Name: ESP32-Based Accident Detection and Alert System with Vehicle Control

Description:

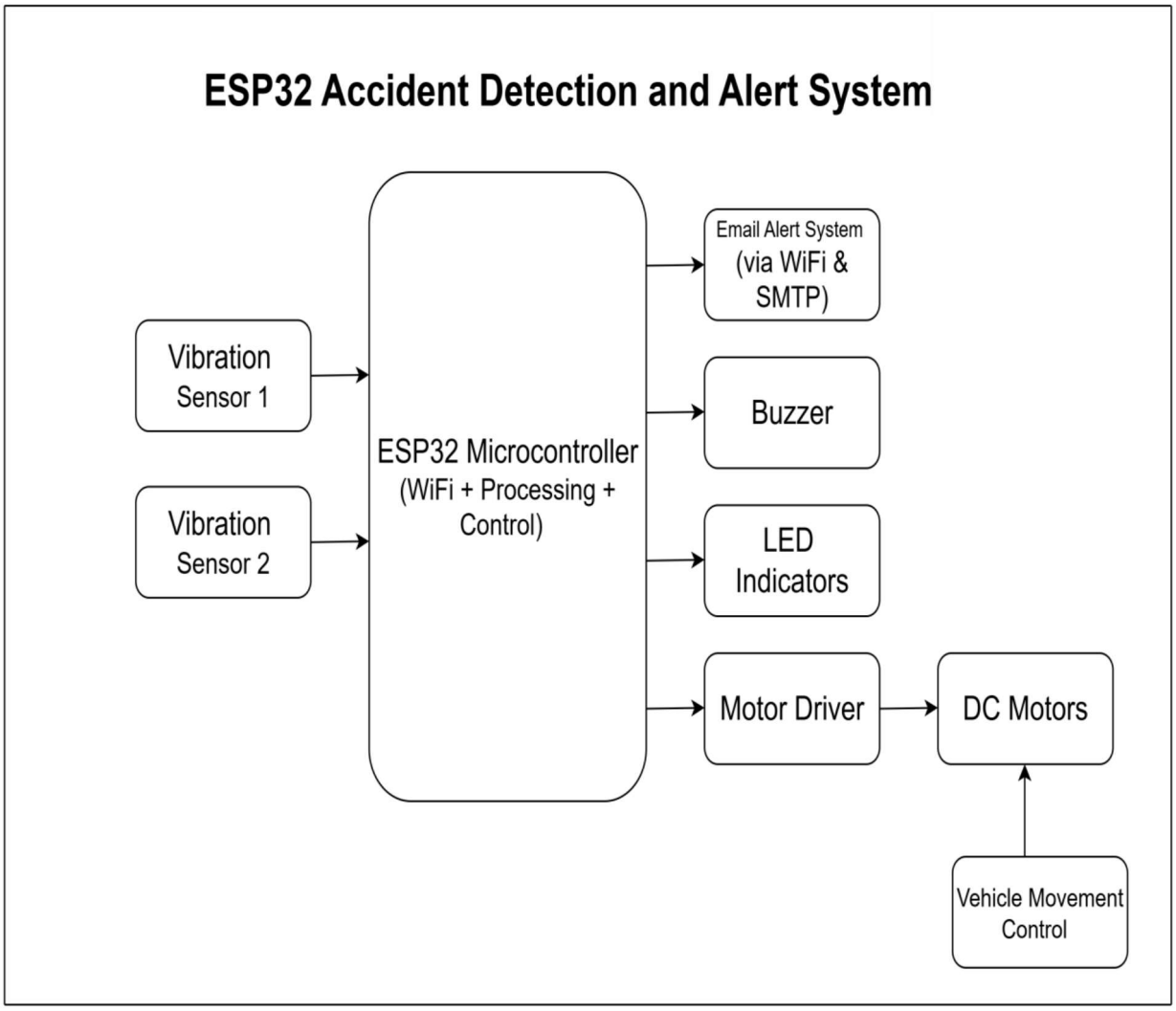
This project is designed to **detect vehicle accidents** using **vibration sensors** and automatically **send alert emails** containing the **vehicle's approximate location** to a predefined recipient. The vehicle is also **Bluetooth-controlled**, allowing manual driving. Upon detecting a collision, the system stops the vehicle, activates LEDs and buzzer, and sends an emergency alert through the internet.

