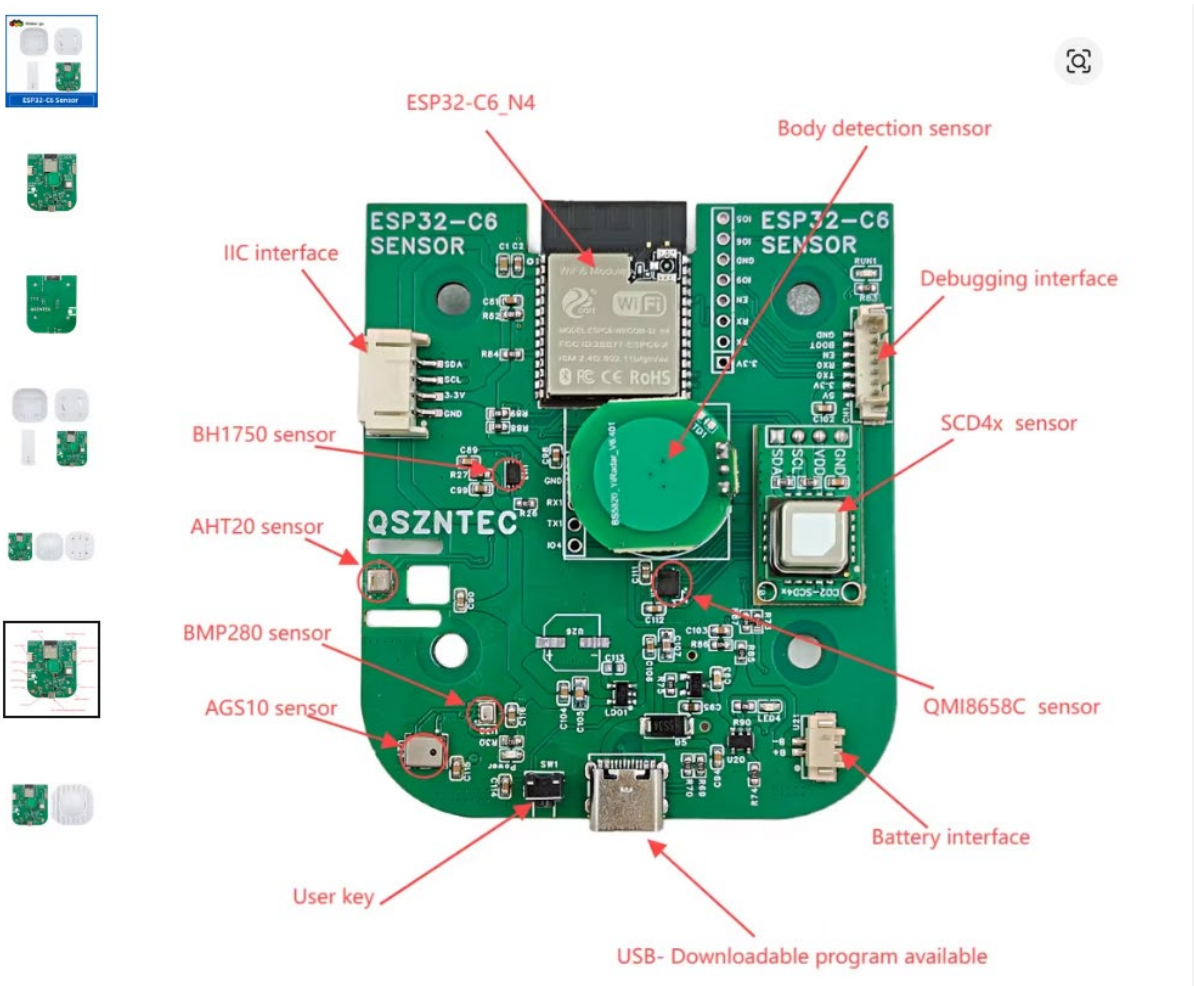
Nell’ottica di uno spirito Collaborativo e propositivo Sistema CO2 AppSensor.

1. Introduzione

Il sistema si compone di un modulo ESP32-C6_Msensor dotato del sensore CO_2 Sensirion SCD40, di uno sketch Arduino che configura il modulo in modalità Access Point e trasmette le misure di CO_2 via UDP, e di un’app Android che si connette all’AP per ricevere e visualizzare i valori misurati. L’obiettivo è monitorare in tempo reale la concentrazione di CO_2 in ambienti chiusi al fine di valutare la qualità dell’aria e prevenire rischi per la salute umana.

