

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	5
2	Strumenti e software utilizzati	6
3	Sviluppo Progetto	7
3.1	Collegamento dispositivi	7
3.2	Configurazione Raspberry	8
3.3	FreeRTOS	14
3.4	ESP32, MQTT Broker e MQTT Client	18
3.5	Sensori	19
3.5.1	BMP280	20
3.5.2	BH1750	22
3.5.3	RTC	23
3.6	Bot Telegram	24
3.7	Creazione Database e caricamento dati	26
3.8	Grafici	33
4	Considerazioni finali	38

Elenco delle figure

1	Struttura progetto	5
2	Schema di cablatura del progetto	8
3	Raspberry Pi Imager	9
4	Abilitazione SSH	10
5	Modifica connessione	10
6	Configurazione IP statico	11
7	Visualizzazione IP	11
8	Visualizzazione IP da terminale	12
9	File Service	14
10	Stato MQTT Broker	14
11	Alert superamento soglia temperatura	21
12	Bot Telegram	26
13	GUI Xampp	27
14	Temperatura inviata al broker	29
15	Pressione inviata al broker	30
16	Database	31
17	Comando Bot "leggi"	33
18	Messaggio di errore	33
19	Grafico andamento temperatura	35
20	Grafico andamento pressione	36
21	Grafico andamento luminosità	36

Listings

1	Creazione Task	15
2	Rilevazione sensori e sincronizzazione data e ora	16
3	Inizializzazione sensori	16
4	NTP Client	16
5	Sincronizzazione RTC	17
6	Connessione WiFi	17
7	Dichiarazione costanti e variabili globali	18
8	Connessione MQTT Broker	18
9	Connessione WiFi e MQTT Broker	19
10	Sottoscrizione topic	20
11	Task lettura dati sensore BMP280	20
12	Task lettura dati sensore BH1750	22
13	Task lettura data e ora sensore RTC	23
14	Codice identificativo bot Telegram	24
15	Funzione invio messaggi Telegram	24
16	Funzione ricezione messaggi Telegram	25
17	Memorizzazione chat ID	26
18	Dati configurazione database	28
19	Connessione al database	28
20	Configurazione e avvio Client MQTT	28
21	Comando start	31
22	Comando leggi con estrazione dati	32
23	Comando leggi con estrazione dati	32
24	Calcolo statistiche	34
25	Definizione istruzioni main	37

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**

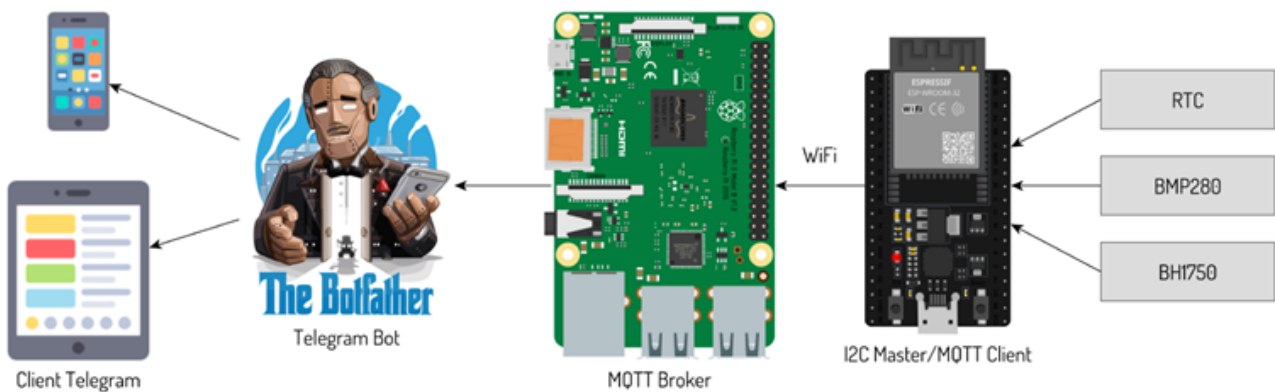


Figura 1: Struttura progetto

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 programmazione di dispositivi remoti mediante 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:

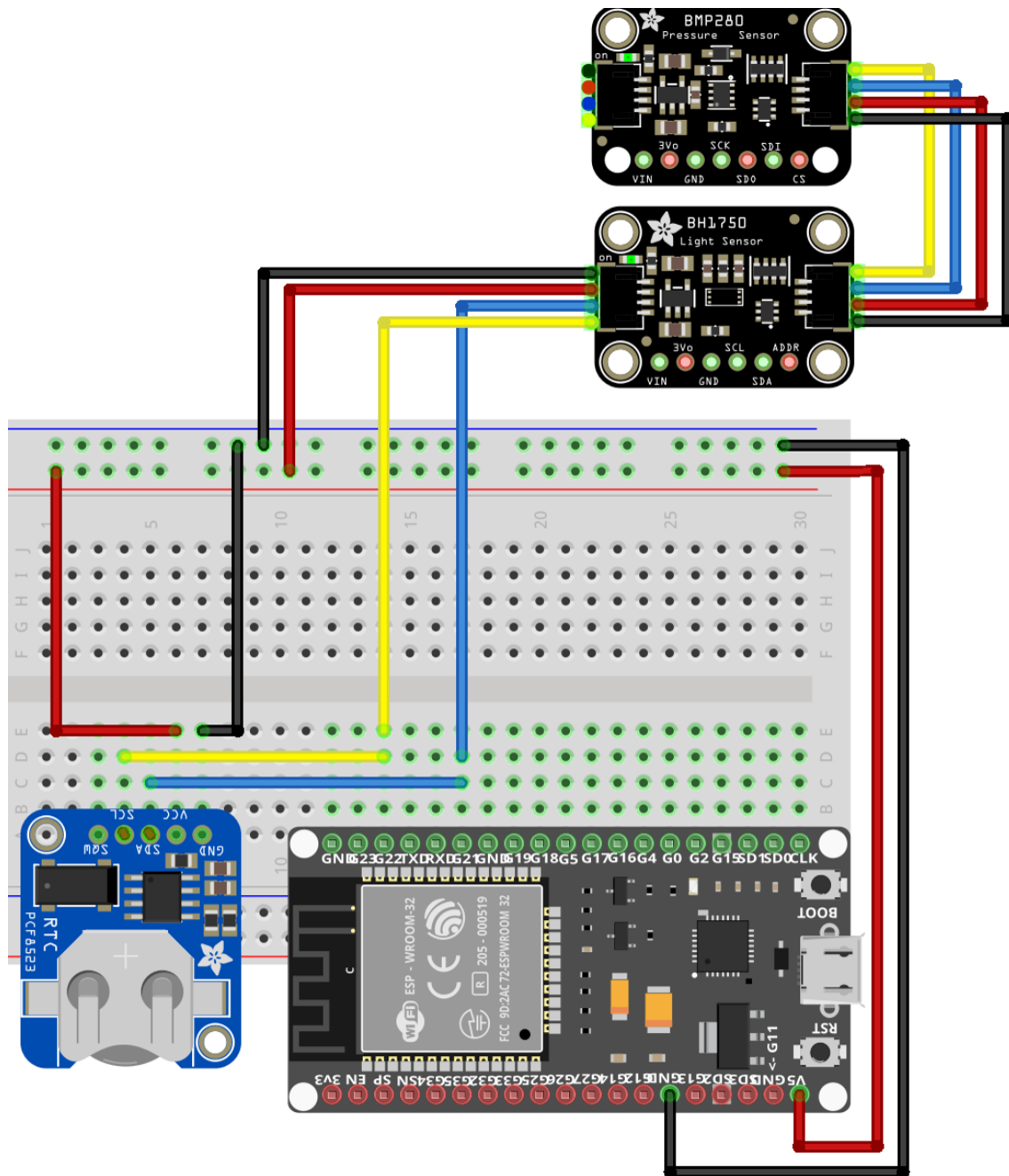


Figura 2: Schema di cablaggio del progetto

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).

Il protocollo **SSH** o Secure Shell permette una comunicazione client-server sicura tra dispositivi.

Per configurare la connessione SSH, viene utilizzato **PuTTY**, che 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 (da terminale inviare comando **hostname**)
- **nome utente:** nel nostro caso **brugia**

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

Dalle impostazioni del Raspberry, sulla sezione **Interfacce**, viene abilitato il servizio SSH, come da figura sottostante.