This ensures **faster response in emergencies**, combining **IoT, automation, and wireless communication** in one system.

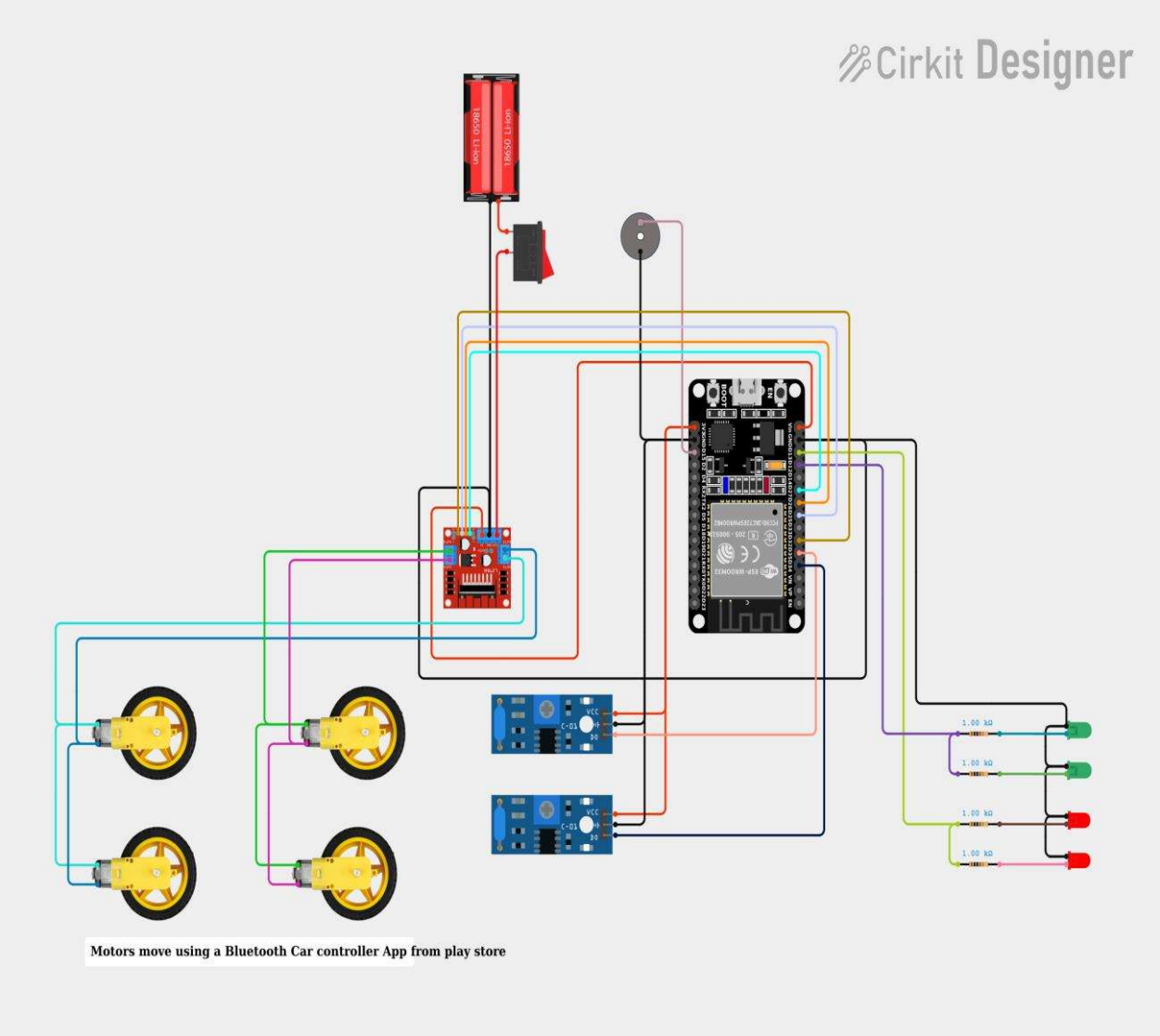
Materials Used:

S.No	Component	Description
1	**ESP32**	Main microcontroller with built-in WiFi and Bluetooth for IoT and control.
2	**Vibration Sensor (x2)**	Detects shocks or impacts to identify accidents.
3	**L298N Motor Driver**	Controls motor direction and speed.
4	**DC Motors (x2) - BO Motors**	Used for vehicle movement (left and right wheels).
6	**LEDs (Front & Back)**	Indicate movement and alert states.
7	**Buzzer**	Gives sound alert during accident detection.

