

**Sistema IoT per il Monitoraggio Cardiaco e la Qualità
dell'Aria con Raspberry Pi, Realtek AMB23
(RTL8722DM), MAX30102 e SGP30: Node-RED
Dashboard e InfluxDB Database**

Dipartimento di Ingegneria dell'Informazione

Corso: Sistemi Operativi Dedicati

Università Politecnica delle Marche, Ancona



PROFESSORE

Daniele Marcozzi

STUDENTI

Giacinto Masella

Dianel Ago

Indice

1	Introduzione	1
2	Componenti Hardware e Software	3
2.1	Raspberry PI	3
2.2	Scheda Realtek AMB23 (RTL8722DM Mini)	4
2.3	Sensore per la misurazione della qualità d'aria SGP30	5
2.4	Sensore per la misurazione del livello d'ossigeno nel sangue e battito cardiaco MAX30102	6
2.5	Modulo RTC PCF8523	7
2.6	OS Raspbian GNU/Linux (bullseye)	8
2.7	Node-Red	9
2.8	Database InfluxDB	10
3	Sviluppo del progetto	14
3.1	Realizzazione del circuito elettrico	14
3.2	Installazione e configurazione del broker MQTT	16
3.2.1	Configurazione di Mosquitto con username, password e SSL/TLS	18
3.3	Installazione e Configurazione Node-Red	19
3.4	Programmazione dell'Ameba AMB23	25
3.4.1	Installazione e configurazione Arduino IDE	25
3.4.2	Installazione delle librerie	26

INDICE

3.4.3 Implementazione del codice	28
4 Conclusioni	37
4.1 Problemi Riscontrati	37
4.2 Sviluppi Futuri	38
5 Appendice	39
5.1 Codici Script Linux	39
5.2 Codici Script Arduino	43
Sitografia	51

1. Introduzione

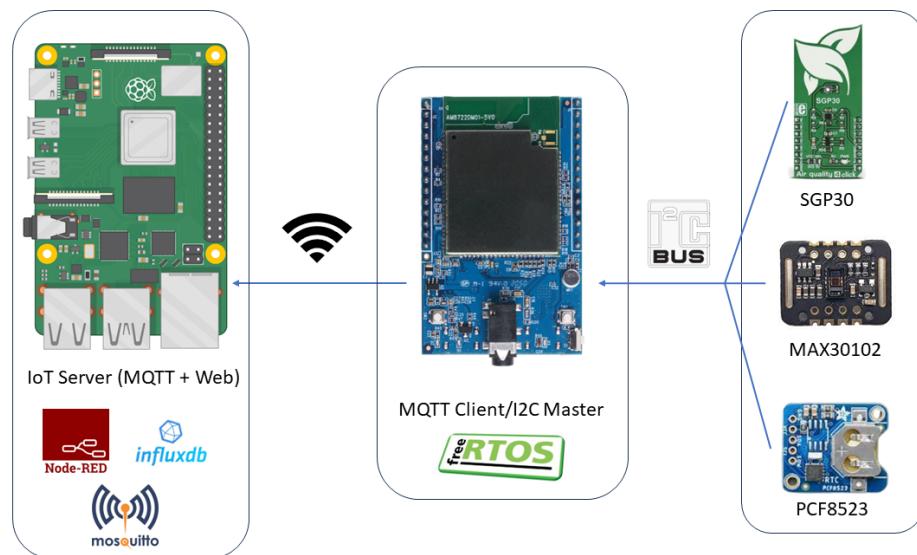


Figura 1.1: Schema IoT.

Il nostro progetto prevede l'utilizzo di Raspberry Pi 4, in combinazione con la scheda Ameba RTL8722DM Mini e diversi sensori, al fine di misurare il livello di saturazione di ossigeno nel sangue e il battito cardiaco (MAX30102), la qualità dell'aria (SGP30) e un modulo RTC (Real-Time Clock) PCF8523 per la gestione della data e dell'ora. L'obiettivo è quello di creare un sistema di monitoraggio IoT in cui i dati raccolti dai sensori vengono visualizzati in tempo reale attraverso una dashboard creata con Node-RED, salvati nel database InfluxDB per l'analisi e protetti mediante la crittografia TLS/SSL.