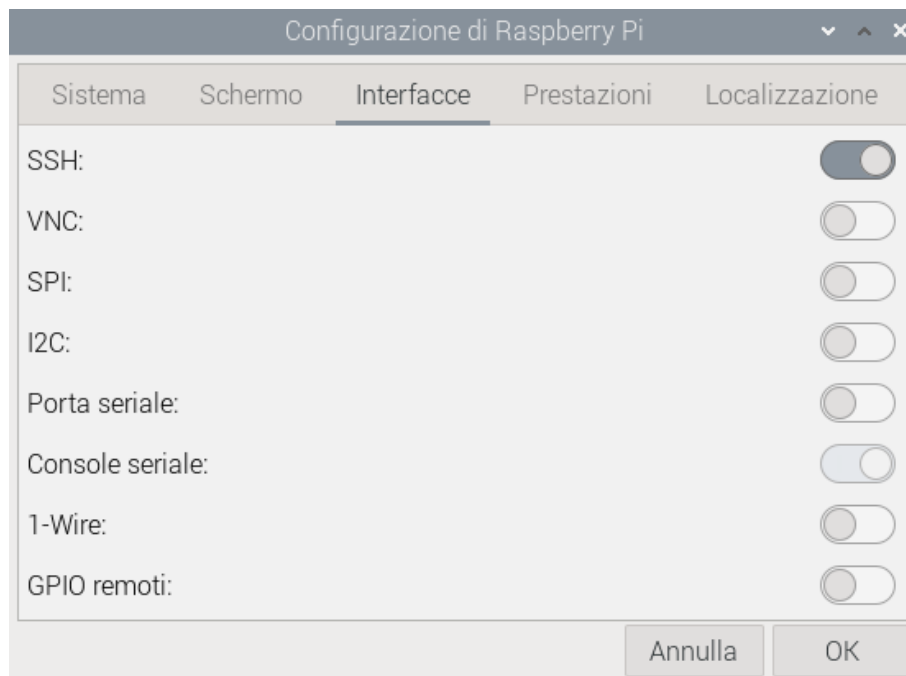


Figura 4: Abilitazione SSH

Essendo il Raspberry connesso alla rete ha un proprio indirizzo IP, che nel nostro caso è: **192.168.1.12**. Per assegnare al Raspberry un indirizzo IP statico, bisogna:

1. Inviare da terminale il seguente comando per modificare la connessione:

```
1 sudo nmtui edit "Nome connessione"
```

```
brugia@raspberrypi:~ $ sudo nmtui edit TIM-63619464
brugia@raspberrypi:~ $
```

Figura 5: Modifica connessione

2. Nella sezione **CONFIGURAZIONE IPv4** dell'editor che si è aperto, cambiare l'opzione **Automatico** in **Manuale**
3. Cliccare su **Mostra** ed inserire sotto il campo **Indirizzi**, l'indirizzo IP statico da assegnare al dispositivo.
4. Cliccare **OK** per salvare la configurazione.
5. Riavviare il servizio di rete, inviando da terminale:

```
sudo systemctl restart NetworkManager
```

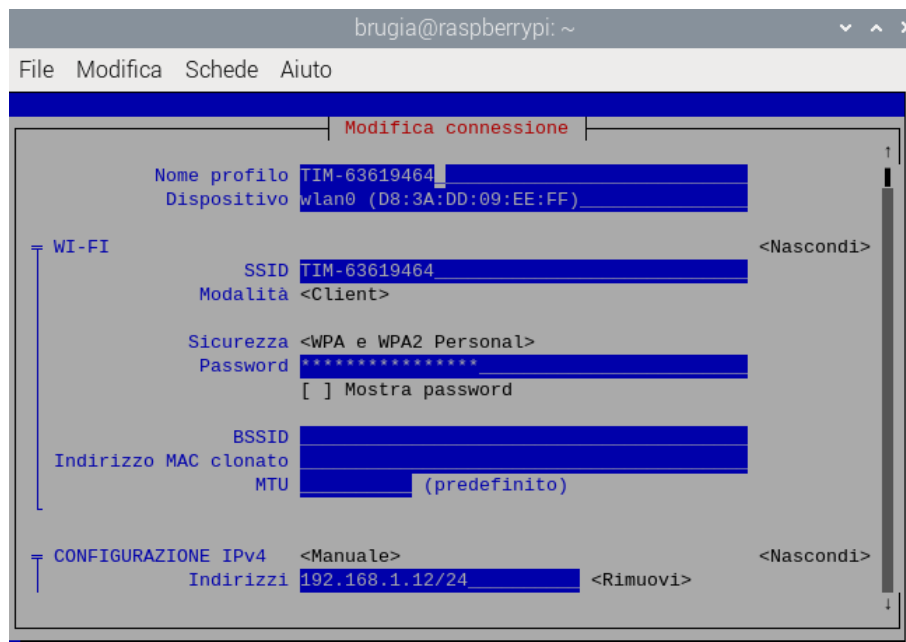


Figura 6: Configurazione IP statico

Per verificare l'indirizzo IP assegnato al Raspberry si può trascinare il mouse sopra l'icona di rete oppure da terminale basta inviare come comando **hostname -I**.

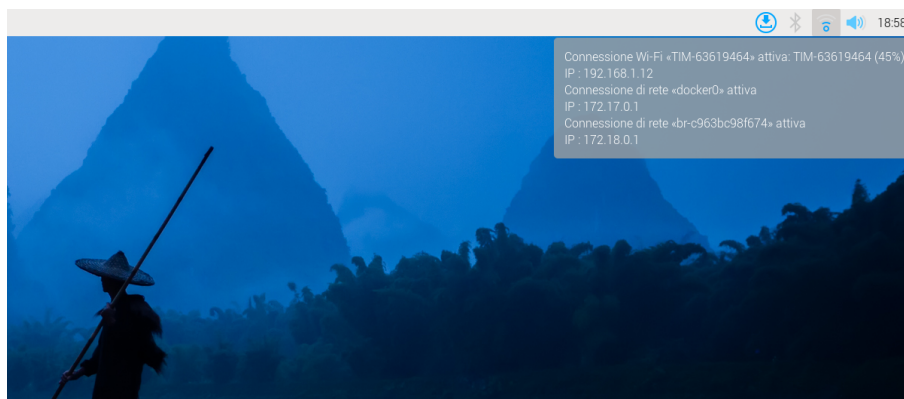


Figura 7: Visualizzazione IP

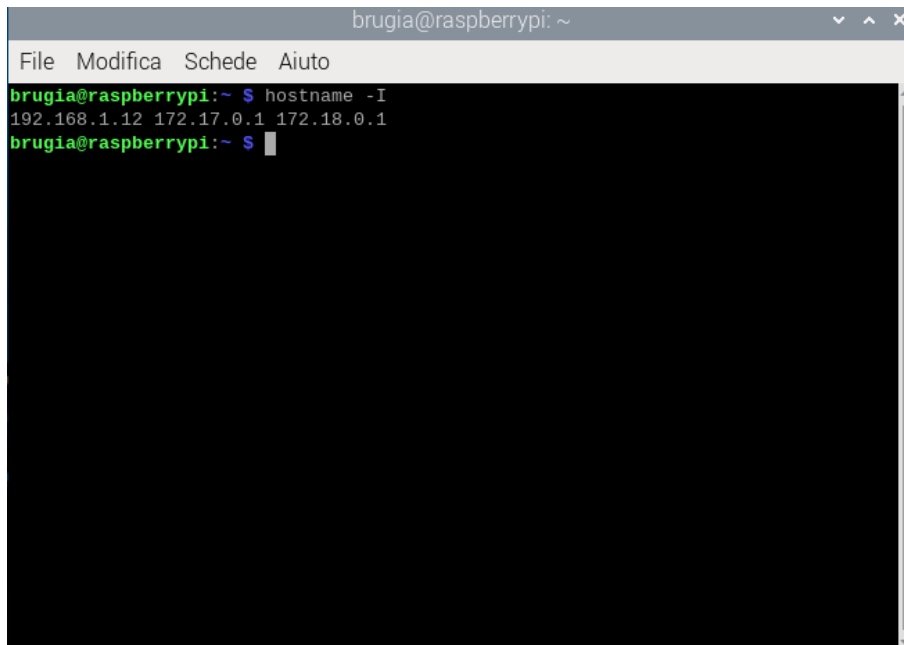


Figura 8: Visualizzazione IP da terminale

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, attraverso un broker, consente lo scambio di messaggi tra dispositivi.

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.

Per far avviare in automatico il servizio MQTT è necessario creare un file **service** all'interno della cartella **systemd** del Raspberry. Il file è composto da tre sezioni:

1. **Unit:**

- **Description:** nome servizio
- **After:** determina quando inizia il servizio.

2. **Service:**

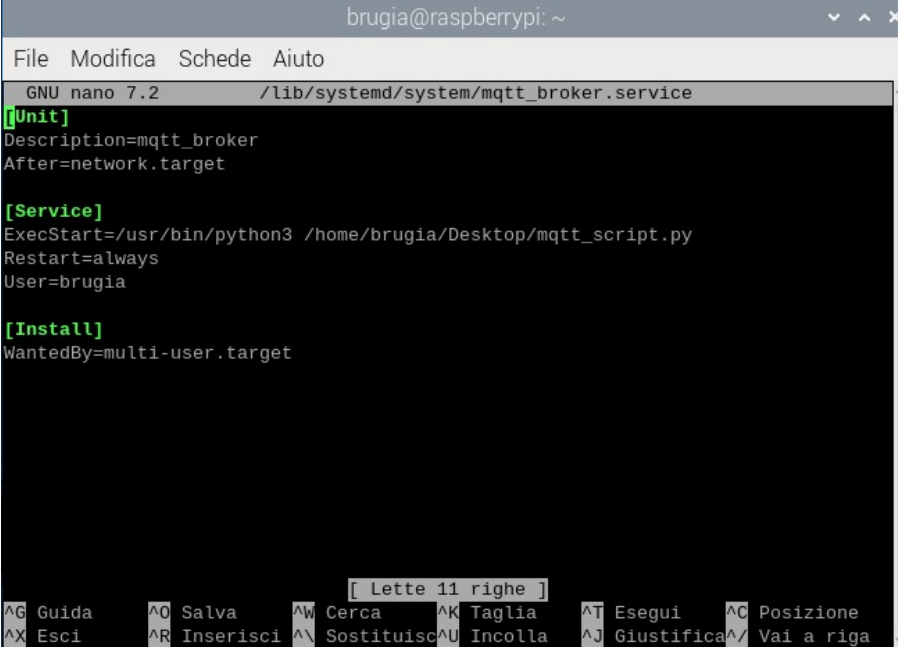
- **ExecStart:** contiene il percorso dello script che deve essere eseguito in automatico all'avvio.
- **Restart:** indica se il servizio deve essere riavviato in caso di uscita.
- **User:** identifica il nome del Raspberry.