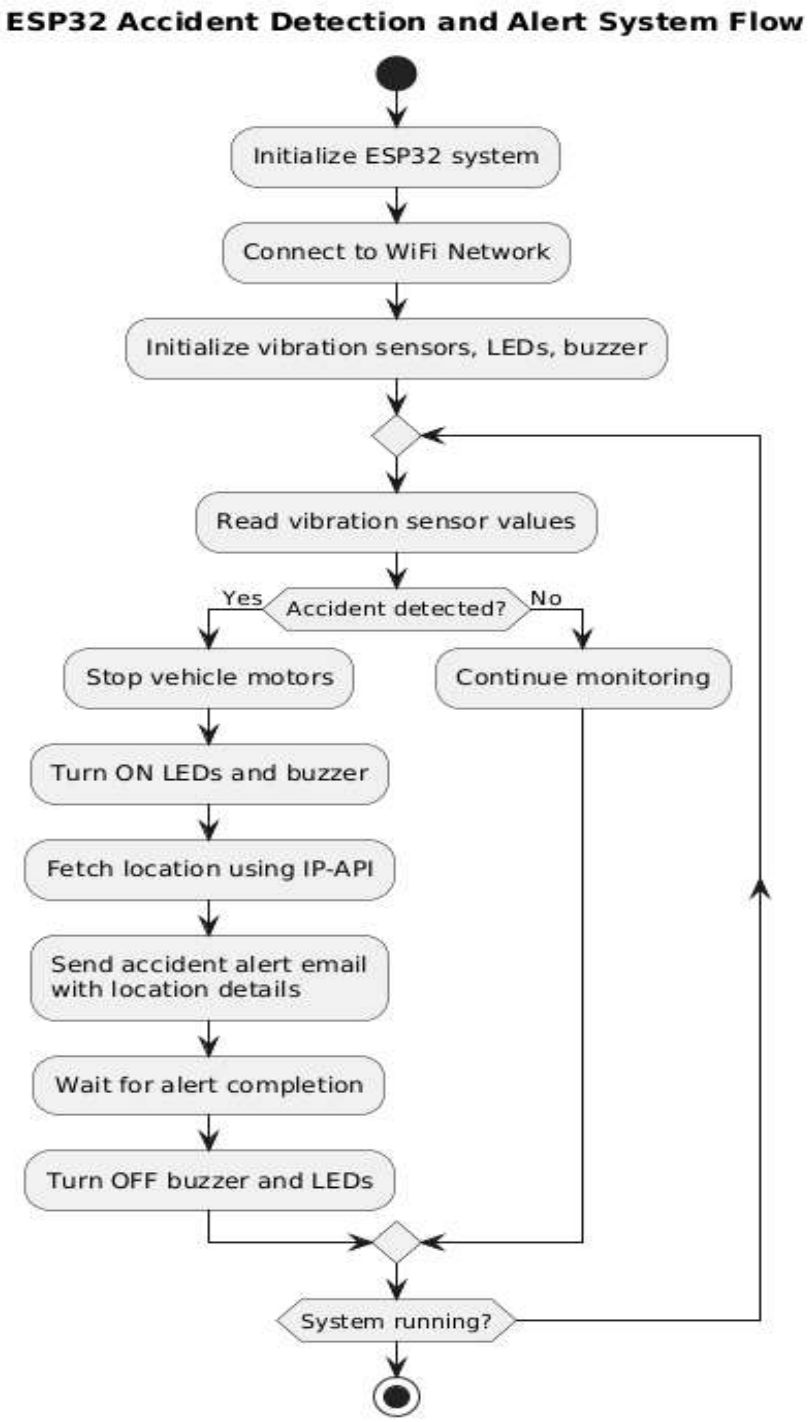
Block Diagram:



Circuit Diagram:



Flow Chart:



How to Explain the Project:

- The system uses ESP32 to detect accidents automatically using vibration sensors.
- When a strong impact is detected, the motors stop immediately.
- LEDs and buzzer turn on to indicate an accident.
- The ESP32 connects to WiFi and fetches the current location from the internet.
- It sends an email alert with the latitude, longitude, and Google Maps link.
- After sending the alert, the buzzer and LEDs turn off and the system resumes monitoring.