Inizieremo configurando il Raspberry Pi 4 con il sistema operativo Raspbian, progettato appositamente per questa scheda, in quanto fornisce gli strumenti necessari per lo sviluppo del nostro progetto.

Successivamente, sfrutteremo la scheda Ameba RTL8722DM Mini per leggere i dati dai

CAPITOLO 1: INTRODUZIONE

sensori e inviarli al server broker su Raspberry Pi tramite il protocollo MQTT, in modo da garantire una comunicazione efficiente e a basso consumo energetico.

I sensori utilizzati sono il MAX30102, per misurare la saturazione di ossigeno nel sangue e il battito cardiaco, e il sensore SGP30, impiegato per misurare la qualità dell'aria e in grado di rilevare inquinanti come composti organici volatili e anidride carbonica. Infine, il PCF8523 sarà utilizzato per fornire il timestamp di ogni lettura, sincronizzato con quello della Raspberry Pi.

Al fine di gestire in modo efficiente i task dei sensori e del client MQTT, adotteremo il sistema operativo in tempo reale FreeRTOS. Grazie a FreeRTOS, potremo creare task indipendenti, garantendo una gestione efficiente dei dati e una corretta sincronizzazione.

Per visualizzare in tempo reale i dati raccolti, faremo uso di Node-RED, un ambiente di sviluppo visuale che ci consentirà di creare facilmente flussi di dati e una dashboard personalizzata.

I dati raccolti verranno salvati nel database InfluxDB, ottimizzato per i dati di serie temporali, il che ci permetterà di archiviarli e analizzarli in modo efficiente.

Infine, per garantire la sicurezza dei dati, implementeremo la crittografia TLS/SSL, al fine di proteggere la comunicazione tra i dispositivi e garantire l'autenticazione e la riservatezza dei dati.

In sintesi, il nostro progetto combina Raspberry Pi, la scheda Ameba RTL8722DM Mini, i sensori MAX30102 e SGP30, il modulo RTC PCF8523, FreeRTOS, Node-RED, InfluxDB e la crittografia TLS/SSL per creare un sistema di monitoraggio IoT completo. Grazie a questo sistema, saremo in grado di raccogliere, visualizzare, archiviare e analizzare i dati in tempo reale, contribuendo così a migliorare la qualità della vita e a prendere misure appropriate basate sui dati raccolti.

2. Componenti Hardware e Software

Di seguito vengono elencati i componenti usati con le loro caratteristiche.

2.1 Raspberry PI

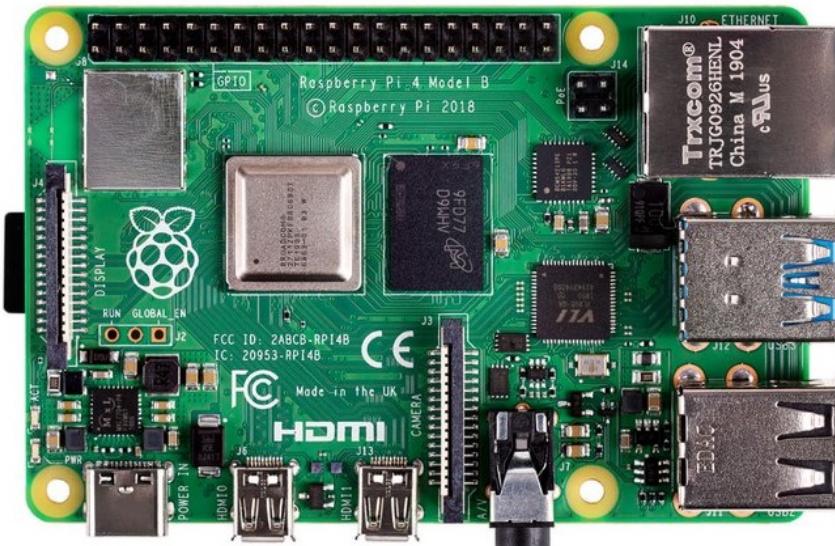


Figura 2.1: Raspberry Pi

Raspberry Pi è una serie di computer a singola scheda progettate per svariati progetti informatici. Sviluppato dalla Raspberry Pi Foundation, offre una soluzione economica per l'apprendimento della programmazione, la prototipazione di dispositivi IoT (Internet delle cose) e l'esecuzione di applicazioni leggere.

