



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Progetto 3

Candidato:

Cichella Lorenzo
Antonini Adelio
Garzarella Fiore
Gioiello Flavia

Relatore:

Prof. Marcozzi Daniele

Anno Accademico 2022-2023

Indice

1	Introduzione	1
2	Materiali e metodi	2
2.1	Componenti Hardware	2
2.1.1	ESP32	2
2.1.2	Raspberry Pi 4	4
2.1.3	Sensore BMP280	5
2.1.4	Display OLED	6
2.1.5	Ventola Noctua NF-A4x10	7
2.1.6	Modulo RTC	8
2.2	Componenti Software	9
2.2.1	Arduino IDE	9
2.2.2	Visual Studio Code	10
2.2.3	Raspbian GNU/Linux 11	10
2.2.4	FreeRTOS (Real-Time Operating System)	11
2.3	Dettagli del funzionamento	12
3	Configurazione e setup	13
3.1	Collegamento del Sensore BMP280	13
3.2	Collegamento del display OLED	14
3.3	Collegamento del modulo RTC	14
3.4	Collegamento ventola NOCTUA NF-A4x10 5V PWM	15
3.5	Setup Raspberry Pi	15
3.6	Setup ESP32	16
3.7	Verifica del corretto cablaggio	17
3.8	Avvio e configurazione della comunicazione tra l'ESP32 e la Raspberry Pi 4	18
4	Script e Programmazione dei Dispositivi	21
4.1	Librerie in Arduino	21
4.2	Implementazione software scheda ESP32	21
4.2.1	Task ventola	22
4.2.2	Task Sensore e Display OLED	23
4.2.3	Task invio file JSON	25
4.2.4	Task lettura RPM	27
4.2.5	Task connessione MQTT	28

Indice

4.3	Installazione pacchetti Raspberry Pi 4	30
4.4	Implementazione software Raspberry Pi 4	31
4.4.1	Codice app.py	31
4.4.2	Codice inviohtml.py	32
4.4.3	Codice index.html	35
4.4.4	Sincronizzazione orario tra Raspberry Pi 4 e ESP32	36
5	Risultati	40
6	Conclusioni e sviluppi futuri	45

Elenco delle figure

2.1	ESP32 DevKitC V4	3
2.2	ESP32 DevKitC V4 Pinout	3
2.3	Raspberry Pi 4 Model B	4
2.4	Sensore BMP280	5
2.5	Schermo Oled	6
2.6	Ventola Noctua NF-A4x10 5V PWM	7
2.7	Modulo RTC	8
2.8	Logo Arduino IDE	9
2.9	Logo Visual Studio Code	10
2.10	Logo Raspberry Pi OS	11
2.11	Logo FreeRTOS	11
2.12	Schema di funzionamento del sistema	12
3.1	Configurazione finale del sistema, con i collegamenti tra ESP32, il sensore BMP280, lo schermo OLED, il modulo RTC e la ventola	13
3.2	Update e Upgrade RPi	16
3.3	Script per effettuare lo scanner dei dispositivi I2C	17
3.4	Scanner I2C	18
3.5	Sezione per la configurazione WiFi	18
3.6	Sezione per la configurazione dell'indirizzo della RPi	19
3.7	Sezione per la configurazione dell'indirizzo del broker	19
4.1	Codice funzione taskFanControl	23
4.2	Codice funzione taskSensors	24
4.3	Codice funzione leggiBmp280	24
4.4	Codice funzione JSONPublish	26
4.5	Codice funzione JSONPublish	26
4.6	Codice funzione readRPMTTask	28
4.7	Codice funzione taskWiFi_MQTT	29
4.8	Codice funzione callback	29
4.9	Codice funzione reconnect	30
4.10	Script app.py	32
4.11	Script inviohtmlv2.0.py	34
4.12	Script pagina HTML	35
4.13	File di configurazione	36
4.14	Configurazione NTP	36

Elenco delle figure

4.15 Script orario.py	37
4.16 Servizio orario.service	38
4.17 Abilitazione e avvio servizio	38
4.18 Verifica stato sevizio	38
4.19 Stato del sevizio	39
5.1 Monitor esp32	40
5.2 Monitor esp32	41
5.3 Server Flask	41
5.4 Client MQTT	42
5.5 Pagina Web	43
5.6 Pagina Web da smartphone	44

Capitolo 1

Introduzione

Il seguente lavoro si concentra sull'implementazione di un sistema IoT completo che combina componenti hardware e software per acquisire, elaborare e condividere dati ambientali in tempo reale.

Il sistema è costituito da due componenti chiave: la Raspberry Pi 4 e l'ESP32, che operano in modo collaborativo. Questi due elementi, insieme a vari altri dispositivi, tra cui il sensore di temperatura e pressione BMP280, una ventola controllata tramite segnale PWM, un display OLED e un modulo RTC, si integrano sinergicamente per formare un sistema complesso.

L'obiettivo principale di questo progetto è, quindi, quello di creare un sistema che sia in grado di effettuare un monitoraggio continuo dei parametri ambientali di temperatura e pressione, di rispondere alle loro variazioni in tempo reale, controllando la velocità della ventola e, infine, di condividere queste informazioni in modo efficiente attraverso un protocollo MQTT alla RPi, che ha il compito di pubblicarle su una pagina web.

Nel corso di questa relazione, verrà analizzato in dettaglio il funzionamento di ciascun componente e il suo ruolo nel sistema generale. Inoltre, verranno presentati i risultati e le sfide affrontate durante il processo di sviluppo.

Capitolo 2

Materiali e metodi

2.1 Componenti Hardware

Di seguito sono riportati nel dettaglio i componenti hardware utilizzati nel progetto e il loro funzionamento:

- **ESP32:** si occupa dell'acquisizione dei dati dal sensore BMP280, del controllo della ventola tramite segnale PWM e della visualizzazione dei dati su un display OLED. Inoltre, funge da client MQTT e trasmette i dati alla Raspberry Pi 4.
- **Raspberry Pi 4:** agisce come il componente principale del sistema ed è responsabile di fungere da broker MQTT e visualizzatore web per i dati raccolti. Inoltre, coordina il funzionamento generale del sistema.
- **Sensore BMP280:** rileva i dati ambientali, tra cui temperatura e pressione atmosferica.
- **Ventola Noctua NF-A4x10 5V PWM:** è utilizzata per regolare la temperatura del sistema. La sua velocità è controllata mediante segnali PWM in proporzione alla temperatura rilevata.
- **Display OLED:** è utilizzato per visualizzare i dati rilevati, tra cui temperatura, pressione e velocità della ventola.
- **Modulo RTC (Real-Time Clock):** è utilizzato per la sincronizzazione del tempo all'interno del sistema e può essere comandato dalla Raspberry Pi 4 per assicurare che tutti i dispositivi siano sincronizzati.

2.1.1 ESP32

Nel progetto viene utilizzata la versione DevKitC V4 della scheda di sviluppo ESP32, in figura 2.1. Di seguito vengono riportate alcune specifiche sulla ESP32 DevKitC V4 [1]:



Figura 2.1: ESP32 DevKitC V4

- **Microcontrollore ESP32:** la scheda è dotata del microcontrollore ESP32, che è il cuore del sistema. L'ESP32 è basato su un processore dual-core Xtensa LX6 a 32 bit e offre una vasta gamma di funzionalità, tra cui connettività Wi-Fi, GPIO, UART, SPI e I2C.
- **Connettività Wi-Fi:** una delle caratteristiche distintive dell'ESP32 è la sua connettività Wi-Fi e Bluetooth integrata. Questo lo rende ideale per progetti IoT, automazione domestica, dispositivi indossabili e molte altre applicazioni.
- **GPIO e Pinout:** la scheda dispone di un set di pin GPIO accessibili (figura 2.2) che possono essere utilizzati per il collegamento di sensori, attuatori e altri componenti esterni. La disposizione dei pin è progettata per semplificare la prototipazione e la connessione di periferiche.

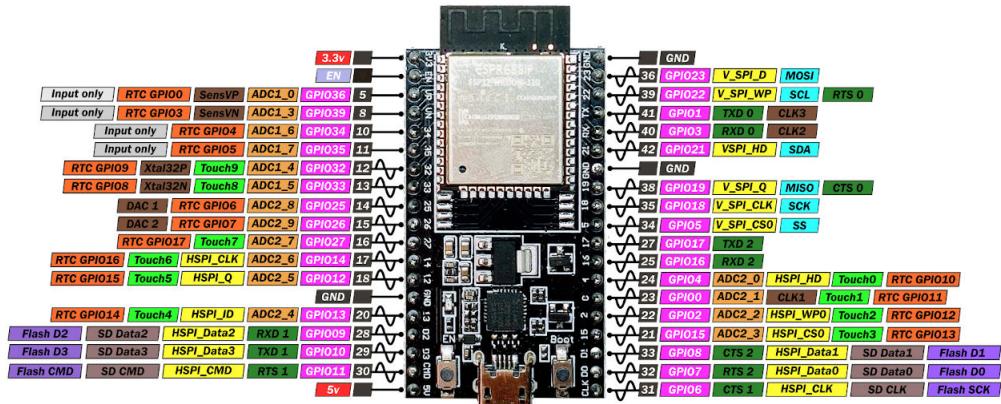


Figura 2.2: ESP32 DevKitC V4 Pinout

- **Interfaccia USB:** la ESP32 DevKitC V4 è dotata di un'interfaccia USB che può essere utilizzata per alimentare la scheda e per la programmazione. È possibile caricare il firmware e sviluppare applicazioni tramite un cavo USB.

- **LED di stato:** sono presenti LED di stato che possono essere utilizzati per scopi di debug o per indicare lo stato del sistema.
- **Compatibilità con Arduino:** la ESP32 DevKitC V4 è compatibile con l'IDE di sviluppo Arduino, il che semplifica la programmazione e lo sviluppo di applicazioni utilizzando l'ecosistema Arduino e la sua vasta libreria di supporto.
- **Compatibilità con FreeRTOS:** la ESP32 è compatibile con il sistema operativo in tempo reale FreeRTOS, che consente di eseguire task paralleli in modo efficiente.

2.1.2 Raspberry Pi 4

La Raspberry Pi 4 Model B è un computer a scheda singola (SBC), progettato e prodotto dalla Raspberry Pi Foundation, in figura 2.3. È la quarta generazione della serie di computer Raspberry Pi ed è stata rilasciata come successore della Raspberry Pi 3 Model B+. [2]



Figura 2.3: Raspberry Pi 4 Model B

Ecco alcune delle caratteristiche principali della Raspberry Pi 4 Model B:

- **Processore:** è equipaggiata con un processore quad-core ARM Cortex-A72 a 64 bit, che offre prestazioni notevolmente migliori rispetto ai modelli precedenti.
- **RAM:** dispone di 8 GB di RAM, quantità significativamente maggiore rispetto alle versioni precedenti della Raspberry Pi, consentendo di eseguire applicazioni più complesse e un multitasking più fluido.
- **Porte micro HDMI:** supporta il collegamento a due monitor tramite le sue due porte micro HDMI, consentendo la visualizzazione di contenuti in doppio schermo o esteso.
- **USB 3.0:** è dotata di due porte USB 3.0 ad alta velocità oltre a due porte USB 2.0, permettendo il collegamento di dispositivi USB ad elevate prestazioni.

- **Connettività di rete:** è dotata di una porta Ethernet Gigabit per una connessione di rete veloce e stabile, oltre al supporto per la connettività wireless Wi-Fi 802.11ac e Bluetooth 5.0 integrati.
- **GPIO (General Purpose Input/Output):** mantiene un layout di pin GPIO compatibile con le versioni precedenti della Raspberry Pi, che permette di interfacciarsi con una varietà di sensori, attuatori e dispositivi esterni.
- **Archiviazione:** utilizza una scheda microSD per l'archiviazione del sistema operativo e dei dati, ma ha anche porte USB per collegare dispositivi di archiviazione esterni.
- **Sistema operativo:** può eseguire diversi sistemi operativi Linux, tra cui Raspberry Pi OS (precedentemente noto come Raspbian), Ubuntu e altri.

2.1.3 Sensore BMP280

Il BMP280 è un sensore di pressione atmosferica e temperatura molto popolare (figura 2.4), spesso utilizzato in applicazioni di monitoraggio meteorologico o sistemi di navigazione.[3]



Figura 2.4: Sensore BMP280

Esso offre:

- **Misurazione della Pressione Atmosferica:** il BMP280 è in grado di misurare la pressione atmosferica con una precisione elevata. Questa informazione può essere utilizzata per calcolare l'altitudine, prevedere le condizioni meteorologiche o monitorare le variazioni di pressione.
- **Misurazione della Temperatura:** oltre alla pressione, il BMP280 fornisce anche dati di temperatura con alta precisione. Questi dati possono essere utilizzati per calibrazione o per compensare le variazioni di temperatura nelle letture di pressione.
- **Interfaccia Comune:** il BMP280 utilizza un'interfaccia di comunicazione standard come I2C o SPI, ciò semplifica la connessione a microcontrollori e microprocessori.

