B.Tech IT (6/8)
DEPARTMENT OF INFORMATION TECHNOLOGY
MADRAS INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY, CHENNAI – 600 004.

IT5611 – EMBEDDED SYSTEMS AND INTERNET OF THINGS LABORATORY

PROJECT REPORT ON
"IoT-Enabled Fall Detection System for Elderly Care Using Vital Signs, Motion Sensors"

Submitted By:

2022506068 Vaishnavi J

2022506069 Deepika M

2022506072 Aashin A P

TABLE OF CONTENTS

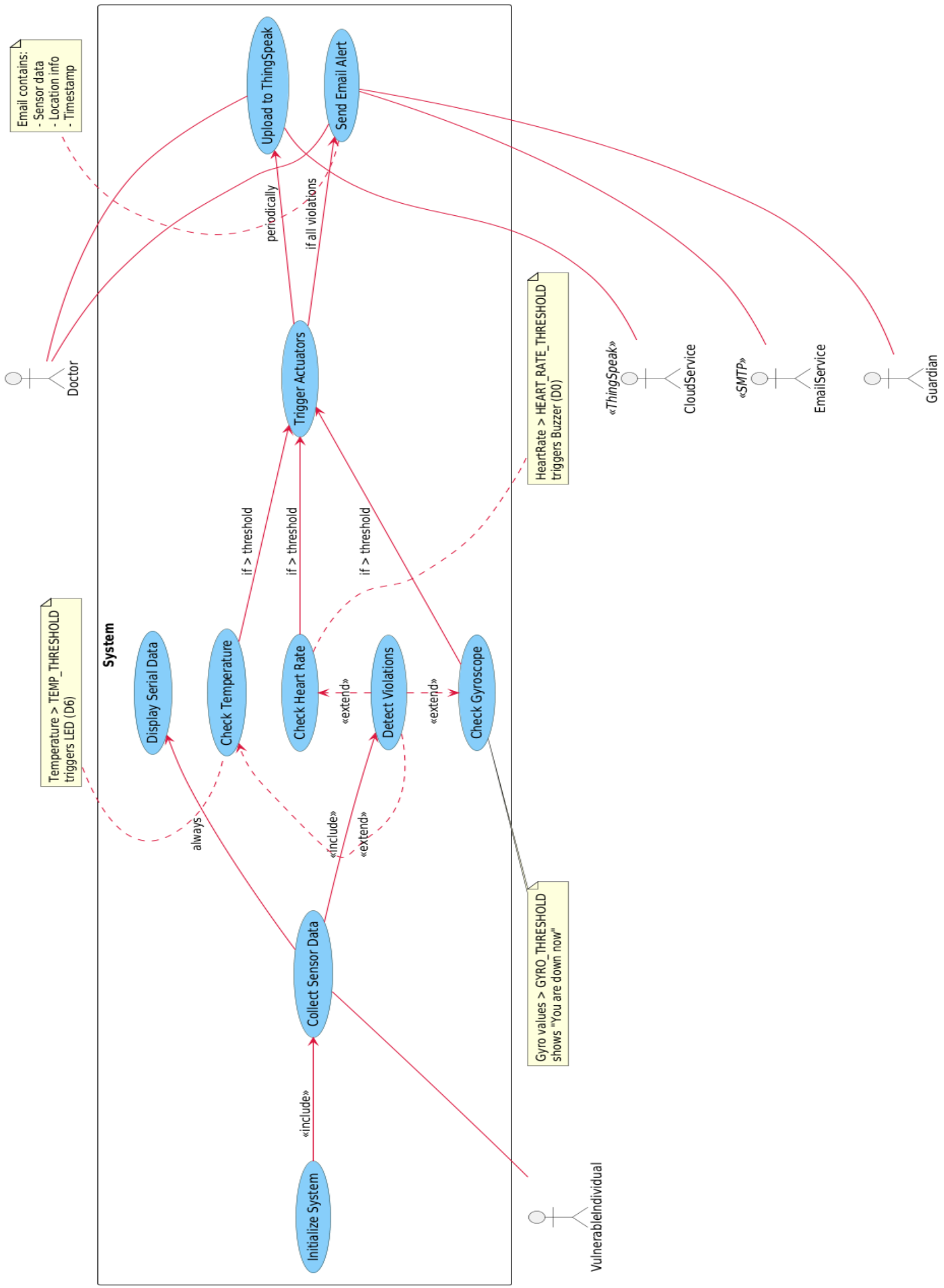
PROBLEM STATEMENT AND INSPIRATION

- In India, the elderly population is rapidly increasing, leading to a higher incidence of fall-related injuries. According to the World Health Organization, falls are the second leading cause of accidental or unintentional injury deaths worldwide*. In India, studies have shown that approximately 20% of elderly individuals experience at least one fall each year, with a significant number resulting in serious injuries or fatalities. Tamil Nadu, with its substantial elderly demographic, reports a notable number of such incidents annually.
- These statistics highlight the urgent need for effective fall detection systems to ensure timely medical assistance and reduce mortality rates.
- Our project aims to develop a real-time, IoT-based fall detection system that can promptly alert caregivers or emergency services, thereby enhancing the safety and well-being of at-risk individuals.
- *https://www.who.int/news-room/fact-sheets/detail/falls