3. **Install:**

- **WantedBy:** specifica l'attivazione di un'unità. Impostandola su **multi-user.target**, si può stabilire una relazione di dipendenza che eseguirà il programma quando il controllo del sistema passa alla shell multiutente.

Nel nostro caso, il servizio che viene eseguito in automatico è l'MQTT. In particolare, viene fatto eseguire lo script che si occupa della connessione all'MQTT broker e dell'invio dei dati rilevati dai sensori.

Nella figura sottostante si può vedere il file **service** che è stato creato all'interno della cartella **systemd** con le relative informazioni.



```
brugia@raspberrypi: ~  
File Modifica Schede Aiuto  
GNU nano 7.2 /lib/systemd/system/mqtt_broker.service  
[Unit]  
Description=mqtt_broker  
After=network.target  
  
[Service]  
ExecStart=/usr/bin/python3 /home/brugia/Desktop/mqtt_script.py  
Restart=always  
User=brugia  
  
[Install]  
WantedBy=multi-user.target  
  
[ Lette 11 righe ]  
^G Guida ^O Salva ^W Cerca ^K Taglia ^T Esegui ^C Posizione  
^X Esci ^R Inserisci ^_ Sostituisci ^U Incolla ^J Giustifica ^_ Vai a riga
```

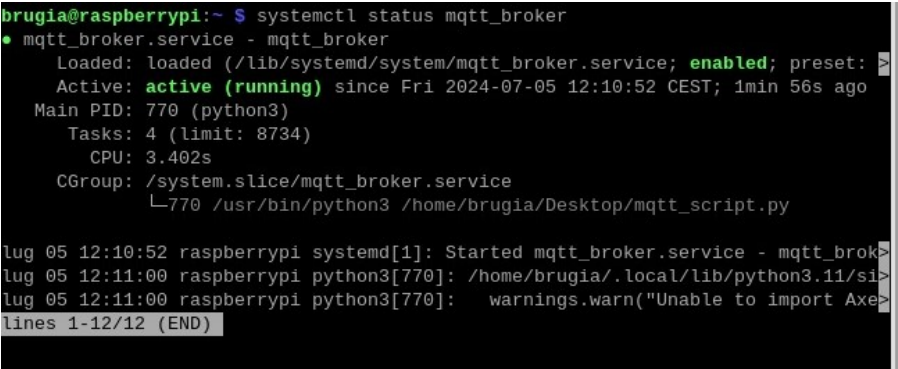
Figura 9: File Service

Una volta salvato il file, si effettua il riavvio del dispositivo.

Per verificare che il servizio venga effettivamente eseguito in automatico, si invia da terminale il comando:

```
systemctl status mqtt_broker
```

e si controlla se lo stato di **Active** sia **active (running)**, come da figura sottostante.



```
brugia@raspberrypi:~$ systemctl status mqtt_broker  
● mqtt_broker.service - mqtt_broker  
   Loaded: loaded (/lib/systemd/system/mqtt_broker.service; enabled; preset: >  
   Active: active (running) since Fri 2024-07-05 12:10:52 CEST; 1min 56s ago  
     Main PID: 770 (python3)  
       Tasks: 4 (limit: 8734)  
          CPU: 3.402s  
       CGroup: /system.slice/mqtt_broker.service  
               └─770 /usr/bin/python3 /home/brugia/Desktop/mqtt_script.py  
  
lug 05 12:10:52 raspberrypi systemd[1]: Started mqtt_broker.service - mqtt_brok>  
lug 05 12:11:00 raspberrypi python3[770]: /home/brugia/.local/lib/python3.11/si>  
lug 05 12:11:00 raspberrypi python3[770]: warnings.warn("Unable to import Axe>  
lines 1-12/12 (END)
```

Figura 10: Stato MQTT Broker

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);
```

Listing 1: Creazione Task

che prendono come parametri:

- Puntatore alla funzione – il nome del task
- Descrizione del task
- Dimensione stack
- Parametri stack
- Priorità: possono essere assegnati valori da 0 a

```
(configMAX_PRIORITIES - 1).
```

Quest'ultimo viene definito all'interno del file di configurazione **FreeRTOSConfig.h**. Per task che hanno pari priorità, come nel nostro caso, viene assegnato valore 1. Valori bassi di priorità indicano attività a bassa priorità.

- Possibilità di accedere al task dall'esterno
- I valori 0 o 1 indicano il numero di indice della CPU a cui l'attività deve essere assegnata

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.

Per essere sicuri che la data e l'ora rilevata sia quella reale è stata inclusa la libreria

```
NTPClient
```

3 Sviluppo Progetto

che consente la sincronizzazione della data e dell'ora da un server NTP.

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

```
void setup() {  
    Serial.begin(115200);  
    connectToWiFi();  
4    client.setServer(mqttBroker, mqttPort);  
  
    if (!bmp.begin()) {  
        Serial.println("Impossibile trovare un sensore valido BMP280, controlla  
        il cablaggio!");  
        while (1);  
9    }  
  
    if (!lightSensor.begin()) {  
        Serial.println("Impossibile trovare un sensore valido BH1750, controlla  
        il cablaggio!");  
        while (1);  
14    }  
  
    if (!rtc.begin()) {  
        Serial.println("Impossibile trovare un sensore valido RTC, controlla il  
        cablaggio!");  
        while (1);  
19    }  
  
    syncTimeWithNTP();  
}
```

Listing 2: Rilevazione sensori e sincronizzazione data e ora

Ogni sensore viene però prima inizializzato richiamando l'apposita libreria:

```
Adafruit_BMP280 bmp;  
2 BH1750 lightSensor;  
RTC_PCF8523 rtc;  
WiFiUDP ntpUDP;  
NTPClient timeClient(ntpUDP, "pool.ntp.org", 3600, 60000);
```

Listing 3: Inizializzazione sensori

Viene definito un NTP Client per richiedere la data e l'ora ad un server NTP, così da avere sempre data e ora aggiornate. Sono stati specificati anche l'offset e l'intervallo di aggiornamento, espresso in millisecondi.

```
1 WiFiUDP ntpUDP;  
NTPClient timeClient(ntpUDP, "pool.ntp.org", 3600, 60000);
```

Listing 4: NTP Client

e viene inizializzato l'NTP Client all'interno del **void loop()**:

```
1    timeClient.begin();
```

Dopodiché nel setup viene richiamata la funzione

```
    syncTimeWithNTP();
```

che verifica se la data e l'ora del Client corrisponde con quella del server NTP. Se non corrisponde la aggiorna con la corrispondenza della data epocale rilevata dalla funzione

```
    getEpochTime();
```


3 Sviluppo Progetto

La funzione sopra citata restituisce il tempo trascorso dall'anno in cui fu introdotto il GMT. Viene utilizzata la funzione

```
rtc.adjust(DateTime(epochTime))
```

per sincronizzare la data dell'RTC con data e ora rilevata dal server NTP.

```
void syncTimeWithNTP() {  
    while (!timeClient.update()) {  
        timeClient.forceUpdate();  
    }  
  
    unsigned long epochTime = timeClient.getEpochTime();  
    rtc.adjust(DateTime(epochTime));  
}
```

Listing 5: Sincronizzazione RTC

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. Finché lo stato della connessione non risulta essere connesso, si ritenta la connessione.

```
void loop() {  
    if (!client.connected()) {  
        connectToMQTT();  
    }  
    client.loop();  
    handleTelegramMessages(); // Gestisci i messaggi in entrata di Telegram  
}  
  
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");  
    timeClient.begin();  
}
```

Listing 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)
```

Per semplificare la gestione del codice, vengono definite all'inizio le costanti e le variabili globali relative ai task, al WiFi, alla porta, all'indirizzo IP e ai topic MQTT.

```
TaskHandle_t bmpTaskHandle, bh1750TaskHandle, rtcTaskHandle;
WiFiClient espClient;
4 PubSubClient client(espClient);
const char *ssid = "TIM-63619464";
const char *password = "uc4DfzHJHd36CZFJ";
const char *mqttBroker = "192.168.1.12";
const int mqttPort = 1883;
9
// Costanti per i topic MQTT
const char *temperatureTopic = "temperatura";
const char *pressureTopic = "pressione";
const char *lightTopic = "luminosit ";
14 const char *dateTopic = "data";
const char *timeTopic = "ora";
```

Listing 7: Dichiarazione costanti e variabili globali

3.4 ESP32, MQTT Broker e MQTT Client

L'acquisizione delle grandezze fisiche richieste avviene mediante i sensori collegati alla EPS32 che funge da nodo IoT. Questo microcontrollore, grazie alla connessione WiFi integrata e al protocollo MQTT, invierà i dati acquisiti al broker.

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**, appositata per la messaggistica MQTT, che permette di ricevere e inviare messaggi.

