

Università Politecnica delle Marche

Corso di Laurea Magistrale

Ingegneria Informatica e dell'Automazione



Sistemi Operativi Dedicati

Studenti

Accattoli Eleonora - S1106361
Brugiavini Alessio - S1106565

Anno Accademico 2022/2023

Indice

1	Introduzione	3
2	Strumenti e software utilizzati	4
3	Sviluppo Progetto	5
3.1	Collegamento dispositivi	5
3.2	Configurazione Raspberry	6
3.3	FreeRTOS	8
3.4	ESP32, MQTT Broker e MQTT Client	11
3.5	Sensori	13
3.5.1	BMP280	13
3.5.2	BH1750	14
3.5.3	RTC	15
3.6	Bot Telegram	16
3.7	Creazione Database e caricamento dati	18
3.8	Grafici	25
4	Considerazioni finali	29

1 Introduzione

L'obiettivo del progetto consiste nel creare un sistema in grado di rilevare e comunicare in tempo reale i dati rilevati di temperatura, pressione, umidità e luminosità dai sensori nell'ambiente. Il sistema sarà composto da:

- **Raspberry Pi 4B**
- **ESP32**: microcontrollore utilizzato per la progettazione di dispositivi connessi e di sistemi IoT.
- Sensore di temperatura e pressione **BMP280**
- Sensore di luminosità **BH1750**
- Modulo RTC **PCF8523**
- **Broker e Client MQTT**
- **Bot Telegram**

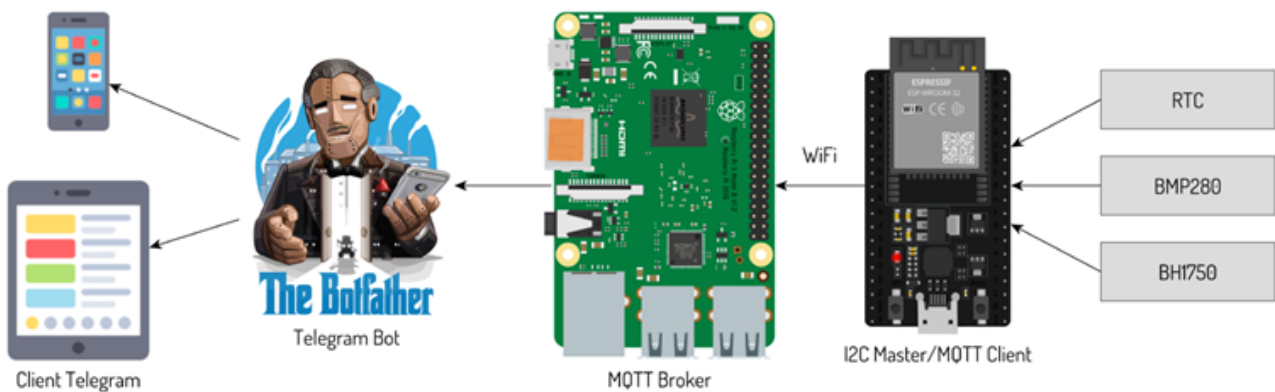


Figura 1: Struttura progetto

I due sensori ed il modulo RTC vengono collegati fisicamente all'ESP32, il quale si occuperà di trasmettere i dati rilevati dai sensori al Raspberry Pi, mediante WiFi. L'ESP32 è configurato come MQTT Client con l'utilizzo del kernel FreeRTOS, il quale permette il coordinamento dei vari task. Il Raspberry Pi sarà configurato come server IoT, il quale svolgerà le funzioni di MQTT Broker, storage dati ed invio alert.

2 Strumenti e software utilizzati

Per lo sviluppo del progetto sono stati utilizzati i seguenti strumenti:

- **Raspberry Pi 4B:** computer a scheda singola basato sul sistema operativo Linux che, opportunamente configurato, permette l'utilizzo in vari campi dell'automazione. Si basa su un system-on-a-chip, ovvero è formato da un circuito integrato che in un solo chip contiene un intero sistema e utilizza una micro SD per il boot e per la memoria ROM non volatile. All'interno del Raspberry Pi è possibile installare, tramite apposito software, un qualsiasi sistema operativo.
- **BMP280:** sensore di temperatura, umidità e pressione.
- **BH1750:** sensore di luminosità.
- **Modulo RTC PCF8523:** è un real-time clock, ovvero un dispositivo con funzione di orologio.

Mentre i software utilizzati sono i seguenti:

- **Visual Studio Code:** editor di codice sorgente sviluppato da Microsoft per Windows, Linux e Mac OS. Permette, tramite apposita estensione, la connessione SSH.
- **FreeRTOS:** sistema operativo open source in tempo reale per microcontrollori e microprocessori. Verrà utilizzato nell'ESP32.
- **Mosquitto:** viene utilizzato come broker MQTT open-source all'interno del Raspberry Pi. Il broker MQTT è responsabile della ricezione e del filtraggio dei messaggi, dell'identificazione dei client iscritti e l'invio dei messaggi ad essi.
- **Raspberry Pi OS:** è il sistema operativo installato sul Raspberry Pi.

3 Sviluppo Progetto

3.1 Collegamento dispositivi

I dispositivi per poter funzionare e lavorare tra loro necessitano di un cablaggio su una bread-board.

Sono stati collegati i seguenti componenti:

- **BMP280:** sensore di temperatura e pressione.
- **BH1750:** sensore di luminosità.
- **Modulo RTC PCF8523:** è un real-time clock, ovvero un dispositivo con funzione di orologio.
- **ESP32:** microcontrollore nel quale caricare il codice per gestire i vari task.

Il cablaggio è stato effettuato considerando lo schema seguente:

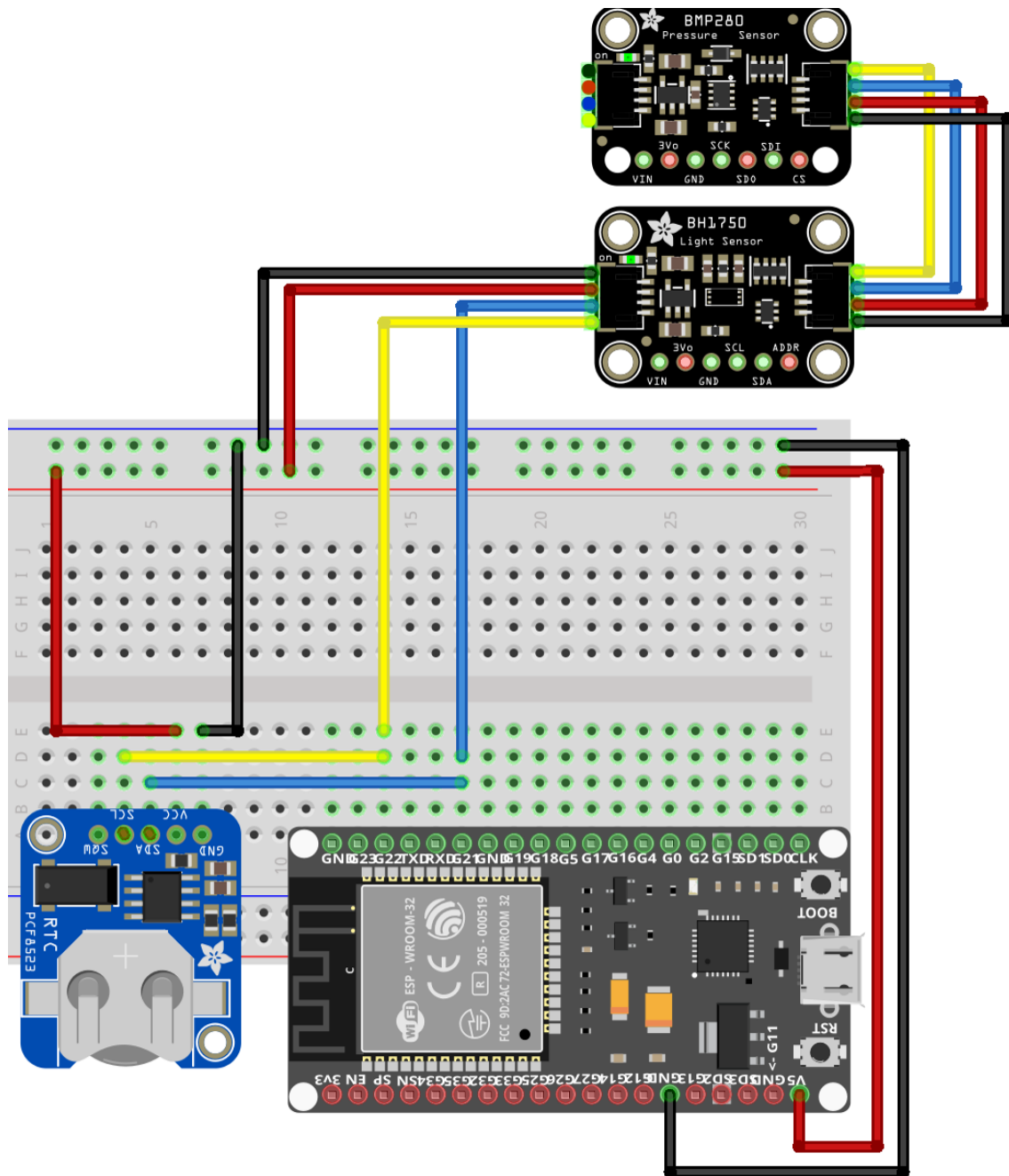


Figura 2: Schema di cablaggio del progetto

In seguito l'ESP32 è stato collegato al PC utilizzando un cavo Micro USB.