2. Sensore CO_2 SCD40

Il sensore SCD40 utilizza tecnologia fotoacustica per misurare la CO_2 in un intervallo da 0 a 40 000 ppm, con accuratezza di ±(50 ppm + 5 % della lettura) nel range 400–2 000 ppm e risoluzione di 1 ppm. La comunicazione con il microcontrollore avviene tramite interfaccia I²C all’indirizzo 0x62.



3. Tabelle di pericolosità della CO_2 per l'uomo

Concentrazione CO_2 (ppm)	Effetti sulla salute umana
≤ 400	Concentrazioni atmosferiche esterne normali (no effetti)
400 – 1 000	Tipiche in ambienti interni ben ventilati (nessun effetto avverso)
1 000 – 2 000	Sonnolenza, lieve riduzione della concentrazione e senso di fastidio
2 000 – 5 000	Mal di testa, stanchezza, aumento della frequenza cardiaca, nausea
5 000	Limite di esposizione lavorativa (8 h TWA) secondo OSHA e NIOSH
15 000	Stimolazione respiratoria lieve (alcuni individui)
30 000	Stimolazione respiratoria moderata, aumento della pressione arteriosa
40 000	IDLH (Immediately Dangerous to Life or Health) secondo NIOSH
≥ 70 000 – 100 000	Suffocazione, perdita di coscienza e rischio di morte in pochi minuti