All'interno del **setup** 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.

```
void setup() {
2   Serial.begin(115200);
   connectToWiFi();
   client.setServer(mqttBroker, mqttPort);
}
```

Listing 8: Connessione MQTT Broker

3 Sviluppo Progetto

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();  
    handleTelegramMessages(); // Gestisci i messaggi in entrata di Telegram  
}  
  
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");  
    timeClient.begin();  
}  
  
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);  
        }  
    }  
}
```

Listing 9: Connessione WiFi e MQTT Broker

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 dell'utente iscritto al bot Telegram.

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.

Per fare ciò è necessario sottoscrivere il client MQTT ai topic relativi alle misure rilevate dai vari sensori, richiamando nel **setup** la funzione

```
subscribeToTopics()
```

così da permettere la verifica della corrispondenza tra i topic caricati e quelli richiesti dall'utente.

```
void subscribeToTopics() {  
    if (client.connected()) {  
        client.subscribe(temperatureTopic);  
        client.subscribe(pressureTopic);  
        client.subscribe(lightTopic);  
        client.subscribe(dateTopic);  
        client.subscribe(timeTopic);  
    }  
}
```

Listing 10: Sottoscrizione topic

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;  
    TickType_t xLastWakeTime = xTaskGetTickCount();  
  
    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
```

3 Sviluppo Progetto

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

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

24 // Riattiva il monitoraggio della soglia se la temperatura scende
    sotto la soglia
        if (!temperatureThresholdEnabled && temperature <=
        temperatureThreshold) {
            temperatureThresholdEnabled = true;
        }

29 } else {
    Serial.println("Fallito nell'inviare la temperatura al Broker MQTT");
}

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

39 vTaskDelayUntil(&xLastWakeTime, 10000 / portTICK_PERIOD_MS); // Ritardo
    di 10 secondi
}
}
```

Listing 11: Task lettura dati sensore 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.

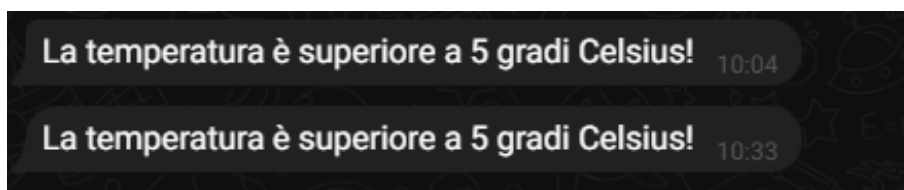


Figura 11: Alert superamento soglia temperatura

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.

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

void readBH1750Task(void *parameter) {
    (void)parameter;
    TickType_t xLastWakeTime = xTaskGetTickCount();

6    float lightThreshold = 500; //luminosit in Lumen

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

16        // Controllo della soglia solo se il monitoraggio della soglia
        abilitato
            if (lightThresholdEnabled && lux > lightThreshold) {
                sendTelegramMessage("La luminosit superiore a 500 lumen!");
                lightThresholdEnabled = false; // Disattiva il monitoraggio della
                soglia
21            }

            // Riattiva il monitoraggio della soglia se la luminosit scende
            sotto la soglia
            if (!lightThresholdEnabled && lux <= lightThreshold) {
                lightThresholdEnabled = true;

26            }
        } else {
            Serial.println("Fallito nell'inviare la luminosit al Broker MQTT");
        }

31    vTaskDelayUntil(&xLastWakeTime, 10000 / portTICK_PERIOD_MS); // Ritardo
    di 10 secondi
    }
}
```

Listing 12: Task lettura dati sensore BH1750

3.5.3 RTC

Come descritto precedentemente, per mantenere data e ora dell'RTC aggiornata, viene chiamata nel **setup** la funzione:

```
syncTimeWithNTP()
```

Nel task relativo all'RTC, vengono definiti due vettori per memorizzare rispettivamente le date e le ore 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, visualizzando, in caso di avvenuto invio, la data e l'ora inviata, altrimenti un messaggio di errore.

```
void readRTCTask(void *parameter) {
    (void)parameter;
    TickType_t xLastWakeTime = xTaskGetTickCount();

4   while (1) {
        // Leggi l'ora corrente dal RTC
        DateTime now = rtc.now();
        char dateBuffer[11];
9       snprintf(dateBuffer, sizeof(dateBuffer), "%04d/%02d/%02d", now.year(),
            now.month(), now.day());
        char timeBuffer[9];
        snprintf(timeBuffer, sizeof(timeBuffer), "%02d:%02d:%02d", now.hour(),
            now.minute(), now.second());

        // Pubblica i dati del RTC su MQTT
14      if (client.publish(dateTopic, dateBuffer)) {
            Serial.print("Data inviata al Broker MQTT: ");
            Serial.println(dateBuffer);
        } else {
            Serial.println("Fallito nell'inviare la data al Broker MQTT");
19      }

        if (client.publish(timeTopic, timeBuffer)) {
            Serial.print("Ora inviata al Broker MQTT: ");
            Serial.println(timeBuffer);
24      } else {
            Serial.println("Fallito nell'inviare l'ora al Broker MQTT");
        }

        vTaskDelayUntil(&xLastWakeTime, 10000 / portTICK_PERIOD_MS); // Ritardo
        di 10 secondi
29    }
}
```

Listing 13: Task lettura data e ora sensore RTC

La rilevazione dei dati dai sensori avviene ogni 10 secondi:

```
1 vTaskDelayUntil(&xLastWakeTime, 10000 / portTICK_PERIOD_MS);
```

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. Più utenti possono iscriversi al Bot Telegram, inserendo il token identificativo. Ogni utente ha un proprio chat ID, che verrà memorizzato all'interno del vettore **chatIDs**.

```
const char *telegramBotToken = "6728633709:
    AAFXKIkfqvrAS2ub1CwPKIJ5PIdrKqdgEps";
std::vector<String> chatIDs; // Vettore per memorizzare gli ID chat
```

Listing 14: Codice identificativo bot Telegram

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 è stata inclusa la libreria **HTTPClient**, che permette di creare con facilità richieste HTTP a un server web. Con l'ausilio di questa libreria, viene implementata la funzione **sendTelegramMessage** per l'invio dei messaggi.

L'ESP32 invia una richiesta HTTP alle API di Telegram, di cui abbiamo specificato l'url, le quali restituiscono una risposta al client (ESP32).

Per instaurare la comunicazione viene utilizzata la funzione

```
http.begin(url)
```

dove l'**url** è stato ottenuto combinando la parte base dell'URL delle API di Telegram, il token del bot Telegram, il chat ID ed il messaggio inviato dall'utente e tramite il **GET()** viene inviata la richiesta di dati di una risorsa specifica. Se la richiesta restituisce valore positivo, viene visualizzata la conferma dell'invio, altrimenti viene mostrato il messaggio di errore.

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



```
19 }  
}
```

Listing 15: Funzione invio messaggi Telegram

Per la gestione dei messaggi in arrivo di Telegram è stata sviluppata la funzione

```
handleTelegramMessages()
```

Prima di tutto si definisce un URL combinando la parte base dell'URL delle API di Telegram, il token del bot Telegram ed il percorso `/getUpdates`. Dopo aver avviato la connessione HTTP, viene inviata una richiesta con il `GET()` e se la risposta è positiva, viene memorizzata all'interno della stringa `response`.

Dopodiché si crea:

- **Documento JSON:** in cui verranno memorizzate le risposte alle richieste effettuate, dopo averle analizzate.

```
deserializeJson(doc, response);
```

- **Vettore JSON:** utilizzato per convertire l'array `result` dalla risposta JSON.

```
JsonArray updates = doc["result"].as<JsonArray>();
```

Per ogni oggetto appartenente al vettore sopra definito, viene estratto l'ID della chat che verrà aggiunto, se non presente, all'elenco degli ID, mediante la funzione:

```
addChatID(chatID)
```

```
void handleTelegramMessages() {  
    HTTPClient http;  
    String url = "https://api.telegram.org/bot" + String(telegramBotToken) + "  
    /getUpdates";  
4    http.begin(url);  
    int httpResponseCode = http.GET();  
    if (httpResponseCode > 0) {  
        String response = http.getString();  
        Serial.println(response);  
9  
        DynamicJsonDocument doc(1024);  
        deserializeJson(doc, response);  
        JsonArray updates = doc["result"].as<JsonArray>();  
14  
        for (JsonObject update : updates) {  
            String chatID = update["message"]["chat"]["id"].as<String>();  
            addChatID(chatID); // Aggiungi l'ID chat all'elenco  
        }  
    } else {  
19    Serial.print("Errore nell'ottenere gli aggiornamenti da Telegram. Error  
    code: ");  
        Serial.println(httpResponseCode);  
    }  
    http.end();  
}
```

Listing 16: Funzione ricezione messaggi Telegram

```
1 void addChatID(const String& chatID) {  
    if (std::find(chatIDs.begin(), chatIDs.end(), chatID) == chatIDs.end()) {  
        chatIDs.push_back(chatID);  
        Serial.print("Aggiunto nuovo chatID: ");  
        Serial.println(chatID);  
6    }  
}
```

Listing 17: Memorizzazione chat ID

L'utente potrà inviare sulla chat i 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.

Entrambe le funzionalità verranno spiegate nel paragrafo successivo.

