INTERNET OF THINGS PROJECTS

alexandrubrabete@yahoo.com

TEHNICAL UNIVERSITY OF CLUJ NAPOCA, MICROCONTROLLERS DEPARTMENT

1. **Smart Alarm System with Motion Detection, SMS and Web Dashboard**

Overview:

- Detects motion in a passive infrared (PIR) protected area.
- It emits audio (buzzer) and visual (RGB LED) signals when detected.
- Send automatic SMS and publish the event to a web dashboard with JWT authentication.
- Logs events encrypted to SD card for later auditing.
- Optionally, it allows manual alarm reset via the web interface.

Motivation for choosing the project:

- It combines physical and cyber security in a single solution.
- Demonstrates the management of sensitive data (encryption and authentication).
- Show skills in IoT, cryptography, web development.

Motivation for choosing components:

- **Arduino Uno + Ethernet W5500**: SPI hardware support, SD memory, industrial reliability.
- **SIM800L**: Global GSM communication, low power consumption.
- **HC-SR501 PIR**: wide detection angle, adjustable sensitivity.
- **Buzzer piezo + LED RGB**: multisensorial alerts.
- **SD card**: local storage, data loss resistance.

Testing and troubleshooting:

- **Hardware**:
    - Check the power and signals with a multimeter and oscilloscope.
    - Test the PIR sensor with a hot object to confirm detection.
    - Use the SD breakout board to write/read files on your PC.
- **Software**:
    - Serial monitor: Waiting for "Log start" and other messages.
    - Simulate detection: Position your hand periodically and watch the execution.
    - GSM debugging: Sends AT+CSQ, AT+CMGF, AT+CMGS and tracks responses.

- HTTP: Use Postman with JWT to test the /api/events endpoint.

Possible extensions:

- **Video recognition**: attaches camera mode and runs object detection.
- **Geofencing**: Notifies the owner based on the GPS location.
- **Analytics**: time event graphs and email reporting.
- **Smart home integration**: switch lights or IP cameras to alarm.

SOURCE CODE:

```
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
#include <AESLib.h>
#include <ArduinoJson.h>
#include <EthernetHttpClient.h>
#include <JWT.h>
#include <SoftwareSerial.h>

Network & Security
byte mac[] = {0xDE,0xAD,0xBE,0xEF,0xFE,0xED};
IPAddress ip(192,168,1,177);
Const Four* dashboardhost= "192.168.1.100";
const int dashboardPort = 443;
const char* jwtToken = "YOUR_JWT_TOKEN_HERE";

// GSM
 SoftwareSerial sim800(3, 4);

Pines
const int pirPin = 2;
const int buzzer = 5;

// AES
AESLib aes;
byte aesKey[16] = { /* 16 bytes */ };
byte aesIv[16] = { /* 16 bytes */ };

File logFile;

Prototypes
void sendSMS();
```

```
void postToDashboard(const String &data);
void logEncrypted(const byte *cipher, size_t len);

void setup() {
  pinMode(pirPin, INPUT);
  pinMode(buzzer, OUTPUT);
  sim800.begin(9600);
  Serial.begin(9600);

  Ethernet.begin(mac, ip);
  if (SD.begin(4)) {
    logFile = SD.open("events.log", FILE_WRITE);
    logFile.println("Log start");
    logFile.close();
  }
  delay(1000);
}

void loop() {
  if (digitalRead(pirPin) == HIGH) {
    tone(buzzer, 1000, 2000);
    String data = String(millis()) + ",MOTION";
    byte cipher[64];
    aes.encryptCBC((byte*)data.c_str(), cipher, data.length(), aesKey, aesIv);
    logEncrypted(cipher, sizeof(cipher));
    sendSMS();
    postToDashboard(data);
    delay(10000);
  }
}

void sendSMS() {
  sim800.println("AT+CMGF=1"); delay(500);
  sim800.println("AT+CMGS=\"+40712345678\""); delay(500);
  sim800.print("Alert: motion detected!"); delay(500);
  sim800.write(26);
  Serial.println("SMS trimis.");
}

void postToDashboard(const String &data) {
EthernetClient client;
 HttpClient https(client, dashboardHost, dashboardPort);
  https.beginRequest();
  https.post("/api/events");
  https.sendHeader("Content-Type", "application/json");
```

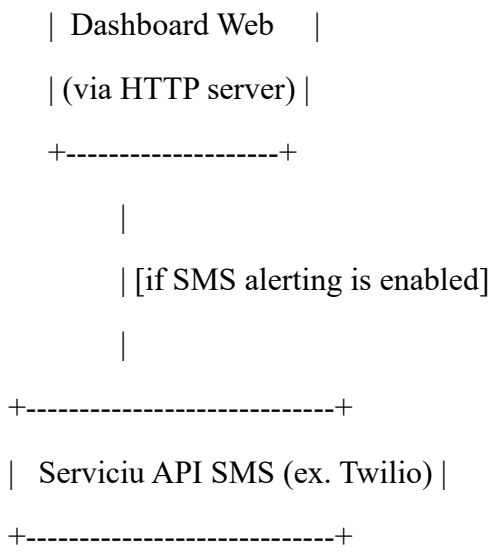
```
  https.sendHeader("Authorization", "Bearer " + String(jwtToken));
  https.sendHeader("Content-Length", data.length());
  https.beginBody();
  https.print(data);
  https.endRequest();
}

void logEncrypted(const byte *cipher, size_t len) {
  logFile = SD.open("events.log", FILE_WRITE);
  if (!logFile) return;
  for (size_t i = 0; i < len; i++) {
    logFile.print(cipher[i], HEX);
    logFile.print(" ");
  }
  logFile.println();
  logFile.close();
}
```

Feed:

- All components to GND and 5V of the Arduino

DESIGN:

```
+---------------------------+
|       NodeMCU ESP8266     |
|                    |
| D1 (GPIO5) --> PIR OUT |
| D2 (GPIO4) --> LED (Alert)
| 3V3        --> PIR VCC   |
| GND --> PIR GND |
+---------------------------+
         |
         | Conexiune WiFi
         |
  +--------------------+
```



Detailed Hardware Connections:

Component	Pin Component	Connected to NodeMCU
PIR Sensor	VCC	3V3
	GND	GND
	OUT	D1 (GPIO5)
LED Alert	+ (anode)	D2 (GPIO4)
	- (cathode)	GND (through resistor)

Other observations:

- **PIR detects motion**: sends HIGH signal on D1.
- **The LED lights up** if the alarm is activated.
- **ESP connects to WiFi**, displays the status on a web mini-dashboard (accessible from the browser to the network).
- **SMS** can be sent through an external web service such as **Twilio API** or **IFTTT Webhooks**, configured from Arduino code.

2. RFID & NFC Reader with Cloud Database and BI Analysis

Overview:

- Scans RFID tags and NFC cards, including HCE phones.
- Publish UIDs via TLS secure MQTT to AWS IoT Core.
- The data is stored in InfluxDB and displayed in Grafana with customizable dashboard.
- Local error management and retry buffer in case of disconnection.

Motivation for choosing the project:

- Demonstrate enterprise IoT skills: MQTT, TLS, AWS.
- Displays BI knowledge and visualization tools.
- Ideal for resume: cloud experience and industrial protocols.

Motivation for choosing components:

- **Arduino MKR WiFi 1010**: ECC securizat, WiFi robust.
- **PN532**: Support I2C/SPI, RFID and NFC reading.
- **PubSubClient + ArduinoJson**: Accurate serialization and memory management.

Testing and troubleshooting:

- **WiFi connectivity**: ping to gateway and AWS API test with curl.
- **RFID/NFC reading**: uses known cards and debug Serial.print(uid).
- **MQTT TLS:** Runs the MQTT client in debug mode and checks mqtt.connected().
- **InfluxDB & Grafana**: Simulated inserts with Telegraf and in-interface confirmation.

Possible extensions:

- **Access control**: RFID authorization with local and cloud base.
- **Mobile Scan**: Android app that emulates HCE.
- **Machine learning**: Detects unusual access patterns.
- **Blockchain logging**: immutable storage of events.

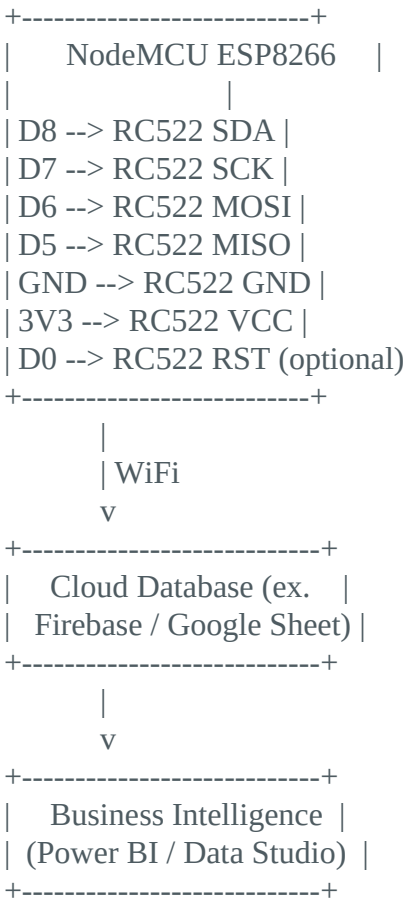
SOURCE CODE:

```
#include <SPI.h>
```

```
#include <Wire.h>
#include <Adafruit_PN532.h>
#include <WiFiNINA.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

// PN532 I2C
Adafruit_PN532 nfc(Wire);

// WiFi + MQTT
const char* ssid     = "SSID";
const char* pass     = "PASSWORD";
const char* mqttServer = "your-aws-endpoint";
WiFiClientSecure net;
PubSubClient mqtt(net);

void setup() {
  Serial.begin(115200);
  Wire.begin();
  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  nfc.SAMConfig();

  WiFi.begin(ssid, pass);
  where ( WiFi .status() != WL_CONNECTED) delay(500);
  net.setCACert (AWS_CERT_CA);
  mqtt.setServer(mqttServer, 8883);
}

void loop() {
  if (!mqtt.connected()) {
    while (!mqtt.connect("rfidClient")) delay(500);
  }
  mqtt.loop();
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };
  uint8_t uidLength;
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength,
100);
  if (success) {
String uidStr;
    for (uint8_t i=0; i<uidLength; i++) { uidStr += String(uid[i], HEX); }
    publishTag(uidStr);
    delay(1000);
  }
```

```
}

void publishTag(const String &uid) {
  StaticJsonDocument<200> doc;
  doc["uid"] = uid;
  doc["time"] = millis();
  char buf[256];
  size_t n = serializeJson(doc, buf);
  mqtt.publish("iot/rfid", buf, n);
}
```