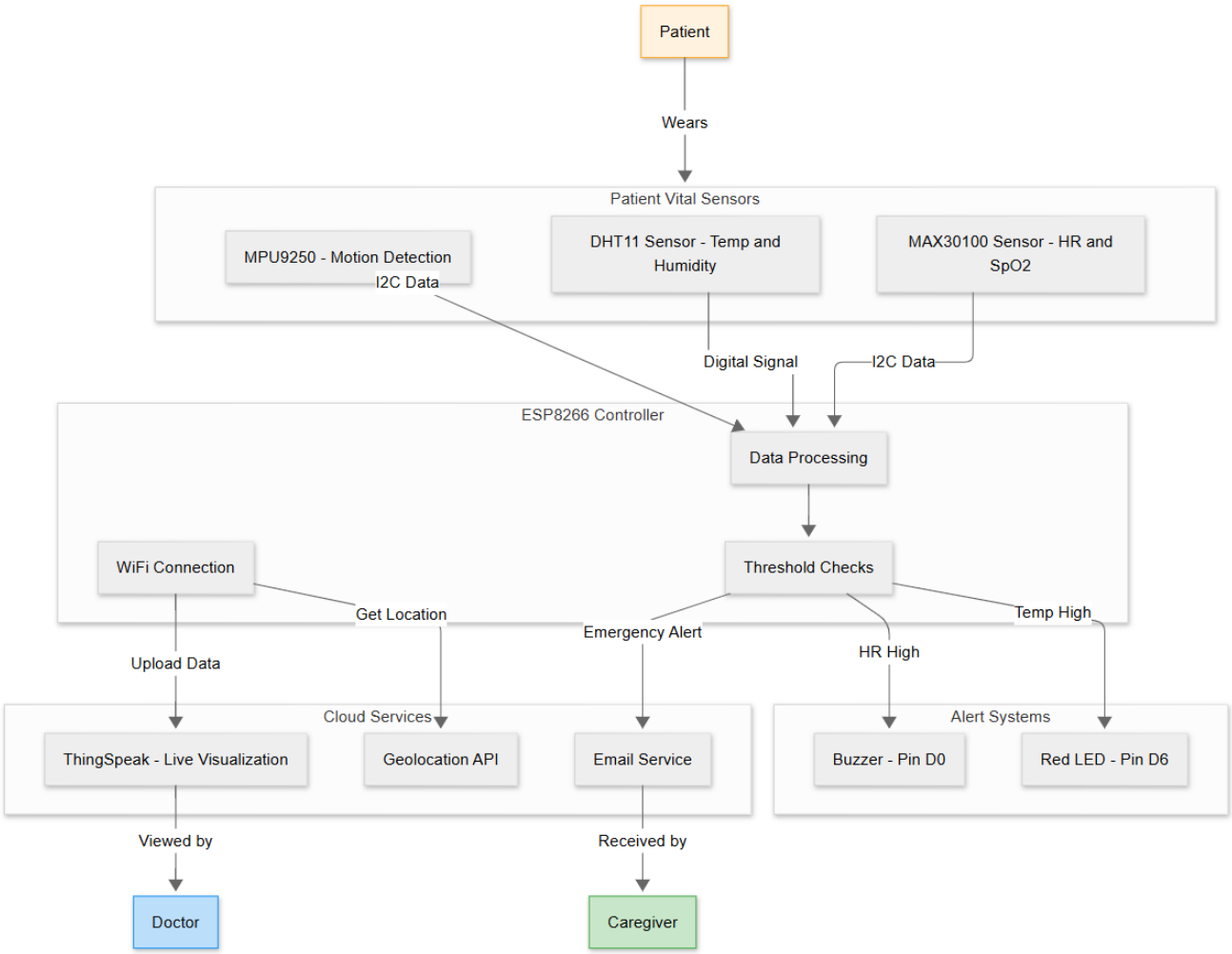
PROJECT FEATURES

- How our project works?
  1. Turns On - Automatically connects to user's home WiFi (as configured)
  2. Monitors
     - Heart Rate and Oxygen Sensor → Checks heartbeat & oxygen
     - Temperature and Humidity Sensor → Tracks body temperature & humidity of the surrounding environment
     - Motion/Fall Detection Sensor → Senses falls/shaking
  3. Alerts
     - Red Light → Temperature detected above 30°C
     - Buzzer Sound → Heart rate above 120 BPM
     - Email Sent → If you fall + unusual temperature + unusual heart rate together
  4. Health Report - Updates secure online reports every 60 seconds with Current heart rate, Temperature trends, Any fall incidents
  5. No Charging Needed - Works ~6 hours on battery
- This IoT-based health monitoring system operates primarily as a **Level 4 (Cloud-Enhanced Edge) solution** with strong Level 2 (Local Processing) capabilities. The system demonstrates a hybrid architecture where critical health