4. Sketch Arduino

```
#include <Arduino.h>
#include <Wire.h>
#include <SensirionI2cScd4x.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include <ArduinoJson.h>

// Macro per errori Sensirion
#ifdef NO_ERROR
#undef NO_ERROR
#endif
#define NO_ERROR 0

// I2C pins per ESP32-C6 MSensor board
#define I2C_SDA_PIN 6
#define I2C_SCL_PIN 7

// WiFi AP credentials
const char* ap_ssid     = "LAB312-CO2-METER";
const char* ap_password = "lab312pass123";

// UDP settings
```

```
const unsigned int udpPort = 32123;
WiFiUDP udp;
IPAddress broadcastIP;

// Measurement interval (ms)
const unsigned long MEAS_INTERVAL = 10000UL;

// Sensirion SCD41 sensor instance
SensirionI2cScd4x sensor;
static char errorMessage[64];
static int16_t errorCode;

// Measurement variables
uint16_t co2_ppm     = 0;
float    temperature = 0.0f;
float    humidity    = 0.0f;

// Prototipi
void initializeSensorSingleShot();
bool performSingleShotMeasurement(uint16_t &co2, float &temp, float &hum);
void sendSensorData(uint16_t co2, float temp, float hum);

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);
  Serial.println("=== ESP32-C6 MSensor + SCD41 UDP Broadcast ===");

  // Inizializza I2C
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);

  // Inizializza sensore
  sensor.begin(Wire, SCD41_I2C_ADDR_62);

  // Imposta target ASC a 400 ppm (livello baseline CO2)
  errorCode = sensor.setAutomaticSelfCalibrationTarget(400);
  if (errorCode != NO_ERROR) {
    Serial.println("Errore impostazione target ASC");
    }

  initializeSensorSingleShot();

  // Avvia AP
  WiFi.softAP(ap_ssid, ap_password);
  IPAddress apIP   = WiFi.softAPIP();
  IPAddress subnet = WiFi.softAPSubnetMask();
  Serial.printf("AP '%s' started. IP: %s\n", ap_ssid, apIP.toString().c_str());

  // Calcola broadcast
  for (int i = 0; i < 4; i++) {
    broadcastIP[i] = apIP[i] | (~subnet[i]);
```

```
  }
  Serial.printf("Broadcast IP: %s\n", broadcastIP.toString().c_str());

  // Inizializza UDP
  udp.begin(udpPort);
  Serial.printf("UDP on port %u\n", udpPort);

  // Avvia prima misura (wakeUp() gestito internamente)
  sensor.wakeUp();
  sensor.measureSingleShot();
}

void loop() {
  static unsigned long lastMillis = 0;
  unsigned long now = millis();

  if (now - lastMillis >= MEAS_INTERVAL) {
    lastMillis = now;

    // Esegui misura
    uint16_t tempCO2;
    float    tempT;
    float    tempH;
    bool ok = performSingleShotMeasurement(tempCO2, tempT, tempH);
    if (ok) {
      co2_ppm     = tempCO2;
      temperature = tempT;
      humidity    = tempH;
      Serial.printf("[Misura OK] CO2: %u ppm, T: %.1f C, RH: %.1f %%\n",
                    co2_ppm, temperature, humidity);
      sendSensorData(co2_ppm, temperature, humidity);
    } else {
      Serial.println("[Misura FALLITA] invio saltato");
    }

    // Richiedi prossima misura
    sensor.measureSingleShot();
  }

  // Gestione comandi UDP in ingresso (opzionale)
  int packetSize = udp.parsePacket();
  if (packetSize) {
    char cmd[256];
    int len = udp.read(cmd, sizeof(cmd) - 1);
    if (len > 0) {
      cmd[len] = '\0';
      Serial.printf("Received UDP: %s\n", cmd);
      // Qui parse JSON per comandi
    }
  }
```

```
}

// Inizializzazione sensore per single-shot
void initializeSensorSingleShot() {
  delay(30);
  errorCode = sensor.wakeUp();
  if (errorCode != NO_ERROR) {
    errorToString(errorCode, errorMessage, sizeof(errorMessage));
    Serial.printf("Errore wakeUp(): %s\n", errorMessage);
  }
  errorCode = sensor.stopPeriodicMeasurement();
  if (errorCode != NO_ERROR) {
    errorToString(errorCode, errorMessage, sizeof(errorMessage));
    Serial.printf("Errore stopPeriodicMeasurement(): %s\n", errorMessage);
  }
  errorCode = sensor.reinit();
  if (errorCode != NO_ERROR) {
    errorToString(errorCode, errorMessage, sizeof(errorMessage));
    Serial.printf("Errore reinit(): %s\n", errorMessage);
  }
  uint64_t serial;
  errorCode = sensor.getSerialNumber(serial);
  if (errorCode != NO_ERROR) {
    errorToString(errorCode, errorMessage, sizeof(errorMessage));
    Serial.printf("Errore getSerialNumber(): %s\n", errorMessage);
  } else {
    Serial.printf("SCD41 SN: 0x%llX\n", serial);
  }
}

// Esegui misura single-shot completa
bool performSingleShotMeasurement(uint16_t &co2, float &temp, float &hum) {
  errorCode = sensor.wakeUp();
  if (errorCode != NO_ERROR) return false;
  errorCode = sensor.measureSingleShot();
  if (errorCode != NO_ERROR) return false;
  errorCode = sensor.measureAndReadSingleShot(co2, temp, hum);
  if (errorCode != NO_ERROR) return false;
  return true;
}

// Invia dati via UDP broadcast
void sendSensorData(uint16_t co2, float temp, float hum) {
  StaticJsonDocument<192> doc;
  doc["co2_ppm"]     = co2;
  doc["temperature"] = temp;
  doc["humidity"]    = hum;
  char buf[256];
  size_t len = serializeJson(doc, buf, sizeof(buf));
  udp.beginPacket(broadcastIP, udpPort);
```

```
  udp.write((uint8_t*)buf, len);
  udp.endPacket();
  Serial.printf("UDP -> %s\n", buf);
}
```