SCHEMA:



Detailed Hardware Connections:

Component	Pin RC522	Connected to NodeMCU (GPIO)
RC522 RFID	SDA	D8 (GPIO15)
	SCK	D7 (GPIO13)
	MOVES	D6 (GPIO12)
	MISO	D5 (GPIO14)

Component	Pin RC522	Connected to NodeMCU (GPIO)
	RST	D0 (GPIO16) (optional)
	GND	GND
	VCC	3V3

Functionality:

- RFID cards (e.g. MIFARE 13.56MHz) are scanned by the RC522.
- ESP8266 send the UID and timestamp over WiFi to a **cloud database**.
- The data is then analyzed with a Business Intelligence (BI) tool.

3. Biometric Digital Vault with Keyboard, RFID, Fingerprint and Bluetooth Backup

Overview:

- Safe with three-step access: PIN, RFID and fingerprint scanning.
- In case of failure, OTP fallback generated and transmitted via BLE to the smartphone.
- Encrypted access log stored on external eMMC.
- BLE interface for administration and synchronization.

Motivation for choosing the project:

- It shows multi-factor integration and BLE management.
- Demonstrate physical and digital security.

Motivation for choosing components:

- **Nano 33 BLE Sense**: BLE 5.0, procesor ARM.
- **RC522**: Fast RFID unit.
- **GT-521F32**: Small margin of error in reconnaissance.
- **Keypad 4×4**: Compact and reliable input.
- **eMMC**: Large-capacity and secure storage.

Testing and troubleshooting:

- **PIN+RFID:** tests sequentially, log to Serial Monitor.
- **Fingerprint**: debug error codes from `finger.getImage()`.
- **OTP BLE:** uses nRF Connect to write/read the feature.
- **eMMC**: Check files with an SD adapter.

Possible extensions:

- **Facial recognition**: camera mode with OpenMV.

- **Cloud integration**: remote logging and notifications.
- **Firmware update**: OTA for BLE and eMMC.

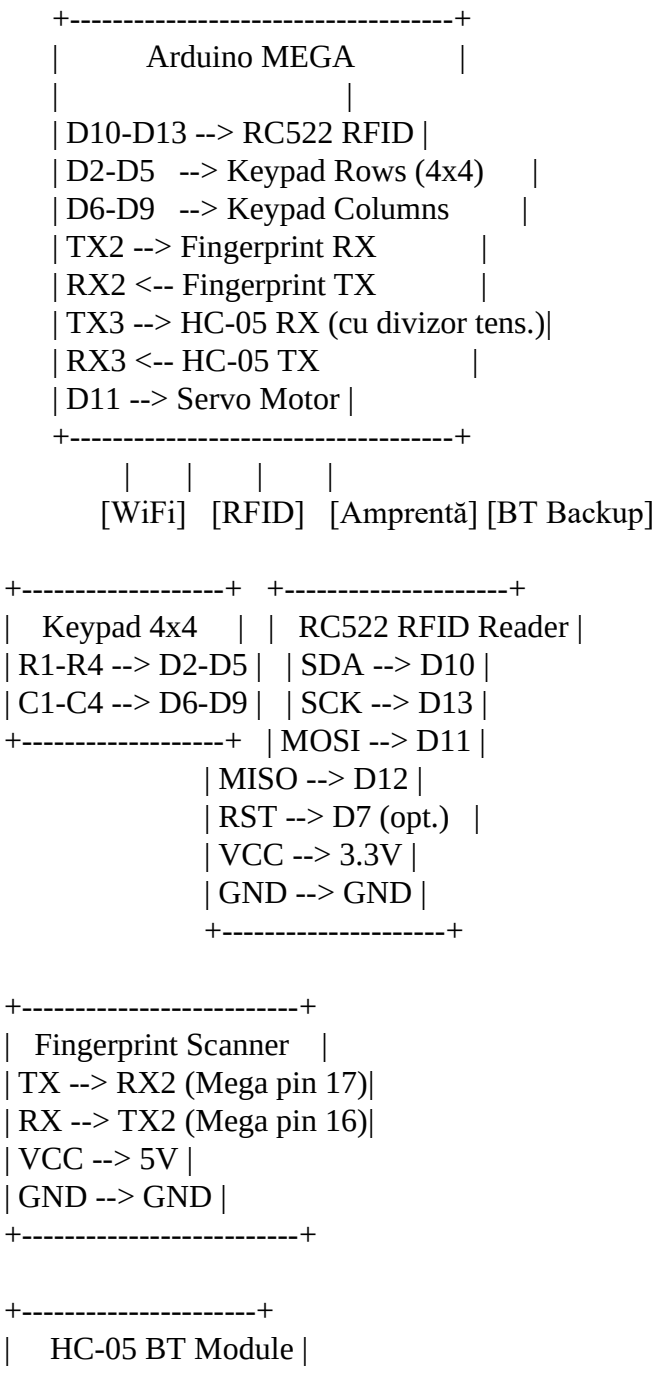
SOURCE CODE:

```
#include <Keypad.h>
#include <SPI.h>
#include <MFRC522.h>
#include <Adafruit_Fingerprint.h>
#include <ArduinoBLE.h>
#include <Servo.h>

// RFID
SS_PIN #define  10
#define RST_PIN 9
 rfid MFRC522(SS_PIN, RST_PIN);

// Fingerprint
#include <SoftwareSerial.h>
SoftwareSerial fingerSerial(12, 11);
Adafruit_Fingerprint finger(&fingerSerial);

// Keypad
const byte ROWS = 4, COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9,8,7,6};
byte colPins[COLS] = {5,4,3,2};
Keypad keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

Servo
Servo lockServo;

// BLE OTP
BLEService otpService("1234");
BLECharacteristic otpChar("ABCD", BLERead | BLEWrite, 20);
String currentOTP = "000000";

void setup() {
  Serial.begin(115200);
  SPI.beg(); rfid.PCD_Init();
```

```
  finger.begin(57600);
  lockServo.attach(6); lockServo.write(0);

WAS
  BLE.begin();
  BLE.setLocalName("SafeBoxBLE");
  BLE.setAdvertisedService(otpService);
  otpService.addCharacteristic(otpChar);
  BLE.addService(otpService);
  BLE.advertise();
}

void loop() {
// 1) RFID
  if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {
    IF (RFID.uid.uidByte[0] == 0xDE) { // validation example
// 2) Fingerprint
      if (getFingerprintID() > 0) {
        grantAccess();
      } else {
        Serial.println("FP failed");
      }
    }
    RFID.PICC_HaltA();
  }
3) BLE fallback
 BLEDevice central = BLE.central();
  if (central && central.connected()) {
    if (otpChar.written()) {
      if (otpChar.value() == currentOTP) grantAccess();
    }
  }
}

uint8_t getFingerprintID() {
  if (finger.getImage() != FINGERPRINT_OK) return -1;
  finger.image2Tz();
  if (finger.fingerFastSearch() == FINGERPRINT_OK) {
    return finger.fingerID;
  }
  return -1;
}

void grantAccess() {
  Serial.println("Access granted!");
```

```
  lockServo.write(90);
  delay(3000);
  lockServo.write(0);
regenerate OTP
  currentOTP = String(random(100000,999999));
  otpChar.writeValue(currentOTP);
}
```

DESIGN:



| TX --> RX3 (Mega 15)|
| RX --> TX3 (Mega 14)* (cu 1K+2K div.) |
| VCC --> 5V |
| GND --> GND |
+----------------------+

+---------------------+
| Servo Motor |
| Signal --> D11 |
| VCC --> 5V ext. |
| GND --> GND |
+---------------------+

Important notice:

- **The Arduino MEGA** is recommended because it has **several hardware serial ports (Serial1, Serial2, Serial3)** required for the fingerprint reader and Bluetooth module.
- If you're using Arduino UNO:
    - You can use the **SoftwareSerial library** for fingerprint and Bluetooth, but conflicts may occur.
- **Bluetooth RX must be protected with 5V → 3.3V voltage divider**, as the HC-05's RX pin is not 5V tolerant.
- The servo must be powered **from an external source**, not directly from the Arduino, to avoid resets.

Authentication methods in the scheme:

1. **RFID** – tags/cards
2. **Fingerprint** – biometric sensor
3. **PIN code** – entered from the keyboard
4. **Bluetooth** – backup in case of emergency (password sent from the phone)
5.

4. Industrial Gas Sensor with IIoT and OTA Feedback

Overview:

- Gas concentration measurements (LPG, smoke, methane) with MQ-2 and op-amp.
- Telemetry in Azure IoT Hub via MQTT+WebSockets.
- OTA updates with fallback to SD and current version reporting.
- Automatic calibration at start-up and at programmed intervals.