Code:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Arduino_JSON.h>
#include <ESP_Mail_Client.h>
#include <WiFiManager.h>
#include "BluetoothSerial.h"
#include "time.h"

// ====================== Pin Definitions ========================
#define VIB_SENSOR_1 35
#define VIB_SENSOR_2 34

#define IN1 27   // Left motor IN1
#define IN2 26   // Left motor IN2
#define IN3 25   // Right motor IN3
#define IN4 33   // Right motor IN4
#define EN 32    // ENA + ENB shorted (common enable)

#define FRONT_LED 12
#define BACK_LED 13
#define BUZZER 15

// ====================== SMTP Configuration ======================
#define SMTP_HOST "smtp.gmail.com"
#define SMTP_PORT 465
#define AUTHOR_EMAIL "your_Email"
#define AUTHOR_PASSWORD "Your_Author_Password"
#define RECIPIENT_EMAIL " your_Email "

SMTPSession smtp;
const char *apiURL = "http://ip-api.com/json/";

// ====================== Bluetooth ===============================
BluetoothSerial SerialBT;

// ====================== Vibration Detection =====================
const int vibConsecutiveThreshold = 2;   // Lowered sensitivity threshold
```

```
int vib1Count = 0;
int vib2Count = 0;
unsigned long lastVibSample = 0;
const unsigned long vibSampleInterval = 50; // ms

// ====================== Control Flags ==========================
unsigned long lastEmailTime = 0;
const unsigned long emailCooldown = 60000; // 1 min cooldown
bool isAlertActive = false;

// ====================== Function Prototypes ====================
void stopMotors();
void forward();
void backward();
void left();
void right();
void flushBluetoothInput();
String fetchLocation();
void sendEmail(String locationData);
void triggerAccident();
void resetAlertOutputs();
void smtpCallback(SMTP_Status status);
bool setSystemTime();
bool ensureWiFiConnected();
void testPing();

// ============================ SETUP ============================
void setup() {
  Serial.begin(115200);
  delay(300);

  // Motor setup
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(EN, OUTPUT);
  stopMotors();

  // Vibration sensors
  pinMode(VIB_SENSOR_1, INPUT);
  pinMode(VIB_SENSOR_2, INPUT);
```

```
  // LEDs & Buzzer
  pinMode(FRONT_LED, OUTPUT);
  pinMode(BACK_LED, OUTPUT);
  pinMode(BUZZER, OUTPUT);
  resetAlertOutputs();

  // Bluetooth
  SerialBT.begin("ESP32_Car");
  Serial.println("☑ Bluetooth started: ESP32_Car");

  // WiFi setup
  WiFiManager wm;
  if (!wm.autoConnect("ESP32_Config")) {
    Serial.println("WiFi connection failed! Restarting...");
    ESP.restart();
  }
  Serial.print("🌐 WiFi connected. IP: ");
  Serial.println(WiFi.localIP());

  // Test network connectivity
  testPing();

  // Sync system time (important for Gmail SSL)
  if (!setSystemTime()) {
    Serial.println("⚠ Time sync failed! Email may not work.");
  }

  Serial.println("🚀 System initialized successfully.");
}

// ============================ LOOP ============================
void loop() {
  unsigned long now = millis();

  // ---------- Bluetooth Car Control ----------
  if (!isAlertActive && SerialBT.available()) {
    char cmd = SerialBT.read();
    Serial.printf("BT cmd: %c\n", cmd);
    switch (cmd) {
      case 'F': forward();  break;
```

```
      case 'B': backward(); break;
      case 'L': left();     break;
      case 'R': right();    break;
      case 'S': stopMotors(); break;
      default: stopMotors(); break;
    }
  }