3.2 Configurazione Raspberry

Per poter utilizzare il Raspberry è stato necessario installare all'interno della sua scheda SD il sistema operativo dedicato, attraverso il software Raspberry Pi Imager. Come si può vedere dall'immagine sottostante, dalla schermata iniziale, si sceglie il modello del Raspberry, il sistema operativo da installare e la periferica su cui farlo; nel nostro caso, è stato scelto il modello Raspberry Pi 4B, il Raspberry Pi OS e la scheda SD.



Figura 3: Raspberry Pi Imager

Dopodiché viene inserita la scheda SD sul Raspberry, vengono collegati i dispositivi necessari, quali display, tastiera, mouse e cavo Ethernet e vengono impostate le credenziali (nome utente e password).

Per permettere una comunicazione client-server sicura tra dispositivi, viene utilizzato il protocollo *SSH* o Secure Shell.

PuTTY è un client SSH gratuito portabile che permette la gestione dei dispositivi da remoto.

PuTTY viene installato nella macchina su cui si intende lavorare da remoto (nel nostro caso una macchina Windows), specificando sotto il campo “*Host Name (or IP address)*” il nome del nostro Raspberry e si seleziona come “*Connection type*” l’opzione “*SSH*”.

Dopo aver cliccato “*Open*”, verranno inserite le credenziali di Raspberry sulla finestra visualizzata.

In questo modo è stata instaurata una connessione SSH tra il PC ed il dispositivo.

Per effettuare la programmazione del Raspberry direttamente dal PC, è necessario installare l’estensione *REMOTE-SSH* su Visual Studio Code, aggiungere una nuova macchina ed inserire i dati della connessione instaurata:

- **nome utente:** *nome Raspberry Pi*
- **host:** *nome host — vedere su impostazioni*
- **nome utente:** nel nostro caso *brugia*

Dopodiché come conferma, si dovrà inserire la password impostata nel Raspberry.

Essendo il Raspberry connesso alla rete ha un proprio indirizzo IP: **192.168.1.7**

Il Raspberry Pi dovrà svolgere la funzione di server IoT, ovvero si occuperà di svolgere la funzione di MQTT Broker, storage dati ed invio alert.

L'**MQTT** è un protocollo di messaggistica standard per l'IoT, che consente lo scambio di messaggi tra dispositivi, attraverso un broker di messaggistica.

Il broker MQTT è il sistema backend, responsabile della ricezione e del filtraggio dei messaggi, della decisione di chi è interessato a loro e della pubblicazione del messaggio a tutti i client iscritti.

Il client MQTT è un qualsiasi dispositivo che utilizza il servizio MQTT.

Tra i vari broker disponibili, viene usato **Mosquitto**, che è il più comune, il quale è stato installato sul Raspberry tramite VS Code, con i seguenti passaggi:

1. Viene aggiornato il sistema:

```
sudo apt update && sudo apt upgrade
```

2. Si installa Mosquitto :

```
sudo apt install -y mosquitto mosquitto-clients
```

3. Si lancia il comando per avviare Mosquitto automaticamente all'avvio:

```
sudo systemctl enable mosquitto.service
```

4. Si testa l'installazione:

```
mosquitto -v
```

Viene poi abilitato l'accesso remoto senza autenticazione:

1. Si apre il file *mosquitto.conf*:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

2. Si aggiungono alla fine del file le seguenti istruzioni per configurare l'accesso non autenticato:

```
listener 1883
```

```
allow_anonymous true
```

3. Si salva il file e si riavvia Mosquitto.

3.3 FreeRTOS

FreeRTOS è un sistema operativo real-time in grado di gestire più attività in parallelo e di rispettare i tempi di esecuzione dei processi.

Per fare ciò vengono utilizzati i *task*, ovvero delle funzioni contenenti una serie di istruzioni da eseguire in loop.

Per lo svolgimento del progetto, sono stati creati ed implementati i seguenti task:

- **readBMP280Task**: viene usato per leggere i dati del sensore BMP280, quali temperatura, pressione e umidità.
- **readBH1750Task**: viene usato per leggere i dati del sensore di luminosità BH1750.

- **readRTCTask:** viene usato per leggere i dati del sensore RTC, quali data e ora.

Per creare i task vengono utilizzate le seguenti funzioni definite all'interno del *setup*:

```
xTaskCreatePinnedToCore(readBMP280Task, "BMP280Task", 4096, NULL, 1, &bmpTaskHandle, 0);
```

```
xTaskCreatePinnedToCore(readBH1750Task, "BH1750Task", 4096, NULL, 1, &bh1750TaskHandle, 0);
```

```
xTaskCreatePinnedToCore(readRTCTask, "RTCTask", 4096, NULL, 1, &rtcTaskHandle, 0);
```

che prendono come parametri:

- Puntatore alla funzione – il nome del task
- Descrizione del task
- Dimensione stack
- Parametri stack
- Priorità
- Possibilità di accedere al task dall'esterno

Dopodiché vengono definiti i task creati, inserendo le istruzioni che dovranno essere svolte in loop.

Le istruzioni definite al loro interno, come per la lettura dei valori rilevati dai sensori, per la pubblicazione dei dati su MQTT e per il controllo delle soglie, verranno descritte in dettaglio successivamente.

All'interno del setup, prima della creazione dei task, come da codice sottostante, viene effettuata la connessione al WiFi, richiamando l'apposita funzione *connectToWiFi()* e vengono effettuati dei controlli iniziali per verificare la rilevazione e l'effettivo funzionamento e collegamento dei dispositivi utilizzati, compresa la sincronizzazione della data e dell'ora.

Nel caso in cui non venga rilevato il sensore, viene visualizzato in output un messaggio di errore.

```
void setup() {  
  Serial.begin(115200);  
  connectToWiFi();  
  client.setServer(mqttBroker, mqttPort);  
  
  if (!bmp.begin()) {  
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");  
    while (1);  
  }  
  
  if (!lightSensor.begin()) {  
    Serial.println("Could not find a valid BH1750 sensor, check wiring!");  
    while (1);  
  }  
  
  if (!rtc.begin()) {  
    Serial.println("Could not find a valid RTC sensor, check wiring!");  
    while (1);  
  }  
  
  if (rtc.lostPower()) {  
    Serial.println("RTC lost power, let's set the time!");  
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));  
  }  
}
```

Figura 4: Codice controllo dispositivi

Ogni sensore viene però prima inizializzato richiamando l'apposita libreria, come si può vedere nella *Figura 5*.

```
// Inizializzazione dei sensori  
Adafruit_BMP280 bmp;  
BH1750 lightSensor;  
RTC_PCF8523 rtc;
```

Figura 5: Inizializzazione Sensori

Nel codice sottostante, si può vedere in dettaglio come avviene la connessione al WiFi tramite *connectToWiFi()*. La connessione alla rete si effettua attraverso la funzione *Wifi.begin(ssid, password)*, alla quale viene passato l'SSID e la password del WiFi.

```
void loop() {
    if (!client.connected()) {
        connectToMQTT();
    }
    client.loop();
}

void connectToWiFi() {
    Serial.println("Connessione al WiFi..");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connettendosi al WiFi...");
    }
    Serial.println("Connesso al WiFi");
}
```

Figura 6: Connessione WiFi

Al fine di far comunicare i vari dispositivi, vengono utilizzate due librerie:

1. **WiFi:** per consentire la connessione alla rete e permette di definire server, client e inviare e ricevere messaggi.
2. **PubSubClient:** per scambiare messaggi MQTT.

Attraverso *WiFiClient* viene definito un nuovo client *espClient* che può connettersi alla rete e scambiare messaggi MQTT: *PubSubClient client(espClient)*.

```
// Dichiarazioni delle costanti e delle variabili globali

TaskHandle_t bmpTaskHandle, bh1750TaskHandle, rtcTaskHandle;
WiFiClient espClient;
PubSubClient client(espClient);
const char *ssid = "TIM-63619464";
const char *password = "uc4DfzHJHd36CZFJ";
const char *mqttBroker = "192.168.1.6";
const int mqttPort = 1883;
```

Figura 7: Dichiarazioni delle costanti e delle variabili globali

3.4 ESP32, MQTT Broker e MQTT Client

La comunicazione tra i dispositivi avviene tramite l'**ESP32**, un microcontrollore che permette l'utilizzo del Wi-Fi e del protocollo **MQTT** per trasmettere i dati rilevati dai sensori.