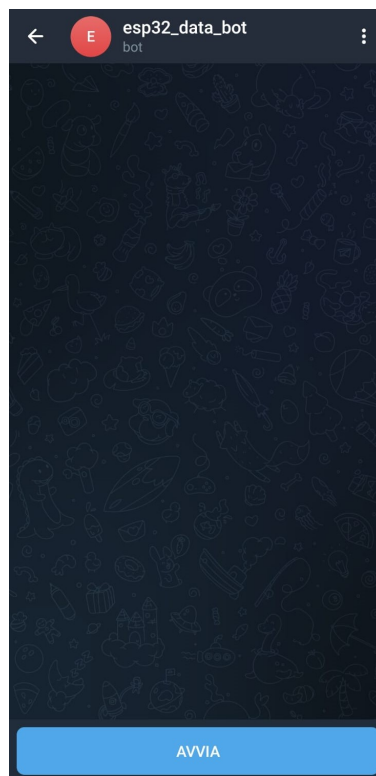


Figura 12: Bot Telegram

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.

3 Sviluppo Progetto

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

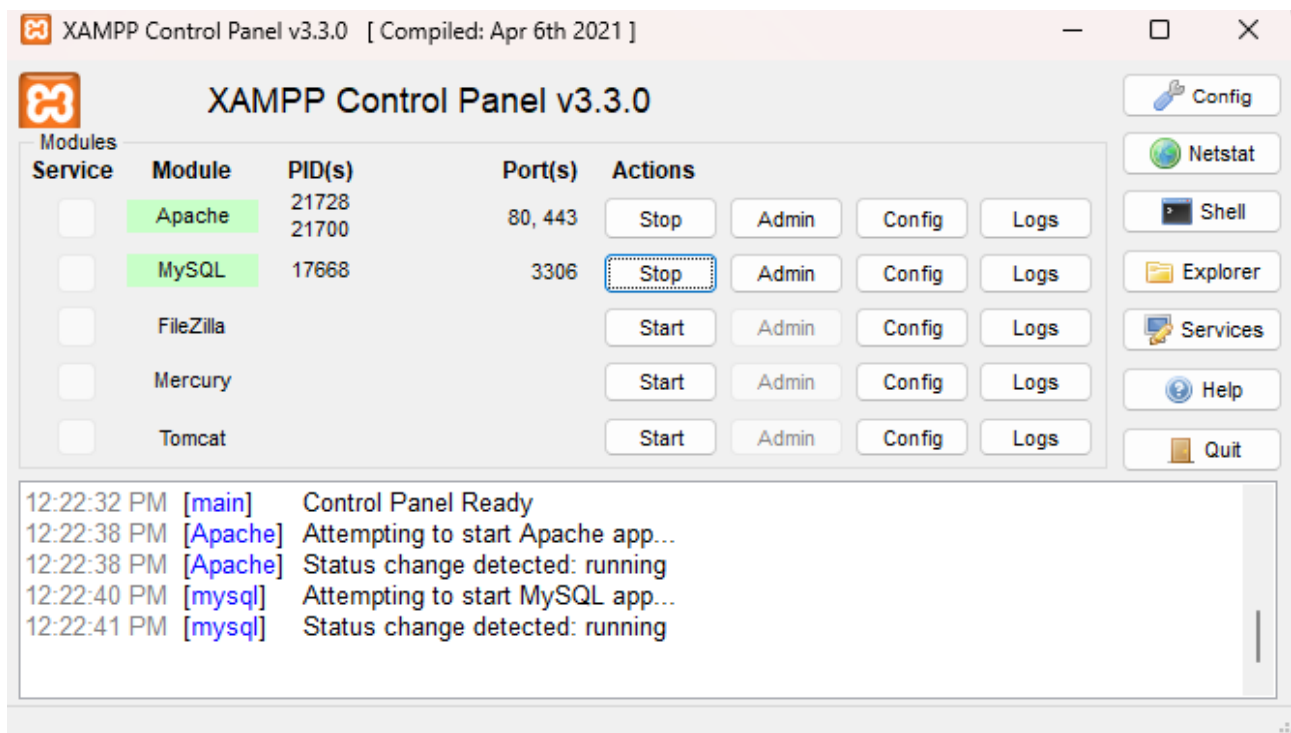


Figura 13: GUI Xampp

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

```
broker_address = '192.168.1.12' # Indirizzo del broker MQTT
2 db_config = {
    'host': '192.168.1.10', # Indirizzo del server MySQL
    'user': 'brugia', # Nome utente del database MySQL
    'password': 'halo3000', # Password del database MySQL
    'database': 'esp32_data' # Nome del database MySQL
7 }
```

Listing 18: Dati configurazione database

```
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()
print("Connessione al database MySQL avvenuta con successo")
```

Listing 19: Connessione al database

Per poter caricare i dati rilevati all'interno del database è necessario utilizzare una funzione apposita per sottoscrivere un client MQTT ai topic necessari. Prima di tutto, con la funzione

```
sniff_mqtt_messages()
```

si crea un nuovo client MQTT, si imposta la funzione **on_message** come callback per gestire i messaggi MQTT ricevuti, si connette il client al broker, lo si iscrive ai topics definiti all'interno dell'array

```
topics = ["temperatura", "pressione", "luminosit ", "data", "ora"]
```

e si avvia il loop di rete per ricevere i messaggi.

