client.connect(MQTT_CLIENT_ID)) {
      Serial.println("Connected to MQTT broker");
    } else {
      Serial.println("Failed to connect to MQTT broker. Retrying...");
      delay(1000);
    }
  }
}
void loop() {
  // Check if any data is available from the SUART (Serial2)
  if (SUART.available() > 0) {
    String x = SUART.readStringUntil('\n');
    Serial.println(x);
    // Publish the string to MQTT
    if (client.connected()) {
      client.publish(MQTT_TOPIC, x.c_str());
      Serial.println("Published to MQTT");
```

```
    }else {
      Serial.println("MQTT not connected. Reconnecting...");

      if (client.connect(MQTT_CLIENT_ID)) {
        Serial.println("Reconnected to MQTT broker");
      } else {
        Serial.println("Still not connected to MQTT broker");
      }
    }
  }

  // Maintain the MQTT connection
  client.loop();
  delay(100);
}
```