Come descritto precedentemente, il Raspberry assume il compito di MQTT Broker, mentre l'ESP32 funge da MQTT Client.

Il funzionamento di MQTT è il seguente:

- Il client MQTT stabilisce una connessione con il broker MQTT
- Una volta connesso, il client invia i messaggi al broker
- Quando il broker riceve il messaggio, lo inoltra ai dispositivi interessati

Per fare ciò, prima di tutto, è stata inclusa la libreria *PubSubClient*, appositamente per la messaggistica MQTT, che permette di ricevere e inviare messaggi.

All'interno del *setup*, come si può vedere sul codice della *Figura 8*, viene effettuata la connessione all'MQTT Broker, attraverso la funzione *client.setServer(mqttBroker, mqttPort)*, a cui vengono passati l'indirizzo dell'MQTT Broker e la sua porta di default.

Per la comunicazione e lo scambio dei dati, è necessario che la connessione con l'MQTT Broker sia sempre attiva, per questo è stato inserito all'interno del *void loop*, il controllo sullo stato della connessione.

Nel caso in cui, lo stato sia disconnesso, si tenta la connessione all'MQTT Broker, attraverso l'apposita funzione *client.connect()* ogni 5 secondi, visualizzando in output l'esito della connessione.

```
void loop() {
    if (!client.connected()) {
        connectToMQTT();
    }
    client.loop();
}

void connectToWiFi() {
    Serial.println("Connessione al WiFi..");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connettendosi al WiFi...");
    }
    Serial.println("Connesso al WiFi");
}

void connectToMQTT() {
    Serial.println("Connessione all MQTT");
    while (!client.connected()) {
        if (client.connect("ESP32Client")) {
            Serial.println("Connesso all MQTT Broker");
        } else {
            Serial.print("Fallita la connessione all MQTT, rc=");
            Serial.print(client.state());
            Serial.println(" Riprova in 5 secondi");
            delay(5000);
        }
    }
}
```

Figura 8: Codice Connessione Wifi e MQTT

L'ESP32 si connette ad un server MQTT, avvia la connessione e attende di ricevere comandi da parte di un client. In questo caso i comandi arriveranno dal Raspberry Pi e dallo smartphone del client.

3.5 Sensori

L'ESP32 dovrà inviare costantemente i valori misurati dai sensori all'MQTT Broker, i quali verranno poi memorizzati all'interno di un database apposito. Nel caso in cui le misure rilevate superino una certa soglia, verrà generato e inviato un alert al client.

Ora, vediamo in dettaglio come sono stati sviluppati i vari task.

3.5.1 BMP280

Il BMP280 è un sensore utilizzato per misurare temperatura e pressione dell'ambiente circostante.

Al fine di memorizzare i dati misurati, vengono definiti due vettori e viene impostata una temperatura di soglia massima, che nel nostro caso è stata impostata pari a 5°C, così da mostrare il funzionamento dell'invio alert.

La lettura della temperatura e della pressione deve essere costante e ciclica, per questo sono state inserite le apposite funzioni *bmp.readTemperature()* e *bmp.readPressure()* all'interno di un ciclo *while()*.

Ogni valore letto, viene inviato dall'ESP32 sotto forma di stringa all'MQTT tramite

```
client.publish(temperatureTopic, String(temperature).c_str())
```

e nel caso in cui il monitoraggio della soglia sia abilitato, *temperatureThresholdEnabled=true*, si verifica se il valore misurato è maggiore della soglia impostata.

Se la temperatura rilevata è superiore a 5°C, viene inviato un alert al bot Telegram e si disattiva il monitoraggio dei valori, impostando a *false* la variabile booleana, così da evitare che arrivino al client continui messaggi. Soltanto quando i valori rilevati torneranno ad essere minori della soglia, verrà riabilitato il monitoraggio di essi.

```
bool temperatureThresholdEnabled = true; // Variabile per tenere traccia dello stato del monitoraggio della soglia

void readBMP280Task(void *parameter) {
    (void)parameter;
    char temperatureTopic[] = "temperatura";
    char pressureTopic[] = "pressione";

    float temperatureThreshold = 5.0; //temperatura in C

    while (1) {
        // Leggi temperatura e pressione dal BMP280
        float temperature = bmp.readTemperature();
        float pressure = bmp.readPressure() / 100.0; // Pressione in hPa

        // Pubblica i dati del BMP280 su MQTT
        if (client.publish(temperatureTopic, String(temperature).c_str())) {
            Serial.print("Temperatura inviata al MQTT Broker: ");
            Serial.println(temperature);

            // Controllo della soglia solo se il monitoraggio della soglia è abilitato
            if (temperatureThresholdEnabled && temperature > temperatureThreshold) {
                sendTelegramMessage("La temperatura è superiore a 5 gradi Celsius!");
                temperatureThresholdEnabled = false; // Disattiva il monitoraggio della soglia
            }

            // Riattiva il monitoraggio della soglia se la temperatura scende sotto la soglia
            if (!temperatureThresholdEnabled && temperature <= temperatureThreshold) {
                temperatureThresholdEnabled = true;
            }
        } else {
            Serial.println("Fallito nell'inviare la temperatura al Broker MQTT");
        }

        if (client.publish(pressureTopic, String(pressure).c_str())) {
            Serial.print("Pressione inviata al Broke MQTT: ");
            Serial.println(pressure);
        } else {
            Serial.println("Fallito nell'inviare la pressione al Broker MQTT");
        }

        vTaskDelay(10000 / portTICK_PERIOD_MS); // Ritardo di 10 secondo
    }
}
```

Figura 9: Lettura ed invio dati BMP280

Anche per la pressione vengono effettuati gli stessi controlli.

Come si può notare nell'immagine sottostante, i due alert sono inviati uno a distanza di mezz'ora dall'altro, a dimostrazione che l'alert si attiva e disattiva quando la soglia viene superata o quando i valori rientrano nella norma.

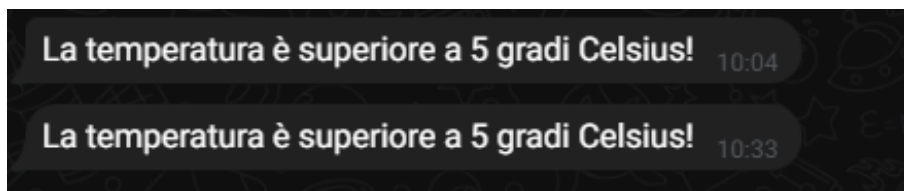


Figura 10: Alert superamento soglia temperatura

3.5.2 BH1750

Il BH1750 è un sensore in grado di rilevare la luminosità dell'ambiente circostante. Anche in questo caso è stata istanziata una variabile booleana per tenere traccia dello stato del monitoraggio della soglia.

Come per la temperatura e la pressione, ogni volta che viene misurata la luminosità, l'ESP32 si occupa di inviare i dati all'MQTT Broker, visualizzando in output l'esito dell'invio.

Ogni valore rilevato viene confrontato con il valore di soglia, che in questo caso è stato impostato a 500 lumen, in quanto corrisponde ad una luminosità molto bassa. Se il valore confrontato è maggiore della soglia, viene inviato un alert al client e viene disattivato il monitoraggio, finché il valore non rientrerà nel range.

```
bool lightThresholdEnabled = true; // Variabile per tenere traccia dello stato del monitoraggio della soglia

void readBH1750Task(void *parameter) {
    (void)parameter;
    char lightTopic[] = "luminosità";

    float lightThreshold = 500; //luminosità in Lumen

    while (1) {
        // Leggi l'intensità luminosa dal BH1750
        float lux = lightSensor.readLightLevel();
        // Pubblica i dati del BMP280 su MQTT
        if (client.publish(lightTopic, String(lux).c_str())) {
            Serial.print("Luminosità inviata al Broke MQTT: ");
            Serial.println(lux);

            // Controllo della soglia solo se il monitoraggio della soglia è abilitato
            if (lightThresholdEnabled && lux > lightThreshold) {
                sendTelegramMessage("La luminosità è superiore a 500 lumen!");
                lightThresholdEnabled = false; // Disattiva il monitoraggio della soglia
            }

            // Riattiva il monitoraggio della soglia se la temperatura scende sotto la soglia
            if (!lightThresholdEnabled && lux <= lightThreshold) {
                lightThresholdEnabled = true;
            }
        } else {
            Serial.println("Fallito nell'inviare la luminosità al Broker MQTT");
        }

        vTaskDelay(10000 / portTICK_PERIOD_MS); // Ritardo di 10 secondo
    }
}
```

Figura 11: Codice *Lettura e invio dati BH1750*

3.5.3 RTC

Per il sensore RTC vengono definiti due vettori, uno per le date ed uno per il tempo, per memorizzare la data e l'ora di rilevamento dei dati dei due sensori BMP280 e BH1750.

L'ora e la data letta dal sensore vengono memorizzate in formati separati, ovvero l'ora viene scomposta, tramite apposita funzione, in ora, minuti e secondi e la data viene suddivisa in anno, mese e giorno.