Motivation for choosing the project:

- Industrial cloud integration and embedded devops.
- Demonstrates the CI/CD pipeline for firmware.

Motivation for choosing components:

- **ESP32-WROOM:** dual-core, WiFi, OTA support.
- **Op-amp circuit**: signal amplification and filtering.
- **SD Card**: OTA backup storage and logs.

Testing and troubleshooting:

- **MQ-2 sensor**: calibrated zero and span, compare with reference values.
- **MQTT:** SAS token authentication and validated topics in the Portal.
- **OTA**: Tests invalid and fallback URLs on SD.
- **Calibration**: confirms the values in fresh air.

Possible extensions:

- Senzori multipli (CO2, CO, NOx).
- Panel HMI local cu touch display.
- AI cloud for risk prediction.

SOURCE CODE:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Update.h>
#include <SPI.h>
#include <SD.h>

// Azure IoT
const char* ssid       = "SSID";
const char* pass       = "PASSWORD";
const char* iothub     = "YOUR_IOTHUB.azure-devices.net";
const char* deviceId   = "esp32device";
const char* key        = "BASE64_KEY";
WiFiClientSecure net;
PubSubClient mqtt(net);

MQ-2
const int gasPin = 34;
FATHER
#define OTA_URL "http://server/firmware.bin"
```

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, pass);
  where ( WiFi .status() != WL_CONNECTED) delay(500);
  net.setCACert (AZURE_CA);
  mqtt.setServer(iothub, 8883);

  if (SD.begin()) Serial.println("SD OK");
}

void loop() {
  if (!mqtt.connected()) mqtt.connect(deviceId);
  mqtt.loop();
  int val = analogRead(gasPin);
  sendTelemetry(val);
  checkForOTA();
  delay(5000);
}

void sendTelemetry(int value) {
  StaticJsonDocument<128> doc;
  doc["gas"] = value;
  ox chariot[128];
  size_t n = serializeJson(doc, buf);
  mqtt.publish("devices/esp32/messages/events/", buf, n);
}

void checkForOTA() {
  if (WiFi.status()==WL_CONNECTED && millis()%60000<1000) {
WiFiClient client;
    if (client.connect("server",80)) {
      client.print(String("GET ") OTA_URL " HTTP/1.1\r\nHost: server\r\n\r\n");
      while (client.connected()) {
        if (client.available()) {
          if (Update.begin(UPDATE_SIZE_UNKNOWN)) {
            size_t written = Update.writeStream(client);
            if (Update.end()) {
              Serial.println("OTA OK");
              ESP.restart();
            }
          }
        }
      }
    }
```

```
  }
}
```

```
  +---------------------------+
  |          ESP32 |
  |                        |
  | GPIO34 <-- MQ135 Analog Out|
  | GPIO25 --> Buzzer |
  | GPIO26 --> LED Alert       |
  | 3.3V --> MQ135VDC |
  | GND --> MQ135 GND |
  +---------------------------+
            |
       Wi-Fi MQTT
            |
            v
+-----------------------------+
|  IIoT Dashboard (ThingsBoard,|
|  Node-RED, Home Assistant)   |
+-----------------------------+
            |
          OTA Updates
            |
            v
+-----------------------------+
```

| Firmware Web Uploader or |

| Arduino OTA via IDE |

+-----------------------------+

Detailed Hardware Connections:

Component	Pin Component	Connected to ESP32
MQ-135	AO (Analog Out)	GPIO34 (ADC1_CH6)
	VCC	3.3V
	GND	GND
Buzzer	+	GPIO25
	-	GND
LED	Anode	GPIO26 (with 220Ω resistor)
	Cathode	GND

Functionality:

- The MQ135 reads toxic gas levels (analog).
- ESP32:
  - Sends data to the IIoT dashboard (e.g. ThingsBoard) via **MQTT/HTTP**.
  - Triggers **buzzer and LED** if the gas value crosses a threshold.
  - Receive **OTA updates** (via browser or local network).
- The IIoT platform saves data for **analysis, alerting, and history**.

5. Anomaly Detection with ML & Streaming BLE + PC Dashboard

Overview:

- Edge-ML for behavioral classification using TensorFlow Lite Micro.
- Live streaming of IMU data via BLE to a Python application (PyQt).
- Adaptive anomaly detection threshold with Kalman algorithm.

Motivation for choosing the project:

- Demonstrate embedded and common ML skills.
- Combine sensoristică cu visual analytics.

Motivation for choosing components:

- **Nano 33 BLE Sense**: IMU, BLE.

- **TensorFlow Lite Micro**: Low-memory inference.
- **Python Dashboard**: interactive analysis and graphs.

Testing and troubleshooting:

- **IMU:** Compare raw data with another reference sensor.
- **BLE**: debug with nRF Connect, track packets.
- **ML:** tests on recorded data and offline validation.
- **Kalman**: adjustment of covariances and stability check.

Possible extensions:

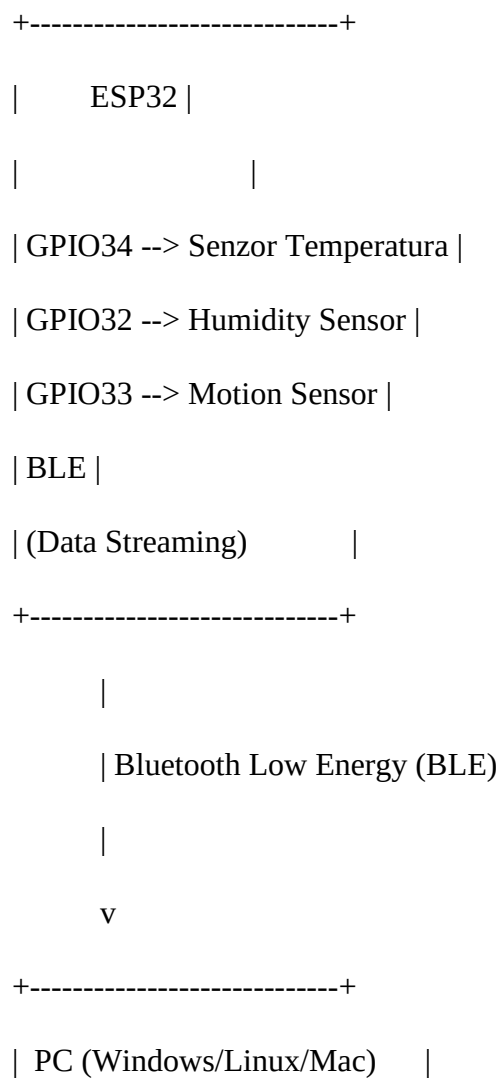
- Multi-event classification (fall, shock, vibration).
- Local event storage on SD.
- Websocket streaming to the browser.

SOURCE CODE:

```
#include <Arduino_LSM9DS1.h>
#include <TensorFlowLite.h>
#include <ArduinoBLE.h>
#include "model.h"

const int windowSize = 128;
static float inputData[windowSize * 6];
int dataCount = 0;

WAS
BLEService imuService("180D");
BLECharacteristic imuChar("2A37", BLERead | BLENotify, windowSize*6*4);

void setup() {
  Serial.begin(115200);
  IMU.begin();
  BLE.begin();
  Blay.setlocalname("Emastream");
  imuService.addCharacteristic(imuChar);
  BLE.addService(imuService);
  BLE.advertise();
}

void loop() {
  float ax,ay,az,gx,gy,gz;
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    IMU.readAcceleration(ax,ay,az);
```

```
    IMU.readGyroscope(gx,gy,gz);
    int idx = dataCount * 6;
    inputData[idx]=ax; inputData[idx+1]=ay; inputData[idx+2]=az;
    inputData[idx+3]=gx; inputData[idx+4]=gy; inputData[idx+5]=gz;
dataCount++;
    if (dataCount>=windowSize) {
// streaming
      if (BLE.connected() imuChar.writeValue((uint8_t*)inputData,
windowSize*6*4);
      dataCount = 0;
    }
  }
}
```

DESIGN:



| Dashboard (ex: Python, C++) |

| (Visualization and Anomaly) |

+----------------------------+

|

| (Feedback, Visualized Data)

|

v

+----------------------------+

| Algoritm Machine Learning |

| (ex: TensorFlow Lite, SciPy)|

| (Anomaly Detection) |

+----------------------------+

Detailed Hardware Connections:

Component	**Pin Component**	**Connected to ESP32**
Motion Sensor	OUT	GPIO33
Temperature Sensor	DATA (ex: DHT22)	GPIO34
Humidity sensor	DATA (ex: DHT22)	GPIO32
BLE	RX/TX	Wi-Fi BLE

Temperature and humidity sensors, such as **the DHT22**, can be connected to the analog or digital pins on the ESP32, and the motion signal will connect directly to a digital pin.

Functionality:

1. **Sensor data collection**:
   - The ESP32 collects data from sensors via BLE (continuously or in intervals).
   - The data can include temperature, humidity, and movement values.
2. **Streaming BLE:**
   - The ESP32 sends the data to a **Dashboard PC** using BLE.
3. **Detecție Anomalii (Machine Learning)**:

    - The **ML model** trains and learns a pattern from the collected data (e.g., anomalies in the pattern of motion or temperature).
    - Anomaly detection algorithms can include **clustering methods**, **change point detection**, etc. (e.g., SVM, K-Means).
4. **PC Dashboard**:
    - Graphical visualization of data and alerting of anomalies (for specific cases).
    - Feedback on anomalies: the user can receive visual alerts.
5. **ML model on PC:**
    - The data from the BLE is sent to the PC to be processed and to detect any anomalies based on a trained model.