Capitolo 2 Materiali e metodi

In seguito verranno descritti in dettaglio alcuni pin del sensore, utili per gli scopi del progetto:

- **VIN (Alimentazione)**: è il pin di alimentazione del sensore e deve essere collegato a una tensione di alimentazione adeguata, di solito 3.3V (sebbene alcuni modelli possono funzionare a 5V).
- **GND (Terra)**: è il pin di terra e deve essere collegato al pin GND dell'ESP32 per stabilizzare il potenziale di riferimento del BMP280.
- **SDI (Serial Data Input)**: viene utilizzato per trasmettere dati dal BMP280 al microcontrollore.
- **SCK (Serial Clock)**: viene utilizzato per sincronizzare la comunicazione tra il microcontrollore e il BMP280.

2.1.4 Display OLED

Il display OLED Adafruit 128x64 è un modulo display a matrice di punti con una risoluzione di 128x64 pixel, in figura 2.5. È uno schermo di piccole dimensioni, con una diagonale di circa 1 pollice, ma offre un'elevata leggibilità grazie all'uso di tecnologia OLED, fornendo un alto contrasto e colori vividi. Il design è completamente compatibile con una tensione di ingresso a 3.3 V, con un regolatore integrato e un convertitore di tensione incorporato.[4] Il display emette luce propria, motivo per cui non è necessaria una retroilluminazione. Ciò riduce notevolmente il consumo di energia necessario per alimentarlo.

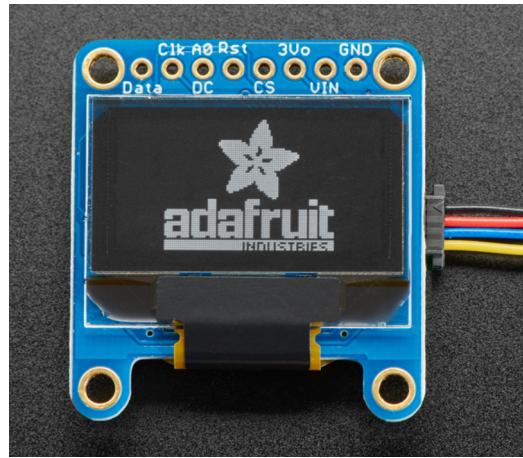


Figura 2.5: Schermo Oled

Il display OLED SSD1306 comunica tramite il protocollo I2C e utilizza quattro pin principali:

- **VIN (Alimentazione)**: è il pin di alimentazione e fornisce la tensione necessaria per alimentare il display OLED (3.3 V).
- **GND (Terra)**: è il pin di terra e deve essere collegato al pin GND dell'ESP32.
- **SDA (Serial Data)**: è il pin di dati seriale e viene utilizzato per trasferire i dati dall' ESP32 al display OLED e viceversa.
- **SCL (Serial Clock)**: è il pin di clock seriale e viene utilizzato per sincronizzare la trasmissione dei dati tra microcontrollore e il display OLED durante la comunicazione I2C.

2.1.5 Ventola Noctua NF-A4x10

Per il raffreddamento del sistema è stata utilizzata una ventola Noctua NF-A4x10 5V PWM, in figura 2.6. La ventola è progettata con caratteristiche avanzate di design aerodinamico, tra cui i "Flow Acceleration Channels" (canali di accelerazione del flusso) e una struttura AAO (Advanced Acoustic Optimization), che contribuiscono a migliorare l'efficienza della ventilazione e a ridurre il rumore. Le sue dimensioni sono di 40x10 mm, rendendola compatta e adatta per l'installazione in spazi ristretti o in situazioni in cui lo spazio è limitato.[5]



Figura 2.6: Ventola Noctua NF-A4x10 5V PWM

Inoltre, è dotata di una funzione PWM (Pulse Width Modulation) che consente di regolare la velocità della ventola in modo dinamico in base alle esigenze di raffreddamento del sistema. Questo significa che può adattarsi automaticamente alle variazioni di temperatura senza dover essere manualmente controllata. Infine la ventola è dotata di un IC PWM personalizzato chiamato NE-FD3 PWM IC, che integra la tecnologia Smooth Commutation Drive. Quest'ultima contribuisce a garantire un funzionamento incredibilmente fluido, riducendo al minimo le vibrazioni e il rumore.

2.1.6 Modulo RTC

Il modulo RTC Adafruit (figura 2.7) basato sul PCF8523 è un dispositivo progettato per fornire funzionalità di orologio in tempo reale (RTC) con alta precisione e un consumo energetico ridotto.[6]

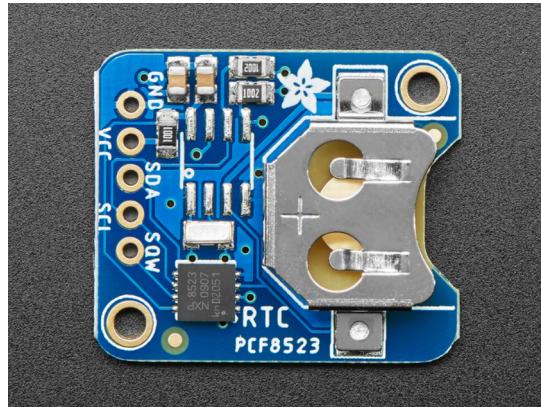


Figura 2.7: Modulo RTC

Di seguito vengono riportate alcune informazioni del modulo RTC;

- **Precisione temporale:** fornisce un'elevata precisione per generare timestamp accurati.
- **Interfaccia I2C:** il PCF8523 comunica tramite l'interfaccia I2C, semplificando la connessione e la comunicazione con schede di sviluppo compatibili con I2C, come Arduino.
- **Batteria di backup:** il modulo RTC è dotato di una batteria di backup al litio, la quale garantisce il mantenimento del corretto funzionamento dell'orologio anche in caso di interruzione dell'alimentazione principale. Questo assicura che l'orologio continui a segnare il tempo correttamente anche dopo un riavvio o una perdita di alimentazione.
- **Funzione di conteggio:** traccia secondi, minuti, ore, giorni della settimana e altri dati temporali.
- **Librerie Adafruit:** per generare un timestamp utilizzando il modulo RTC PCF8523 di Adafruit, è necessario avviare il modulo, acquisire l'ora attuale e poi impiegare questi dati per creare il proprio timestamp. Le librerie Adafruit semplificano notevolmente questa procedura, agevolando l'impiego del modulo RTC per generare timestamp precisi nei progetti.

Il modulo RTC potrebbe gradualmente accumulare una piccola deriva nel tempo, il che può portare a ritardi nel mantenere l'ora esatta. Per affrontare questo problema, è stato implementato uno script eseguito sulla Raspberry Pi che estrae l'orario da

un server NTP (Network Time Protocol) e successivamente lo trasmette all'RTC attraverso il protocollo MQTT. In questo modo, la Raspberry Pi agisce come un "guardiano" del tempo, garantendo un sincronismo adeguato dell'RTC.

2.2 Componenti Software

Di seguito vengono riportate i software utilizzati in questo progetto:

- **Arduino IDE**: ambiente di sviluppo utilizzato per programmare la scheda ESP32.
- **Visual Studio Code**: editor per lo sviluppo del codice python per la Raspberry Pi 4.
- **Raspbian GNU/Linux 11 (bullseye)**: sistema operativo installato sulla Raspberry Pi 4.
- **FreeRTOS (Real-Time Operating System)**: utilizzato per la gestione dei vari task di controllo e comunicazione.

2.2.1 Arduino IDE

Arduino IDE (Integrated Development Environment) è un ambiente di sviluppo software ufficiale utilizzato per programmare e caricare il codice sulle schede Arduino e microcontrollori. Esso offre un'interfaccia utente intuitiva che facilita la scrittura, la modifica e il caricamento del codice C/C++ sulle schede. Gli utenti possono accedere facilmente a tutte le funzionalità necessarie per sviluppare progetti elettronici. Arduino include una vasta libreria standard di funzioni predefinite che semplificano notevolmente il processo di programmazione. Queste librerie contengono codice preconfigurato per molte periferiche comuni, come sensori di temperatura, sensori di luce, motori e display.[7]



Figura 2.8: Logo Arduino IDE

Una delle funzionalità principali di questo ambiente di sviluppo è la possibilità di caricare il codice compilato sulla scheda Arduino attraverso una connessione USB.

Questo processo è noto come "upload" ed è molto semplice da eseguire. Una volta scritto il codice nel progetto, basta premere il pulsante di upload e il software compilerà il codice e lo trasferirà sulla scheda Arduino collegata. Infine, è presente un debugger molto basilare che consente di eseguire il codice passo dopo passo e di visualizzare il valore delle variabili.

2.2.2 Visual Studio Code

Visual Studio Code è un editor cross-platform opensource compatibile con Windows, Linux e macOS che permette di evidenziare la sintassi di ciascun linguaggio di programmazione, integra il supporto per il debugging, controllo Git, IntelliSense (il completamento automatico delle istruzioni), possibilità di mantenere aperti più file affiancandone il contenuto in più schede e molto altro ancora. Il punto di forza di Visual Studio Code sono le estensioni grazie alle quali è possibile ampliare notevolmente le funzionalità del programma.[8]

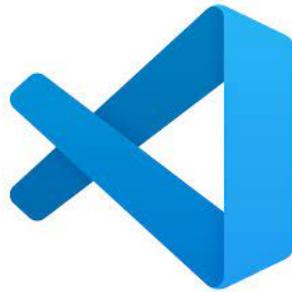


Figura 2.9: Logo Visual Studio Code

VsCode poggia il suo funzionamento su Electron, noto framework con cui è possibile realizzare applicazioni Node.js: è veloce e leggero non soffrendo quindi delle problematiche in termini di performance che affliggono alcune app JavaScript.

2.2.3 Raspbian GNU/Linux 11

Il sistema operativo Raspbian GNU/Linux 11 è la versione più recente del sistema operativo ufficiale per Raspberry Pi. È basato su Debian 11 (bullseye), e include una serie di miglioramenti e nuove funzionalità. Tra questi un nuovo kernel Linux 5.10, che offre miglioramenti per le prestazioni e il supporto hardware.[9]



Figura 2.10: Logo Raspberry Pi OS

Questa versione, a differenza di Raspbian GNU/Linux 11 (bullseye) Lite (che presenta solo un ambiente a linea di comando) include un’interfaccia grafica desktop completa, con una serie di applicazioni preinstallate, tra cui un browser web, un client di posta elettronica, un elaboratore di testi e un foglio di calcolo.

2.2.4 FreeRTOS (Real-Time Operating System)

FreeRTOS (Real-Time Operating System) è un sistema operativo in tempo reale open-source progettato per applicazioni embedded e a tempo critico. È stato creato per fornire una piattaforma affidabile e altamente efficiente per lo sviluppo di sistemi embedded in cui è fondamentale la gestione precisa del tempo e delle risorse. Grazie al suo kernel leggero può essere eseguito su microcontrollore richiedendo risorse di sistema relativamente basse. Questo lo rende adatto per applicazioni embedded in cui la memoria e la potenza di calcolo sono limitate.[10]



Figura 2.11: Logo FreeRTOS

Offre una gestione precisa del tempo e supporta il multitasking preemptive, consentendo di creare applicazioni con più thread o task che possono essere eseguiti concorrentemente. Attraverso strumenti come code, semafori, mutex e altre primitive di sincronizzazione, FreeRTOS consente la comunicazione tra i task in modo sicuro e coerente.

Inoltre, FreeRTOS è altamente portabile e adattabile a una varietà di piattaforme hardware.

2.3 Dettagli del funzionamento

Il principale compito del sistema è quello di monitorare costantemente i parametri ambientali di temperatura e pressione tramite il sensore BMP280.

In particolare, quando la temperatura misurata supera una soglia predefinita, viene attivata la ventola regolando la sua velocità in proporzione alla temperatura rilevata così da mantenere il sistema all'interno di un range di funzionamento accettabile.

I dati rilevati, tra cui temperatura, pressione e velocità della ventola, vengono visualizzati in tempo reale su un display OLED.

Questi stessi dati, insieme ai relativi timestamp, vengono trasmessi utilizzando il protocollo MQTT dalla scheda ESP32 alla Raspberry Pi, che funge da broker MQTT. La Raspberry Pi ha il compito di archiviare i dati e di agire come un server IoT MQTT, consentendo ad altri dispositivi connessi alla stessa rete di visualizzare e accedere ai dati memorizzati.

Il modulo RTC è utilizzato per garantire che tutti i dispositivi all'interno del sistema siano sincronizzati in termini di tempo.