  // ---------- Vibration Detection ----------
  if (now - lastVibSample >= vibSampleInterval) {
    lastVibSample = now;

    int v1 = digitalRead(VIB_SENSOR_1);
    int v2 = digitalRead(VIB_SENSOR_2);

    vib1Count = (v1 == HIGH) ? vib1Count + 1 : max(0, vib1Count - 1);
    vib2Count = (v2 == HIGH) ? vib2Count + 1 : max(0, vib2Count - 1);

    if (!isAlertActive &&
        (vib1Count >= vibConsecutiveThreshold || vib2Count >=
vibConsecutiveThreshold) &&
        (millis() - lastEmailTime > emailCooldown)) {

      Serial.println("⚠ Accident detected! Stopping motors...");
      triggerAccident();
      lastEmailTime = millis();
    }
  }

  delay(5);
}

// ====================== Motor Functions ========================
void forward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(EN, 200);
  digitalWrite(FRONT_LED, HIGH); // Turn on front LED
  digitalWrite(BACK_LED, LOW);   // Turn off back LED
}
```

```
void backward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(EN, 200);
  digitalWrite(FRONT_LED, LOW);  // Turn off front LED
  digitalWrite(BACK_LED, HIGH);  // Turn on back LED
}

void left() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(EN, 200);
  digitalWrite(FRONT_LED, LOW);  // Turn off both LEDs
  digitalWrite(BACK_LED, LOW);
}

void right() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(EN, 200);
  digitalWrite(FRONT_LED, LOW);  // Turn off both LEDs
  digitalWrite(BACK_LED, LOW);
}

void stopMotors() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(EN, 0);
  digitalWrite(FRONT_LED, LOW);  // Turn off both LEDs
  digitalWrite(BACK_LED, LOW);
}

// ================= Accident Trigger ============================
```

```
void triggerAccident() {
  isAlertActive = true;
  stopMotors();
  flushBluetoothInput();

  digitalWrite(FRONT_LED, HIGH);
  digitalWrite(BACK_LED, HIGH);
  digitalWrite(BUZZER, HIGH);

  String locationData = fetchLocation();
  Serial.println(locationData);

  sendEmail(locationData);

  resetAlertOutputs();
  isAlertActive = false;
  Serial.println("☑ Accident alert handled and system reset.");
}

// ==================== Helper Functions ========================
void resetAlertOutputs() {
  digitalWrite(FRONT_LED, LOW);
  digitalWrite(BACK_LED, LOW);
  digitalWrite(BUZZER, LOW);
}

void flushBluetoothInput() {
  while (SerialBT.available()) SerialBT.read();
}

// ==================== Location Fetch ==========================
String fetchLocation() {
  String locationMsg = "";
  if (!ensureWiFiConnected()) {
    Serial.println("WiFi not connected!");
    return "WiFi not connected!";
  }

  HTTPClient http;
  http.begin(apiURL);
  int httpCode = http.GET();
```

```
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    JSONVar data = JSON.parse(payload);

    if (JSON.typeof(data) != "undefined" && String((const char*)data["status"])
== "success") {
      double lat = (double)data["lat"];
      double lon = (double)data["lon"];
      locationMsg = "Accident detected!\n";
      locationMsg += "Latitude: " + String(lat, 6) + "\n";
      locationMsg += "Longitude: " + String(lon, 6) + "\n";
      locationMsg += "Google Maps: https://www.google.com/maps?q=" + String(lat,
6) + "," + String(lon, 6);
    } else {
      locationMsg = "Location unavailable.";
    }
  } else {
    locationMsg = "HTTP error: " + String(httpCode);
  }
  http.end();
  return locationMsg;
}

// ==================== Email Send ==============================
void sendEmail(String locationData) {
  Serial.println("📧 Preparing to send Gmail alert...");

  if (!ensureWiFiConnected()) {
    Serial.println("❌ Email not sent: No WiFi connection.");
    return;
  }
  Serial.println("🌐 WiFi confirmed connected. IP: " +
WiFi.localIP().toString());

  testPing(); // Test network connectivity before sending

  smtp.callback(smtpCallback);

  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;
  session.login.user_domain = "";