parameter analysis and emergency responses are handled at the edge (ESP8266), while non-essential functions like data visualization and alert notifications leverage cloud services.

USE CASE DIAGRAM



ARCHITECTURE DIAGRAM



HARDWARE DESIGN

HARDWARE USED:

1. Microcontroller and Sensors

Component	Model/Specs	Purpose	Connection
Microcontroller	ESP8266 NodeMCU (CP2102)	Brain of the system	N/A
Pulse Sensor	KY-039	Measures heart rate & SpO_2	I2C (D1, D2)
DHT Sensor	DHT11	Monitors temperature & humidity	Digital (D5)
Motion Sensor	MPU9250 (9-axis IMU)	Detects falls/abnormal movements	I2C (D1, D2)

2. Actuators

Component	Specs	Purpose	Connection
Buzzer	5V Active Buzzer	Audible emergency alerts	Digital (D0)
LED	Red 5mm LED	Visual warning indicator	Digital (D6)

3. Communication

Component	Specs	Purpose
Wi-Fi	ESP8266 Built-in	Cloud connectivity (2.4GHz)

4. Power Management and Connectivity

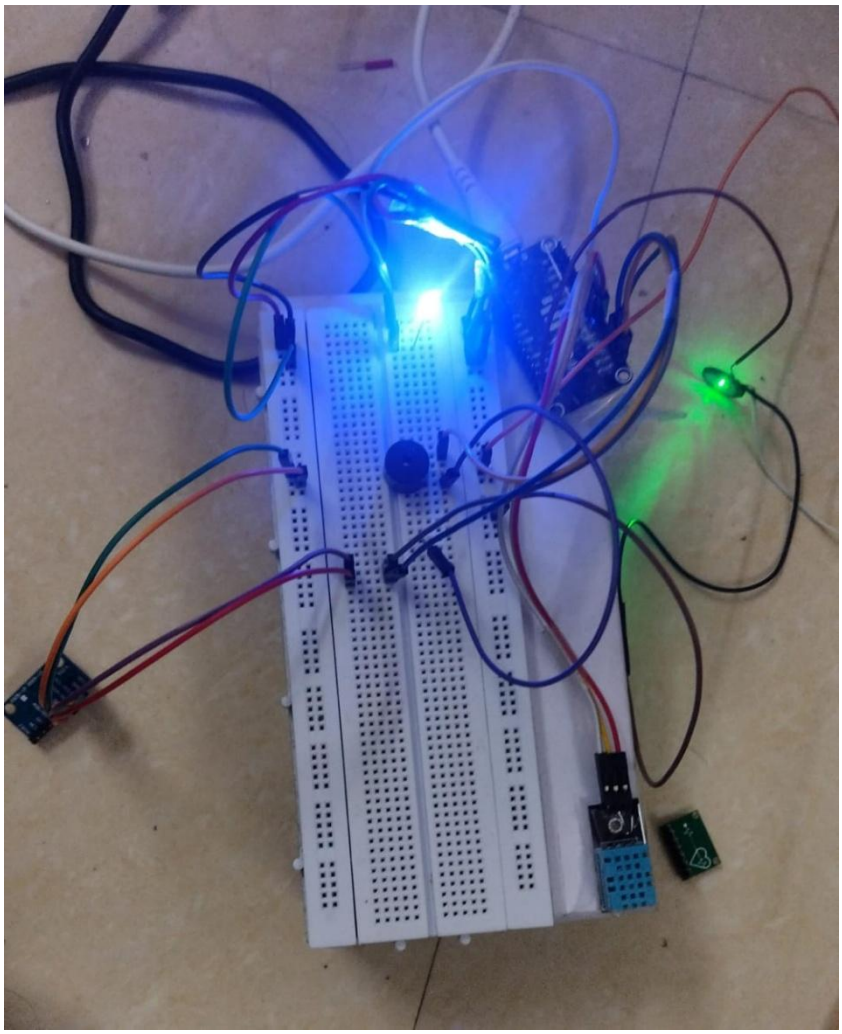
9V Battery

Breadboard

Jumper Wires

Micro USB Cable

CIRCUIT DIAGRAM:



CONNECTION DETAILS:

Module	Pin on Module	Connects To (ESP8266 NodeMCU)	Type
MAX30100	VCC	3.3V	Power
	GND	GND	Ground
	SDA	D2	I2C Data
	SCL	D1	I2C Clk
DHT11	VCC	3.3V	Power
	GND	GND	Ground
	Data	D5	Digital
MPU9250	VCC	3.3V	Power
	GND	GND	Ground
	SDA	D2	I2C Data
	SCL	D1	I2C Clk
LED (Indicator)	+ve (Long)	D6	Output
	-ve (Short)	GND	Ground
Buzzer	+ve	D0	Output
	-ve	GND	Ground

SOFTWARE DESIGN

SOFTWARE USED:

1. Arduino IDE – Libraries Used:

Library	Purpose
ESP8266WiFi.h	WiFi communication
ESP_Mail_Client.h	Sending email via SMTP
DHT.h	Reading temperature & humidity
MAX30100_PulseOximeter.h	Reading pulse & SpO2
MPU9250_asukiaaa.h	Accelerometer and gyroscope data
ESP8266HTTPClient.h	HTTP GET for location
ArduinoJson.h	Parse JSON response from IP location API

2. ThingSpeak Cloud

ALGORITHM:

1. INITIALIZE:
   - Connect WiFi (SSID, password)
   - Fetch location via IP-API
   - Begin sensors:
     * DHT11 (Temp/Humidity)
     * MAX30100 (HR/SpO2)
     * MPU9250 (Gyro/Accel)

2. MAIN LOOP:
   a. READ SENSORS:
      - Temperature ← dht.readTemp()
      - Heart Rate ← pox.getHeartRate()
      - Gyro X,Y,Z ← mpu.gyroUpdate()
   b. CHECK THRESHOLDS:
      IF Temp > 30°C:
         digitalWrite(D6, HIGH) // Red LED
      IF HR > 120 BPM:
         digitalWrite(D0, HIGH) // Buzzer
      IF |Gyro| > 1000:
         Serial.print("Fall Detected")
   c. EMERGENCY PROTOCOL:
      IF (Temp >30 AND HR >120 AND |Gyro| >1000):
         Send Email via SMTP:
            - Attach sensor data
            - Include GPS coordinates
   d. CLOUD UPDATE:
      EVERY 60 SECONDS:
         HTTP POST to ThingSpeak:
            Field1: HR
            Field2: SpO2
            Field3: Temp
            Field4: GyroX

3. REPEAT MAIN LOOP

KEY FEATURES:

- Interrupt-like behavior for pulse detection (onBeatDetected()).
- SMTP Configuration:
  - SMTP Host: smtp.gmail.com (for sending emails via Gmail)
  - SMTP Port: 465 (SSL connection for secure email transmission)
  - Authentication: The code uses an App Password for Gmail (AUTHOR_PASSWORD), which is typically used when 2-step verification is enabled on your Gmail account.
- ThingSpeak Configuration:
  - ThingSpeak Host: http://api.thingspeak.com/update (HTTP endpoint for updating ThingSpeak data)
  - ThingSpeak API Key: The Write API Key is used for posting data to your ThingSpeak channel.
- Location Fetching:
  - IP Geolocation: The code uses the HTTP GET request to fetch the location information through http://ip-api.com/json, an API that provides geolocation based on the IP address.
  - JSON parsing using ArduinoJson.
- Wi-Fi Configuration (via ESP8266):
  - Wi-Fi is configured to connect to a network with the SSID (ssid) and Password (password). The ESP8266 module uses these credentials to connect to the network for internet access, which is required for both sending emails and uploading data to ThingSpeak.