Il tutto è gestito da Freertos in modo da eseguire molteplici task concorrenti con priorità diverse e garantire che le attività abbiano sempre l'accesso alle risorse necessarie. In figura 2.12 è riassunto lo schema di funzionamento del sistema e come i vari componenti interagiscono tra loro.

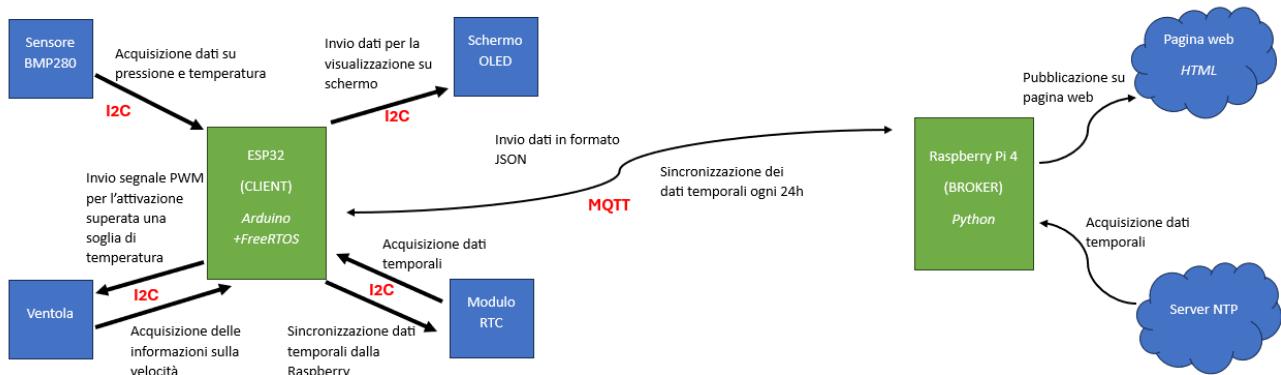


Figura 2.12: Schema di funzionamento del sistema

Capitolo 3

Configurazione e setup

Questo capitolo si propone di esplorare in dettaglio le procedure per configurare correttamente ciascun dispositivo nel sistema e come stabilire connessioni affidabili tra sensori, schermo oled, ventola e ESP32, prestando attenzione per evitare cortocircuiti o errori di collegamento e garantendo che i dati e le informazioni fluiscano senza intoppi tra i componenti chiave del progetto.

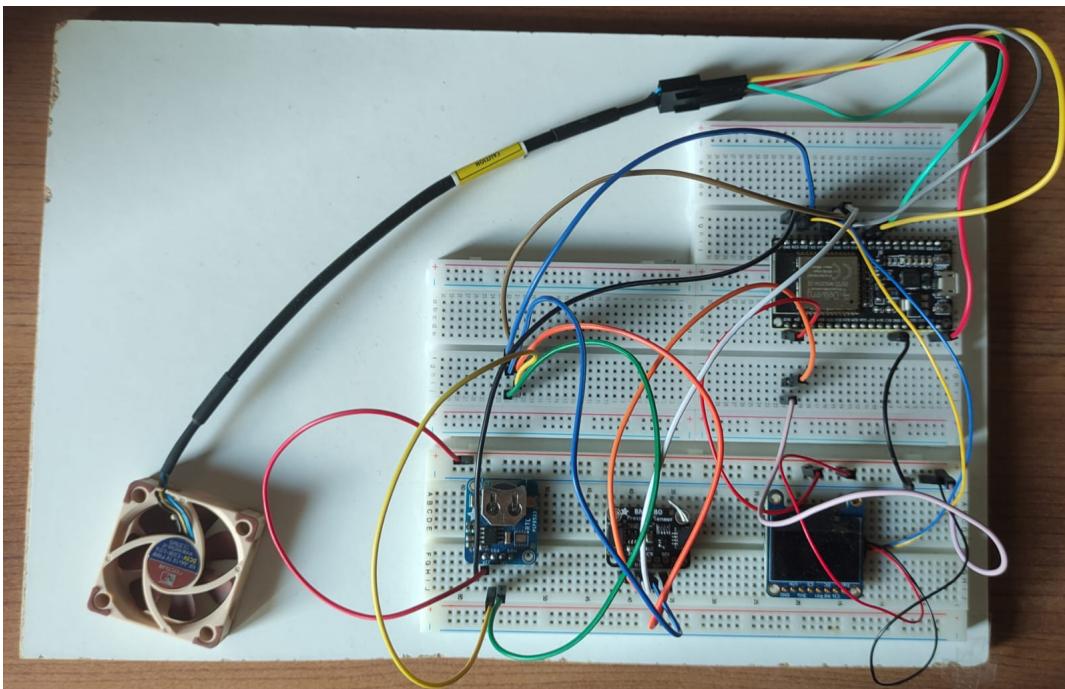


Figura 3.1: Configurazione finale del sistema, con i collegamenti tra ESP32, il sensore BMP280, lo schermo OLED, il modulo RTC e la ventola

3.1 Collegamento del Sensore BMP280

Per collegare il sensore BMP280 all'ESP32, è possibile utilizzare sia l'interfaccia I2C che l'interfaccia SPI. In questo progetto, si è preferito usare I2C per i collegamenti, in quanto è più adatta per i sensori di temperatura e applicazioni a bassa velocità, grazie alla sua semplicità di implementazione e configurazione, il che lo rende una

Capitolo 3 Configurazione e setup

scelta comune per dispositivi con requisiti di comunicazione meno intensi.

In particolare, i collegamenti sono stati effettuati nel seguente modo:

- Collegare il pin VCC (**cavo arancione**) del BMP280 al pin 3.3V dell'ESP32 per l'alimentazione.
- Collegare il pin GND (cavo bianco) del BMP280 al pin GND dell'ESP32 per il collegamento a terra.
- Collegare Il pin SDI (**cavo arancione**) del BMP280 al pin SDA (GPIO21 su ESP32) dell'ESP32 per la comunicazione I2C.
- Collegare Il pin SCK (**cavo blu**) del BMP280 al pin SCL (GPIO22 su ESP32) dell'ESP32 per la comunicazione I2C.

3.2 Collegamento del display OLED

Per il collegamento è stato utilizzato un connettore femmina sullo schermo OLED e un connettore maschio sul microcontrollore ESP32. Questo consente di collegare e scollegare il display OLED in modo semplice e senza dover saldare direttamente i fili. In particolare, i collegamenti sono stati effettuati nel seguente modo:

- Collegare il cavo VIN (**cavo rosso**) del display OLED a una tensione appropriata (solitamente 3,3 V per ESP32).
- Collegare il cavo GND (cavo nero) del display OLED al GND della ESP32.
- Collegare il cavo SDA (**cavo blu**) del display OLED al pin SDA (GPIO21) della ESP32.
- Collegare il pin SCL (**cavo giallo**) del display OLED al pin SCL (GPIO22) della ESP32.

3.3 Collegamento del modulo RTC

I collegamenti per la comunicazione tra il modulo RTC e la Esp32 sono stati realizzati nel seguente modo:

- Collegare il pin VCC (**cavo rosso**) del modulo RTC al pin 3.3V dell'ESP32 per l'alimentazione.
- Collegare il pin GND (cavo nero) del modulo RTC al pin GND dell'ESP32 per stabilire il collegamento a terra.
- Collegare il pin SDA (**cavo giallo**) del modulo RTC al pin GPIO21 dell'ESP32 per la comunicazione dati I2C.
- Collegare il pin SCL (**cavo verde**) del modulo RTC al pin GPIO22 dell'ESP32 per la comunicazione clock I2C.

3.4 Collegamento ventola NOCTUA NF-A4x10 5V PWM

Di seguito la procedura per collegare correttamente la ventola alla scheda ESP32:

- Collegare il pin GND della ventola (cavo grigio) ad uno dei pin GND della scheda per il collegamento a terra.
- Collegare il pin VCC della ventola (**cavo rosso**) al pin 5V della scheda affichè venga alimentata correttamente.
- Collegare il pin TACHO della ventola (**cavo verde**) al pin GPIO16 della scheda. Questo collegamento è fondamentale per l'acquisizione dei dati dal tachimetro, necessaria per il calcolo della velocità della ventola.
- Collegare l'ultimo pin PWM della ventola (**cavo giallo**) al pin GPIO17 della scheda. Questo pin serve per acquisire l'intensità del segnale PWM utilizzato per alimentare la ventola.

Infine, la ventola è stata posizionata in modo da rivolgere il suo flusso d'aria verso il sistema.

3.5 Setup Raspberry Pi

Di seguito vengono riportati i passaggi base per utilizzare Raspberry Pi Imager per configurare la Raspberry Pi con una scheda SD:

1. **Scaricare e installare Raspberry Pi Imager:** dal sito ufficiale di Raspberry Pi Imager (<https://www.raspberrypi.org/software/>) scaricare il programma appropriato in base al proprio sistema operativo.
2. **Avviare Raspberry Pi Imager:** dopo l'installazione, avviare Raspberry Pi Imager.
3. **Selezionare l'immagine del sistema operativo:** nella finestra di Raspberry Pi Imager, fare clic su "*Scegli s.o.*" e selezionare l'immagine del sistema operativo scaricato in precedenza. Per il seguente progetto è stato utilizzato Raspberry PI OS (32 bit).
4. **Selezionare la scheda SD:** fare clic su "*Scegli Scheda SD*" e selezionare la scheda SD che si desidera utilizzare, prestando attenzione a selezionare la scheda SD corretta, in quanto verranno cancellati tutti i dati presenti su di essa.
5. **Scrivere l'immagine:** fare clic su "*Scrivi*" per iniziare il processo di scrittura dell'immagine del sistema operativo sulla scheda SD.

6. **Eseguire l'inserimento e l'accensione:** una volta completato il processo di scrittura, rimuovere la scheda SD in modo sicuro dal computer e inserirla nella Raspberry Pi. Collegare tutte le periferiche necessarie e alimentare la Raspberry Pi.
7. **Configurare il sistema operativo:** seguire le istruzioni di configurazione iniziale che verranno visualizzate sullo schermo. Queste istruzioni guideranno attraverso la creazione di una password, la configurazione della connessione Wi-Fi e altre impostazioni di base.
8. **Aggiornare il sistema:** dopo la configurazione iniziale, è consigliabile eseguire un aggiornamento del sistema operativo per assicurarsi di avere tutte le ultime correzioni di sicurezza e aggiornamenti del software. È possibile farlo tramite il terminale con il comando in figura 3.2.

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

Figura 3.2: Update e Upgrade RPi

9. **Installare le applicazioni desiderate:** ora è possibile installare le applicazioni e configurare la Raspberry Pi secondo le esigenze di progetto.

3.6 Setup ESP32

Di seguito vengono riportati i passaggi base per configurare un modulo ESP32:

1. **Installazione dell'IDE:** bisogna scaricare e installare un ambiente di sviluppo integrato (IDE) sul computer. In questo progetto è stato utilizzato Arduino.
2. **Installazione del Supporto ESP32 nell'IDE:** aprire Arduino IDE. Spostarsi su "*File*" > "*Preferenze*" e nella finestra delle preferenze, inserire l'URL "https://dl.espressif.com/dl/package_esp32_index.json" nella sezione "*URL aggiuntive per il Gestore schede*". Andare su "*Strumenti*" > "*Gestore schede*", cercare "*esp32*" e installare il supporto per ESP32.
3. **Collegare l'ESP32 al Computer:** collegare il modulo ESP32 al computer tramite il cavo USB.
4. **Selezionare la Scheda e la Porta:** nell'IDE, aprire un nuovo progetto. Selezionare il modello di ESP32 utilizzato dal menu "*Strumenti*" > "*Scheda*". In questo progetto viene utilizzato ESP32 Dev Module. Selezionare la porta COM corretta a cui è collegato l'ESP32 dal menu "*Strumenti*" > "*Porta*".

5. **Caricare un programma di esempio sull'ESP32** : per verificare che tutto funzioni correttamente, caricare un programma di esempio. Si può utilizzare l'esempio "WiFi" > "WiFiScan" che rileva le reti wifi nelle vicinanze. Successivamente premere il pulsante "Carica" nell'IDE per caricare il programma sull'ESP32.

3.7 Verifica del corretto cablaggio

Una volta effettuata la configurazione hardware e il setup software, come primo passo è consigliato effettuare uno scan dei dispositivi I2C visibili per verificare il corretto cablaggio dei componenti.