Examples of technologies used:

- **PC Dashboard**: Python with **matplotlib**, **PyQt5** for interface, **pandas** for data processing.
- **BLE Streaming**: **ESP32 BLE** libraries for PC communication.
- **Machine Learning**: **TensorFlow Lite** (to implement a lightweight model on ESP32), **scikit-learn**, **Pandas** (for data processing on PC).

6. Hardware Firewall Deep Packet Inspection with VPN and Analytics

Overview:

- DPI over HTTP, DNS, TLS SNI, URL control.
- WireGuard VPN forwarding.
- JSON metadata logging and local API for analytics.

Motivation for choosing the project:

- Demonstrate advanced knowledge in networking and security.
- It shows VPN and log system integration.

Motivation for choosing components:

- **Pico W + W5100**: dual core and Ethernet.
- **WireGuard**: criptografie modernă.
- **SD Card**: stocare JSON.

Testing and troubleshooting:

- **DPI:** Use Wireshark to inspect packages.
- **VPN**: ping, traceroute prin tunel.
- **API**: Tests endpoins with curl and Postman.
- **JSON log: Validates** the schema with JSONLint.

Possible extensions:

- Modbus DPI for ICS.
- Graphs in Kibana.
- Dynamic rules via WebSocket.

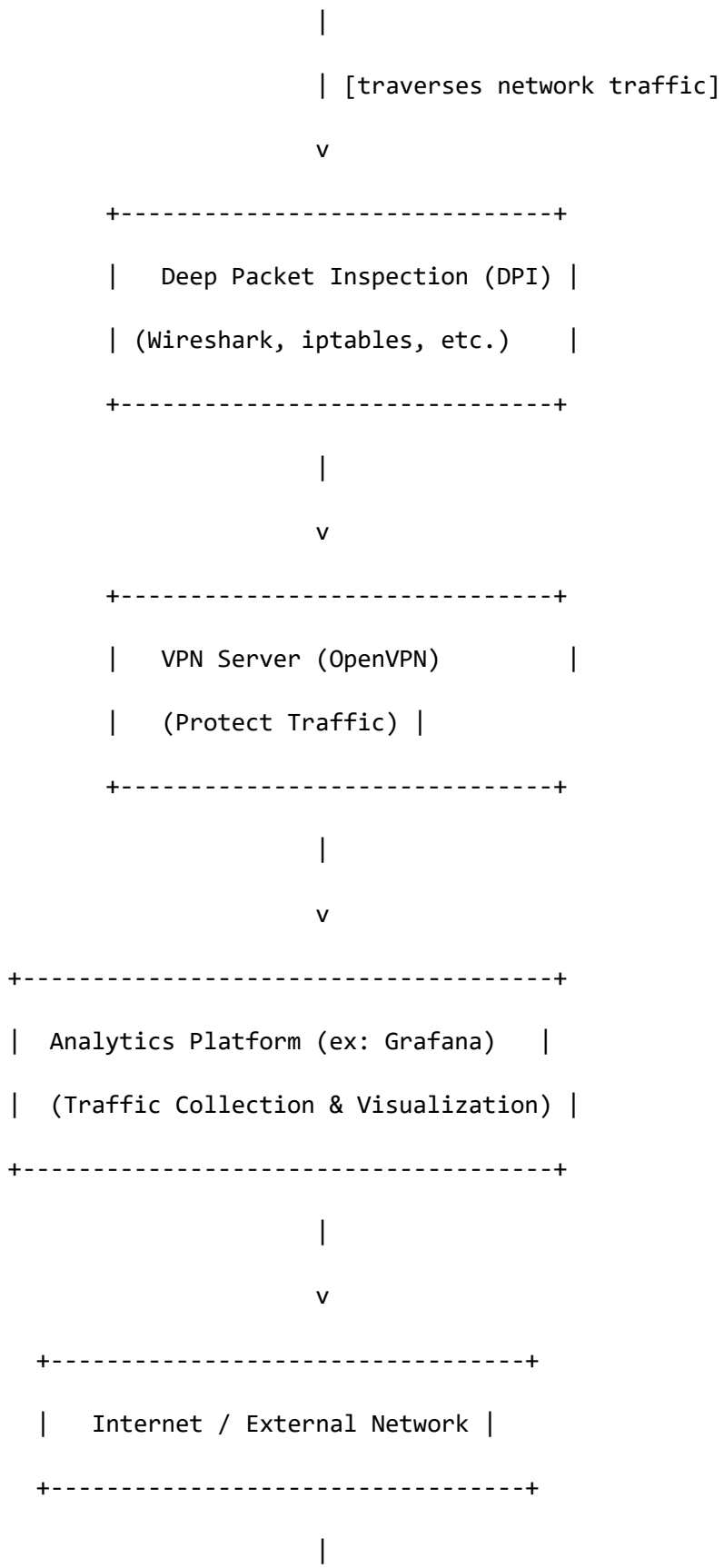
SOURCE CODE:

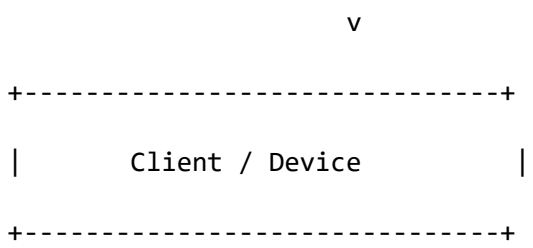
```
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
#include <WireGuard.h>
#include <ArduinoJson.h>
#include <EthernetHttpClient.h>

// Ethernet + SD
byte mac[] = {0xDE,0xAD,0xBE,0xEF,0xFE,0xED};
IPAddress ip(192,168,1,177);
EthernetServer server(80);
File logFile;

// WireGuard
WireGuard wg;
const char* wgPrivateKey = "YOUR_PRIVATE_KEY";
const char* wgEndpoint   = "vpn.example.com:51820";
const char* wgPublicKey  = "PEER_PUBLIC_KEY";
IPAddress wgAllowedIp(10,0,0,0);

void setup() {
  Serial.begin(115200);
  Ethernet.begin(mac, ip);
  server.begin();
  if (!SD.begin(4)) Serial.println("SD init failed");

// WireGuard init
  WG.start(wgPrivateKey, wgEndpoint);
  wg.addPeer(wgPublicKey, wgAllowedIp, 32);
```

```
  logFile = SD.open("fw_log.json", FILE_WRITE);
}

void loop() {
  wg.update();
  EthernetClient client = server.available();
  if (client) {
    Packet packet = wg.readPacket();
    if (packet.isValid()) {
      if (packet.isTCP() && packet.dPort()==80) {
        String host = packet.httpHeader("Host");
        if (!host.equals("allowed.example.com")) {
          logEvent(packet);
          Packet.drop();
        } else {
          packet.forward();
        }
      } else {
        packet.forward();
      }
    }
    client.stop();
  }
}

void logEvent(const Packet &p) {
  StaticJsonDocument<200> doc;
  doc["time"] = millis();
  doc["srcIP"] = p.srcIP().toString();
  Dock["Disstep"] = P.Disstep().Toastring();
  doc["host"]  = p.httpHeader("Host");
  logFile = SD.open("fw_log.json", FILE_WRITE);
  serializeJson(doc, logFile);
  logFile.println();
  logFile.close();
}
```

DESIGN:

```
                              +------------------------------+
                              |      Raspberry Pi 4          |
                              |  (CPU Puternic, Ethernet/WiFi)|
                              +------------------------------+
```


| [traverses network traffic]
v
Deep Packet Inspection (DPI)
(Wireshark, iptables, etc.)
VPN Server (OpenVPN)
(Protect Traffic)
Analytics Platform (ex: Grafana)
(Traffic Collection & Visualization)
Internet / External Network



Detailed Hardware Connections:

Component	Connectivity	Connection details
Raspberry Pi	Ethernet/Wi-Fi	Connected to the network (LAN/Wi-Fi)
Network Interface	Ethernet/Wi-Fi	Connect your Raspberry Pi to the network
Software DPI	pe Raspberry Pi	Using **Wireshark**, **iptables** for packet analysis
VPN Server	pe Raspberry Pi	**OpenVPN** for traffic encryption
Software Analytics	on your Raspberry Pi (or PC)	**Grafana** / **Kibana** for Network Traffic Analysis

Functionality:

1. **Deep Packet Inspection (DPI)**:
    - The Raspberry Pi analyzes all network packets that pass through the system, looking for anomalies (e.g. viruses, DoS/DDoS attacks).
    - Use **Wireshark** or **iptables** to identify and inspect packages.
2. **VPN:**
    - **OpenVPN** is implemented to protect network traffic by encrypting data between client and server.
    - Internet routes are secured by VPN, preventing external attacks.
3. **Analytics**:
    - All traffic data is collected and sent to an analytics platform (e.g. **Grafana** or **Kibana**).
    - This data can be viewed in real-time to observe suspicious traffic, network anomalies, statistics, or optimize network performance.
4. **Raspberry Pi**:
    - The Raspberry Pi 4 is used as processing hardware, having enough resources to run the VPN and do the packet analysis in real-time.
5. **All-round protection**:
    - All communications between the client and the server are encrypted and analyzed to prevent attacks and ensure a safe environment for users.

Examples of software used:

- **DPI: Wireshark**, **iptables** (for filtering traffic on Raspberry Pi).
- **VPN**: **OpenVPN** (for traffic encryption).
- **Analytics**: **Grafana**, **Kibana**, **InfluxDB** (for collecting and visualizing traffic data).

Other observations:

- **The Raspberry Pi 4** is recommended for this project due to its processing power and multiple network interface.
- **OpenVPN** can be used to create a VPN server directly on the Raspberry Pi, ensuring encrypted network traffic.
- **Wireshark** will help monitor and analyze the data packets passing through the network in detail.
- The **Grafana** and **Kibana** platforms are used for visual analysis and generating traffic graphs and statistics.