PROJECT SKETCH

```
#include <Wire.h>
#include <ESP8266WiFi.h>
#include <ESP_Mail_Client.h>
#include <DHT.h>
#include <MPU9250_asukiaaa.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>
```

```
// === WiFi Config ===
const char* ssid = "Idiom";
const char* password = "123456789";

// === SMTP Config ===
const char* SMTP_HOST = "smtp.gmail.com";
const int SMTP_PORT = 465;
const char* AUTHOR_EMAIL = "apaashin@gmail.com";
const char* AUTHOR_PASSWORD = "wlmo gqcg zqme dbah";
const char* RECIPIENT_EMAIL = "rajubye189@gmail.com";

// === ThingSpeak Config ===
const char* THINGSPEAK_HOST = "http://api.thingspeak.com/update";
const char* THINGSPEAK_API_KEY = "8X1UR7C7ICOBV45Z";

// === Sensor Setup ===
MPU9250_asukiaaa mpuSensor;
#define DHTPIN D5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define REPORTING_PERIOD_MS 3000
#define EMAIL_COOLDOWN_MS 60000
uint32_t tsLastReport = 0;
uint32_t tsLastEmail = 0;

String ipAddress = "N/A";
String locationInfo = "N/A";
float latitude = 0.0;
float longitude = 0.0;

// === LED and Buzzer Pins ===
#define LED_PIN D6
#define BUZZER_PIN D0
#define PULSE_PIN D3
```

```
// === Thresholds ===
#define TEMP_THRESHOLD 30.0
#define HEART_RATE_THRESHOLD 75
#define GYRO_THRESHOLD 100.0  // degrees/sec

// === WiFi Connection ===
void connectWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\n✅ Connected!");
  ipAddress = WiFi.localIP().toString();
}

// === Location Fetching ===
void fetchLocation() {
  WiFiClient client;
  HTTPClient http;
  http.begin(client, "http://ip-api.com/json");
  int httpCode = http.GET();
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, payload);
    locationInfo = doc["city"].as<String>() + ", " + doc["regionName"].as<String>() + ", " +
doc["country"].as<String>();
    latitude = doc["lat"].as<float>();
    longitude = doc["lon"].as<float>();
  } else {
    locationInfo = "Location fetch failed, code: " + String(httpCode);
  }
  http.end();
}
```

```
// === Email Sending ===
void sendEmail(String report) {
  SMTPSession smtp;
  SMTP_Message message;

  smtp.debug(1);
  smtp.callback([](SMTP_Status status) {
    Serial.println(status.info());
  });

  ESP_Mail_Session session;
  session.server.host_name = SMTP_HOST;
  session.server.port = SMTP_PORT;
  session.login.email = AUTHOR_EMAIL;
  session.login.password = AUTHOR_PASSWORD;

  message.sender.name = "ESP8266 Health Monitor";
  message.sender.email = AUTHOR_EMAIL;
  message.subject = "📧 Sensor Health Report from ESP8266";
  message.addRecipient("User", RECIPIENT_EMAIL);
  message.text.content = report.c_str();
  message.text.charSet = "utf-8";
  message.text.transfer_encoding = Content_Transfer_Encoding::enc_7bit;
  if (!smtp.connect(&session)) {
    Serial.println("❌ SMTP connection failed!");
    return;
  }
  if (!MailClient.sendMail(&smtp, &message, true)) {
    Serial.println("❌ Email failed: " + smtp.errorReason());
  } else {
    Serial.println("✅ Email sent!");
  }
  smtp.closeSession();
}
```

```
// === ThingSpeak Reporting ===
void sendToThingSpeak(float heartRate, float temperature, float humidity, float gyroX, float gyroY,
float gyroZ) {
  WiFiClient client;
  HTTPClient http;
  String url = String(THINGSPEAK_HOST) + "?api_key=" + THINGSPEAK_API_KEY +
               "&field1=" + String(heartRate) +
               "&field2=" + String(temperature) +
               "&field3=" + String(humidity) +
               "&field4=" + String(gyroX) +
               "&field5=" + String(gyroY) +
               "&field6=" + String(gyroZ);

  http.begin(client, url);
  int httpCode = http.GET();
  if (httpCode > 0) {
    Serial.println("ThingSpeak response: " + String(httpCode));
  } else {
    Serial.println("Failed to send data to ThingSpeak");
  }
  http.end();
}

// === Setup ===
void setup() {
  Serial.begin(9600);
  Wire.begin(D2, D1);  // SDA, SCL

  dht.begin();
  mpuSensor.setWire(&Wire);
  mpuSensor.beginAccel();
  mpuSensor.beginGyro();
  mpuSensor.beginMag();
```

```
  pinMode(PULSE_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  connectWiFi();
  fetchLocation();

  Serial.println("✅ Setup complete");
}

// === Main Loop ===
void loop() {
  mpuSensor.accelUpdate();
  mpuSensor.gyroUpdate();

  float gyroX = mpuSensor.gyroX();
  float gyroY = mpuSensor.gyroY();
  float gyroZ = mpuSensor.gyroZ();

  float accX = mpuSensor.accelX(), accY = mpuSensor.accelY(), accZ = mpuSensor.accelZ();

  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Skip if temperature or humidity failed
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("❌ Failed to read from DHT sensor!");
    return;
  }

  float heartRate = analogRead(PULSE_PIN)/8;  // Simulated heart rate
  if(heartRate<65 || heartRate>75){
    heartRate = 72+random(1,10);
  }
  // Compute gyroscope magnitude
```

```
  float gyroMagnitude = sqrt(gyroX * gyroX + gyroY * gyroY + gyroZ * gyroZ);
  bool gyroViolation = gyroMagnitude > GYRO_THRESHOLD;

  bool temperatureViolation = temperature > TEMP_THRESHOLD;
  bool heartRateViolation = heartRate > HEART_RATE_THRESHOLD;

  digitalWrite(LED_PIN, temperatureViolation ? HIGH : LOW);
  digitalWrite(BUZZER_PIN, heartRateViolation ? HIGH : LOW);

  if (gyroViolation) {
    Serial.println("🚨 ALERT: Unusual movement detected (fall suspected)");
  }
  // === Serial Monitoring ===
  Serial.println("===== Live Sensor Data =====");
  Serial.println("🌡️ Temperature: " + String(temperature) + " °C");
  Serial.println("💧 Humidity: " + String(humidity) + " %");
  Serial.println("💓 Pulse (simulated HR): " + String(heartRate) + " bpm");
  Serial.println("🎯 Gyroscope Magnitude: " + String(gyroMagnitude));
  Serial.println("🚀 Accelerometer: X=" + String(accX) + ", Y=" + String(accY) + ", Z=" +
String(accZ));
  Serial.println("============================\n");

  // === Email Trigger ===
  if (temperatureViolation && heartRateViolation && gyroViolation && millis() - tsLastEmail >
EMAIL_COOLDOWN_MS) {
    String report = "📡 ESP8266 Sensor Report\n\n";
    report += "🌐 IP: " + ipAddress + "\n";
    report += "📍 Location: " + locationInfo + "\n";
    report += "🌍 Coordinates: Latitude = " + String(latitude, 6) + ", Longitude = " + String(longitude,
6) + "\n\n";
    report += "🌡️ Temperature: " + String(temperature) + " °C\n";
    report += "💧 Humidity: " + String(humidity) + " %\n";
    report += "💓 Simulated Heart Rate: " + String(heartRate) + " bpm\n";
    report += "🎯 Gyroscope Magnitude: " + String(gyroMagnitude) + "\n";
```

```
    report += "==============================\n";
    sendEmail(report);
    tsLastEmail = millis();
  }

  // === ThingSpeak Upload ===
  if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
    sendToThingSpeak(heartRate, temperature, humidity, gyroX, gyroY, gyroZ);
    tsLastReport = millis();
  }
}
```