Nello script in figura 3.3 viene utilizzato un ciclo for iterato dall'indirizzo 1 al 126 (da 0x01 a 0x7F), ovvero tutti gli indirizzi validi per la comunicazione, in quanto 0x00 è riservato per il bus generale e 0x7F è riservato per le trasmissioni broadcast.

```
void loop()
{
    byte error, address; //Dichiarazione variabili per gestire errori e indirizzi
    int nDevices; //Variabile intera per tenere traccia dei dispositivi I2C trovati

    Serial.println("Scanning..."); //Stampa messaggio sulla porta seriale per indicare che la scansione è iniziata

    nDevices = 0;
    for(address = 1; address < 127; address++) //ciclo for iterato dall'indirizzo 1 al 126 (0x01 a 0x7F)
    {
        // Lo scanner I2C utilizza il valore restituito da "Wire.endTransmission"
        // per verificare se un dispositivo ha riconosciuto l'indirizzo
        Wire.beginTransmission(address); //viene effettuata una trasmissione I2C verso tutti gli indirizzi
        error = Wire.endTransmission();

        if (error == 0) //se error=0 significa che un dispositivo ha risposto all'indirizzo corrente
        {
            Serial.print("I2C device found at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.print(address,HEX);
            Serial.println(" !");

            nDevices++;
        }
        else if (error==4)
        {
            Serial.print("Unknow error at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.println(address,HEX);
        }
    }
    if (nDevices == 0)
        Serial.println("No I2C devices found\n");
    else
        Serial.println("done\n");

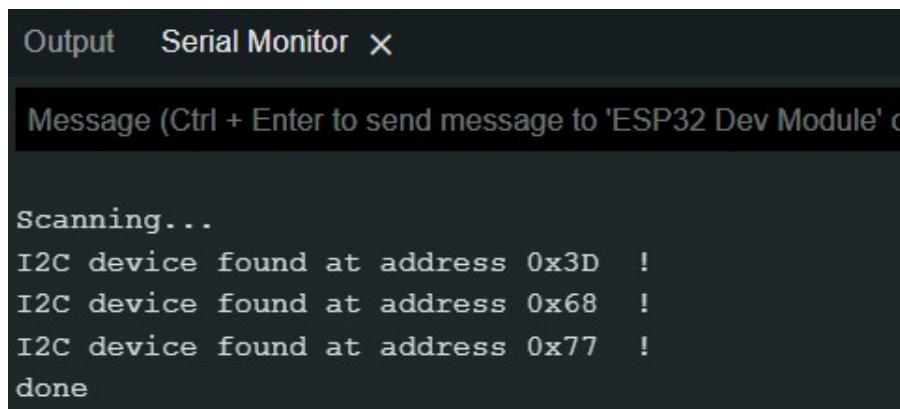
    delay(5000);           // aspetta 5 secondi per la prossima scansione
}
```

Figura 3.3: Script per effettuare lo scanner dei dispositivi I2C

All'interno del ciclo viene effettuata una trasmissione I2C verso ciascun indirizzo utilizzando "Wire.beginTransmission(address)". Successivamente, "Wire.endTransmission()" restituisce un codice di errore che indica se un dispositivo ha risposto all'indirizzo.

Se "error" è uguale a 0, significa che un dispositivo ha risposto all'indirizzo corrente. In tal caso, il programma stampa l'indirizzo I2C sulla porta seriale in formato esadecimale e incrementa il contatore "nDevices". Se "error" è uguale a 4, indica un errore sconosciuto durante la comunicazione con l'indirizzo corrente, e il programma stampa un messaggio di errore. Alla fine del ciclo for, il programma verifica se sono stati trovati dispositivi I2C. Se "nDevices" è uguale a 0, il programma stampa "No I2C devices found", altrimenti stampa "done" per indicare che la scansione è completa.

Se la configurazione hardware è stata eseguita correttamente, sul terminale viene visualizzato l'output mostrato in figura 3.4, dove l'indirizzo 0x3D corrisponde allo schermo OLED, l'indirizzo 0x77 al sensore e il 0x68 al modulo rtc.



```
Output  Serial Monitor ×

Message (Ctrl + Enter to send message to 'ESP32 Dev Module' or another port)

Scanning...
I2C device found at address 0x3D !
I2C device found at address 0x68 !
I2C device found at address 0x77 !
done
```

Figura 3.4: Scanner I2C

3.8 Avvio e configurazione della comunicazione tra l'ESP32 e la Raspberry Pi 4

Di seguito i passaggi per avviare e configurare correttamente la comunicazione lato ESP32:

1. **Configurazione del WiFi:** per avviare la comunicazione tra l'ESP32 e il Raspberry Pi, è necessario configurare le impostazioni WiFi, andando a modificare il codice sorgente. È necessario individuare nel codice sorgente della ESP32 la sezione relativa alla configurazione WiFi, come in figura 3.5, e modificare il nome della rete (SSID) e la password della rete WiFi in uso.

```
const char *ssid = "IlTuoSSID";
const char *password = "LaTuaPassword";
```

Figura 3.5: Sezione per la configurazione WiFi

2. **Modifica dell'Indirizzo della Raspberry Pi 4:** per stabilire la connessione, è fondamentale specificare l'indirizzo IP della Raspberry Pi, modificando nel codice sorgente dell'ESP32 la sezione corrispondente, come in figura 3.6.

```
const char *mqtt_server = "IndirizzoIP_RaspberryPi";
```

Figura 3.6: Sezione per la configurazione dell'indirizzo della RPi

3. **Caricamento del codice sull'ESP32:** dopo aver completato la scrittura e il caricamento del codice sull'ESP32 (seguendo i passaggi del paragrafo 3.6), il dispositivo è pronto per l'esecuzione.
4. **Verifica della connessione:** è sufficiente collegare l'ESP32 alla fonte di alimentazione e il dispositivo inizierà a eseguire il codice automaticamente. È possibile verificare la connessione WiFi tramite il monitor seriale per assicurarsi che l'ESP32 sia connesso correttamente alla rete. In questo modo l'ESP32 è pronto per comunicare con la Raspberry Pi 4 seguendo le impostazioni di rete e l'indirizzo del broker configurato.

Di seguito i passaggi per avviare e configurare correttamente la comunicazione lato Raspberry:

1. **Configurazione indirizzo Rpi:** innanzitutto, è necessario modificare l'indirizzo IP della Raspberry Pi (a cui il broker sarà in ascolto per le connessioni dai client MQTT) in tutti gli script della Raspberry Pi per garantire una comunicazione corretta con l'ESP32, modificandone la sezione corrispondente, come in figura 3.7.

```
broker_address = "IndirizzoIP_RaspberryPi"
```

Figura 3.7: Sezione per la configurazione dell'indirizzo del broker

2. **Modificare path:** nello script "*inviohtmlv2.0.py*" specificare il percorso dello script "*orario.py*" nella variabile "*script_path*".
3. **Avvio del Server Flask:** per avviare il server Flask, scrivere sul terminale "*app.py*".
4. **Invio dati al Server Flask:** dopo aver avviato il server, bisogna eseguire lo script di invio dati "*inviohtmlv2.0.py*" da un altro terminale.
5. **Visualizzazione dei dati sulla pagina HTML:** infine aprire il browser e visitare l'indirizzo "<http://192.168.178.188:5000/pagina>" per visualizzare la

Capitolo 3 Configurazione e setup

pagina HTML, verificando che il file HTML (index.html) sia nella cartella "Templates" all'interno della directory del server Flask.

Capitolo 4

Script e Programmazione dei Dispositivi

Questo capitolo fornisce una panoramica sui codici sviluppati per la realizzazione dei vari task, utilizzando Arduino per la programmazione della ESP32 e Python per la Raspberry Pi 4.

4.1 Librerie in Arduino

Il primo passo fondamentale per la realizzazione degli obiettivi del progetto è quello di installare le librerie direttamente dall'Arduino IDE selezionando "*Gestisci librerie*" e cercando "*nome_libreria*". Di seguito sono riportate le librerie installate ai fini del progetto e le loro principali funzionalità:

- **Connessione I2C:** è stata utilizzata la libreria "*Wire*" per la comunicazione I2C.
- **Sensore BMP280:** è stata utilizzata la libreria "*Adafruit_BMP280*" che semplifica la comunicazione con il sensore.
- **Schermo OLED:** per lo schermo OLED basato sul controller SSD1306, sono state utilizzate le librerie "*Adafruit_SSD1306*" e "*Adafruit_GFX Library*".
- **Modulo RTC :** è stata utilizzata la libreria specifica per il PCF8523, "*RTCLib*", per configurare e leggere l'orologio in tempo reale.
- **WiFi:** per connettere l'ESP32 al wifi è stata utilizzata la libreria "*WiFi*".
- **Invio dati in formato JSON:** per inviare i dati via MQTT in formato JSON è stata utilizzata la libreria "*Arduino_JSON*".
- **FreeRTOS:** per suddividere le attività in task implementando FreeRTOS è stata utilizzata la libreria Arduino "*FreeRTOS*".

4.2 Implementazione software scheda ESP32

La programmazione della scheda ESP32 è stata gestita su Arduino IDE. Il codice implementato è responsabile delle attività di connessione Wi-Fi ed MQTT, dell'acquisizione dei dati, ovvero temperatura, pressione, orario e velocità della ventola,

della stampa di questi ultimi sul display OLED ed infine dell'invio delle informazioni alla Raspberry Pi 4 via MQTT in formato JSON. Il codice utilizza un approccio multitasking per gestire diverse attività in modo concorrente. Questo è stato possibile grazie all'utilizzo del framework FreeRTOS, alla creazione di vari task che vengono eseguiti in parallelo e all'utilizzo di un meccanismo di sincronizzazione, quale i Mutex, per la gestione delle risorse condivise.

4.2.1 Task ventola

Per il task relativo al controllo della ventola è stata realizzata la funzione "*taskFanControl*", in figura 4.1.

Per quanto riguarda la definizione delle variabili iniziali, il pin 17 è stato utilizzato per controllare la ventola mediante segnale PWM mentre il pin 16 per acquisire i giri dal tachimetro e successivamente calcolare la velocità in rpm.

In seguito vengono definite la temperatura desiderata, una temperatura di tolleranza e i range operativi del segnale PWM e della velocità della ventola.

Il codice esegue le seguenti operazioni ciclicamente con intervalli regolari di 1 secondo:

- **Calcolo temperatura:** il codice legge la temperatura dal sensore BMP280 tramite il comando *bmp280.readTemperature()* e calcola la differenza tra la temperatura registrata e quella desiderata.
- **Calcolo intensità della ventola:** in proporzione alla differenza di temperatura viene regolata l'intensità del segnale che pilota la ventola. Maggiore è lo scostamento tra la temperatura letta e quella desiderata e maggiore sarà la velocità della ventola. L'intervallo di tolleranza è stato inserito affinchè il raffreddamento non si attivi istantaneamente con il superamento della soglia, ma solo se lo scostamento aumenta ulteriormente. La velocità della ventola viene impostata utilizzando la funzione "*analogWrite(fanPin, fanSpeed)*", tipica per il controllo di una ventola utilizzando la modulazione di larghezza di impulso (PWM).

```

void taskFanControl(void *pvParameters) {
    const int fanPin = 17; // Imposta il pin per il controllo della ventola
    const int feedbackPin = 32; // Imposta il pin di feedback della ventola
    const float targetTemp = 20.8; // Temperatura desiderata in gradi Celsius
    const float tempTolerance = 0.4; // Tolleranza di temperatura in gradi Celsius
    const int maxFanSpeed = 255; // Velocità massima della ventola (0-255)

    const int PWM_min = 0;
    const int PWM_max = 255;
    const int Velocita_min = 0;
    const int Velocita_max = 5000;

    pinMode(feedbackPin, INPUT_PULLUP);
    pinMode(fanPin, OUTPUT);

    while (true) {
        float currentTemp = bmp280.readTemperature(); // Leggi la temperatura attuale dal tuo sensore
        float tempDifference = currentTemp - targetTemp;

        // Calcola la velocità della ventola in base alla differenza di temperatura
        // fanSpeed = map(tempDifference, -tempTolerance, tempTolerance, 0, maxFanSpeed);
        if (tempDifference > tempTolerance) {
            // Temperatura troppo alta, calcola la velocità in base alla differenza di temperatura
            fanspeed = map(tempDifference, 0, 3, 20, maxFanSpeed);
            fanSpeed = constrain(fanspeed, 0, maxFanSpeed);
        }
        else if (tempDifference < -tempTolerance) {
            // Temperatura troppo bassa, disattiva gradualmente la ventola
            fanspeed = map(tempDifference, -tempTolerance, 0, maxFanSpeed, 0);
            fanSpeed = constrain(fanspeed, 0, maxFanSpeed);
        }
        else {
            // Temperatura nella soglia di tolleranza, mantieni la velocità attuale
        }
        //fanSpeed = constrain(fanSpeed, 0, maxFanSpeed); // Assicurati che la velocità sia nell'intervallo
        Serial.println(fanSpeed);

        Serial.println(tempDifference);

        analogWrite(fanPin, fanSpeed);

        vTaskDelay(pdMS_TO_TICKS(1000)); // Controllo della ventola ogni secondo
    }
}

```

Figura 4.1: Codice funzione taskFanControl

4.2.2 Task Sensore e Display OLED

La funzione "*taskSensors*", in figura 4.2 è responsabile del task di lettura dei dati dal sensore BMP280 e della loro stampa sul display OLED.