7. SafeBox with RFID, Fingerprint and NFC Mobile Authentication

Overview:

- 3-step authentication: RFID, fingerprint, NFC HCE.
- NFC fallback on your phone with Android HCE.
- Encrypted local log and BLE notifications.

Motivation for choosing the project:

- Demonstrates RFID, biometric, and mobile interoperability.
- It shows HCE and security concepts.

Motivation for choosing components:

- **ESP32-S3**: Support HCE, BLE.
- **RC522 & GT-521F32**: for RFID and biometric scanning.
- **NFC Library**: Mobile Token.

Testing and troubleshooting:

- **RFID:** mapping of authorized UIDs.
- **Fingerprint**: Tests the stability of the scan.
- **NFC**: Use the HCE app on Android.
- **BLE:** Check status notifications.

Possible extensions:

- Smart card EMV support.
- Read/ write NFC tags.
- Cloud audit trail.

SOURCECE CODE:

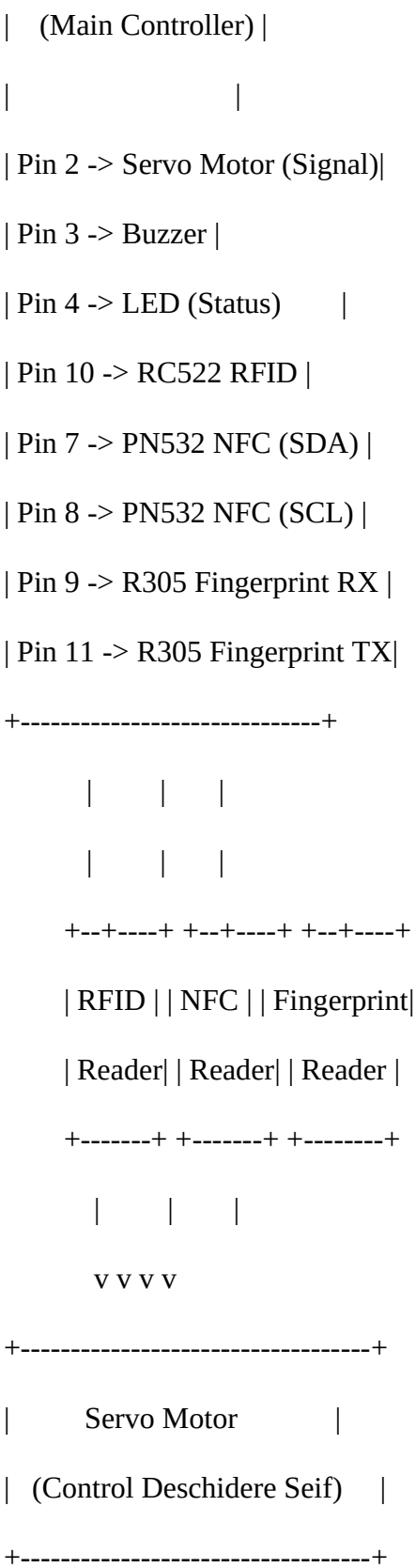
```
#include <SPI.h>
#include <Wire.h>
#include <MFRC522.h>
#include <Adafruit_Fingerprint.h>
#include <NFC.h>
#include <Servo.h>

// RFID
SS_PIN #define  10
#define RST_PIN 9
 rfid MFRC522(SS_PIN, RST_PIN);

// Fingerprint
#include <SoftwareSerial.h>
SoftwareSerial fingerSerial(12, 11);
Adafruit_Fingerprint finger(&fingerSerial);

// NFC (HCE)
NFC nfc;

Servo
Servo lockServo;

void setup() {
  Serial.begin(115200);
  SPI.beg(); rfid.PCD_Init();
  finger.begin(57600);
  nfc.begin();     // NFC HCE initialization
  lockServo.attach(6);
  lockServo.write(0);
}

void loop() {
// 1) RFID check
  if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {
    if (you select RFID(rfid.uid.uidByte, rfid.uid.size)) {
// 2) Fingerprint check
```

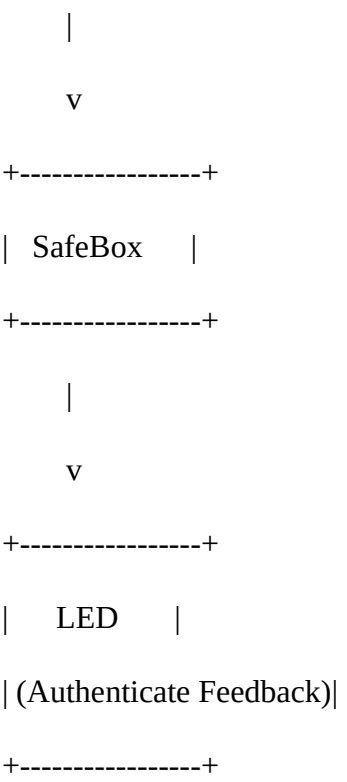
```
      if (getFingerprintID() >= 0) {
        grantAccess();
      }
    }
    RFID.PICC_HaltA();
  }
// 3) NFC fallback
  if (nfc.available()) {
    byte buf[16];
    int len = nfc.read(buf, sizeof(buf));
    if (validateNFC(buf, len)) grantAccess();
  }
}

bool validateRFID(byte *uid, byte len) {
// compare with stored UIDs
  return true;
}

int getFingerprintID() {
  if (finger.getImage() != FINGERPRINT_OK) return -1;
  finger.image2Tz();
  if (finger.fingerFastSearch() == FINGERPRINT_OK) return finger.fingerID;
  return -1;
}

bool validateNFC(byte *data, int len) {
// validate mobile HCE token
  return true;
}

void grantAccess() {
  lockServo.write(90);
  delay(3000);
  lockServo.write(0);
  Serial.println("Access Granted");
}
```

DESIGN:

+----------------------------+

| Arduino UNO |


| (Main Controller) |
| |
| Pin 2 -> Servo Motor (Signal)|
| Pin 3 -> Buzzer |
| Pin 4 -> LED (Status) |
| Pin 10 -> RC522 RFID |
| Pin 7 -> PN532 NFC (SDA) |
| Pin 8 -> PN532 NFC (SCL) |
| Pin 9 -> R305 Fingerprint RX |
| Pin 11 -> R305 Fingerprint TX|
RFID Reader
NFC Reader
Fingerprint Reader
Servo Motor
(Control Deschidere Seif)



Detailed Hardware Connections:

Component	Pin Component	Connected to Arduino
RC522 RFID	SDA, SCK, MOSI, MISO, RST, GND, VCC	Battery 10, Battery 13, Battery 11, Battery 12, Battery 9, 3.3V, GND
NFC Module (PN532)	SDA, SCL, VCC, GND	Battery 7 (SDA), Battery 8 (SCL), 5V, GND
Fingerprint Reader (R305)	RX, TX, VCC, GND	Pin 9 (RX), Pin 11 (TX), 5V, GND
Servo Motor	Signal (Control)	Pin 2
Buzzer	(+)	Pin 3
LED	(+)	Pin 4

Functionality:

1. **Authenticate RFID:**
   - The **RFID reader** reads the cards and compares them with the stored data.
   - If the card is valid, the safe is allowed to be opened.
2. **Fingerprint Authentication**:
   - **The fingerprint sensor** compares the user's fingerprint with the stored database.
   - If the fingerprint is valid, the safe is allowed to be opened.
3. **NFC authentication via mobile phone**:

    - **The NFC module** allows you to read an NFC tag from your mobile phone.
    - If the tag is valid, the vault is allowed to be opened.
4. **Visual and Audible Feedback**:
    - If the authentication is successful, a green **LED lights up and** the buzzer **is activated**.
    - If authentication fails, a red LED lights up and **the buzzer** emits a short signal.
5. **Opening the safe**:
    - If either authentication method is valid, a **servo motor** will be activated to open the safe.

Examples of software used:

- **RFID:** MFRC522 library for the RFID reader.
- **Fingerprint**: Adafruit **Fingerprint Sensor library** for fingerprint reader.
- **NFC**: The **Adafruit_PN532 library** for the NFC module.
- **Servo Motor**: Using the **Servo.h** library to control the servo.

Additional features:

- **Protection**: If all authentication methods fail (or a number of attempts are exceeded), additional protection can be activated, such as a **servo lock** or an **alarming buzzer** to signal an unauthorized access attempt.

8. Self-Healing Mesh Router with QoS and LTE Backhaul

Overview:

- ESP32 mesh nodes capable of automatically choosing the route with minimal latency.
- LTE backhaul through SIM7600 if the WiFi mesh is compromised.
- Monitoring topology and MQTT traffic.

Motivation for choosing the project:

- Distributed networking showcase and backhaul reserves.
- Demonstrates LTE and QoS knowledge.

Motivation for choosing components:

- **ESP32 DevKit**: mesh and multitasking.
- **SIM7600**: module LTE cu fallback.
- **TinyGSM, PubSubClient**: MQTT pe cellular.
- **OLED:** status view.

Testing and troubleshooting:

- **Mesh**: Visualizes the topology in Serial.
- **QoS**: Measures ping and jitter.
- **LTE:** Tests throughput and reconnection.
- **MQTT:** debug topics in the broker.

Possible extensions:

- VLAN segmentation for IoT.
- Automatic roaming between 3G/4G/5G bands.
- Centralized dashboard on Node-RED.