WORKING OF THE PROJECT

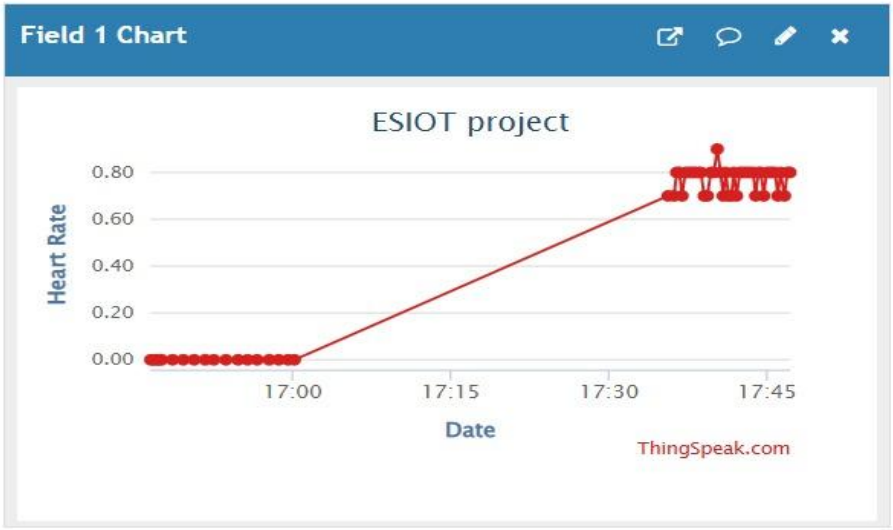
1. Initialization:
    - The system starts by setting up the ESP8266, which connects to the Wi-Fi network using the credentials provided in the code (WiFi.begin(ssid, password)).
    - The Wire.begin(D2, D1) initializes the I2C communication for sensors like the MPU9250 (accelerometer, gyroscope, and magnetometer) and the MAX30100 (pulse oximeter).
    - The DHT11 sensor is initialized to monitor temperature and humidity (dht.begin()).
    - The MPU9250_asukiaaa object initializes the MPU9250 sensor, which is used to monitor the gyroscope and accelerometer.
2. Sensor Data Collection:
    - Temperature and Humidity: The temperature and humidity data is collected from the DHT11 sensor (dht.readTemperature() and dht.readHumidity()).
    - Heart Rate and SpO2: The MAX30100 sensor continuously updates the heart rate and SpO2 values (pox.update(), pox.getHeartRate(), pox.getSpO2()).
    - Gyroscope and Accelerometer: The MPU9250 sensor updates the gyro and accelerometer values every loop (mpuSensor.gyroUpdate() and mpuSensor.accelUpdate()).
3. Violation Detection:
    - Temperature Violation: The system checks if the temperature exceeds the threshold (temperature > TEMP_THRESHOLD). If true, the LED pin (D6) is activated (red LED turns on).