Prima di tutto viene inizializzato lo schermo OLED attraverso il metodo "*begin*", dove "*SSD1306_SWITCHCAPVCC*" è una costante che specifica la modalità di inizializzazione del display, mentre "*SCREEN_ADDRESS*" è l'indirizzo del display OLED. Le righe successive definiscono una serie di variabili locali per memorizzare la temperatura e la pressione.

Il codice esegue le seguenti operazioni ciclicamente ogni 5 secondi (tempo scelto per assicurare l'aggiornamento dei dati dal sensore):

- **Pulisci schermo:** l'istruzione "*display.clearDisplay()*" cancella il contenuto attuale del display OLED, preparandolo per l'aggiornamento con i nuovi dati dei sensori.

Capitolo 4 Script e Programmazione dei Dispositivi

- **Lettura dati dal sensore:** l'istruzione `"leggiBmp280(temp, press)"` legge i dati dal sensore BMP280 e aggiorna le variabili con i nuovi valori di temperatura e pressione.
- **Visualizzazione dati sullo schermo:** viene impostata la posizione della stampa all'interno del display e viene utilizzato `"display.println()"` per scrivere i dati del sensore, la percentuale del PWM e gli RPM della ventola. I dati aggiornati vengono visualizzati sullo schermo OLED tramite `"display.display()"`.

```
void taskSensors(void *pvParameters) {
    display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS); // Inizializza display oled
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    float temp=0.0;
    float press=0.0;
    while (true) {
        // Lettura dei sensori e aggiornamento dei valori
        display.clearDisplay();
        leggiBmp280(temp,press); //lettura valori sensore
        display.setCursor(10, 10);
        display.println("Temp: " + String(temp) + " C"); // Stampa temperatura
        display.setCursor(10, 20);
        display.println("Press: " + String(press) + " Pa"); // Stampa pressione
        display.setCursor(10, 30);
        display.println("PWM: " + String((getPWM() / 255.0) * 100.0) + " %"); // Stampa percentuale PWM
        display.setCursor(10, 40);
        display.println("RPM: " + String(getRPM())); // Stampa RPM
        display.display();

        vTaskDelay(pdMS_TO_TICKS(5000)); // Lettura dei sensori ogni 5 secondi
    }
}
```

Figura 4.2: Codice funzione taskSensors

Inoltre, all'inizio del ciclo while viene richiamata la funzione `"leggiBmp280"`, in figura 4.3. Quest'ultima va a leggere i valori di temperatura e pressione e li memorizza nelle variabili `"a"` e `"b"`. La funzione `"m.lock()"` segna l'inizio di una sezione critica protetta da un meccanismo di blocco. In questo modo si garantisce che i valori di temperatura e pressione vengano utilizzati da un solo thread. Il comando `"m.unlock"` rappresenta l'uscita dalla sezione critica.

```
void leggiBmp280(float &a, float &b){
    m.lock();
    a = bmp280.readTemperature();
    b = bmp280.readPressure();
    m.unlock();
}
```

Figura 4.3: Codice funzione leggiBmp280

4.2.3 Task invio file JSON

La funzione "*taskJSONPublish*", nelle figure 4.4 e 4.5, è responsabile della creazione del file JSON contenente i dati relativi al sensore, alla ventola e i rispettivi dati temporali, e del suo invio alla Raspberry Pi.

Inizialmente viene dichiarato un oggetto di tipo "*StaticJsonDocument*" con una dimensione massima di 200 byte. Questo oggetto verrà utilizzato per creare il documento JSON. Successivamente viene definito un array di caratteri con una dimensione massima di 500 caratteri, che verrà utilizzato per memorizzare la rappresentazione JSON dei dati.

Il codice esegue le seguenti operazioni ciclicamente ogni 5 secondi:

- **Pulisci file:** "*doc.clear()*" cancella il contenuto del documento JSON in modo da prepararlo per l'aggiornamento con nuovi dati.
- **Ottieni dati temporali:** tramite "*DateTime now = rtc.now()*" viene ottenuto l'orario corrente utilizzando un oggetto "*DateTime*" ricavato dal modulo RTC.
- **Formatta l'orario:**
"snprintf(formattedTime, sizeof(formattedTime), "%02d:%02d:%02d", now.hour(), now.minute(), now.second())" formatta l'orario corrente in un formato hh:mm:ss e lo memorizza nella variabile "*formattedTime*".
- **Leggi dati dal sensore:** l'istruzione "*leggiBmp280(temp, press)*" chiama la funzione per leggere i dati dal sensore e aggiornare le variabili con i nuovi valori di temperatura e pressione.
- **Scrivi documento JSON:** vengono aggiunti i dati al documento JSON utilizzando l'istruzione "*doc["nomeCampo"] = valore*". In particolare, vengono aggiunti i campi "*time*", "*temperatura*", "*pressione*", "*Potenza della ventola*" e "*RPM*" con i valori corrispondenti.
- **Conversione file JSON:** "*serializeJson(doc, output)*" converte il documento JSON in una rappresentazione JSON in formato stringa e la memorizza nell'array di caratteri "*output*".
- **Invio file JSON:** se il client è correttamente connesso con il broker MQTT, invia i dati JSON tramite il comando "*client.publish("parametri", output)*". I dati vengono pubblicati su un topic chiamato "*parametri*". Altrimenti, se la connessione MQTT non è disponibile, aggiunge il JSON a una coda chiamata "*jsonQueue*", verificando se c'è spazio disponibile, altrimenti ulteriori dati verranno persi.

Capitolo 4 Script e Programmazione dei Dispositivi

```
// Task per la creazione e l'invio del file JSON alla Raspberry
void taskJSONPublish(void *pvParameters) {
    StaticJsonDocument<200> doc ;
    char output[500];

    // Invio dei dati JSON tramite MQTT
    while (true) {
        doc.clear();
        // Acquisisce l'orario corrente
        DateTime now = rtc.now();
        char formattedTime[10];
        sprintf(formattedTime, sizeof(formattedTime), "%02d:%02d:%02d", now.hour(), now.minute(), now.second());

        // Crea un oggetto JSON e aggiunge i dati
        doc["time"] = formattedTime;

        float temp = 0.0;
        float press = 0.0;
        leggiBmp280(temp, press);

        doc["temperatura"] = temp;
        doc["pressione"] = press;
        doc["Potenza della ventola"] = String((getPWM() / 255.0) * 100.0) + "%";
        doc["RPM"] = getRPM();
    }
}
```

Figura 4.4: Codice funzione JSONPublish

```
serializeJson(doc, output);

if (client.connected()) {
    // Invia il JSON se c'è connessione
    client.publish("parametri", output);
    Serial.println("JSON inviato: ");
    Serial.print(output);
} else {
    // La connessione MQTT non è disponibile, quindi aggiungi il JSON alla coda
    if (((queueRear + 1) % QUEUE_SIZE) != queueFront) {
        strcpy(jsonQueue[queueRear], output);
        queueRear = (queueRear + 1) % QUEUE_SIZE;
        Serial.println("JSON aggiunto alla coda: ");
        Serial.print(output);

    }
}

vTaskDelay(pdMS_TO_TICKS(5000)); // Invio dei dati JSON ogni 5 secondi
}
}
```

Figura 4.5: Codice funzione JSONPublish

Questo task usa le funzioni personalizzate "getRPM" e "getPWM" che restituiscono il valore di RPM e PWM in modo sicuro, assicurandosi che non vengano lette informazioni inconsistenti a causa di interruzioni durante la lettura.

4.2.4 Task lettura RPM

La funzione "*readRPMTask*", in figura 4.6, si occupa della lettura della velocità della ventola e del suo calcolo in RPM.

La riga "(void)*pvParameters*" viene utilizzata per evitare un avviso del compilatore relativo alla variabile "*pvParameters*", indicando al compilatore che la variabile è intenzionalmente non usata in questa funzione. In seguito viene dichiarata la costante "*interval*" che rappresenta l'intervallo di tempo in millisecondi tra le letture delle rotazioni. In questo caso, le letture vengono eseguite ogni secondo. Viene anche tenuta traccia dell'ultima lettura con la variabile "*previousMillis*".

Il codice esegue le seguenti operazioni ciclicamente ogni 100 millisecondi:

- **Ottieni tempo corrente:** tramite "*unsigned long currentMillis = millis()*" viene ottenuto il tempo corrente in millisecondi utilizzando la funzione "*millis()*". Questo rappresenta il momento attuale in cui si sta eseguendo il ciclo.
- **Controllo temporale:** viene utilizzata una condizione if per verificare se è passato l'intervallo di tempo specificato dall'ultima lettura delle rotazioni.
- **Disabilita interruzioni:** "*noInterrupts()*" disabilita temporaneamente le interruzioni, il che è importante per evitare problemi di concorrenza dato che si vuole accedere a variabili condivise come "*pulseCount*".
- **Lettura impulsi ventola:** tramite "*unsigned long currentPulseCount = pulseCount*" viene letta la variabile "*pulseCount*", che rappresenta il conteggio degli impulsi generati dal sensore.
- **Reset conteggio:** tramite "*pulseCount = 0*" viene resettato il conteggio degli impulsi a zero per prepararsi alla prossima lettura.
- **Abilita interruzioni:** "*interrupts()*" riabilita le interruzioni.
- **Calcolo RPM:** viene calcolato il valore degli RPM utilizzando la formula $rpm = (currentPulseCount * 60000UL) / (interval * 2UL)$. Questa formula tiene conto del numero di impulsi generati dal sensore durante l'intervallo di tempo specificato e calcola le rotazioni per minuto.
- **Aggiornamento variabile temporale:** tramite "*previousMillis = currentMillis*" la variabile "*previousMillis*" viene aggiornata con il valore corrente di "*currentMillis*", in modo che il ciclo aspetti nuovamente l'intervallo di tempo specificato prima della prossima lettura.

Capitolo 4 Script e Programmazione dei Dispositivi

```
void readRPMTask(void *pvParameters) {
    (void)pvParameters;

    const unsigned long interval = 1000; // Intervallo di lettura in millisecondi
    unsigned long previousMillis = 0;

    while (true) {
        unsigned long currentMillis = millis();
        if (currentMillis - previousMillis >= interval) {
            noInterrupts();
            unsigned long currentPulseCount = pulseCount;
            pulseCount = 0; // Resetta il conteggio degli impulsi
            interrupts();

            rpm = (currentPulseCount * 60000UL) / (interval * 2UL); // Il sensore dà due impulsi per ogni rotazione completa

            Serial.print("RPM: ");
            Serial.println(rpm);

            previousMillis = currentMillis;
        }

        vTaskDelay(pdMS_TO_TICKS(100)); // Attendi un po' prima di eseguire un nuovo controllo
    }
}
```

Figura 4.6: Codice funzione readRPMTask

4.2.5 Task connessione MQTT

La funzione "*taskWiFi_MQTT*" in figura 4.7, ha il compito di configurare la connessione MQTT con il broker e restare in ascolto per sincronizzare i dati temporali in arrivo dalla Raspberry. Il task si occupa dei seguenti compiti:

- **Connessione alla rete WiFi:** ogni 5 secondi controlla se la connessione alla rete specificata da SSID e password tramite il comando *WiFi.begin(ssid, password)* è andata a buon fine. Se non ancora connesso il task aspetta finchè non è avvenuta la connessione.
- **Configurazione MQTT:** tramite il comando *client.setServer(mqttServer, 1883)* viene configurato il client MQTT, specificando l'indirizzo IP del broker e la porta a cui connettersi.
- **Aggiorna l'orario:** ogni qualvolta riceve un messaggio nel topic specificato (comando *client.subscribe(mqtt_topic)*), il task richiama una funzione di callback che legge il messaggio e aggiorna l'orario del modulo RTC.
- **Riconnessione alla rete:** ciclicamente monitora se il dispositivo è ancora connesso, altrimenti esegue la riconnessione tramite la funzione "*reconnect()*" e tramite "*client.loop()*" si assicura di mantenere il client MQTT sempre attivo e funzionante.

Capitolo 4 Script e Programmazione dei Dispositivi

```

// Definizione del task di connessione MQTT
void taskWiFi_MQTT(void *pvParameters) {
    // Connessione WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        vTaskDelay(pdMS_TO_TICKS(5000));
        Serial.println("Connessione alla rete Wi-Fi...");
    }
    Serial.println("Connesso alla rete Wi-Fi"); // Verifica la connessione WiFi ogni 5 secondi
    client.setServer(mqttServer, 1883);
    // In attesa dell'orario via mqtt
    client.setCallback(callback);
    client.subscribe(mqtt_topic);
    //vTaskDelay(pdMS_TO_TICKS(5000));
    while (true) {
        if (!client.connected()){
            //vTaskDelay(pdMS_TO_TICKS(5000));
            reconnect();
        }
        // Esegui il loop del client MQTT ogni 100 ms
    }
    client.loop();
}
}

```

Figura 4.7: Codice funzione taskWiFi_MQTT

In particolare nel task vengono eseguite ciclicamente due funzioni personalizzate, ovvero *"callback()"* e *"reconnect()"*.