5. App Android Realizzata con l'ambiente free Basic4Android

5.1 Modulo Main Activity

```
#Region  Project Attributes
	#ApplicationLabel: AppSensor
	#VersionCode: 1
	#VersionName:
	'SupportedOrientations possible values: unspecified, landscape or portrait.
	#SupportedOrientations: unspecified
	#CanInstallToExternalStorage: False
#End Region

#Region  Activity Attributes
	#FullScreen:  False
	#IncludeTitle: True
#End Region

Sub Process_Globals
	Private rp As RuntimePermissions
	Public co2ppm As Int=0
	Public temperature As Float
	Public humidity As Float
End Sub

Sub Globals
	Private Gauge As SDGaugeView

	Private lblTemperature As Label
	Private lblHumidity As Label
	Private imgLogOut As ImageView
	Private lblAccuracy As Label
End Sub

Sub Activity_Create(FirstTime As Boolean)
	Activity.LoadLayout("Layout")
	Activity.Title="AppSensor"
	Gauge.ValueMin=0
	Gauge.ValueMax=5000
	Gauge.MinimumFractions=0
	Gauge.Value=0
	Dim Permissions As List
```

```
	Permissions =
Array(rp.PERMISSION_ACCESS_FINE_LOCATION,"android.permission.BLUETOOTH_ADV
ERTISE",rp.PERMISSION_POST_NOTIFICATIONS)
	For Each PER As String In Permissions
		rp.CheckAndRequest(PER)
		Wait For Activity_PermissionResult (PERMISSION As String, Result As Boolean)
		If Result = False Then
			ToastMessageShow("No permission: " & PERMISSION, True)
			Activity.Finish
		End If
	Next
	imgLogOut.Bitmap=TextToBitmap(Chr(0xF090), 28)
	StartService(ForegroundSensor)
End Sub


Public Sub ShowSensorValue

	Gauge.Value=co2ppm
	lblTemperature.Text="Temperatura: " & Round2(temperature,1) & "C°"
	lblHumidity.Text="Umidità: " & Round2(humidity,1) & "%"
	lblAccuracy.Text="+/-" & Round2((50+(0.05*co2ppm)),0)
End Sub

Sub Activity_Resume
	ShowSensorValue
End Sub

Sub Activity_Pause (UserClosed As Boolean)
  If UserClosed =True Then
		StopService(ForegroundSensor)
	End If
End Sub

Sub TextToBitmap (s As String, FontSize As Float) As Bitmap
	Dim bmp As Bitmap
	bmp.InitializeMutable(32dip, 32dip)
	Dim cvs As Canvas
	cvs.Initialize2(bmp)
	Dim h As Double = cvs.MeasureStringHeight(s, Typeface.FONTAWESOME, FontSize)
	cvs.DrawText(s, bmp.Width / 2, bmp.Height / 2 + h / 2, Typeface.FONTAWESOME,
FontSize, Colors.Black, "CENTER")
	Return bmp
End Sub

Private Sub imgLogOut_Click
	StopService(ForegroundSensor)
	Activity.Finish

End Sub
```

5.2 Service Starter

```
#Region  Service Attributes
	#StartAtBoot: False
	#ExcludeFromLibrary: True
#End Region

Sub Process_Globals
	'These global variables will be declared once when the application starts.
	'These variables can be accessed from all modules.

End Sub

Sub Service_Create
	'This is the program entry point.
	'This is a good place to load resources that are not specific to a single activity.

End Sub

Sub Service_Start (StartingIntent As Intent)
	Service.StopAutomaticForeground 'Starter service can start in the foreground state in some
edge cases.
End Sub

Sub Service_TaskRemoved
	'This event will be raised when the user removes the app from the recent apps list.
End Sub

'Return true to allow the OS default exceptions handler to handle the uncaught exception.
Sub Application_Error (Error As Exception, StackTrace As String) As Boolean
	Return True
End Sub

Sub Service_Destroy

End Sub
```