  SMTP_Message message;
  message.sender.name = "ESP32 Accident Alert";
  message.sender.email = AUTHOR_EMAIL;
  message.subject = "🚨 Accident Alert - ESP32 Vehicle";
  message.addRecipient("Owner", RECIPIENT_EMAIL);

  String html = "<h3>🚨 Accident Detected!</h3><p>An accident has been detected
by your ESP32 vehicle system.</p><pre>" + locationData + "</pre>";
  message.html.content = html.c_str();
  message.html.transfer_encoding = "base64";
  message.text.charSet = "us-ascii";

  const int maxRetries = 5;
  int retryCount = 0;
  bool emailSent = false;

  while (!emailSent && retryCount < maxRetries) {
    retryCount++;
    Serial.printf("Attempt %d/%d: Connecting to SMTP server...\n", retryCount,
maxRetries);
    if (!smtp.connect(&session)) {
      Serial.println("❌ Could not connect to mail server: " +
smtp.errorReason());
      if (retryCount < maxRetries) {
        Serial.println("Retrying in 2s...");
        delay(2000);
      }
      continue;
    }

    Serial.println("Sending email...");
    if (MailClient.sendMail(&smtp, &message)) {
      Serial.println("☑ Email sent successfully!");
      emailSent = true;
    } else {
      Serial.println("❌ Sending failed: " + smtp.errorReason());
```

```
      if (retryCount < maxRetries) {
        Serial.println("Retrying in 2s...");
        delay(2000);
      }
    }
    smtp.closeSession();
  }

  if (!emailSent) {
    Serial.printf("❌ Email sending failed after %d retries. Continuing system
operation.\n", maxRetries);
  }
}

// ====================== NTP Time Setup ========================
bool setSystemTime() {
  Serial.println("⏱️ Syncing time via NTP...");
  configTime(19800, 0, "pool.ntp.org", "time.google.com", "time.nist.gov"); //
Added third server
  struct tm timeinfo;
  int retries = 0;
  const int maxRetries = 20;

  while (!getLocalTime(&timeinfo) && retries < maxRetries) {
    Serial.print(".");
    delay(1000);
    retries++;
  }

  if (retries >= maxRetries) {
    Serial.println("\n❌ Time sync failed after " + String(maxRetries) + "
retries.");
    return false;
  }

  Serial.println("\n☑️ Time synced successfully: " + String(timeinfo.tm_year +
1900) + "-" +
                 String(timeinfo.tm_mon + 1) + "-" + String(timeinfo.tm_mday) + "
" +
                 String(timeinfo.tm_hour) + ":" + String(timeinfo.tm_min) + ":" +
String(timeinfo.tm_sec));
  return true;
```

```
}

// ====================== WiFi Management ========================
bool ensureWiFiConnected() {
  if (WiFi.status() == WL_CONNECTED) {
    return true;
  }

  Serial.println("🌐 WiFi disconnected! Attempting to reconnect...");
  WiFi.reconnect();
  int retries = 0;
  const int maxRetries = 10;

  while (WiFi.status() != WL_CONNECTED && retries < maxRetries) {
    Serial.print(".");
    delay(500);
    retries++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\n🌐 WiFi reconnected. IP: " + WiFi.localIP().toString());
    return true;
  } else {
    Serial.println("\n❌ WiFi reconnection failed after " + String(maxRetries) +
" retries.");
    return false;
  }
}

// ====================== Ping Test ============================
void testPing() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin("http://8.8.8.8");
    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("☑ Ping to 8.8.8.8 successful.");
    } else {
      Serial.println("❌ Ping to 8.8.8.8 failed. HTTP code: " +
String(httpCode));
    }
```

```
    http.end();
  } else {
    Serial.println("❌ WiFi not connected for ping test.");
  }
}

// ===================== SMTP Callback ========================
void smtpCallback(SMTP_Status status) {
  Serial.println(status.info());
  if (status.success()) {
    Serial.println("📧 Email delivery confirmed by callback!");
  } else {
    Serial.println("Callback: Email not delivered yet.");
  }
}
```

Future Scope:

- **Integrate with Blynk or IoT dashboard** for live status monitoring.
- **Add voice alerts** or **automatic emergency calling** features.
- **Camera module** to capture image upon impact for safety verification.