Le schede Raspberry Pi presentano un sistema su chip Broadcom con architettura ARM, che offre elevate capacità di elaborazione. Sono dotate anche di varie porte, tra cui USB, HDMI, Ethernet e GPIO (General Purpose Input/Output), consentendo la connessione con periferiche e schede di espansione.

Uno dei principali vantaggi del Raspberry Pi è la sua flessibilità, che consente agli utenti di eseguire diversi sistemi operativi, come Raspbian (basato su Debian), Ubuntu o addirittura distribuzioni specializzate come RetroPie per scopi di gioco. Supporta linguaggi di programmazione come Python e C/C++, rendendolo adatto a una vasta gamma di applicazioni.

La comunità di Raspberry Pi è vivace e diversificata, con ampie risorse online, forum e progetti disponibili. Ciò lo rende una scelta popolare per appassionati, educatori e professionisti, consentendo loro di esplorare e innovare con una piattaforma di calcolo a basso costo.

In particolare, il Raspberry Pi può essere utilizzato per creare un broker MQTT. MQTT è un protocollo di messaggistica leggero ampiamente utilizzato nell'ambito dell'IoT per la comunicazione tra dispositivi. Grazie alla sua potenza di calcolo e alle sue capacità di connettività, il Raspberry Pi può fungere da centralina per gestire il flusso di messaggi MQTT tra i dispositivi IoT.

2.2 Scheda Realtek AMB23 (RTL8722DM Mini)

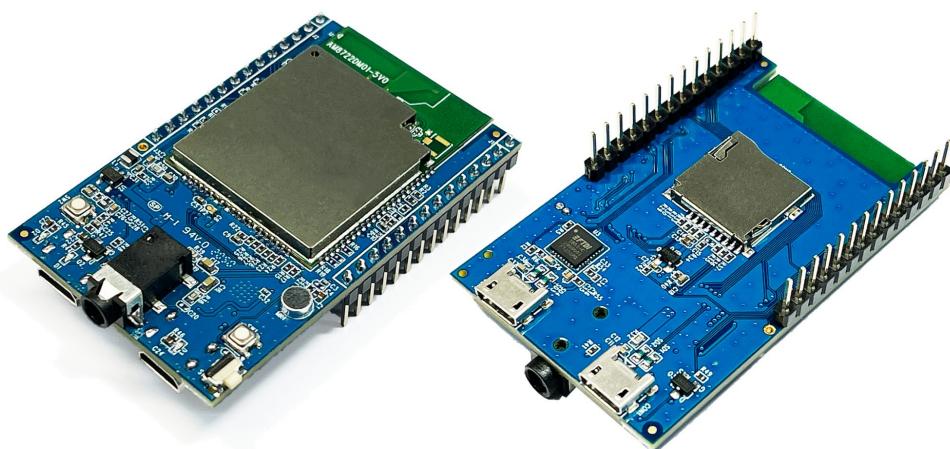


Figura 2.2: Ameba RTL8722DM Mini

La scheda **Ameba RTL8722DM Mini** è una scheda di sviluppo basata sul microcontrollore Realtek RTL8722DM. Questa scheda offre un'ampia gamma di funzionalità e un'architettura versatile che la rende adatta per sviluppare applicazioni IoT.

Il microcontrollore RTL8722DM integra un processore ARM Cortex-M4 ad alta velocità, con connettività Wi-Fi 802.11b/g/n e Bluetooth 5.0, consentendo una facile connessione ai dispositivi IoT e alle reti wireless.