Ciascun dato viene poi inviato dall'ESP32 all'MQTT Broker.

```
void readRTCtask(void *parameter) {
    (void)parameter;
    char dateTopic[] = "data";
    char timeTopic[] = "ora";
    while (1) {
        // Leggi l'ora corrente dal RTC
        DateTime now = rtc.now();
        // Pubblica i dati del RTC su MQTT
        if (client.publish(dateTopic, (String(now.year()) + "/" + String(now.month()) + "/" + String(now.day())).c_str())) {
            Serial.print("Data inviata al Broker MQTT: ");
            Serial.print(now.year());
            Serial.print("/");
            Serial.print(now.month());
            Serial.print("/");
            Serial.println(now.day());
        } else {
            Serial.println("Fallito nell'inviare la data al Broker MQTT");
        }
        if (client.publish(timeTopic, (String(now.hour()) + ":" + String(now.minute()) + ":" + String(now.second())).c_str())) {
            Serial.print("Ora inviata al Broker MQTT");
            Serial.print(now.hour());
            Serial.print(":");
            Serial.print(now.minute());
            Serial.print(":");
            Serial.println(now.second());
        } else {
            Serial.println("Fallito nell'inviare l'ora al Broker MQTT");
        }
        vTaskDelay(10000 / portTICK_PERIOD_MS); // Ritardo di 10 secondi
    }
}
```

Figura 12: Codice *readRTCtask*

3.6 Bot Telegram

Telegram è un servizio di messaggistica che permette la creazione di bot. Di fatti, il bot Telegram permette di far interagire il client attraverso messaggi Telegram, in particolare permetterà all'utente di visualizzare i valori misurati, gli eventuali alert ed i grafici sugli andamenti delle grandezze, in base ad un particolare range.

Per implementare un nuovo bot, viene utilizzato BotFather, che è il bot padre da cui si possono creare altri bot. Si cerca su Telegram *@BotFather* e si invia sulla chat il comando */newbot*. Dopodiché il BotFather chiederà il nome e l'identificativo da associare al nuovo bot, per poi fornire il token identificativo, da salvare, come da immagine sottostante.

```
//Costanti per il bot Telegram
const char *telegramBotToken = "6728633709:AAFXXIKfqvrAS2ub1CwPKIJ5PIdrKqdgEps";
const char *telegramChatID = "364476654";
```

Figura 13: Credenziali Bot

Una volta generato il bot, si passa all'implementazione del codice. Nel nostro caso, il bot viene configurato sia per l'invio che per la ricezione dei messaggi.

Per consentire la comunicazione tra i dispositivi sono state incluse le seguenti librerie:

- **UniversalTelegramBot:** permette l'utilizzo dell'API di Telegram Bot
- **HTTPClient:** permette di creare con facilità richieste HTTP a un server web

Tramite l'ausilio della libreria `HTTPClient`, viene implementata la funzione `sendTelegramMessage` per l'invio dei messaggi. L'ESP32 invia una richiesta HTTP al Raspberry (MQTT Broker), il quale restituisce una risposta al client (ESP32).

Per instaurare la comunicazione viene utilizzata la funzione `http.begin(url)`, dove `url` contiene i riferimenti del bot creato con il messaggio inviato dall'utente e tramite il `GET()` viene inviata la richiesta di dati di una risorsa specifica.

```
// Funzione per inviare un messaggio a un bot Telegram
void sendTelegramMessage(const char *message) {
    HTTPClient http;
    String url = "https://api.telegram.org/bot" + String(telegramBotToken) + "/sendMessage?chat_id=" + String(telegramChatID) + "&text=" + String(message);
    http.begin(url);
    int httpStatusCode = http.GET();
    if (httpStatusCode > 0) {
        Serial.print("Messaggio Telegram inviato. Response code: ");
        Serial.println(httpStatusCode);
    } else {
        Serial.print("Errore nell'inviare il messaggio Telegram. Error code: ");
        Serial.println(httpStatusCode);
    }
    http.end();
}
```

Figura 14: Codice Messaggio Bot

Il client potrà inviare sulla chat uno dei seguenti comandi:

1. **start:** per avviare la comunicazione ed il bot invierà una risposta preimpostata da noi.
2. **leggi:** per leggere i dati memorizzati all'interno del database relativi ad un range di tempo, che spiegheremo nel paragrafo successivo.

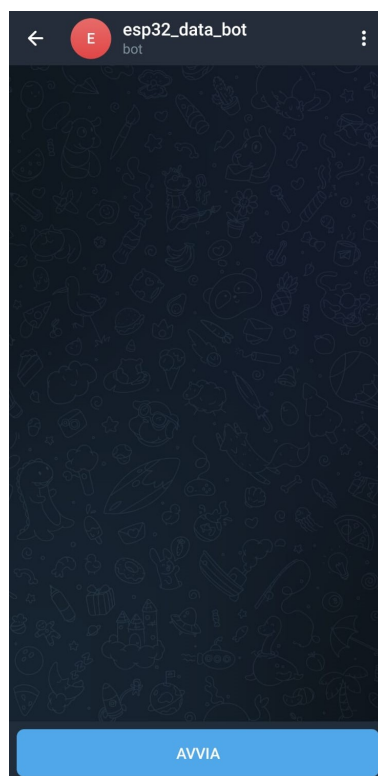


Figura 15: Bot Telegram

3.7 Creazione Database e caricamento dati

Il database utilizzato per questo progetto è un database MySQL, gestito con phpMyAdmin e viene eseguito su una macchina Windows locale ed un server Apache. Per poter configurare e gestire facilmente il DB, è stato installato XAMPP e creato il DB tramite l'interfaccia utente di phpMyAdmin.

XAMPP è un software che permette di testare il database in locale, phpMyAdmin è un software che permette la gestione di MySQL.

Nella schermata principale di XAMPP vengono mostrati i vari servizi disponibili; nel nostro caso vengono selezionati Apache e MySQL. Cliccando su *Admin* in corrispondenza di MySQL, si aprirà la pagina di *phpMyAdmin*.

Per creare il database, viene effettuato il login con l'account root di default e poi si clicca su *create database* e a seguire su *create table* per generare la tabella *sensor data* che conterrà tutti i parametri dei sensori. Gli attributi della tabella sono stati scelti in maniera tale da avere degli attributi di tipo:

- **float:** per pressione e temperatura
- **int autoincrementale:** per gli ID
- **DATETIME:** per data e ora del sensore RTC

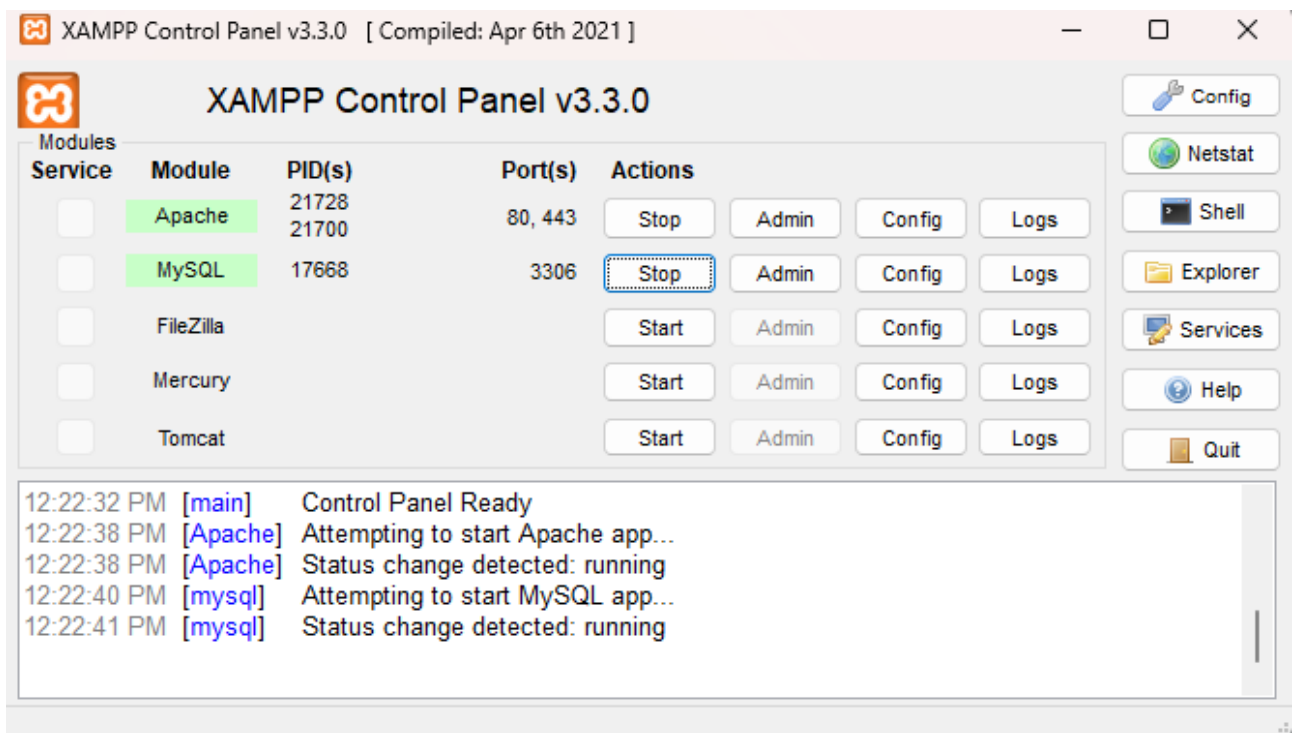


Figura 16: GUI Xampp

Una volta creato il database, si effettua la connessione ad esso, tramite apposita funzione *mysql.connector.connect* a cui vengono passati:

- L'indirizzo del server MySQL
- Il nome utente associato al database
- La password del database
- Il nome del database

Per poter caricare i dati rilevati all'interno del database è necessario utilizzare una funzione apposita per sniffare i messaggi MQTT. Prima di tutto, si crea un nuovo client, poi si imposta la funzione *on_message* come callback per gestire i messaggi MQTT ricevuti, si effettua la configurazione di esso e lo si connette al broker.