La funzione **on_message** viene quindi chiamata ogni volta che si riceve un messaggio su uno dei topics definiti. Nel dettaglio, la funzione **on_message** decodifica il payload del messaggio e verifica la corrispondenza tra il topic inserito dal client con quelli presenti 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()**.

```
def on_message(client, userdata, message):
    topic = message.topic
    payload = message.payload.decode('utf-8')
4
    # Verifica se il topic presente nel mapping
    if topic in topics:
        # Prepara la query SQL per l'inserimento
        query = f"INSERT INTO sensor_data (temperatura, pressione,
9         luminosit , data, ora) VALUES (%s, %s, %s, %s, %s)"

        # Esegui l'inserimento dei dati
        cursor.execute(query, (payload, payload, payload, payload, payload))
        conn.commit()
        print(f"Dato inserito correttamente per il topic {topic}: {payload
14     }")
    else:
        print(f"Percorso del messaggio non trovato nel mapping: {topic}")

# Funzione per configurare e avviare il client MQTT
def sniff_mqtt_messages(broker_address, topics):
19     client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    client.on_message = on_message
```

```
24 # Connessione al broker MQTT
    client.connect(broker_address)
    print("Connessione al broker MQTT avvenuta con successo")

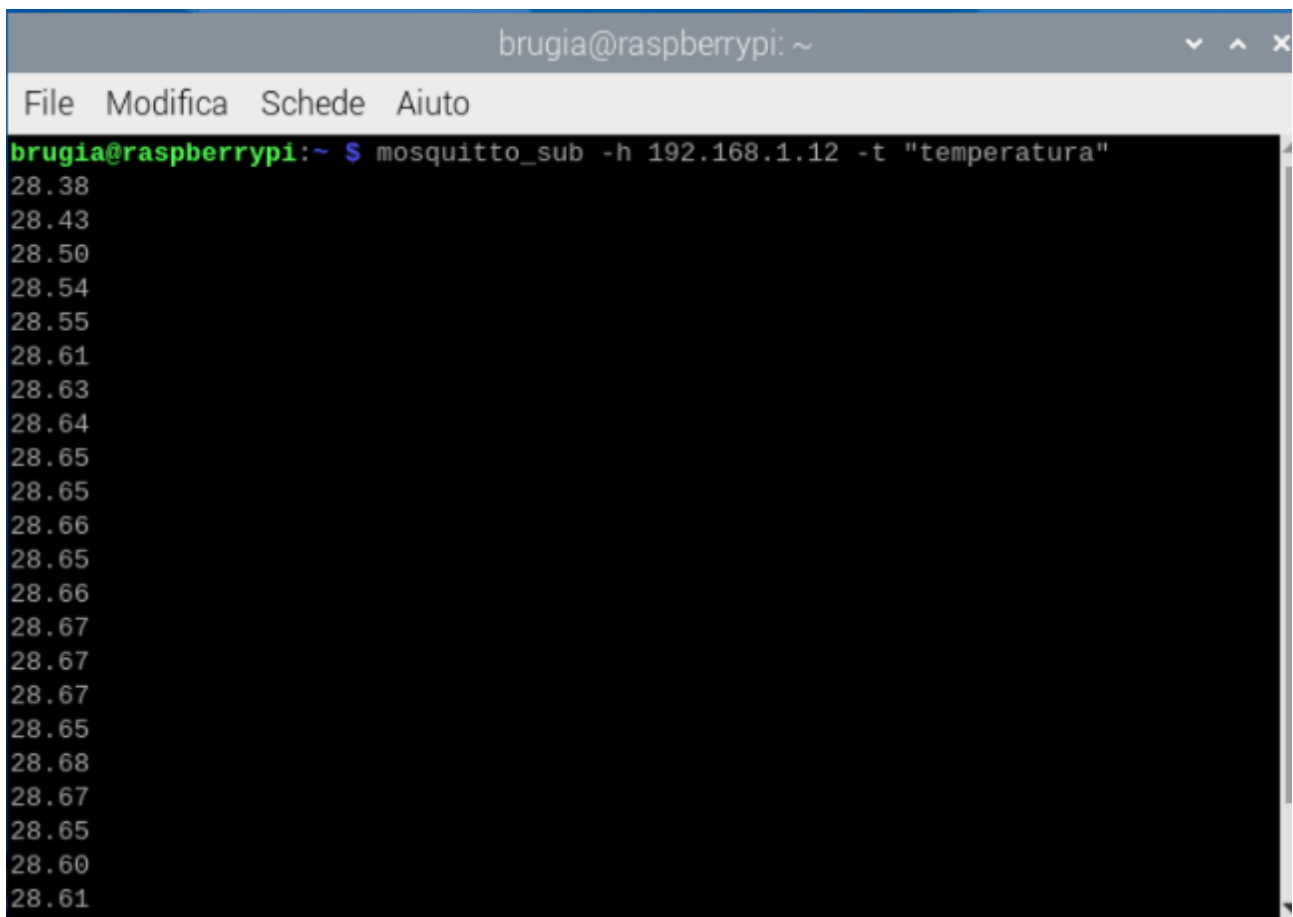
# Sottoscrizione ai topic
29 for topic in topics:
    client.subscribe(topic)
    print(f"Sottoscritto al topic: {topic}")

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

Listing 20: Configurazione e avvio Client MQTT

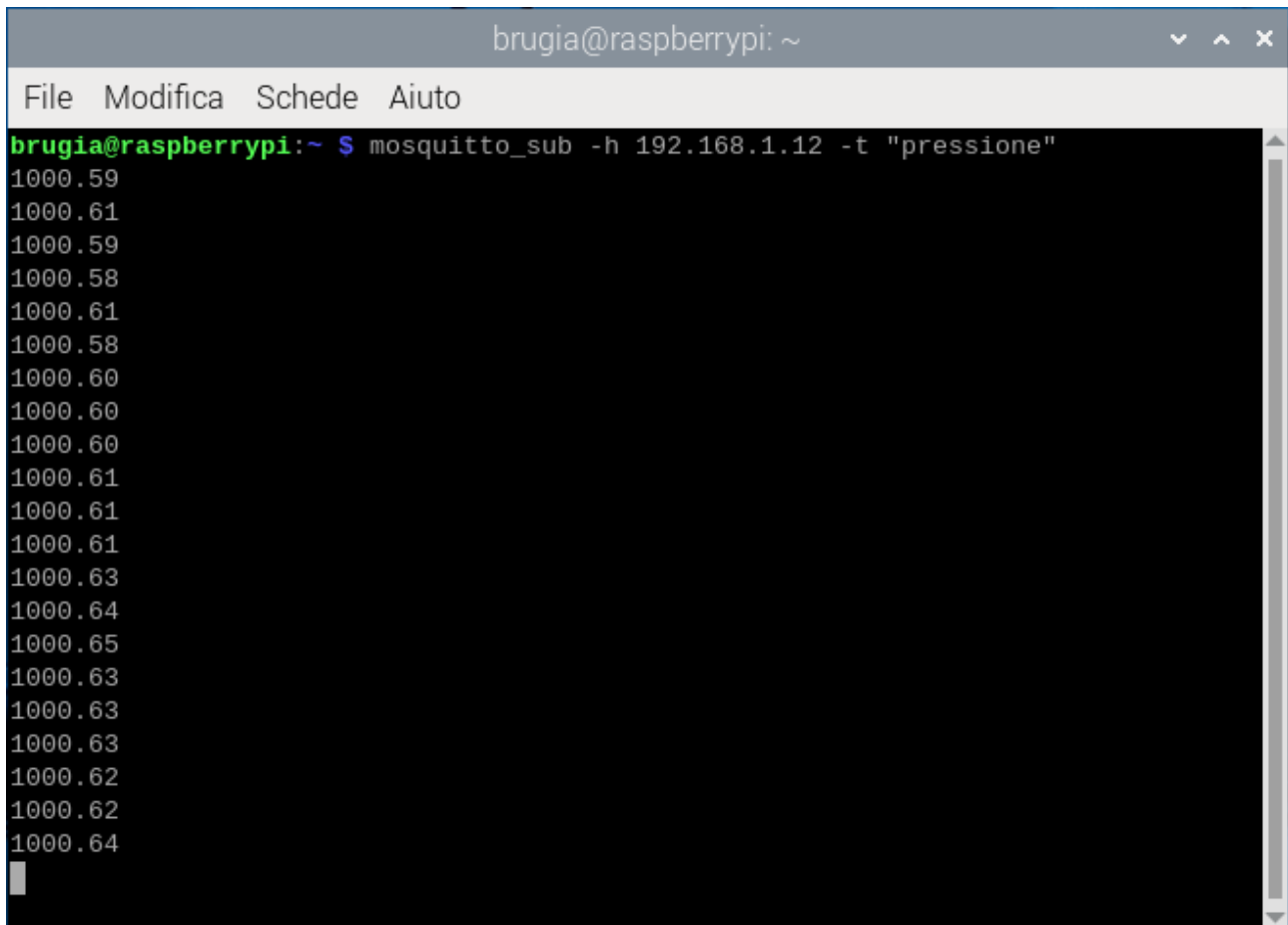
Per controllare se il broker riceve i dati correttamente basta utilizzare il comando

```
1 mosquitto\_sub -h "ip\_broker" -t "nome\_topic"
```



```
brugia@raspberrypi: ~
File Modifica Schede Aiuto
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 14: Temperatura inviata al broker



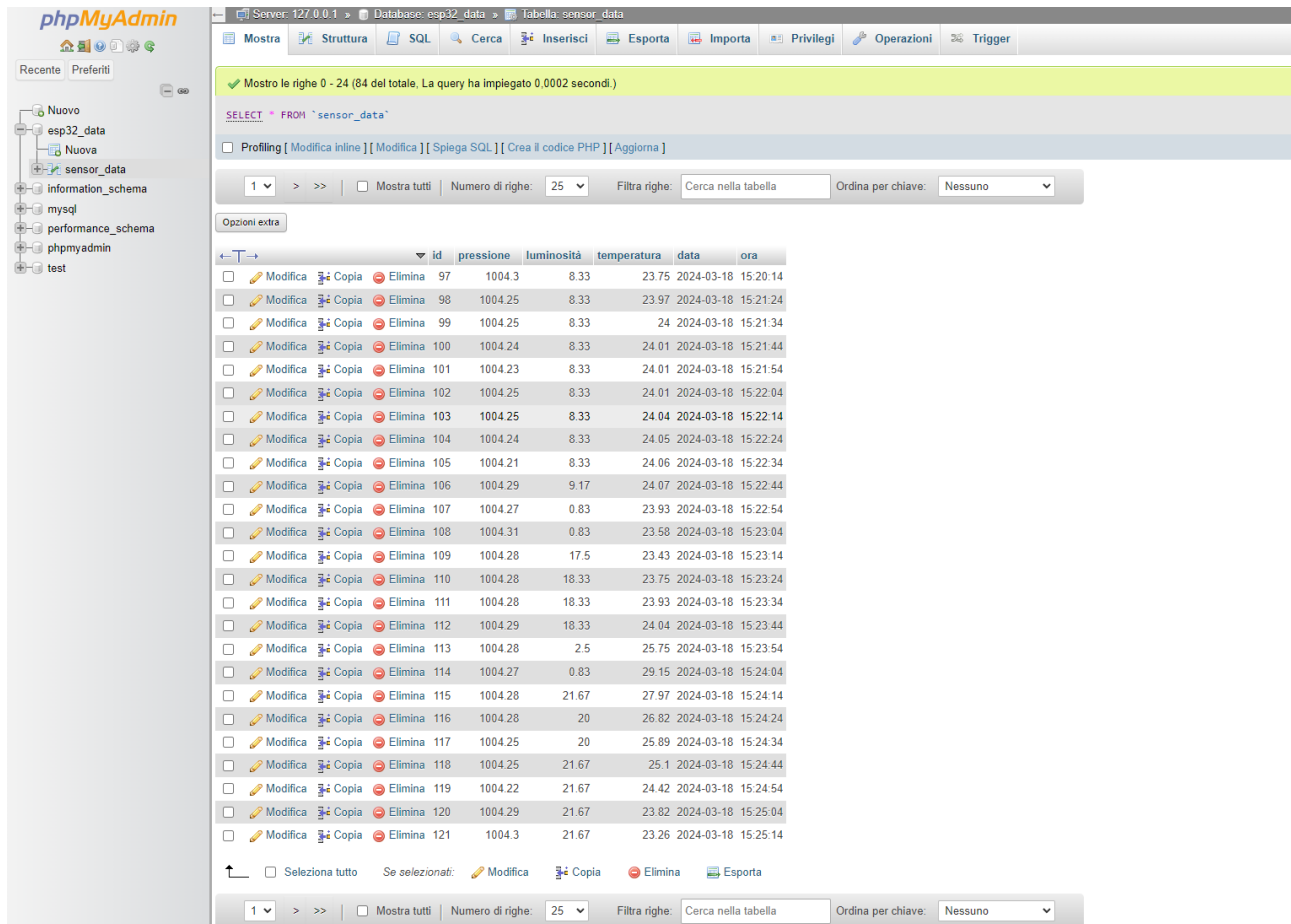
A terminal window titled 'brugia@raspberrypi: ~' with a menu bar containing 'File', 'Modifica', 'Schede', and 'Aiuto'. The terminal shows the command 'mosquitto_sub -h 192.168.1.12 -t "pressione"' being executed. Below the command, a list of 20 numerical values representing pressure is displayed, each on a new line. The values range from 1000.58 to 1000.65. 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 15: Pressione inviata al broker

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

3 Sviluppo Progetto



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

Figura 16: Database

Inviando sulla chat Telegram il comando `/start`, verrà visualizzato il messaggio di benvenuto con la richiesta di inserimento informazioni per ottenere i dati.

```
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /start is issued."""
    user = update.effective_user
    await update.message.reply_html(
        rf"Ciao {user.mention_html()}, inserisci read+data+topic per
        ottenere i dati!",
        reply_markup=ForceReply(selective=True),
    )
```

Listing 21: Comando start

Inviando nella chat il comando `/leggi`, seguita da data, ora e topic, l'utente otterrà come risposta l'elenco dei dati memorizzati nel database o nel caso in cui il formato inserito non sia valido, verrà inviato all'utente un messaggio di errore comunicando il formato corretto da utilizzare per la richiesta.