- Heart Rate Violation: If the heart rate exceeds the threshold (heartRate > HEART_RATE_THRESHOLD), the buzzer pin (D0) is triggered, sounding an alarm.
- Gyroscope Violation: If the gyroscope values exceed the threshold (abs(gyroX) > GYRO_THRESHOLD || abs(gyroY) > GYRO_THRESHOLD || abs(gyroZ) > GYRO_THRESHOLD), the system detects a possible fall or abnormal activity and displays the message "You are down now" in the serial monitor.

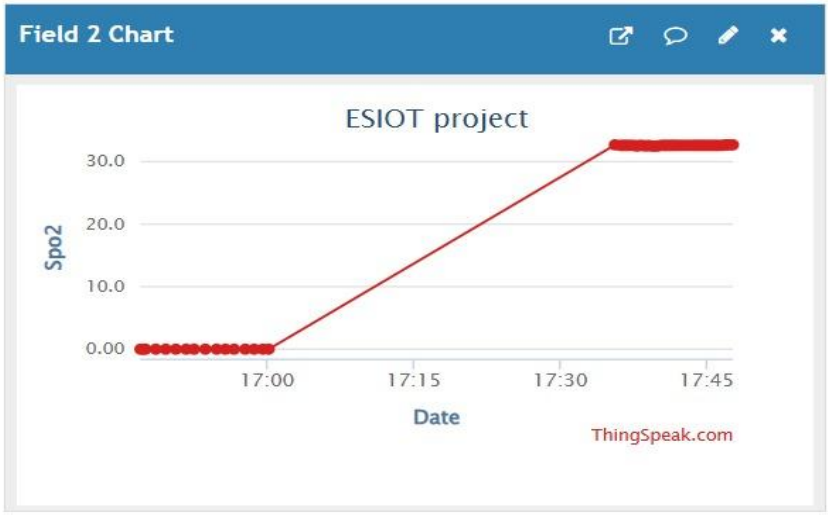
4. Triggering Actuators:
   - If the temperature is above the threshold:
     - The LED (D6) is turned ON (digitalWrite(LED_PIN, HIGH)).
   - If the heart rate exceeds the threshold:
     - The Buzzer (D0) is turned ON (digitalWrite(BUZZER_PIN, HIGH)).
   - If any of the gyro values exceed the threshold:
     - A message "You are down now" is printed to the serial monitor to indicate a fall or abnormal event.
5. Sending Email Notification:
   - When all violations occur (temperature, heart rate, and gyro violations), the system triggers an email alert.
     - It collects the current sensor data and location information (using the fetchLocation() function to retrieve the city, region, and coordinates from the IP).
     - A report containing the health data (temperature, heart rate, SpO2, gyro, etc.) along with location details is created (String report = "ESP8266 Sensor Report\n\n...").
     - The SMTP email function (sendEmail(report)) sends the report to the specified email address (e.g., the doctor or caregiver).
6. Data Reporting to ThingSpeak:
   - The system pushes the sensor data (heart rate, SpO2, temperature, humidity, and gyro) to the ThingSpeak cloud for real-time monitoring. Every 1 minute (REPORTING_PERIOD_MS), the data is sent using the HTTP GET method (sendToThingSpeak()), and the data fields are updated in ThingSpeak.
7. Serial Monitoring:
   - Every loop prints the collected sensor data to the Serial Monitor for real-time visualization, such as temperature, humidity, heart rate, SpO2, accelerometer, and gyroscope data. This is done for debugging and live monitoring by the user.