Nel dettaglio, la funzione *on_message* verifica se il topic inserito dal client ha una corrispondenza nella tabella del database. In caso affermativo, prepara la query con i valori rilevati della temperatura, pressione, luminosità, data e ora ed esegue l'inserimento all'interno del database, tramite *cursor.execute()*.

Dopodiché, una volta avvenuta la connessione del client al broker MQTT, si sottoscrive il client a ciascun topic specificato, questo perché, dovendo sniffare continuamente i dati in ingresso, per rendere tutto più efficiente si è deciso di creare una mappa dei topic di interesse, con i rispettivi nomi di *temperatura*, *pressione*, *luminosità*, *data* e *ora*, in seguito si avvia il loop di rete del client per ricevere i messaggi.

```
# Funzione di callback per la ricezione dei messaggi MQTT
def on_message(client, userdata, message):
    topic = message.topic
    payload = message.payload.decode('utf-8')

    # Verifica se il topic è presente nel mapping
    if topic in mapping:
        # Prendi il nome della colonna dal mapping
        column_name = mapping[topic]

        # Prepara la query SQL per l'inserimento
        query = f"INSERT INTO sensor_data (temperatura, pressione, luminosità, data, ora) VALUES (%s, %s, %s, %s, %s)"

        # Esegui l'inserimento dei dati
        cursor.execute(query, (payload, payload, payload, payload, payload))
        conn.commit()
        print("Dati inseriti correttamente.")
    else:
        print("Percorso del messaggio non trovato nel mapping.")

def sniff_mqtt_messages(broker_address, topics):

    # Imposta la funzione on_message come callback per gestire i messaggi MQTT ricevuti
    client.on_message = on_message

    # Connetti il client MQTT al broker MQTT
    client.connect(broker_address)

    # Sottoscrivi il client MQTT a ciascun topic specificato
    for topic in topics:
        client.subscribe(topic)
        print("Funzione di sniff&push avvenuta con successo")

    # Avvia il loop di rete del client MQTT per ricevere i messaggi
    client.loop_forever()

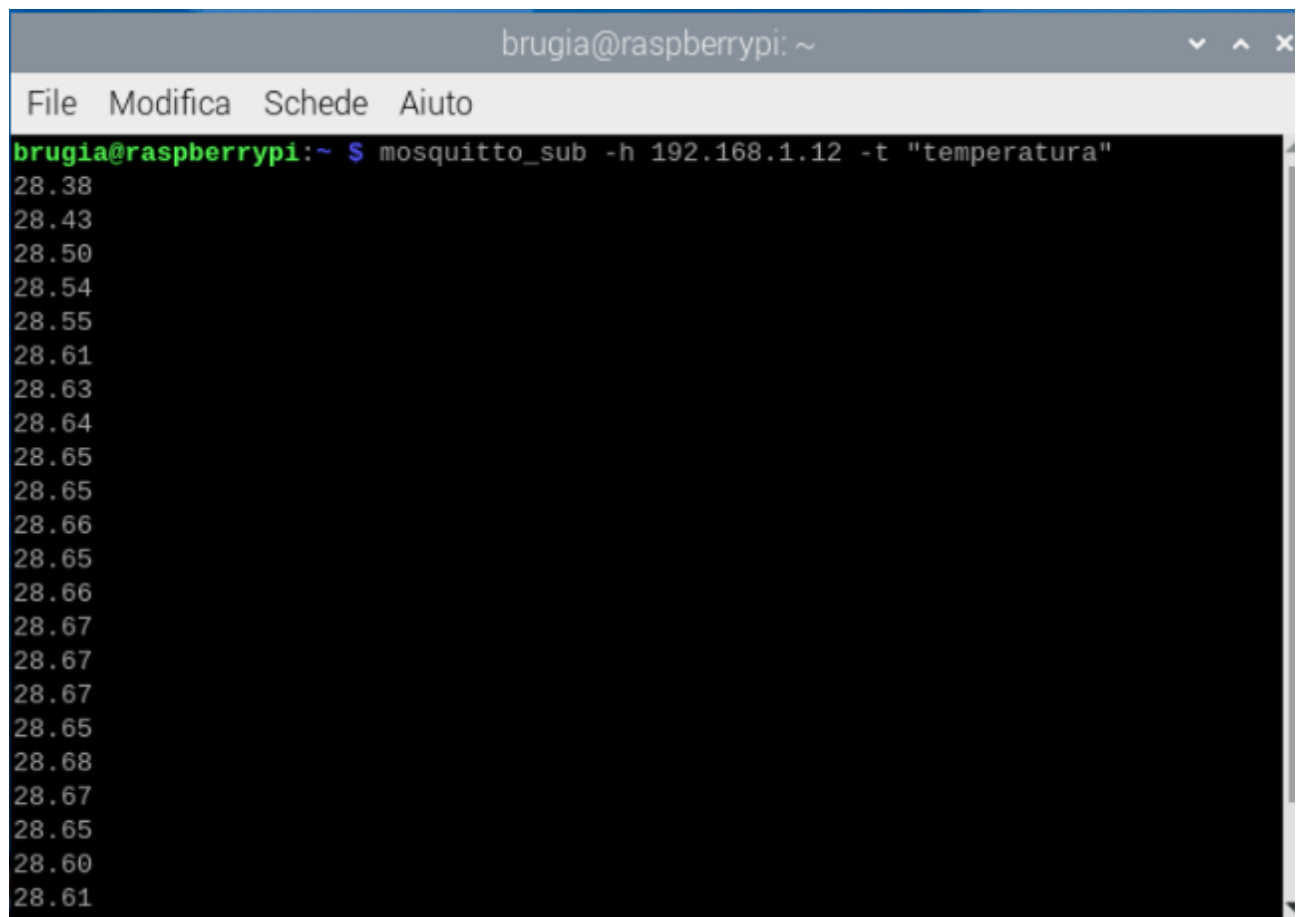
# Configurazione del client MQTT
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_message = on_message

# Connessione al broker MQTT
broker_address = '192.168.1.6' # Inserisci l'indirizzo del broker MQTT
client.connect(broker_address)
client.subscribe('topic_da_sottoscrivere') # Sostituisci con il topic che desideri sottoscrivere

sniff_mqtt_messages(broker_address, topics)
```