La funzione *callback()*, in figura 4.8, viene richiamata ogni volta che viene ricevuto un messaggio sul topic a cui il client è sottoscritto.

Innanzitutto controlla il payload ricevuto, imposta il carattere nullo alla fine del messaggio ricevuto per renderla una stringa C valida ed estraendo tre numeri interi tramite *sscanf()*.

Se ha successo, crea un oggetto *DateTime* con il valore delle ore, minuti e secondi estratti dal payload e aggiorna l'orario del modulo RTC tramite *rtc.adjust()*.

Infine invia al broker un messaggio di conferma per comunicare la corretta ricezione dell'orario.

```

void callback(char* topic, byte* payload, unsigned int length) {

    payload[length] = '\0'; // Assicura che la stringa sia terminata correttamente

    if (sscanf((char*)payload, "%d:%d:%d", &hours, &minutes, &seconds) == 3) {
        // Imposta solo l'orario RTC con i valori ottenuti
        rtc.adjust(DateTime(0, 0, 0, hours, minutes, seconds));
    }
    client.publish("Conferma","Orario ricevuto con successo"); // Invia una conferma
}

```

Figura 4.8: Codice funzione callback

La funzione `reconnect()`, in figura 4.9, gestisce la riconnessione in caso di perdita di rete.

Finchè la connessione non è ristabilita, verifica se è possibile connettersi con il nome cliente `"ESP32Client"`, tramite il comando `client.connect("ESP32Client")` e tenta di riconettersi al broker.

Se la comunicazione è stata ripristinata allora invia tutti i pacchetti accumulati durante la disconnessione nella coda `"jsonQueue"` al topic `"parametri"`. Infine sottoscrive il client nuovamente al topic MQTT specificato.

Se, invece, la connessione al broker MQTT non riesce, effettua un nuovo tentativo, rieseguendo l'intera funzione dopo 5 secondi.

```
void reconnect() {
    // Loop fino a quando non ci si è riconnessi al broker MQTT
    while (!client.connected()) {
        Serial.println("Connessione al broker MQTT...");
        if (client.connect("ESP32Client")) {
            Serial.println("Connesso al broker MQTT");

            // La connessione è stata ripristinata, invia i messaggi dalla coda
            while (queueFront != queueRear) {
                vTaskDelay(pdMS_TO_TICKS(500));
                client.publish("parametri", jsonQueue[queueFront]);
                Serial.println("JSON della coda inviato: ");
                Serial.print(jsonQueue[queueFront]);
                queueFront = (queueFront + 1) % QUEUE_SIZE;
            }

            client.subscribe(mqtt_topic); // Sottoscrivi al topic solo se la connessione è stata stabilita
        } else {
            Serial.print("Errore di connessione, rc=");
            Serial.print(client.state());
            Serial.print(" Riprovo tra 5 secondi...");
            vTaskDelay(pdMS_TO_TICKS(5000));
        }
    }
}
```

Figura 4.9: Codice funzione reconnect

4.3 Installazione pacchetti Raspberry Pi 4

Prima dell'implementazione software su Raspberry è stato necessario installare diversi pacchetti e dipendenze, fondamentali per il corretto funzionamento dell'applicazione IoT.

- **Installazione del broker MQTT Mosquitto:** Mosquitto è uno dei broker MQTT più popolari ed è disponibile come pacchetto preinstallato su molti sistemi operativi Raspberry Pi come Raspbian. E' possibile installare Mosquitto direttamente dal terminale con il comando `"sudo apt-get install mosquitto"`.

Successivamente si avvia il servizio con "`sudo systemctl start mosquitto`" e infine, con il comando "`sudo systemctl enable mosquitto`" si configura Mosquitto in modo che si avvii automaticamente all'accensione della Raspberry.

- **Installazione del client MQTT:** Per comunicare con il broker MQTT dalla Rasberry è necessario installare una libreria MQTT, in questo caso è stata utilizzata *Paho MQTT* per python. Il comando da utilizzare sul terminale è "`pip install paho-mqtt`".
- **Pacchetto NTP (Network Time Protocol):** Questo pacchetto è utilizzato per sincronizzare l'orologio della Raspberry Pi con un server di tempo NTP. Per installarlo è necessario assicurarsi che la Raspberry sia collegata ad internet e lanciare i seguenti comandi da terminale: "`sudo apt-get update`" e successivamente "`sudo apt-get install ntp`".

4.4 Implementazione software Raspberry Pi 4

La programmazione della Raspberry Pi 4 è stata realizzata utilizzando il linguaggio Python. I codici implementati si occupano della gestione della connessione MQTT, del ricevere i pacchetti dall'ESP32, aggiornando la visualizzazione dei dati sulla pagina web. Infine, sono responsabili della sincronizzazione del tempo del modulo RTC ogni 24 ore.

4.4.1 Codice app.py

Lo script riportato in Figura 4.10 è un'applicazione Flask che funge da server per ricevere dati inviati dalla ESP32 (tramite una richiesta POST) e visualizza questi dati su una pagina web.

Di seguito una spiegazione dettagliata del codice:

- **Importazioni:** vengono importati i moduli necessari da Flask. *Flask* è il modulo principale per creare un'applicazione web, *request* è utilizzato per gestire le richieste HTTP e *render_template* è utilizzato per renderizzare pagine HTML.
- **Inizializzazione app:** viene creata un'istanza dell'app Flask, *template_folder* specifica la cartella in cui verranno cercati i file dei modelli HTML.
- **Rotta¹ per ricevere dati:** questa funzione accetta richieste POST alla rotta "`/ricevi-dati`", estrae i dati JSON dalla richiesta e aggiorna le variabili globali temperatura, pressione, e velocità_ventola con i dati ricevuti.

¹Le rotte in un'applicazione web si riferiscono a percorsi specifici o URL a cui l'applicazione risponde quando viene effettuata una richiesta HTTP e determinano come l'applicazione risponde alle diverse azioni degli utenti o alle richieste dei client

- **Rotta per visualizzare dati:** questa funzione restituisce i dati correnti quando viene effettuata una richiesta GET alla rottta "/visualizza-dati".
- **Rotta per la pagina web:** questa funzione renderizza la pagina HTML "index.html" utilizzando i dati correnti di temperatura, pressione e velocità della ventola.
- **Avvio dell'applicazione:** l'app viene eseguita sulla porta 5000 rendendola accessibile all'esterno da qualsiasi indirizzo IP collegato alla stessa rete locale tramite host='0.0.0.0'.

```
from flask import Flask, request, render_template

app = Flask(__name__, template_folder='templates')

temperatura = None
pressione = None
velocità_ventola= None
@app.route('/ricevi-dati', methods=['POST'])
def ricevi_dati():
    global temperatura, pressione, velocità_ventola
    dati = request.get_json()

    temperatura = dati.get("temperatura")
    pressione = dati.get("pressione")
    velocità_ventola = dati.get("velocità_ventola")
    return 'Dati ricevuti con successo'

@app.route('/visualizza-dati', methods=['GET'])
def visualizza_dati():
    global temperatura, pressione, velocità_ventola
    dati = {"temperatura": temperatura, "pressione": pressione, "velocità_ventola": velocità_ventola}
    return dati

@app.route('/pagina', methods=['GET'])
def pagina():
    return render_template('index.html', temperatura=temperatura, pressione=pressione, velocità_ventola= velocità_ventola)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Figura 4.10: Script app.py

4.4.2 Codice inviohtml.py

Lo script in figura 4.11 gestisce l'invio e la ricezione di messaggi tra ESP32 e Raspberry Pi 4 su due topic distinti. Di seguito una spiegazione delle principali parti dello script:

- **Importazioni:**

- *paho.mqtt.client*: viene utilizzato per gestire la comunicazione con il broker MQTT tramite il protocollo MQTT.
- *requests*: è una libreria per semplificare l'invio di richieste HTTP.
- *json*: gestisce la codifica e la decodifica di dati JSON.
- *subprocess*: consente di eseguire comandi esterni.
- *time*: offre funzionalità relative al tempo.

- **Configurazione del broker MQTT:** vengono definite le variabili riferite all'indirizzo IP e alla porta del broker MQTT.
- **Funzioni di callback:** le funzioni "*on_connect*" e "*on_message*" sono definite per gestire gli eventi di connessione al broker MQTT e la ricezione di messaggi. Queste funzioni sono richiamate automaticamente dalla libreria Paho MQTT quando si verificano gli eventi corrispondenti. In particolare la funzione "*on_connect*" si occupa di iscrivere il client ai due topic desiderati ("*topic1*" e "*topic2*") dopo una connessione riuscita. La funzione "*on_message*" decodifica il payload del messaggio e gestisce il messaggio in base al topic a cui è stato inviato. Se il messaggio è sul topic 1, chiama la funzione "*invia_dati_al_server*" con i dati decodificati. Se il messaggio è sul topic 2, chiama la funzione "*messaggio_orario*" con il payload del messaggio.
- **Funzione per inviare dati al server Flask:** questa funzione riceve un parametro dati, contenente i dati inviati dall'ESP32. La funzione poi compone una richiesta HTTP POST per inviare questi dati al server Flask che è in ascolto sulla rottura "*/ricevi-dati*".
- **Funzione di conferma della ricezione dell'orario:** imposta un flag a True quando viene ricevuto un messaggio sul topic 2, ovvero quando l'orario è stato ricevuto correttamente dalla ESP32.
- **Funzione per eseguire uno script se non riceve conferma:** questa funzione esegue uno script solo se non è stato ricevuto un messaggio sul topic 2, ovvero nel caso in cui l'ESP32 non abbia ricevuto l'orario dalla RPi o non sia arrivata la conferma alla RPi. Nella funzione viene controllato il flag di "*messaggio_ricevuto*"; nel caso in cui è False viene eseguito lo script "*orario.py*", utilizzando il modulo *subprocess*.
- **Ciclo principale:** questo ciclo controlla se sono trascorsi 5 minuti e aggiorna il timestamp, infine esegue lo script se necessario.

Capitolo 4 Script e Programmazione dei Dispositivi

```

import paho.mqtt.client as mqtt
import requests
import json
import subprocess
import time

# Configurazione del broker MQTT
broker_address = "192.168.7.229" # Sostituisci con l'indirizzo IP del tuo broker MQTT
broker_port = 1883
topic1 = "parametri" # Sostituisci con il tuo topic MQTT
topic2= "conferma"

# Flag per verificare se è stato ricevuto un messaggio sul topic "conferma"
messaggio_ricevuto = False

# Variabile di timestamp per la logica dei 5 minuti
timestamp_5_minuti = time.time()

def on_connect(client, userdata, flags, rc):
    print("Connesso al broker MQTT con codice:", rc)
    client.subscribe(topic1)
    client.subscribe(topic2)

def on_message(client, userdata, msg):
    print("Messaggio ricevuto sul topic", msg.topic)
    print("Contenuto:", msg.payload.decode("utf-8"))
    if msg.topic == topic1 :
        # Invia i dati al server Flask
        dati = json.loads(msg.payload.decode("utf-8"))
        invia_dati_al_server(dati)
    elif msg.topic == topic2 :
        messaggio_orario(msg.payload.decode("utf-8"))

def invia_dati_al_server(dati):
    url = "http://192.168.7.229:5000/ricevi-dati"
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, json=dati, headers=headers)
    print("Risposta dal server:", response.text)

def messaggio_orario(payload):
    global messaggio_ricevuto
    print("Messaggio orario ricevuto:", payload)
    # Imposta il flag a True quando ricevi un messaggio sul topic "orario"
    messaggio_ricevuto = True

def esegui_script_se_necessario():
    global messaggio_ricevuto
    # Se non è stato ricevuto nessun messaggio sul topic "conferma", esegui lo script
    if not messaggio_ricevuto:
        # Specifica il percorso completo dello script che vuoi eseguire
        script_path = "/home/fiore/Desktop/orario_mqtt/orario.py"

        try:
            # Esegui lo script utilizzando subprocess
            subprocess.run(["python", script_path], check=True)
            print("Script eseguito con successo.")
        except subprocess.CalledProcessError as e:
            print("Errore durante l'esecuzione dello script:", e)

    # Funzione che imposta messaggio_ricevuto a False ogni 5 minuti
def aggiorna_time():
    global timestamp_5_minuti
    # Aggiorna il timestamp per la prossima verifica dei 5 minuti
    timestamp_5_minuti = time.time()

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(broker_address, broker_port, 60)
client.loop_start()

# Ciclo principale
while True:
    # Verifica dei 5 minuti
    if time.time() - timestamp_5_minuti >= 5 * 60:
        aggiorna_time()
        # Chiamare questa funzione quando si desidera eseguire lo script se necessario
        esegui_script_se_necessario()
    # Intervallo di attesa di 1 secondo
    time.sleep(1)

```

Figura 4.11: Script inviohtmlv2.0.py

4.4.3 Codice index.html

Il codice in figura 4.12 crea una pagina web che visualizza i dati di temperatura, pressione e velocità della ventola.

In particolare nella sezione "*head*" definisce il titolo che appare nella barra del titolo del browser e include la libreria jQuery, che è una libreria JavaScript molto utilizzata per semplificare la gestione di operazioni comuni lato client come l'invio di richieste AJAX².

La sezione "*body*" contiene il contenuto visibile della pagina web, visualizzando i dati ambientali, e utilizza degli identificatori per aggiornare dinamicamente il contenuto dei dati tramite JavaScript.

La sezione "*script*" incorpora JavaScript all'interno della pagina HTML per definire le funzioni e le operazioni necessarie. Viene definita la funzione *aggiornaDati()* che esegue ogni 5 secondi una richiesta AJAX per ottenere i dati di temperatura, pressione e velocità della ventola dal server Flask. Questa chiamata AJAX imposta una richiesta GET verso l'URL '/visualizza-dati' del server Flask. Quando la richiesta ha successo, i dati ricevuti in formato JSON vengono utilizzati per aggiornare dinamicamente il contenuto degli identificatori.