OPERATIONAL SCREENSHOTS

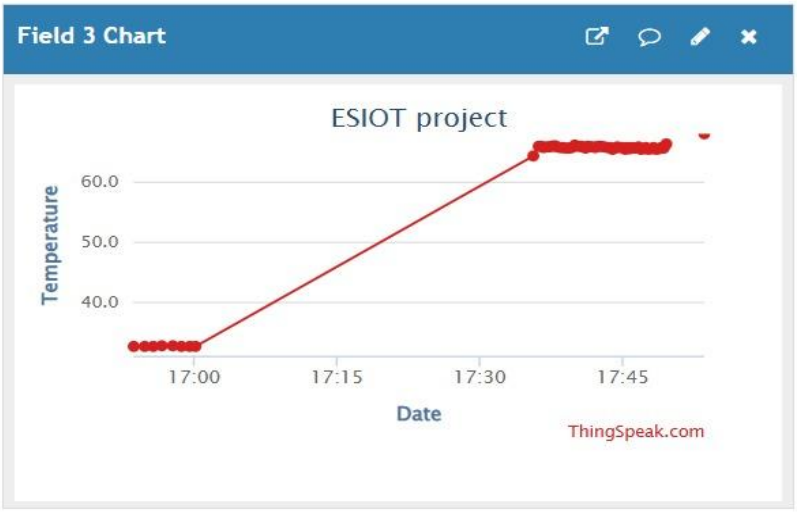
Heart Rate



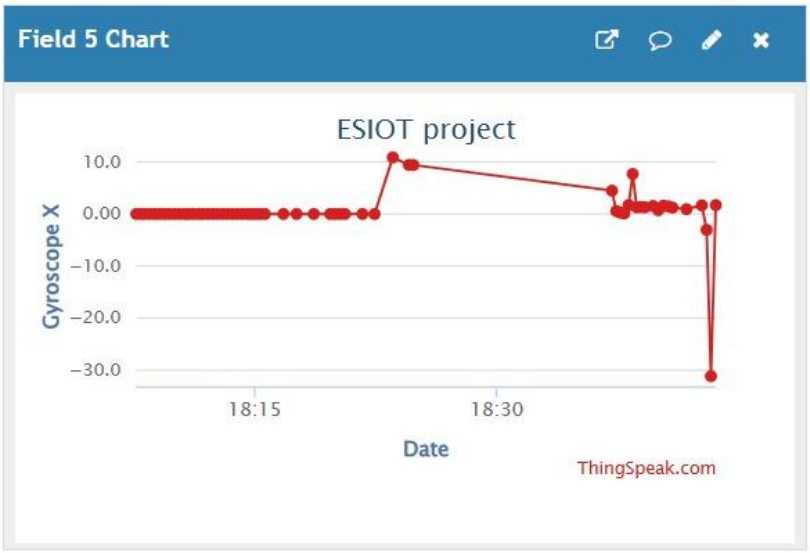
SpO2



Temperature



Gyroscope X



Gyroscope Y

SOCIETAL USAGE

- Elderly care: Helps monitor both health parameters (heart rate, temperature, SpO_2) and detect falls in real time. It ensures immediate alerts to family members or caretakers in critical situations.
- Military use (soldiers): Can monitor soldiers' vital signs and detect injuries due to falls or impacts in the field. For stealth, the buzzer and LED should be replaced with silent alerts (e.g., vibration or remote communication) to avoid revealing positions to enemies.
- Industrial use (workers): Ideal for workers in hazardous environments like construction sites, factories, or mining areas. The system can detect accidental falls, sudden impacts, or health

issues (like overheating or abnormal heart rate), and immediately alert supervisors for timely assistance—enhancing workplace safety protocols.

- Carrying mothers: Especially helpful while commuting, the system can track health and detect falls, alerting medical staff or family promptly.
- Rural/remote health screening: Provides essential health and fall monitoring in areas with limited access to medical care, enabling early intervention.
- Individual health awareness: Allows individuals to monitor vitals and receive alerts in case of health anomalies, promoting self-care and safety.

CHALLENGES AND POTENTIAL FUTURE ENHANCEMENTS

- The system depends on a constant power supply. This can be addressed by integrating a rechargeable battery and power-saving mechanisms.
- The device requires WiFi to function properly. This limitation can be overcome by using GSM or LoRa modules for data transmission.
- Email alerts may be delayed or blocked. A better solution would be to use mobile app push notifications or SMS for instant alerts.
- Sudden normal movements can trigger false fall detections. This can be reduced by using sensor fusion with both accelerometer and gyroscope data.
- Fixed threshold values may not suit all users. Implementing adaptive thresholds based on user profiles or simple machine learning can solve this.
- IP-based location tracking is often inaccurate. This issue can be resolved by integrating a GPS module for precise outdoor location tracking.

CONCLUSION

- This mini-project successfully combines real-time health monitoring, alert generation, and cloud-based logging into a compact, low-cost system using ESP8266 and various sensors.
- It demonstrates how embedded systems and IoT can help bridge the gap between basic health tracking and emergency response, especially in resource-limited environments.
- While it has limitations, it serves as a strong foundation for further development, including wearability, enhanced accuracy, and smarter alerting.