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 dall'utente, 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

3 Sviluppo Progetto

generato un messaggio di errore oppure verrà indicato che non esiste alcun dato nella data e ora richiesti.

```
1 async def leggi(update, context):
    user = update.effective_user

    if len(context.args) < 2:
        await context.bot.send_message(chat_id=user.id, text="Devi
specificare una data e un'ora nel formato 'YYYY-MM-DD HH:MM:SS' e un
topic.")
6         return

    print("Valore di context.args[0]:", context.args[0])
    print("Valore di context.args[1]:", context.args[1])

11    try:
        datetime_str = context.args[0] + ' ' + context.args[1]
        datetime_filter = datetime.strptime(datetime_str, '%Y-%m-%d %H:%M:%S
',)

        query = "SELECT * FROM sensor_data WHERE DATE(data) >= %s AND TIME(
ora) >= %s"
16        cursor.execute(query, (datetime_filter.date(), datetime_filter.time
()))
        results = cursor.fetchall()

        if results:
            # Verifica il tipo di dato del primo elemento per determinare il
formato
21            if isinstance(results[0][0], int):
                # Se il timestamp un intero (UNIX timestamp)
                timestamps = [datetime.fromtimestamp(row[0]) for row in
results]
            elif isinstance(results[0][0], str):
                # Se il timestamp una stringa
26                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 n un
intero n una stringa")

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

Listing 22: Comando leggi con estrazione dati

```
2         if results:
            for row in results:
                message = f"Pressione [Pa]: {row[1]}, Luminosit [Lumen
]: {row[2]}, Temperatura[C]: {row[3]}, Data: {row[4]}, ora: {row[5]}"
                await context.bot.send_message(chat_id=user.id, text=
message)
            else:
                await context.bot.send_message(chat_id=user.id, text="Nessun
valore trovato per la data e l'ora specificate.")
7        except ValueError:
            await context.bot.send_message(chat_id=user.id, text="Formato data e
ora non valido. Utilizza 'YYYY-MM-DD HH:MM:SS'.")
```

Listing 23: Comando leggi con estrazione dati

Pressione [Pa]: 999.54, Luminosità [Lumen]: 30.83, Temperatura[C]: 22.43, Data: 2024-03-18, ora: 18:09:55	11:22
Pressione [Pa]: 999.53, Luminosità [Lumen]: 30.83, Temperatura[C]: 22.4, Data: 2024-03-18, ora: 18:10:05	11:22
Pressione [Pa]: 999.5, Luminosità [Lumen]: 30.83, Temperatura[C]: 22.39, Data: 2024-03-18, ora: 18:10:15	11:22
Pressione [Pa]: 999.53, Luminosità [Lumen]: 30.83, Temperatura[C]: 22.37, Data: 2024-03-18, ora: 18:10:25	11:22
Pressione [Pa]: 999.58, Luminosità [Lumen]: 27.5, Temperatura[C]: 22.36, Data: 2024-03-18, ora: 18:10:35	11:23
Pressione [Pa]: 999.52, Luminosità [Lumen]: 25.83, Temperatura[C]: 22.32, Data: 2024-03-18, ora: 18:10:45	11:23
Pressione [Pa]: 999.53, Luminosità [Lumen]: 25.0, Temperatura[C]: 22.34, Data: 2024-03-18, ora: 18:10:55	11:23
Pressione [Pa]: 999.54, Luminosità [Lumen]: 25.0, Temperatura[C]: 22.32, Data: 2024-03-18, ora: 18:11:45	11:23
Pressione [Pa]: 999.52, Luminosità [Lumen]: 25.83, Temperatura[C]: 22.31, Data: 2024-03-18, ora: 18:11:55	11:23
Pressione [Pa]: 998.65, Luminosità [Lumen]: 27.5, Temperatura[C]: 22.17, Data: 2024-03-18, ora: 20:49:01	11:23
Pressione [Pa]: 998.67, Luminosità [Lumen]: 26.67, Temperatura[C]: 22.18, Data: 2024-03-18, ora: 20:49:11	11:23
Pressione [Pa]: 998.63, Luminosità [Lumen]: 26.67, Temperatura[C]: 22.91, Data: 2024-03-18, ora: 21:03:31	11:23
Pressione [Pa]: 998.64, Luminosità [Lumen]: 27.5, Temperatura[C]: 22.9, Data: 2024-03-18, ora: 21:03:41	11:23
Pressione [Pa]: 998.65, Luminosità [Lumen]: 28.33, Temperatura[C]: 22.81, Data: 2024-03-18, ora: 21:04:41	11:23
Pressione [Pa]: 998.61, Luminosità [Lumen]: 27.5, Temperatura[C]: 23.2, Data: 2024-03-18, ora: 21:12:41	11:23
Pressione [Pa]: 998.65, Luminosità [Lumen]: 27.5, Temperatura[C]: 23.22, Data: 2024-03-18, ora: 21:12:51	11:23
Pressione [Pa]: 998.61, Luminosità [Lumen]: 27.5, Temperatura[C]: 23.22, Data: 2024-03-18, ora: 21:13:01	11:23
Pressione [Pa]: 998.63, Luminosità [Lumen]: 27.5, Temperatura[C]: 23.1, Data: 2024-03-18, ora: 21:15:11	11:23

Figura 17: Comando Bot "leggi"



Figura 18: Messaggio di errore

3.8 Grafici

L'utente 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 per il comando **leggi** e si calcola per ogni misura il valore massimo, il valore minimo e la media dei valori, tramite le apposite funzioni **mean()**, **max()** e **min()**, del periodo selezionato.

Per ogni topic verrà realizzato un grafico, contenente i valori memorizzati nel database a partire dalla data e ora richiesta dall'utente, mostrando anche le statistiche calcolate.

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.

```
pressione_media = np.mean(pressione)
pressione_minimo = np.min(pressione)
pressione_massimo = np.max(pressione)

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

    temperatura_media = np.mean(temperatura)
10    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):
15        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}')
20
        plt.legend(loc='upper left')
        plt.xlabel('Data e Ora')
        plt.ylabel('Valore')
        plt.title(f'{parametro} nel Tempo')
25
        # 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')
30
        # Nascondi l'asse X
        plt.gca().xaxis.set_visible(False)