```
<!DOCTYPE html>
<html>
<head>
    <title>Dati di Temperatura e Pressione</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h1>Dati di Temperatura e Pressione</h1>
    <p>Temperatura: <span id="temperatura">{{ temperatura }}</span> °C</p>
    <p>Pressione: <span id="pressione">{{ pressione }}</span> Pa</p>
    <p>Velocità della Ventola: <span id="velocità_ventola">{{ velocità_ventola }}</span> RPM</p>
    <script>
        function aggiornaDati() {
            $.ajax({
                url: '/visualizza-dati',
                type: 'GET',
                dataType: 'json',
                success: function(data) {
                    $('#temperatura').text(data.temperatura);
                    $('#pressione').text(data.pressione);
                    $('#velocità_ventola').text(data.velocità_ventola);
                }
            });
        }
        // Aggiorna i dati ogni 5 secondi
        setInterval(aggiornaDati, 5000);
    </script>
</body>
</html>
```

Figura 4.12: Script pagina HTML

²Le richieste AJAX (Asynchronous JavaScript and XML) sono richieste HTTP asincrone effettuate dal browser web verso un server web e consentono di scambiare dati tra il browser e il server in background senza interrompere o ricaricare la visualizzazione della pagina. Le richieste AJAX sono tipicamente utilizzate per ottenere dati aggiornati, inviare dati al server, o eseguire altre operazioni senza creare interruzioni visive per gli utenti.

4.4.4 Sincronizzazione orario tra Raspberry Pi 4 e ESP32

Per garantire che l'orario tra la Raspberry Pi e l'ESP32 sia preciso e sincronizzato, viene utilizzato un componente chiave chiamato client NTP.

Il client NTP, o Network Time Protocol, è un software che consente a un dispositivo di sincronizzare il proprio orario con orologi precisi e affidabili su Internet.

Sulla Raspberry Pi, è stato installato un client NTP che si connette a server NTP noti (ad esempio, 0.pool.ntp.org, 1.pool.ntp.org, 2.pool.ntp.org) per acquisire un orario accurato. Questo assicura che la Raspberry Pi abbia sempre un orologio preciso.

Quando si desidera sincronizzare l'orario tra la Raspberry Pi e l'ESP32, è possibile sfruttare il fatto che la Raspberry Pi dispone di un orologio preciso grazie al client NTP. Viene utilizzato un codice Python (Figura 4.15) sulla Raspberry Pi per ottenere l'orario corrente da inviare all'ESP32.

In questo modo entrambi i dispositivi sono allineati temporalmente.

Di seguito una guida più chiara su come creare un servizio sulla Raspberry Pi che invia automaticamente all'accensione l'orario alla ESP32:

1. **Configurazione del client NTP:** una volta installato il server NTP, come spiegato nella sezione 4.3, è necessario configurarlo per utilizzare server NTP esterni. Quindi bisogna modificare il file di configurazione di NTP. Per aprire il file di configurazione usare il comando:

```
sudo nano /etc/ntp.conf
```

Figura 4.13: File di configurazione

All'interno di questo file, è presente una sezione che permette di specificare i server NTP da utilizzare. Modificare il file aggiungendo i seguenti server NTP e riavviare il servizio per applicare le modifiche:

```
server 0.pool.ntp.org  
server 1.pool.ntp.org  
server 2.pool.ntp.org  
  
sudo service ntp restart
```

Figura 4.14: Configurazione NTP

2. **Creazione di uno script per il servizio:** il codice "*orario.py*", in figura 4.15, legge l'orario da un server NTP ogni 24 ore e lo invia alla ESP32 tramite MQTT. Nel dettaglio:

Capitolo 4 Script e Programmazione dei Dispositivi

- Importa i moduli necessari, inclusi *datetime* per la gestione dell'orario, *paho.mqtt.client* per la connessione MQTT, *ntplib* per la richiesta NTP e *time* per il controllo del tempo.
- Configura la connessione MQTT specificando l'indirizzo del broker MQTT e la porta, nonché il topic MQTT a cui verrà inviato l'orario.
- Definisce una funzione di callback "*on_connect*" che verrà chiamata quando il client MQTT si connette con successo al broker.
- Inizializza il client MQTT, impostando la funzione di callback di connessione e connette il client al broker utilizzando l'indirizzo e la porta specificati.
- Entra in un ciclo while che si ripete ogni 24h per eseguire il processo di acquisizione e invio dell'orario dove ottiene l'orario attuale dal server NTP utilizzando "*ntp_client.request*". Questo valore di orario è un timestamp in formato UNIX. Quindi formatta il timestamp e utilizza il client MQTT per pubblicare l'orario formattato sul topic specificato.

```

from datetime import datetime
import paho.mqtt.client as mqtt
import ntplib
import time

# Configurazione MQTT
mqtt_broker = "192.168.178.188"
mqtt_port = 1883
mqtt_topic = "orario"

# Funzione di callback per quando il client MQTT si connette
def on_connect(client, userdata, flags, rc):
    print("Connesso al broker MQTT")

# Inizializza il client MQTT
client = mqtt.Client()
client.on_connect = on_connect

# Connotti il client MQTT al broker
client.connect(mqtt_broker, mqtt_port, 60)
client.loop_start()

# Inizializza il client NTP
ntp_client = ntplib.NTPClient()

while True:
    try:
        # Ottieni l'orario attuale dal server NTP
        response = ntp_client.request('0.pool.ntp.org', version=3)
        timestamp = response.tx_time
        print("NTP Server response:", response)
        print("Timestamp:", response.tx_time)

        # Formatta l'orario in HH:MM:SS
        formatted_time = datetime.fromtimestamp(timestamp).strftime('%H:%M:%S')
        print("Formatted Time:", formatted_time)

        # Invia l'orario alla ESP32 tramite MQTT
        client.publish(mqtt_topic, formatted_time)

    except ntplib.NTPException as e:
        print("NTP Exception:", e)
    except Exception as e:
        print("An error occurred:", e)

    # Attendi 24 ore prima di inviare nuovamente l'orario
    time.sleep(24 * 60 * 60)

```

Figura 4.15: Script orario.py

3. Creazione servizio systemd:

utilizzare il terminale per creare il file di servizio systemd e inserire il path dello script "orario.py":

```
sudo nano /etc/systemd/system/orario.service
```

```
1 [Unit]
2 Description=Avvio automatico orario.py
3 After=network-online.target
4
5 [Service]
6 ExecStart=/usr/bin/python3 /home/fiore/Desktop/orario_mqtt/orario.py
7 ExecStartPre=/bin/sleep 20
8 WorkingDirectory=/home/fiore/Desktop/orario_mqtt/
9 StandardOutput=inherit
10 StandardError=inherit
11 RestartSec=30
12 User=fiore
13
14 [Install]
15 WantedBy=multi-user.target
16 |
```

Figura 4.16: Servizio orario.service

4. Abilitazione e avvio del servizio: dopo aver salvato il file, abilitare il servizio tramite:

```
sudo systemctl enable orario.service
sudo systemctl start orario.service
```

Figura 4.17: Abilitazione e avvio servizio

Infine per verificare lo stato del servizio:

```
sudo systemctl status orario.service
```

Figura 4.18: Verifica stato sevizio

Questo comando darà le informazioni sullo stato del servizio, e dovrebbe risultare attivo e in esecuzione senza errori, come mostrato in figura 4.19.

Capitolo 4 Script e Programmazione dei Dispositivi

```
fiore@raspberrypi: ~
File Modifica Schede Aiuto
● orario.service - Avvio automatico orario.py
  Loaded: loaded (/etc/systemd/system/orario.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2023-10-09 12:37:50 CEST; 4h 23min ago
    Main PID: 1099 (python3)
       Tasks: 2 (limit: 4915)
         CPU: 3.083s
        CGroup: /system.slice/orario.service
                  └─1099 /usr/bin/python3 /home/fiore/Desktop/orario_mqtt/orario.py

ott 09 12:37:30 raspberrypi systemd[1]: Starting Script per creare una cartella vuota all'avvio...
ott 09 12:37:50 raspberrypi systemd[1]: Started Script per creare una cartella vuota all'avvio.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
lines 1-11/11 (END)
```

Figura 4.19: Stato del servizio

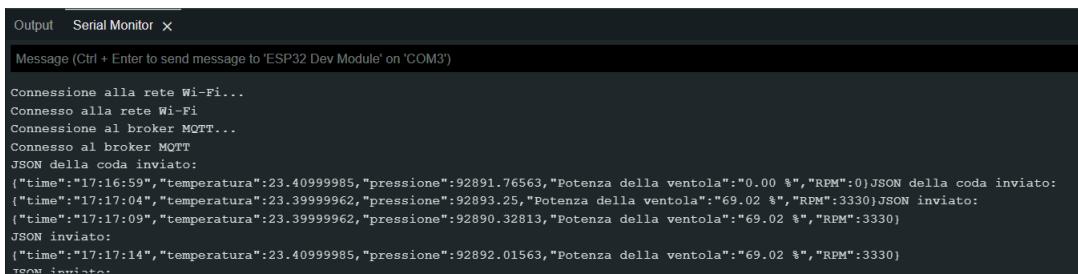
Capitolo 5

Risultati

In questa sezione vengono mostrati i punti chiave del funzionamento complessivo della comunicazione MQTT tra ESP32 e Raspberry Pi.

1. Per l'ESP32:

Appena collegata all'alimentazione, l'ESP32 esegue il codice incorporato. La prima azione consiste nella connessione WiFi, seguita dalla connessione MQTT con la RPi e inizia a inviare i dati di temperatura, pressione e velocità ventola, come illustrato nella figura 5.1.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor". The main area displays the following log output:

```
Output Serial Monitor ×
Message (Ctrl + Enter to send message to 'ESP32 Dev Module' on 'COM3')
Connessione alla rete Wi-Fi...
Connesso alla rete Wi-Fi
Connessione al broker MQTT...
Connesso al broker MQTT
JSON della coda inviato:
{"time": "17:16:59", "temperatura": 23.40999985, "pressione": 92891.76563, "Potenza della ventola": "0.00 %", "RPM": 0} JSON della coda inviato:
{"time": "17:17:04", "temperatura": 23.39999962, "pressione": 92893.25, "Potenza della ventola": "69.02 %", "RPM": 3330} JSON inviato:
{"time": "17:17:09", "temperatura": 23.39999962, "pressione": 92890.32813, "Potenza della ventola": "69.02 %", "RPM": 3330}
JSON inviato:
{"time": "17:17:14", "temperatura": 23.40999985, "pressione": 92892.01563, "Potenza della ventola": "69.02 %", "RPM": 3330}
JSON inviato:
```

Figura 5.1: Monitor esp32

Inoltre mostra i dati (temperatura, pressione, potenza ventola e velocità) sullo schermo OLED, come in figura 5.2.

Capitolo 5 Risultati

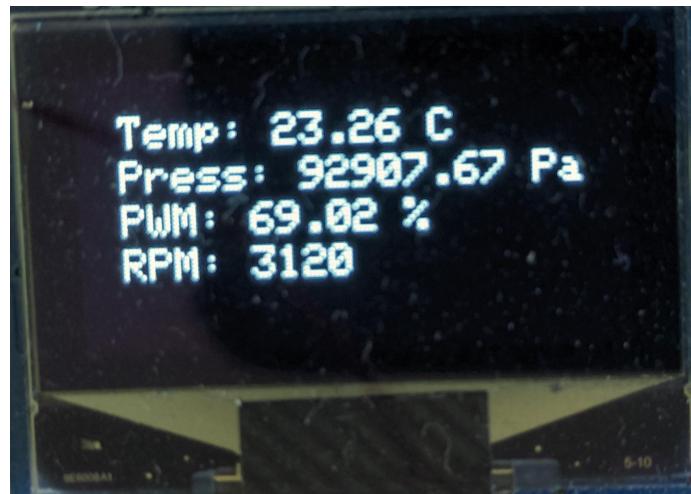
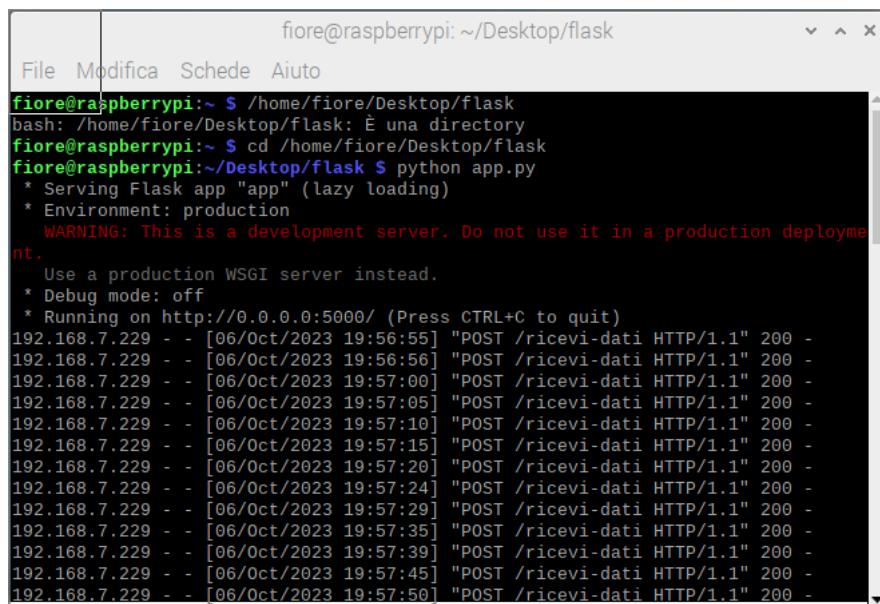


Figura 5.2: Monitor esp32

2. Ricezione dati sulla RPi:

Dopo aver avviato il server Flask, sul terminale viene illustrata la comunicazione MQTT tra ESP32 e RPi.



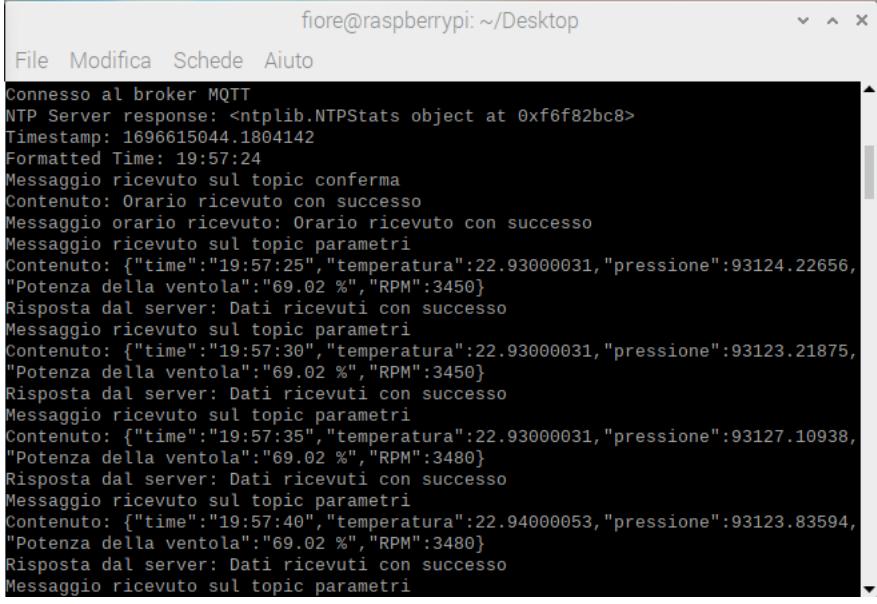
A screenshot of a terminal window titled "fiore@raspberrypi: ~/Desktop/flask". The window contains the following text:

```
File Modifica Schede Aiuto
fiore@raspberrypi:~ $ /home/fiore/Desktop/flask
bash: /home/fiore/Desktop/flask: È una directory
fiore@raspberrypi:~ $ cd /home/fiore/Desktop/flask
fiore@raspberrypi:~/Desktop/flask $ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
192.168.7.229 - - [06/Oct/2023 19:56:55] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:56:56] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:00] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:05] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:10] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:15] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:20] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:24] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:29] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:35] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:39] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:45] "POST /ricevi-dati HTTP/1.1" 200 -
192.168.7.229 - - [06/Oct/2023 19:57:50] "POST /ricevi-dati HTTP/1.1" 200 -
```

Figura 5.3: Server Flask

Mentre su un altro terminale vengono ricevuti i dati dalla ESP32.

Capitolo 5 Risultati



fiore@raspberrypi: ~/Desktop

File Modifica Schede Aiuto

```
Connesso al broker MQTT
NTP Server response: <ntplib.NTPStats object at 0xf6f82bc8>
Timestamp: 1696615044.1804142
Formatted Time: 19:57:24
Messaggio ricevuto sul topic conferma
Contenuto: Orario ricevuto con successo
Messaggio orario ricevuto: Orario ricevuto con successo
Messaggio ricevuto sul topic parametri
Contenuto: {"time": "19:57:25", "temperatura": 22.93000031, "pressione": 93124.22656,
"Potenza della ventola": "69.02 %", "RPM": 3450}
Risposta dal server: Dati ricevuti con successo
Messaggio ricevuto sul topic parametri
Contenuto: {"time": "19:57:30", "temperatura": 22.93000031, "pressione": 93123.21875,
"Potenza della ventola": "69.02 %", "RPM": 3450}
Risposta dal server: Dati ricevuti con successo
Messaggio ricevuto sul topic parametri
Contenuto: {"time": "19:57:35", "temperatura": 22.93000031, "pressione": 93127.10938,
"Potenza della ventola": "69.02 %", "RPM": 3480}
Risposta dal server: Dati ricevuti con successo
Messaggio ricevuto sul topic parametri
Contenuto: {"time": "19:57:40", "temperatura": 22.94000053, "pressione": 93123.83594,
"Potenza della ventola": "69.02 %", "RPM": 3480}
Risposta dal server: Dati ricevuti con successo
Messaggio ricevuto sul topic parametri
```

Figura 5.4: Client MQTT

Come notiamo dalla figura 5.4, prima di visualizzare i dati ricevuti dalla ESP32, compare il messaggio di conferma di orario ricevuto da parte dell'ESP32, questo significa che i due dispositivi sono sincronizzati temporalmente tra di loro.

3. Visualizzazione dati sulla pagina web:

Visitando https://Indirizzo_ip_rpi:5000/pagina, è possibile osservare i parametri che vengono aggiornati automaticamente. Questo conferma che i dati inviati dalla ESP32 vengono ricevuti correttamente e visualizzati sulla pagina web, come chiaramente illustrato nella figura 5.5.

Capitolo 5 Risultati

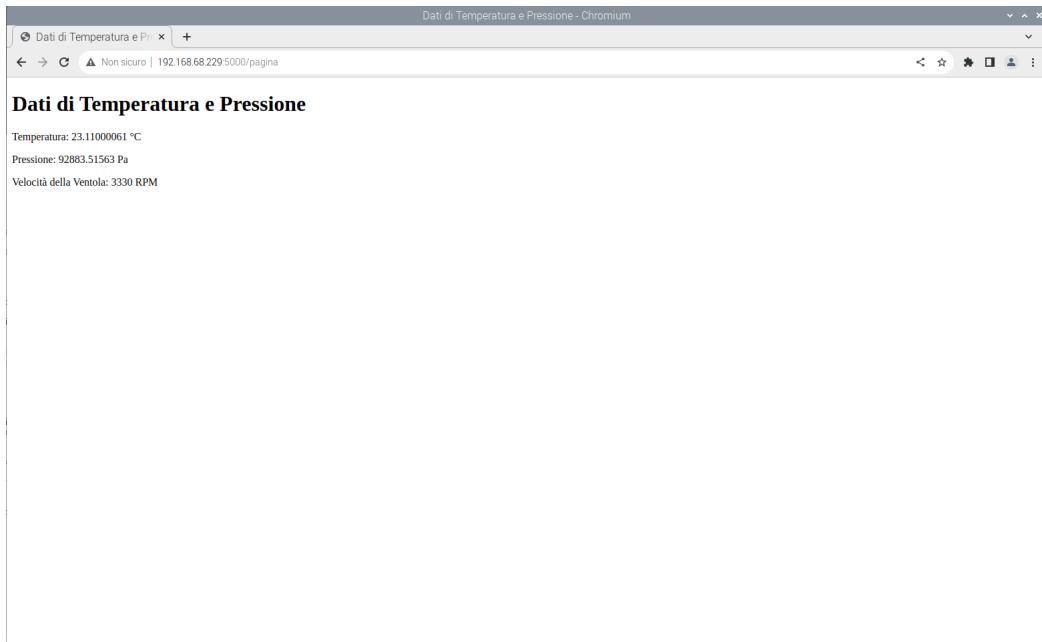
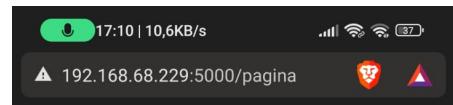


Figura 5.5: Pagina Web

La pagina web è raggiungibile da qualsiasi dispositivo connesso alla stessa rete WiFi, come nell'esempio qui in figura 5.6 dove la pagina web viene mostrata sul browser di uno smartphone:

Capitolo 5 Risultati



Dati di Temperatura e Pressione

Temperatura: 23.30999947 °C

Pressione: 92888.23438 Pa

Velocità della Ventola: 3360 RPM



Figura 5.6: Pagina Web da smartphone

Capitolo 6

Conclusioni e sviluppi futuri

In conclusione, il progetto di monitoraggio delle temperature e della pressione, con il controllo di una ventola di raffreddamento, ha dimostrato l'efficacia dell'integrazione di sensori e dispositivi di controllo in un sistema IoT (Internet of Things). Il sensore ha permesso di rilevare con precisione le variazioni delle condizioni ambientali, consentendo una gestione proattiva del sistema di raffreddamento. L'utilizzo di un server MQTT ha semplificato la comunicazione tra i dispositivi, garantendo una trasmissione affidabile dei dati.

Per lo sviluppo futuro del progetto, ci sono molte opportunità di miglioramento e ampliamento come l'espansione del sistema per monitorare più parametri ambientali, ad esempio l'umidità tramite sensore BME280.

Un'altra possibilità di sviluppo potrebbe essere l'integrazione di un sistema di allarmi che possa avvisare gli utenti in caso di condizioni anomale, ad esempio temperature troppo elevate o basse, o valori di pressione fuori range.

Infine si potrebbe migliorare l'interfaccia utente della pagina web. Questo potrebbe includere l'introduzione di meccanismi di sicurezza per proteggere i dati sensibili e l'accesso al sistema, ad esempio tramite autenticazione utente o crittografia, oppure l'introduzione di grafici dinamici e personalizzati per rendere più intuitiva la visualizzazione dei dati e l'interazione con il sistema, implementando anche un sistema di archiviazione dei dati a lungo termine per analisi storiche e generazione di report.

In definitiva, questo progetto di monitoraggio e controllo di parametri ambientali è un punto di partenza solido per la creazione di sistemi più complessi e sofisticati, adatti a una vasta gamma di applicazioni, dall'automazione domestica all'industria. L'evoluzione costante delle tecnologie IoT offre infinite opportunità per l'innovazione e il miglioramento delle soluzioni esistenti.

Bibliografia

- [1] Espressif Systems. Esp32-devkitc v4 getting started guide. Ottobre 8, 2016.
- [2] Raspberry Pi (Trading) Ltd. Raspberry pi 4 datasheet. Giugno 2019.
- [3] Bosch Sensortec. Data sheet bmp280 digital pressure sensor. Ottobre 15, 2015.
- [4] SOLOMON SYSTECH SEMICONDUCTOR TECHNICAL DATA. Advance information ssd1306 128 x 64 dot matrix oled/pled segment/common driver with controller. Aprile 2008.
- [5] Noctua. Noctua nf-a4x10 5v pwm premium fan datasheet.
- [6] lady ada. Adafruit pcf8523 real time clock. Agosto 2023.
- [7] Massimo Banzi. Betabook, il manuale di arduino. *Apogeo*, 2012.
- [8] code.visualstudio.com. Why visual studio code? 30 marzo 2023.
- [9] raspbian.org. About raspbian.
- [10] FreeRTOS. Freertos documentation. 2016.