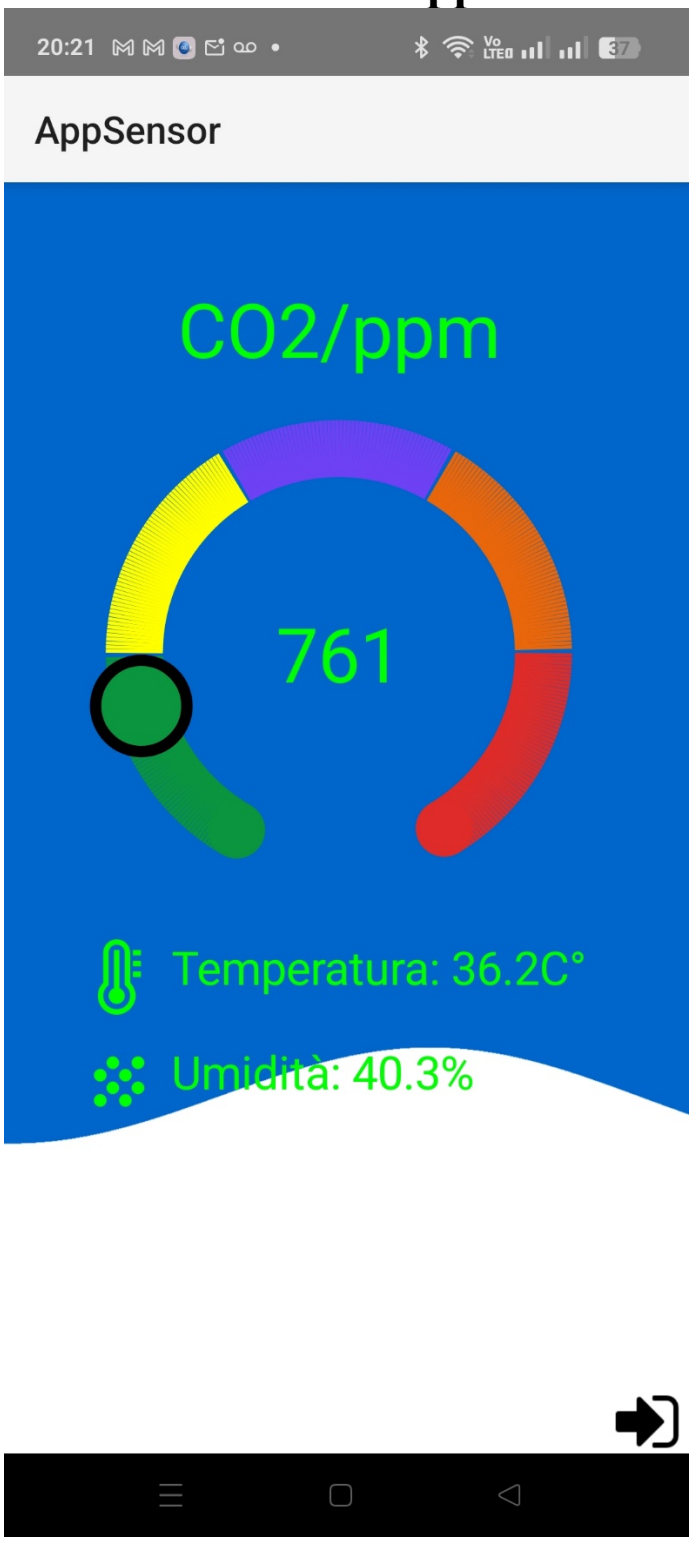
5.3 Service ForegroundSensor

```
#Region  Service Attributes
	#StartAtBoot: False

#End Region


Sub Process_Globals
	Private udpSock As UDPSocket
	Private udpPort As Int = 32123    ' porta su cui ascoltare i broadcast
```

```
End Sub

Sub Service_Create

End Sub

Sub Service_Start (StartingIntent As Intent)
    Dim n As Notification
    n.Initialize
    n.Icon = "icon"
    n.SetInfo("Lettura Sensori Attiva","Lettura in corso...", Main)
    Service.StartForeground(1, n)
    StartListening

End Sub

Sub Service_Destroy
        udpSock.Close
        Service.StopForeground(1)
End Sub


Sub StartListening
        Try
                udpSock.Initialize("UDP",udpPort,255)
        Catch
                Log(LastException.Message)
        End Try
End Sub


Sub UDP_PacketArrived (Packet As UDPPacket)
        Dim msg As String
        msg = BytesToString(Packet.Data, Packet.Offset, Packet.Length,"UTF8")
        Try
      Dim jp As JSONParser
      jp.Initialize(msg)
      Dim root As Map = jp.NextObject
      Main.co2ppm = root.Get("co2_ppm")
      Main.temperature = root.Get("temperature")
                Main.humidity = root.Get("humidity")
                CallSub(Main,"ShowSensorValue")
                'ShowNotification
   Catch
      Log(LastException.Message)
   End Try
End Sub

Sub ShowNotification
        Dim n As Notification
        n.Initialize2(n.IMPORTANCE_HIGH)
```

```
        n.Icon = "icon"   ' icona drawable
        Dim date_string As String
        DateTime.DateFormat = "dd/MM/yyyy"
        DateTime.TimeFormat = "HH:mm:ss"
        date_string=DateTime.Date(DateTime.Now) & " " & DateTime.Time(DateTime.Now)
        n.SetInfo("Dati Rilevati"  &  CRLF & date_string,"CO2: " & Main.co2ppm & "ppm" &
CRLF & "Temperatura: " & Round2(Main.temperature,1) & "C°   "  & "Umidità: " &
Round2(Main.humidity,1) & "%", Main)
        n.Notify(2)
End Sub
```

6. Screenshoot dell’app android



Il Costo del sistema è relativo al singolo modulo ESP32 C6 Msensor che si aggira intorno ai 55 euro iva compresa e senza costi di spedizione acquistabile tramite lo store Aliexpress.