Figura 17: Codice per sniffare e caricare i dati

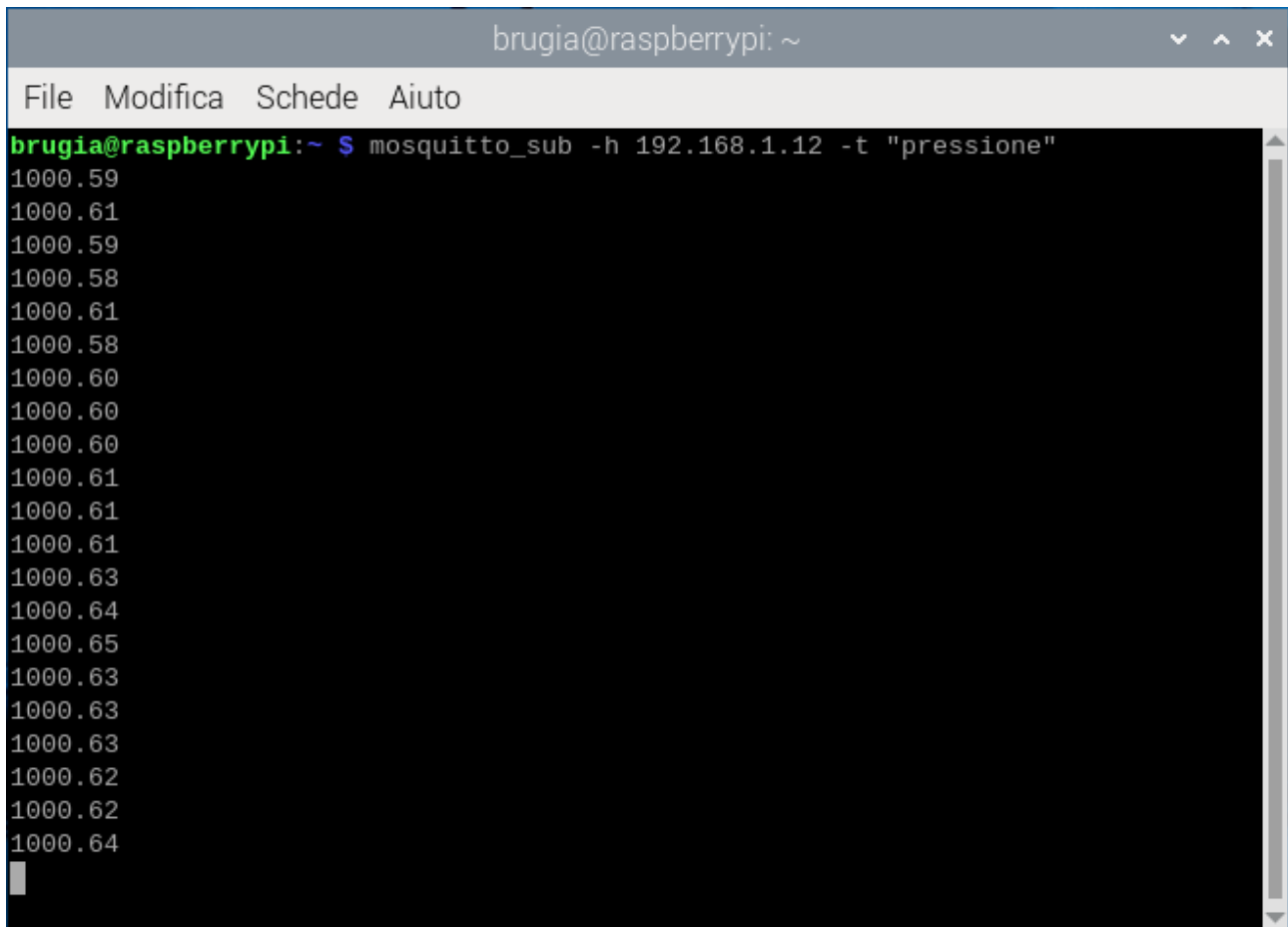
Per controllare se il broker riceve i dati correttamente basta utilizzare il comando *mosquitto_sub -h "ip_broker" -t "nome_topic"*.



A terminal window titled 'brugia@raspberrypi: ~' with a menu bar containing 'File', 'Modifica', 'Schede', and 'Aiuto'. The terminal shows the command 'mosquitto_sub -h 192.168.1.12 -t "temperatura"' being executed. Below the command, a list of temperature values is displayed, ranging from 28.38 to 28.61.

```
brugia@raspberrypi:~ $ mosquitto_sub -h 192.168.1.12 -t "temperatura"
28.38
28.43
28.50
28.54
28.55
28.61
28.63
28.64
28.65
28.65
28.66
28.65
28.66
28.67
28.67
28.67
28.65
28.68
28.67
28.65
28.60
28.61
```

Figura 18: Temperatura inviata al broker



A terminal window titled 'brugia@raspberrypi: ~' with a menu bar containing 'File', 'Modifica', 'Schede', and 'Aiuto'. The terminal shows the command 'mosquitto_sub -h 192.168.1.12 -t "pressione"' being executed. Below the command, a series of numerical values representing pressure are displayed on each line, ranging from 1000.59 to 1000.64. A vertical scrollbar is visible on the right side of the terminal window.

```
brugia@raspberrypi:~ $ mosquitto_sub -h 192.168.1.12 -t "pressione"
1000.59
1000.61
1000.59
1000.58
1000.61
1000.58
1000.60
1000.60
1000.60
1000.61
1000.61
1000.61
1000.63
1000.64
1000.65
1000.63
1000.63
1000.63
1000.62
1000.62
1000.64
```

Figura 19: Pressione inviata al broker

Nell'immagine sottostante possiamo vedere il database creato con i valori caricati.

Server: 127.0.0.1 Database: esp32_data Tabella: sensor_data

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni Trigger

Mostrare le righe 0 - 24 (84 del totale, La query ha impiegato 0.0002 secondi.)

SELECT * FROM `sensor_data`

Profiling [Modifica inline] [Modifica] [Spiega SQL] [Crea il codice PHP] [Aggiorna]

1 > >> ☐ Mostra tutti Numero di righe: 25 Filtra righe: Cerca nella tabella Ordina per chiave: Nessuno

Opzioni extra

				id	pressione	luminosità	temperatura	data	ora
<input type="checkbox"/>	Modifica	Copia	Elimina	97	1004.3	8.33	23.75	2024-03-18	15:20:14
<input type="checkbox"/>	Modifica	Copia	Elimina	98	1004.25	8.33	23.97	2024-03-18	15:21:24
<input type="checkbox"/>	Modifica	Copia	Elimina	99	1004.25	8.33	24	2024-03-18	15:21:34
<input type="checkbox"/>	Modifica	Copia	Elimina	100	1004.24	8.33	24.01	2024-03-18	15:21:44
<input type="checkbox"/>	Modifica	Copia	Elimina	101	1004.23	8.33	24.01	2024-03-18	15:21:54
<input type="checkbox"/>	Modifica	Copia	Elimina	102	1004.25	8.33	24.01	2024-03-18	15:22:04
<input type="checkbox"/>	Modifica	Copia	Elimina	103	1004.25	8.33	24.04	2024-03-18	15:22:14
<input type="checkbox"/>	Modifica	Copia	Elimina	104	1004.24	8.33	24.05	2024-03-18	15:22:24
<input type="checkbox"/>	Modifica	Copia	Elimina	105	1004.21	8.33	24.06	2024-03-18	15:22:34
<input type="checkbox"/>	Modifica	Copia	Elimina	106	1004.29	9.17	24.07	2024-03-18	15:22:44
<input type="checkbox"/>	Modifica	Copia	Elimina	107	1004.27	0.83	23.93	2024-03-18	15:22:54
<input type="checkbox"/>	Modifica	Copia	Elimina	108	1004.31	0.83	23.58	2024-03-18	15:23:04
<input type="checkbox"/>	Modifica	Copia	Elimina	109	1004.28	17.5	23.43	2024-03-18	15:23:14
<input type="checkbox"/>	Modifica	Copia	Elimina	110	1004.28	18.33	23.75	2024-03-18	15:23:24
<input type="checkbox"/>	Modifica	Copia	Elimina	111	1004.28	18.33	23.93	2024-03-18	15:23:34
<input type="checkbox"/>	Modifica	Copia	Elimina	112	1004.29	18.33	24.04	2024-03-18	15:23:44
<input type="checkbox"/>	Modifica	Copia	Elimina	113	1004.28	2.5	25.75	2024-03-18	15:23:54
<input type="checkbox"/>	Modifica	Copia	Elimina	114	1004.27	0.83	29.15	2024-03-18	15:24:04
<input type="checkbox"/>	Modifica	Copia	Elimina	115	1004.28	21.67	27.97	2024-03-18	15:24:14
<input type="checkbox"/>	Modifica	Copia	Elimina	116	1004.28	20	26.82	2024-03-18	15:24:24
<input type="checkbox"/>	Modifica	Copia	Elimina	117	1004.25	20	25.89	2024-03-18	15:24:34
<input type="checkbox"/>	Modifica	Copia	Elimina	118	1004.25	21.67	25.1	2024-03-18	15:24:44
<input type="checkbox"/>	Modifica	Copia	Elimina	119	1004.22	21.67	24.42	2024-03-18	15:24:54
<input type="checkbox"/>	Modifica	Copia	Elimina	120	1004.29	21.67	23.82	2024-03-18	15:25:04
<input type="checkbox"/>	Modifica	Copia	Elimina	121	1004.3	21.67	23.26	2024-03-18	15:25:14

Seleziona tutto Se selezionati: Modifica Copia Elimina Esporta

1 > >> ☐ Mostra tutti Numero di righe: 25 Filtra righe: Cerca nella tabella Ordina per chiave: Nessuno

Figura 20: Database

Il client può richiedere la lettura dei dati memorizzati nel database, inviando nella chat la parola *leggi* seguita da data e ora. Nel caso in cui il formato inserito non sia valido, verrà inviato all'utente un messaggio di errore comunicando il formato con cui inviare i dati.

Per estrarre i dati dal database viene effettuata un'interrogazione di esso mediante una query in cui si vanno a selezionare dalla tabella *sensor_data* i valori relativi alla data e all'ora richiesta dal client, attraverso la funzione *cursor.fetchall()*.

Se esiste una corrispondenza dei dati con la data e l'ora richiesti, verranno inviati al bot Telegram rispettivamente i valori di pressione, luminosità e temperatura. Altrimenti, verrà generato un messaggio di errore oppure verrà indicato che non esiste alcun dato nella data e ora richiesti.