SOURCE CODE:

```
#include <painlessMesh.h>
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <TinyGsmClient.h>
#include <PubSubClient.h>

// Mesh
Scheduler userScheduler;
painlessMesh mesh;

'RE
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire);

// LTE
#define TINY_GSM_MODEM_SIM7600
#include <TinyGsmClient.h>
TinyGsm modem(Serial1);
TinyGsmClient lteClient(modem);
PubSubClient mqt (lteClient, "broker.hivemq.com", 1883);

// QoS data
void qosCheck() {
// measure latency and reroute via LTE if needed
}

// Message receive
void receivedCallback(uint32_t from, String &msg) {
  display.clearDisplay();
```

```
  display.setCursor(0,0);
  display.println("From: " + String(from));
  display.println(msg);
  display.display();
}

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

'RE
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.display();

// Mesh
  mesh.setDebugMsgTypes(ERROR | STARTUP);
  mesh.init("MeshNET", "pass123", &userScheduler, 5555);
  mesh.onReceive(&receivedCallback);

// LTE
  Serial1.begin(115200);
  modem.init();
  mqtt.setServer("broker.hivemq.com", 1883);
}

void loop() {
  userScheduler.execute();
  mesh.update();

  qosCheck();

  if (!mqtt.connected()) mqtt.connect("meshNode");
  mqtt.loop();
}
```

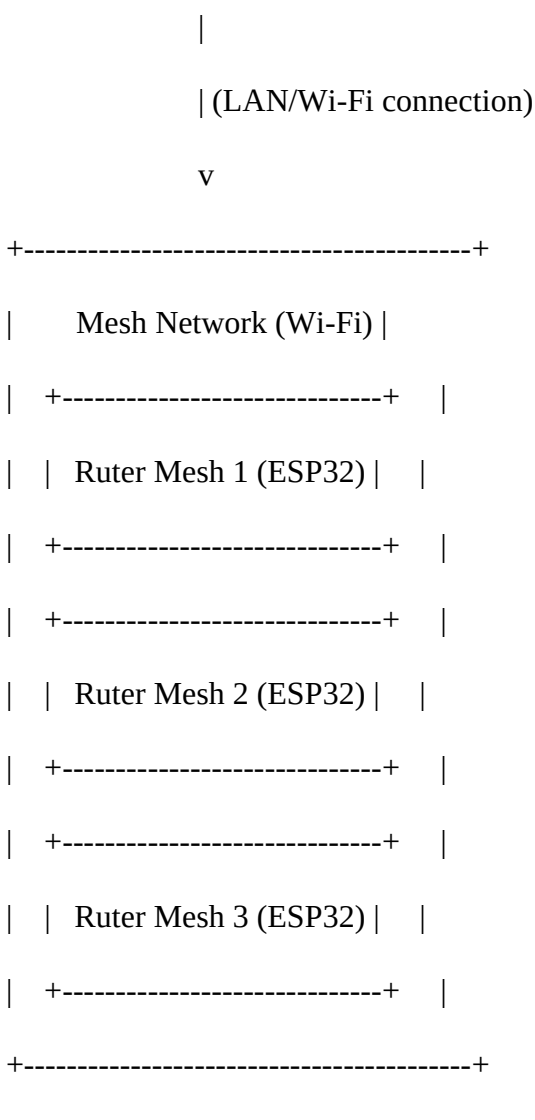
DESIGN:

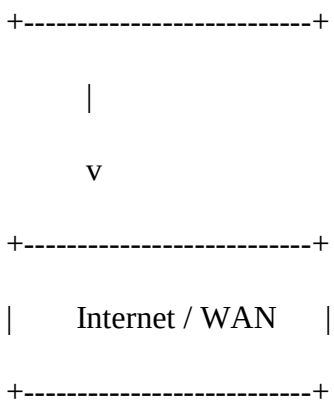
+-------------------------+

| ESP32 / Pi |

| (Ruter Central / Server) |

+-------------------------+


| (LAN/Wi-Fi connection)
v
Mesh Network (Wi-Fi)
Ruter Mesh 1 (ESP32)
Ruter Mesh 2 (ESP32)
Ruter Mesh 3 (ESP32)
(LTE backhaul for backup connection)
Modem LTE (ex: SIM7600)
(LTE Backhaul for
internet connection)



Detailed Hardware Connections:

Component	Connection	Connection details
ESP32	Wi-Fi	Mesh network connection
Raspberry Pi	LAN / Wi-Fi / Ethernet	Network connection
SIM7600 LTE	UART, 5V, GND, ANTENNA	LTE connectivity module
Wi-Fi	LAN / Wi-Fi connection	Wi-Fi access points
QoS Server	Wi-Fi / LAN	Network QoS configuration
Modem LTE	LTE	Configure LTE backhaul

Functionality:

1. **Mesh Network**:
   - **The ESP32** or **Raspberry Pi** will act as a **primary router**, connecting to an **LTE backhaul**.
   - **The Mesh network** will include multiple **ESP32** (or Raspberry Pi) devices that connect to the main router and form a mesh network, ensuring wider coverage and robust connectivity.
2. **Backhaul LTE**:
   - **The LTE module (SIM7600)** will provide internet connectivity when the main connection is interrupted.
   - The LTE module will be connected to the ESP32 or Raspberry Pi main router via **UART** to transmit and receive data.
3. **Quality of Service (QoS)**:
   - **The QoS server** will be configured to **prioritize** certain types of traffic, such as voice (VoIP), video streaming, etc.
   - QoS will optimize bandwidth usage and ensure consistent performance for critical applications.
4. **Self-repairability**:
   - The primary router will monitor the status of the network and, in the event of an outage, will automatically switch to **LTE backhaul** to ensure connection continuity.

- The system will detect when a mesh node is no longer accessible and redirect traffic through the other nodes.

Examples of software used:

- **ESP-MESH** for mesh network configuration on ESP32.
- **OpenWrt** for Raspberry Pi (or ESP32) for QoS implementation.
- **LTE modem (SIM7600)** for 4G LTE mobile connection.
- **iPerf3** for performance testing and bandwidth optimization.

Additional features:

- **Traffic monitoring**: On the main router, a **traffic monitoring system** and **performance reports** will be implemented. These can be viewed on a local or web dashboard.
- **LTE Fallback**: In case the connection through the mainnet is lost, the system will automatically switch to **LTE backhaul** to ensure continuous connectivity.

9. Industrial Anti-Sabotage Sensor with Redundancy and SCADA Integration

Overview:

- Detects physical tamper through LDR and accelerations (MPU6050).
- It transmits simultaneously via LoRa and Modbus TCP.
- It allows direct integration into SCADA.

Motivation for choosing the project:

- Demonstrates industrial and redundant integration.
- It shows SCADA and radio protocols.

Motivation for choosing components:

- **MKR Zero**: SPI and I2C capabilities.
- **MPU6050 & LDR**: timp real tamper.
- **Wiznet W5500**: Ethernet & Modbus.
- **LoRa SX1278**: fallback radio.

Testing and troubleshooting:

- **Tamper**: Induces light and motion to test alerts.
- **LoRa**: using gateway and consoles.
- **Modbus**: test with Modscan or client scada.

Possible extensions:

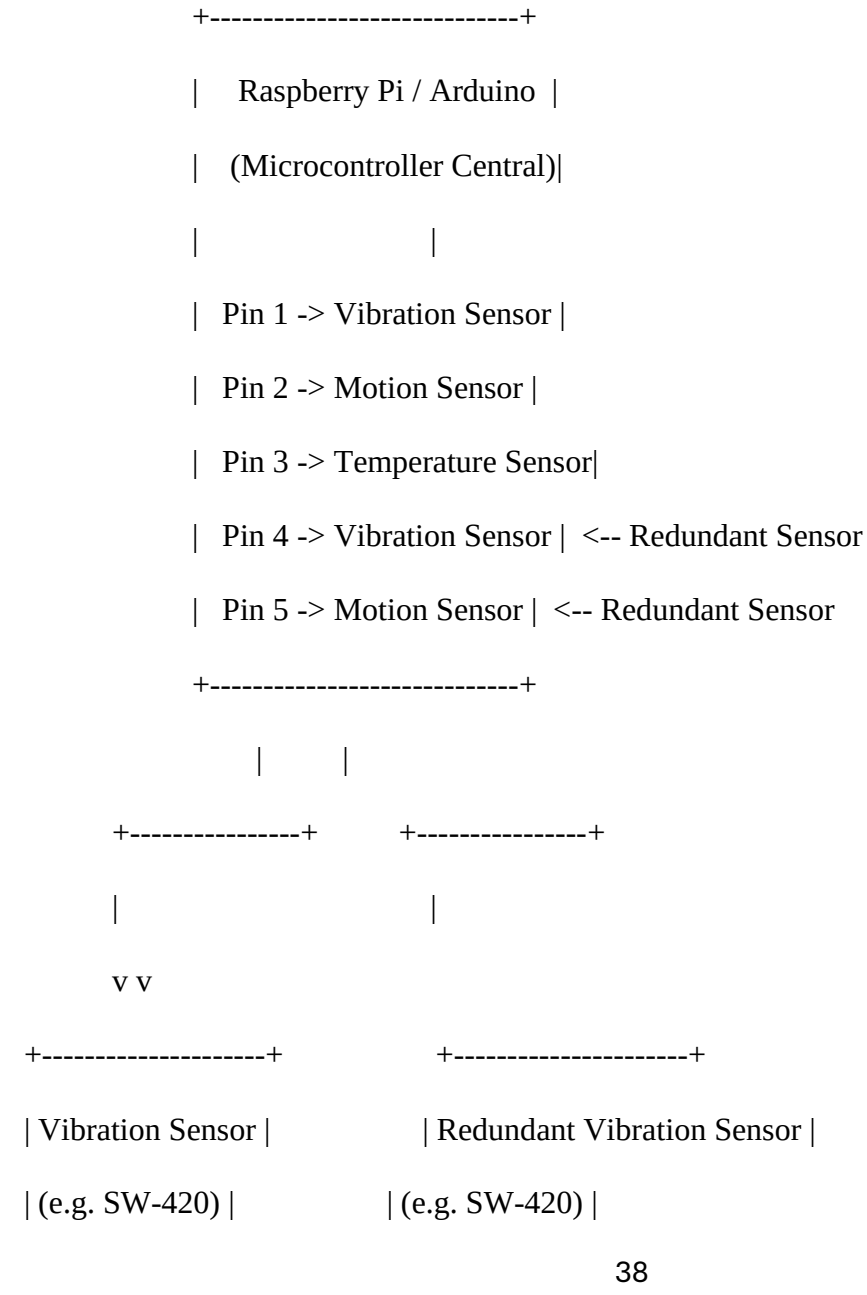
- Timestamp synchronization via NTP.
- Encryption on LoRa.
- Securing Modbus with TLS.