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

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

    for timestamps, valori, parametro, media, minimo, massimo,
nome_file in grafici:
45        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)
```

50

Listing 24: Calcolo statistiche

Di seguito si possono vedere degli esempi dei grafici ottenuti:

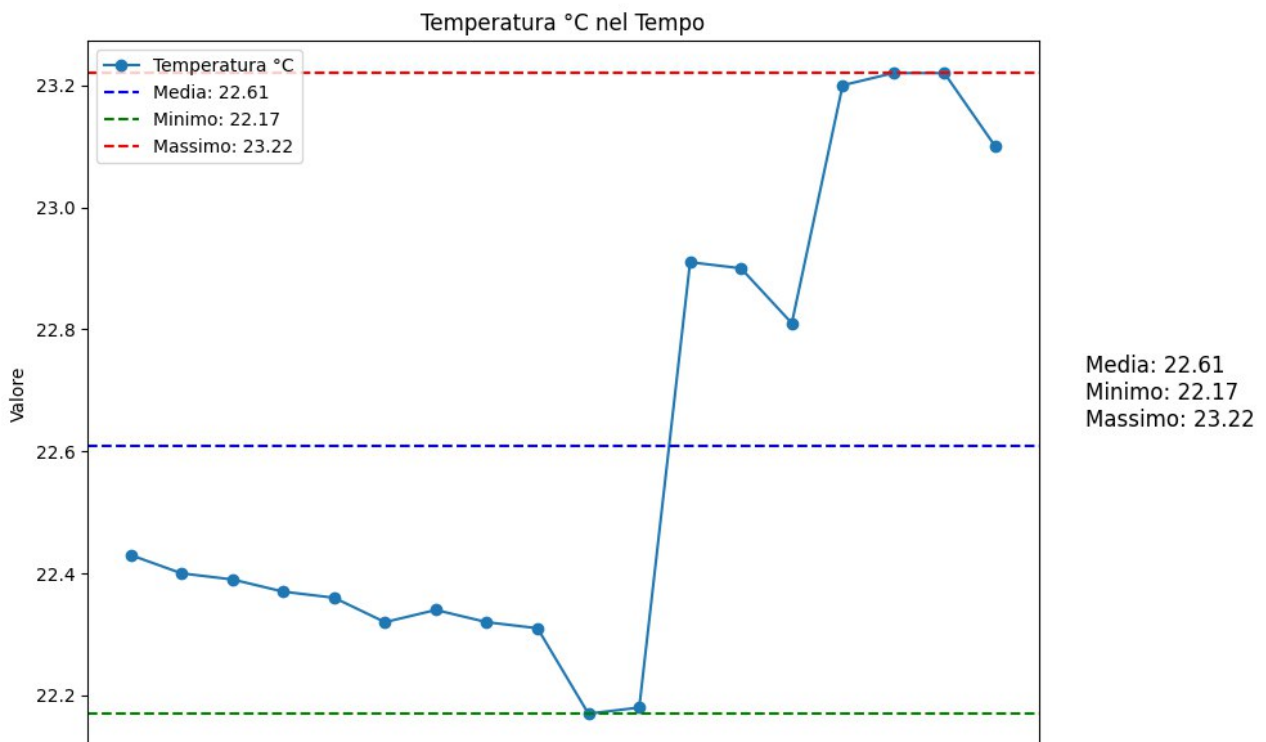


Figura 19: Grafico andamento temperatura

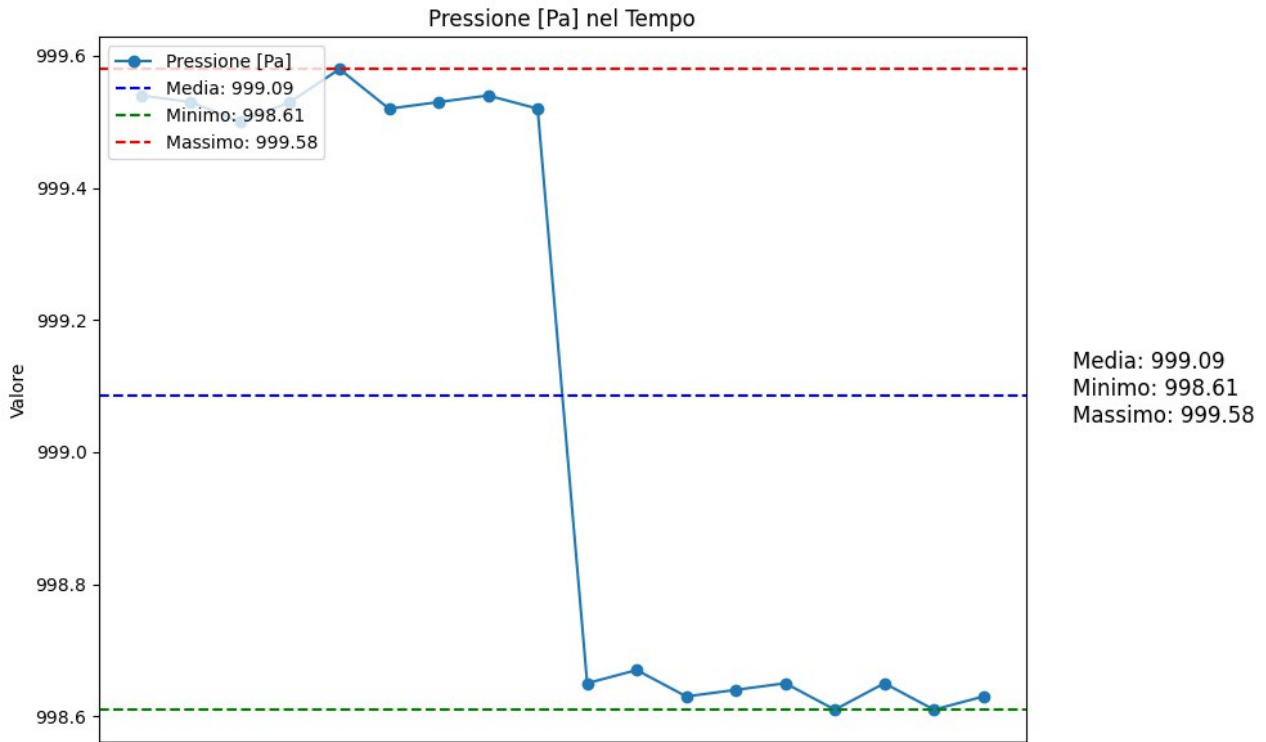


Figura 20: Grafico andamento pressione

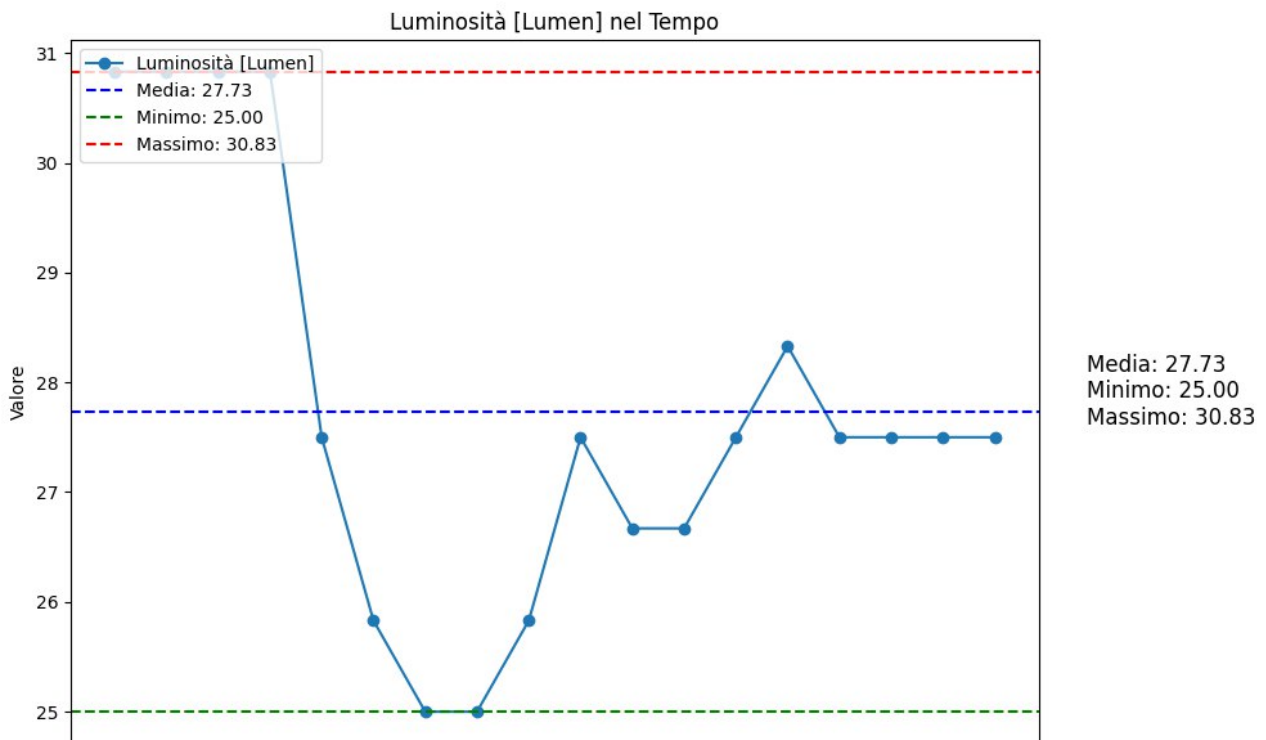


Figura 21: Grafico andamento luminosità

Per poter gestire correttamente tutte le funzionalità, all'interno del **main** devono essere richiamate le funzioni per la gestione dei comandi, per l'avvio del bot e per l'esecuzione del client

3 Sviluppo Progetto

MQTT in un thread separato.

Quindi, prima di tutto si crea l'applicazione Telegram tramite la funzione

```
application = Application.builder().token(TOKEN).build
```

a cui viene passato il token identificativo del bot.

Per consentire la gestione dei comandi `/start` e `/leggi` si utilizza la funzione

```
CommandHandler("start", start)
```

che prende come argomenti il comando e la funzione di callback.

Dopodiché si inizializza e si avvia il bot Telegram tramite:

```
run_polling()
```

Il client MQTT viene gestito in un thread separato, così da consentire contemporaneamente sia l'ascolto dei messaggi MQTT che la gestione dei comandi provenienti dal bot Telegram.

Viene quindi creato un nuovo thread per eseguire

```
run_mqtt_client()
```

il quale richiama

```
sniff_mqtt_messages()
```

per gestire il client MQTT indipendentemente dal bot.

Tramite

```
mqtt_thread.start()
```

si avvia l'esecuzione del thread, che verrà eseguito in parallelo al thread principale.

Richiamando la funzione `main()` viene avviata l'applicazione del bot Telegram.

```
def main():
    tracemalloc.start()

    application = Application.builder().token(TOKEN).build()

    # Gestione del comando /start
    application.add_handler(CommandHandler("start", start))

    # Gestione del comando read
    application.add_handler(CommandHandler("leggi", leggi))

    # Avvia il bot
    application.run_polling(allowed_updates=Update.ALL_TYPES)

# Esegui il client MQTT in un thread separato
def run_mqtt_client():
    sniff_mqtt_messages(broker_address, topics)

19 if __name__ == '__main__':
    mqtt_thread = threading.Thread(target=run_mqtt_client)
    mqtt_thread.start()
    main()
```

Listing 25: Definizione istruzioni main

4 Considerazioni finali

Durante lo sviluppo del progetto non abbiamo riscontrato difficoltà rilevanti.

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

Per permettere all'utente di inviare richieste dal bot Telegram, avevamo previsto l'utilizzo della libreria UniversalBotTelegram, ma in fase di test, nonostante la richiesta venisse inviata correttamente alle API di Telegram, non riuscivamo ad ottenere una risposta.

Per questo motivo, è stata utilizzata la libreria HTTPClient.

Si è scelto di implementare il database su una macchina Windows, in quanto sul Raspberry abbiamo riscontrato problemi con i permessi Root.

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 all'utente 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.