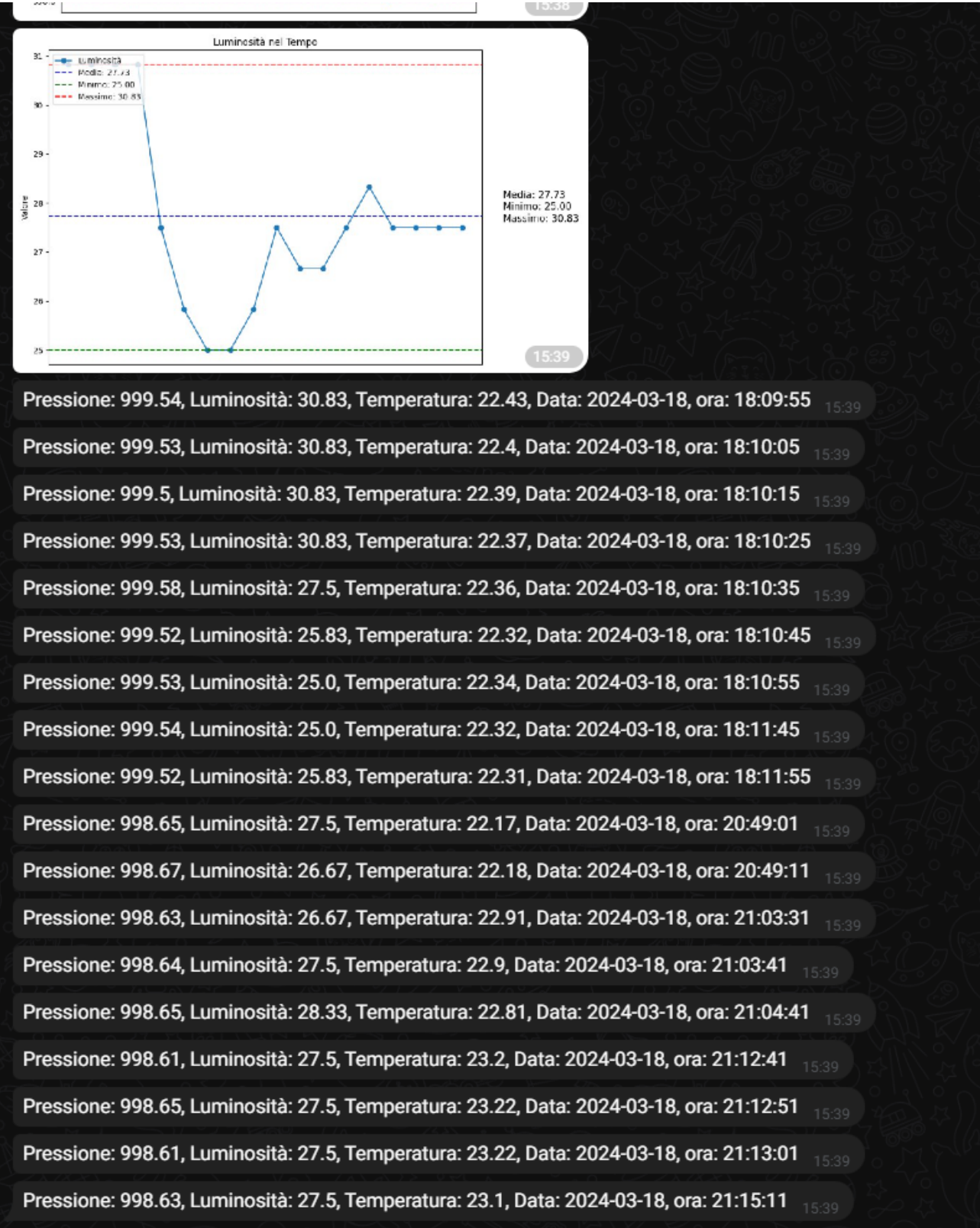


Figura 21: Comando Bot "leggi"



Figura 22: Messaggio di errore

3.8 Grafici

Il client può richiedere la generazione dei grafici a partire da una determinata data ed ora.

Viene effettuata l'estrazione dei dati dal database per ogni topic, come fatto in precedenza e viene calcolata per ogni misura la media dei valori, il valore massimo ed il valore minimo, tramite le apposite funzioni *mean()*, *max()* e *min()*

Per ogni topic verrà realizzato un grafico, contenente i valori memorizzati nel database a partire dalla data e ora richiesta dall'utente. Verranno visualizzati anche le statistiche calcolate precedentemente: valore medio, valore minimo e massimo del range selezionato.

Una volta generato il grafico, viene salvato come immagine, inviato al bot Telegram tramite l'apposita funzione *context.bot.send_photo()* e viene poi eliminato il file.

```

if results:
    # Verifica il tipo di dato del primo elemento per determinare il formato
    if isinstance(results[0][0], int):
        # Se il timestamp è un intero (UNIX timestamp)
        timestamps = [datetime.fromtimestamp(row[0]) for row in results]
    elif isinstance(results[0][0], str):
        # Se il timestamp è una stringa
        timestamps = [datetime.strptime(row[0], '%Y-%m-%d %H:%M:%S') for row in results]
    else:
        raise TypeError("Il tipo di dato del timestamp non è né un intero né una stringa")

    pressione = [row[1] for row in results]
    luminosita = [row[2] for row in results]
    temperatura = [row[3] for row in results]

    # Calcola media, minimo e massimo per ciascun parametro
    pressione_media = np.mean(pressione)
    pressione_minimo = np.min(pressione)
    pressione_massimo = np.max(pressione)

    luminosita_media = np.mean(luminosita)
    luminosita_minimo = np.min(luminosita)
    luminosita_massimo = np.max(luminosita)

    temperatura_media = np.mean(temperatura)
    temperatura_minimo = np.min(temperatura)
    temperatura_massimo = np.max(temperatura)

    # Funzione per creare il grafico e salvarlo
    def crea_grafico(timestamps, valori, parametro, media, minimo, massimo, nome_file):
        plt.figure(figsize=(10, 6))
        plt.plot(timestamps, valori, label=parametro, marker='o')
        plt.axhline(y=media, color='blue', linestyle='--', label=f'Media: {media:.2f}')
        plt.axhline(y=minimo, color='green', linestyle='--', label=f'Minimo: {minimo:.2f}')
        plt.axhline(y=massimo, color='red', linestyle='--', label=f'Massimo: {massimo:.2f}')

        plt.legend(loc='upper left')
        plt.xlabel('Data e Ora')
        plt.ylabel('Valore')
        plt.title(f'{parametro} nel Tempo')

        # Aggiungi le statistiche sul lato del grafico
        plt.annotate(f'Media: {media:.2f}\nMinimo: {minimo:.2f}\nMassimo: {massimo:.2f}',
                    xy=(1.05, 0.5), xycoords='axes fraction', fontsize=12, ha='left', va='center')

        # Nascondi l'asse X
        plt.gca().xaxis.set_visible(False)

        plt.tight_layout()
        plt.savefig(nome_file)
        plt.close()

    # Creare e salvare i grafici
    grafici = [
        (timestamps, temperatura, 'Temperatura', temperatura_media, temperatura_minimo, temperatura_massimo, 'grafico_temperatura.png'),
        (timestamps, pressione, 'Pressione', pressione_media, pressione_minimo, pressione_massimo, 'grafico_pressione.png'),
        (timestamps, luminosita, 'Luminosità', luminosita_media, luminosita_minimo, luminosita_massimo, 'grafico_luminosita.png')
    ]

    for timestamps, valori, parametro, media, minimo, massimo, nome_file in grafici:
        crea_grafico(timestamps, valori, parametro, media, minimo, massimo, nome_file)

    # Invia i grafici al bot Telegram
    for _, _, _, _, _, nome_file in grafici:
        await context.bot.send_photo(chat_id=user.id, photo=open(nome_file, 'rb'))
        os.remove(nome_file)

```

Figura 23: Codice per generare i grafici

I grafici ottenuti saranno come i seguenti:

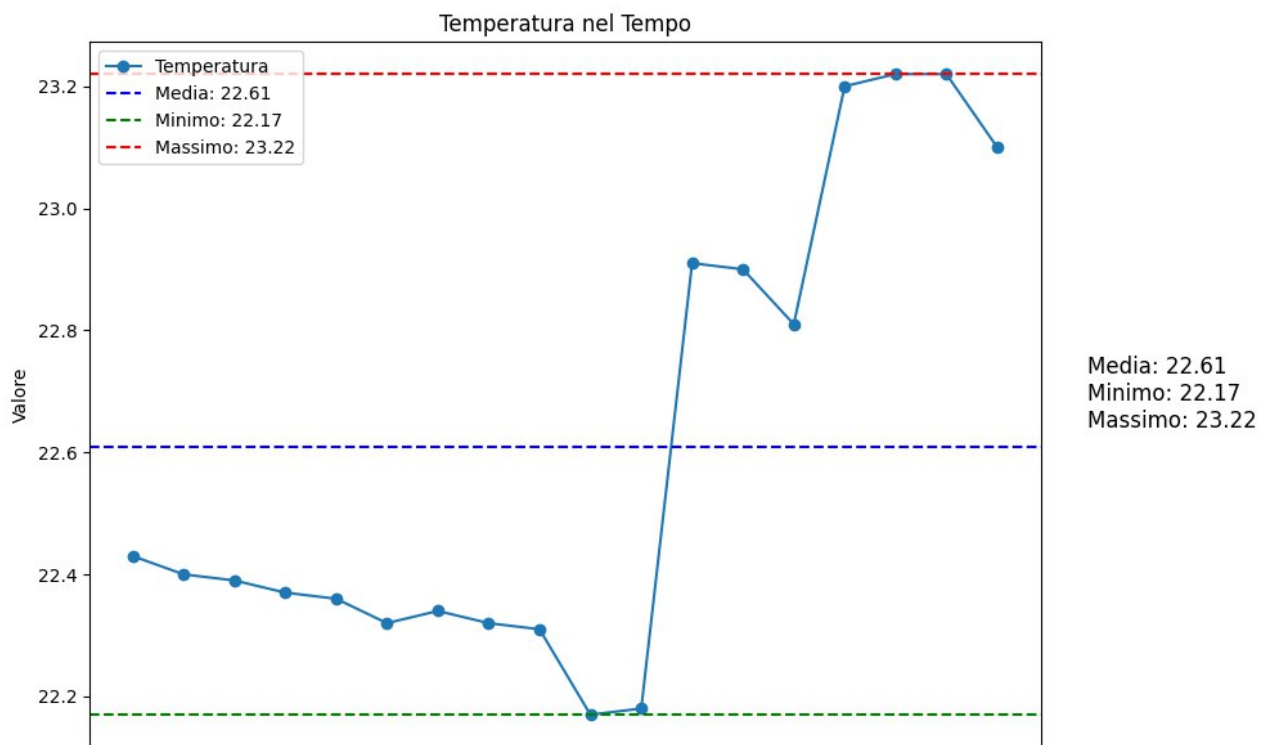


Figura 24: Grafico andamento temperatura

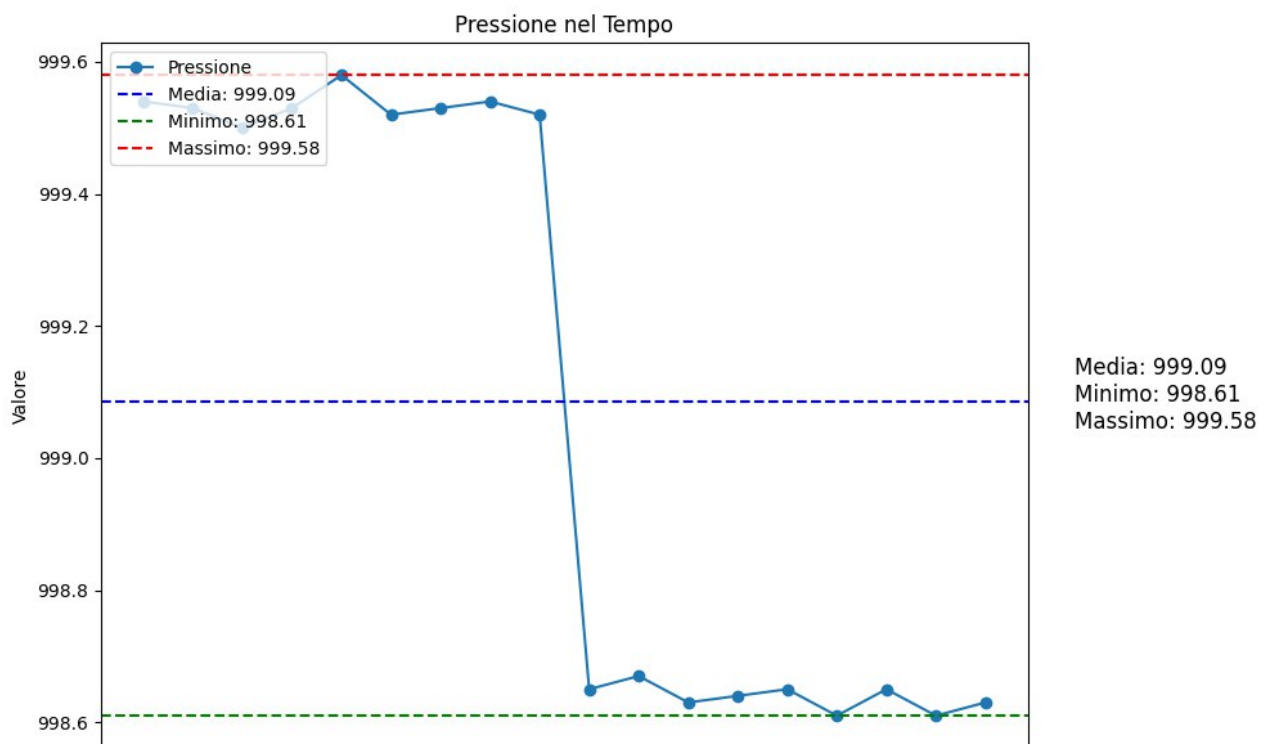


Figura 25: Grafico andamento pressione

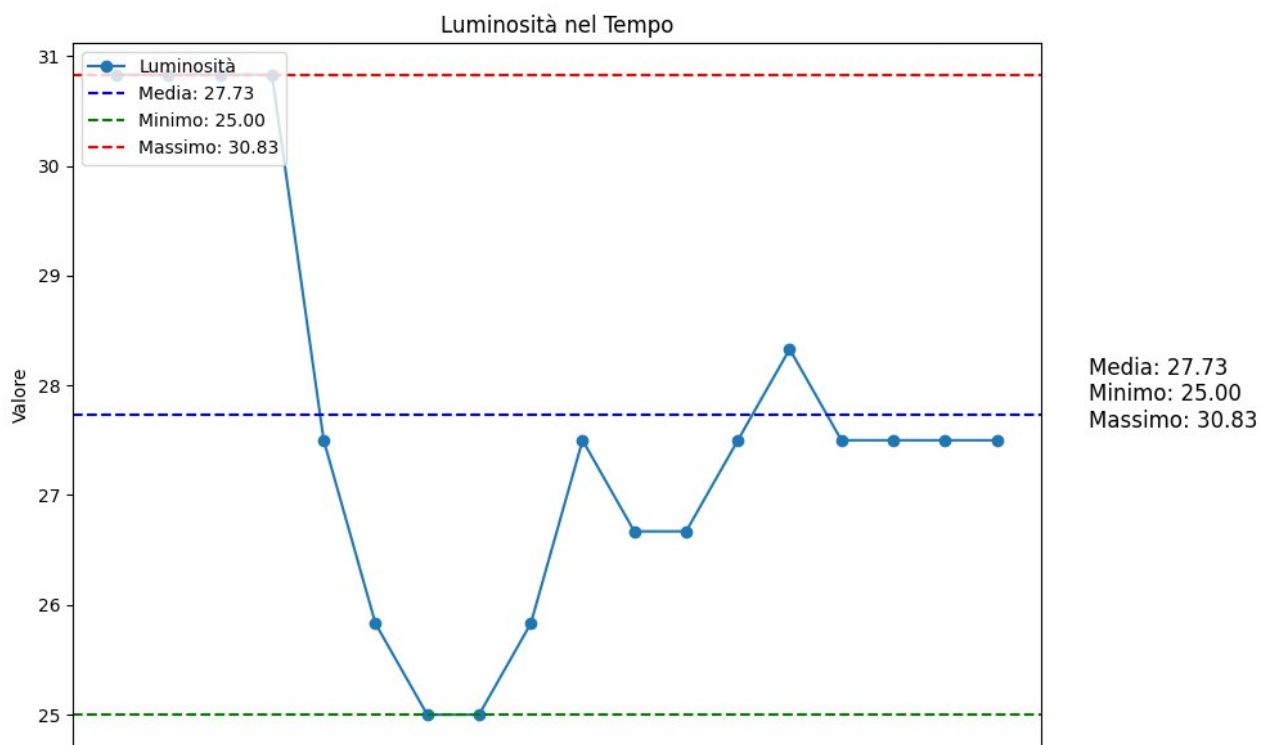


Figura 26: Grafico andamento luminosità

4 Considerazioni finali

Durante lo sviluppo del progetto non abbiamo riscontrato difficoltà rilevanti.

Studiando i vari sensori, abbiamo notato che il sensore BMP280 non rileva l'umidità, ma solo temperatura e pressione.

Diverse funzioni che venivano riportate nelle documentazioni delle librerie non erano compatibili con il dispositivo utilizzato, come ad esempio per la connessione all'MQTT, per la creazione del database ed il caricamento dei relativi dati.

Infine, avendo posizionato i sensori all'interno di una stanza, non siamo riusciti ad ottenere valori che si discostano molto tra loro.

Con questo progetto è stato realizzato un sistema che permette al client di monitorare l'ambiente circostante, controllando i dati rilevati dai vari sensori. L'utilità e le funzionalità fornite, integrate con altri dispositivi, possono essere impiegate quotidianamente sia a casa che negli ambienti lavorativi.