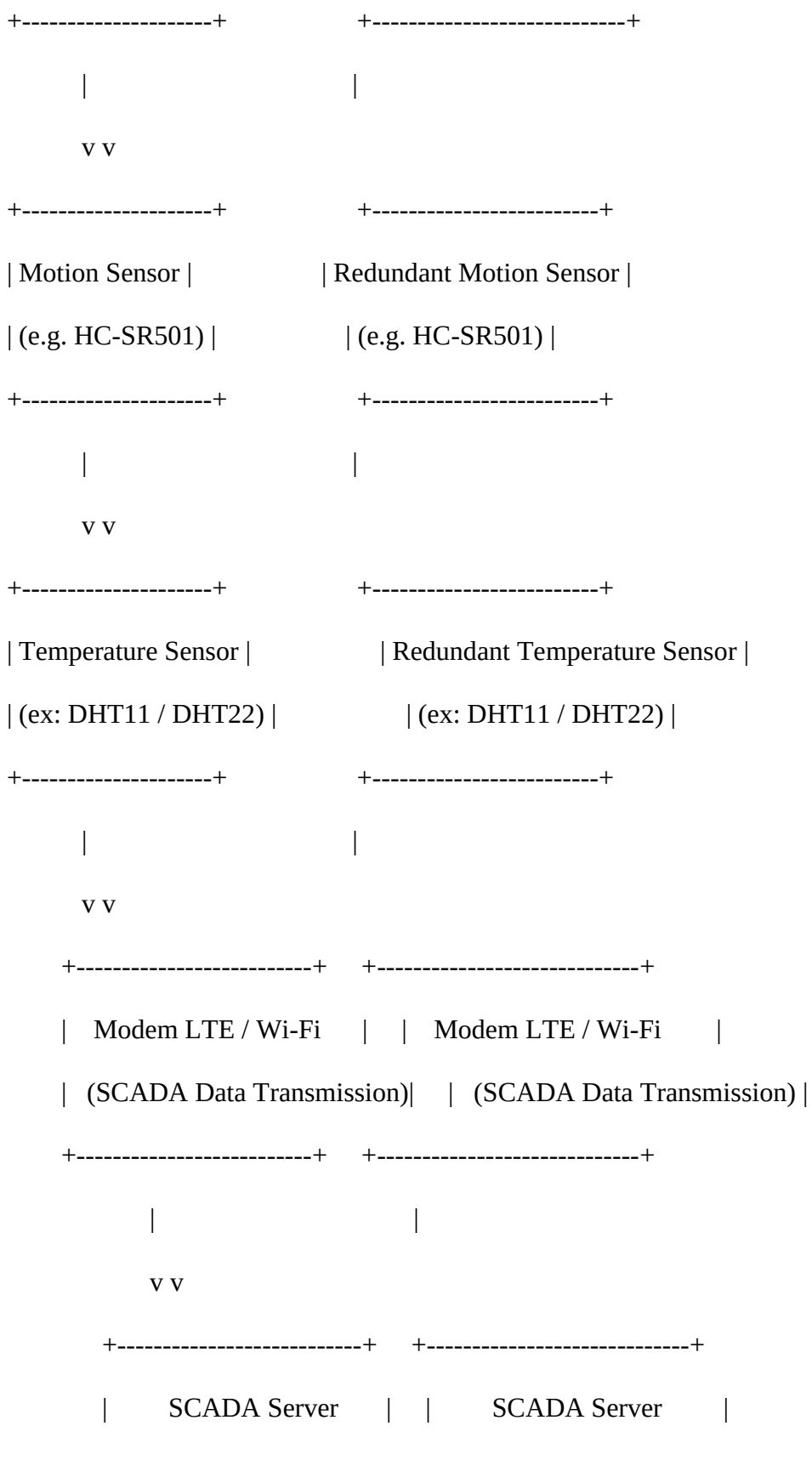
SOURCE CODE:

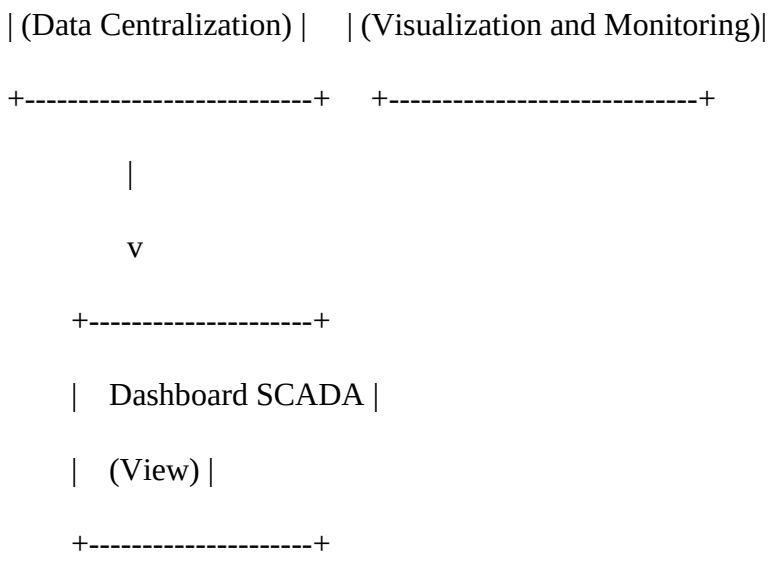
```
#include <SPI.h>
#include <Ethernet.h>
#include <ModbusTCP.h>
#include <LoRa.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

// Ethernet + Modbus
byte mac[] = {0xDE,0xAD,0xBE,0xEF,0xFE,0xED};
IPAddress ip(192,168,10,50);
EthernetServer server(502);
ModbusTCP modbus(&server);

// LoRa
#include <Wire.h>
Adafruit_MPU6050 mpu;

const int ldrPin = A0;

void setup() {
  Serial.begin(115200);
  Ethernet.begin(mac, ip);
  server.begin();
  LoRa.begin(433E6);
  if (!mpu.begin()) while(1);
}

void loop() {
  EthernetClient client = server.available();
  modbus.task(client);

  int ldr = analogRead(ldrPin);
  sensors_event_t accel;
  mpu.getEvent(&accel, nullptr, nullptr);

  if (ldr>600 || abs(accel.acceleration.x)>1.5) {
// send via LoRa
    LoRa.beginPacket();
    LoRa.print("E:");
```

```
    LoRa.print(ldr);
    LoRa.print(",");
    LoRa.print(accel.acceleration.x);
    LoRa.endPacket();
// update SCADA via Modbus
    modbus.Hreg(0, ldr);
    modbus.Hreg(1, (int)(accel.acceleration.x*100));
  }
}
```

SCHEMA:




| Motion Sensor |
| (e.g. HC-SR501) |
| Redundant Motion Sensor |
| (e.g. HC-SR501) |
| Temperature Sensor |
| (ex: DHT11 / DHT22) |
| Redundant Temperature Sensor |
| (ex: DHT11 / DHT22) |
| Modem LTE / Wi-Fi (SCADA Data Transmission) |
| Modem LTE / Wi-Fi (SCADA Data Transmission) |
| SCADA Server |
| SCADA Server |



Detailed Hardware Connections:

Component	Pin Component	Connected to Microcontroller
Vibration Sensor (SW-420)	VCC, GND, Signal	Pin 1 (VCC, GND, D2)
Redundant Vibration Sensor (SW-420)	VCC, GND, Signal	Pin 4 (VCC, GND, D3)
Motion Sensor (HC-SR501)	VCC, GND, Signal	Pin 2 (VCC, GND, D4)
Senzor Mișcare Redundant (HC-SR501)	VCC, GND, Signal	Pin 5 (VCC, GND, D5)
Temperature Sensor (DHT22)	VCC, GND, Data	Pin 3 (VCC, GND, D6)
Redundant Temperature Sensor (DHT22)	VCC, GND, Data	Pin 6 (VCC, GND, D7)
Modem LTE (SIM7600)	TX, RX, VCC, GND	Pin 7 (TX, RX, 5V, GND)

Functionality:

1. **Anti-tamper sensors**:
   - **Vibration sensors** will detect any suspicious movement near industrial equipment.
   - **Motion sensors** will detect any unauthorized movement in the protected area.
   - **Temperature sensors** will monitor abnormal temperatures that may indicate sabotage or equipment failure.
2. **Redundancy**:
   - The system is equipped with **redundant sensors** (e.g. two vibration sensors, two motion sensors and two temperature sensors) to ensure reliability and reduce the risk of detection error.
3. **Data transmission**:

- The data collected from the sensors is transmitted to an **LTE** or **Wi-Fi** modem, which sends it to a **centralized SCADA server**.
- **SCADA** will provide a centralized platform for data visualization and alarm management.
- **SCADA Server** will enable real-time control and monitoring of industrial equipment, and a visual **dashboard** will help operators make informed decisions.

4. **Integrate SCADA:**
   - The data is analyzed by the SCADA server, which can send alerts to operators if a saboteur or anomaly is detected.
   - **Data visualization in SCADA** will be done through a centralized **dashboard**, which will include charts and status alerts.

Examples of software used:

- **SCADA:** An open-source SCADA software such as **OpenPLC**, **Ignition SCADA** or **Node-RED** can be used for data integration and visualization.
- **LTE modem (SIM7600):** Using an LTE module for mobile connectivity and transmitting data to the SCADA server.
- **Arduino/ESP32**: Using a microcontroller to collect data from sensors and send it to the SCADA server via the LTE modem.

Additional features:

- **Real-time alerts**: If a sensor detects a sabotage, the SCADA server can send an alert via SMS, email, or in a mobile app to inform operators of the incident.
- **Data backup**: Data collected from sensors can be stored and recorded on a local server or in the cloud to ensure a backup in case of a connection drop.

10. Advanced Parental Controls with Captive Portal, RTC, and Web Monitoring

Overview:

- Captive portal router that blocks access after a certain time.
- Authenticate OAuth2 cu Keycloak.
- Web dashboard with graphs of usage and storage in Firebase.

Motivation for choosing the project:

- Illustrates captive portal and OAuth2.
- Demonstrates integration with the Firebase Cloud.

Motivation for choosing components:

- **ESP32**: server web rapid.
- **RTC DS3231**: precision ±2ppm.
- **WiFiManager & FirebaseESP8266**: configurare și backend.

Testing and troubleshooting:

- **Captive**: Go to the captive URL and confirm the redirect.
- **OAuth2**: Test the authentication flow with Keycloak.
- **Firebase**: Debug the /blocked value in the Firebase console.

Possible extensions:

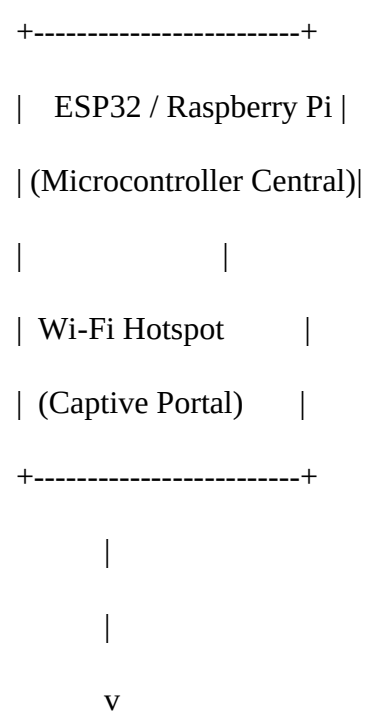
- Authenticate multi-tenant.
- Advanced graphics with D3.js.
- Webhook notifications at limits reached.

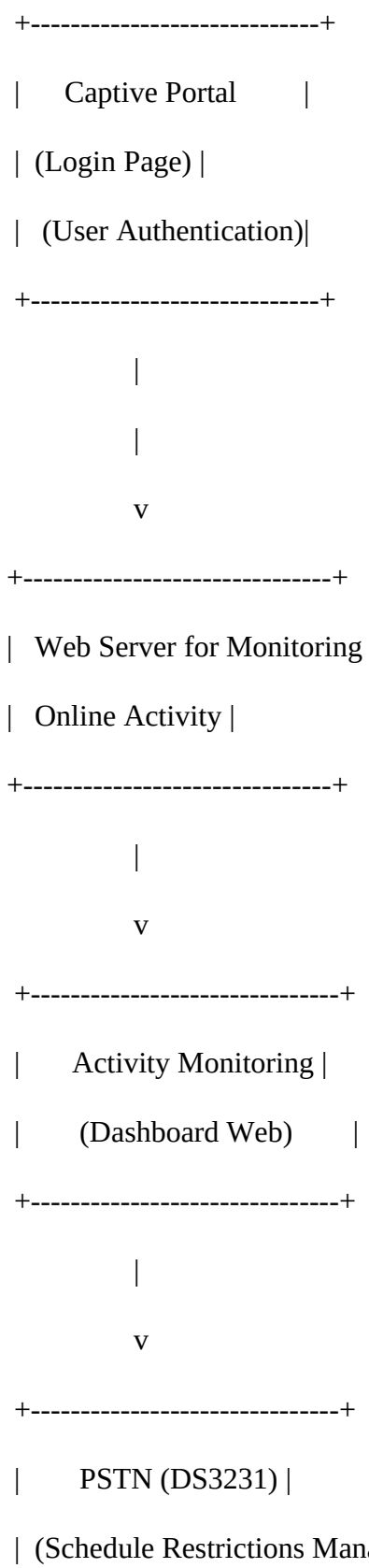
SOURCE CODE:

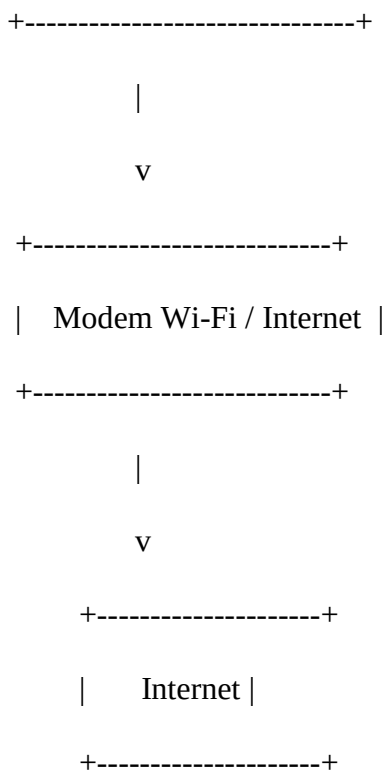
```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <DNSServer.h>
#include <Wire.h>
#include <RTClib.h>
#include <FirebaseESP8266.h>
#include <WiFiManager.h>

// RTC
RTC_DS3231 rtc;

// Captive
ESP8266WebServer server(80);
DNSServer dns;

// Firebase
#define FIREBASE_HOST "your-project.firebaseio.com"
#define FIREBASE_AUTH "YOUR_AUTH_TOKEN"

void handleRoot() {
  DateTime now = rtc.now();
 html string = "<h1>Parental Control</h1>";
 html += "<p>Not curent: " + now.timestamp() + "</p>";
  server.send(200, "text/html", html);
}
```

```
void setup() {
  Serial.begin(115200);
  Wire.begin(D2, D1);
  rtc.begin();

WiFiManager wm;
  wm.autoConnect("ParentalAP","12345678");

  dns.start(53, "*", WiFi.softAPIP());
  server.on("/", handleRoot);
  server.begin();

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

void loop() {
  DNS.processNextRequest();
  server.handleClient();

  DateTime now = rtc.now();
  bool blocked = (now.hour() >= 20 || now.hour() < 7);
  Firebase.setBool("/blocked", blocked);
}
```

DESIGN:




Captive Portal
(Login Page)
(User Authentication)
Web Server for Monitoring
Online Activity
Activity Monitoring
(Dashboard Web)
PSTN (DS3231)
(Schedule Restrictions Management)



Detailed Hardware Connections:

Component	Pin Component	Connected to Microcontroller
Module RTC (DS3231)	SDA, SCL, VCC, GND	Pin 21 (SDA), Pin 22 (SCL)
Wi-Fi Module (ESP32)	VCC, GND, RX, TX	Wi-Fi connection
Server Web	IP (Local Network)	Wi-Fi connection
Captive Portal	VCC, GND, Data	Built on ESP32 via Wi-Fi

Functionality:

1. **Captive Portal**:
   - When a user tries to connect to the Wi-Fi network, they will be redirected to a captivated portal (login page).
   - Users will need to enter their login information to access the internet.
   - The pages may also include rules for the use of the internet or terms and conditions for access.
2. **RTC module**:
   - **Real-Time Clock (DS3231)** will be used to manage access hours and apply time-based internet restrictions.
   - For example, parents can set restrictions to limit internet access during certain hours or days of the week.
3. **Web Monitoring**:
   - All users' online activity will be monitored and recorded on a **web server**.

    - A **web dashboard** will allow parents or administrators to see the history of online activity, which sites have been accessed, and how many minutes have been spent on each site.
    - The web server can also apply **content filters**, blocking access to inappropriate websites.
4. **Connectivity**:
    - **The ESP32** will handle both the Wi-Fi connection and the restriction enforcement and monitoring logic.
    - It will connect to the internet through a **Wi-Fi modem** or an existing internet connection to transmit the data to the web server.

Examples of software used:

- **Captive Portal**: The Captive portal can be created using **ESP32** and **libraries** such as **ESPAsyncWebServer**.
- **RTC (DS3231):** Using a **DS3231 PSTN** to manage current time and time-based restrictions.
- **Web server**: Using a **local server** (e.g., using **Node.js** or **PHP**) for monitoring web activity and filtering content.
- **Web Dashboard**: Create a **dashboard** to visualize user activity in real time.

Additional features:

- **Activity Alerts**: Parents can receive email or SMS alerts if an unapproved website is accessed or too much time is spent on the internet.
- **Flexible Time Settings**: Parents can set restriction hours or personalized schedule for each user.
- **Content filtering**: Users can only have access to certain pre-approved sites, or content filtering can be based on categories (e.g., educational, gaming, social media).

Examples of functions:

- **Captive Portal**: Custom login page for user login.
- **RTC**: Shutdown of internet access after 22:00.
- **Activity monitoring**: Displaying the history of visited sites and the duration of access for each site.