La scheda Ameba RTL8722DM Mini dispone di porte GPIO, UART, SPI e I2C, che offrono flessibilità nella connessione di sensori, attuatori e altri dispositivi esterni. Inoltre, è compatibile con l'ambiente di sviluppo Arduino IDE, semplificando la programmazione e l'integrazione con librerie e sketch Arduino.

Grazie alle sue dimensioni ridotte e al consumo energetico efficiente, la scheda Ameba RTL8722DM Mini è adatta per applicazioni embedded, wearable e IoT a basso consumo.

Utilizzando la scheda Ameba RTL8722DM Mini, è possibile sviluppare progetti IoT avanzati, come sistemi di monitoraggio remoto, domotica, automazione industriale e molto altro ancora.

2.3 Sensore per la misurazione della qualità d'aria SGP30



Figura 2.3: SGP30 Sensirion (Mikroe)

Il sensore **SGP30** è un modulo di rilevamento della qualità dell'aria che fornisce misurazioni precise della concentrazione di composti organici volatili (VOC) e della presenza

di anidride carbonica (CO₂) nell'ambiente circostante.

Dotato di un sensore a semiconduttore MOX (Metal Oxide) avanzato, il SGP30 è in grado di rilevare una vasta gamma di VOC comuni e offre una compensazione automatica della temperatura e dell'umidità per garantire letture accurate.

Il sensore SGP30 è dotato di un'interfaccia di comunicazione I2C, che consente una facile integrazione con microcontrollori e sistemi embedded come Arduino e Raspberry Pi. È supportato da librerie software che semplificano la lettura dei dati e l'interpretazione delle misurazioni.

Con il sensore SGP30, è possibile monitorare la qualità dell'aria in ambienti interni ed esterni, rilevare l'efficacia dei sistemi di ventilazione, valutare l'impatto dell'inquinamento indoor e monitorare la qualità dell'aria all'interno dei dispositivi IoT.

Grazie alle sue dimensioni compatte e alla facilità di utilizzo, il sensore SGP30 è un'opzione ideale per progetti di monitoraggio ambientale, controllo della qualità dell'aria e applicazioni di sicurezza indoor.

2.4 Sensore per la misurazione del livello d'ossigeno nel sangue e battito cardiaco MAX30102

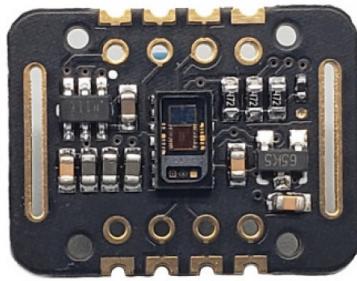


Figura 2.4: MAX30102

Il sensore **MAX30102** è un modulo di rilevamento del battito cardiaco e della saturazione di ossigeno nel sangue (SpO₂) basato sulla tecnologia di riflessione ottica. È in grado di fornire misurazioni accurate e non invasive del battito cardiaco e dei livelli di

ossigenazione.

Utilizzando la tecnologia a due LED (rosso e infrarosso) e un fotosensore, il MAX30102 è in grado di rilevare i cambiamenti di riflessione della luce attraverso il tessuto corporeo. Questi dati vengono quindi elaborati per calcolare il battito cardiaco in battiti al minuto (bpm) e la saturazione di ossigeno nel sangue in percentuale (%).

Il sensore MAX30102 è dotato di un'interfaccia di comunicazione I2C, che consente una facile integrazione con microcontrollori e sistemi embedded come Arduino e Raspberry Pi. Sono disponibili anche librerie software che semplificano la lettura dei dati e l'interpretazione delle misurazioni.

Con il sensore MAX30102, è possibile monitorare il battito cardiaco e la SpO₂ in tempo reale, consentendo il monitoraggio della salute, l'analisi dell'attività fisica e l'utilizzo in dispositivi indossabili per il fitness e il benessere.

Grazie alle sue dimensioni compatte e alla facilità di utilizzo, il sensore MAX30102 è ampiamente utilizzato in applicazioni di monitoraggio medico, fitness e progetti IoT per la salute e il benessere.

2.5 Modulo RTC PCF8523

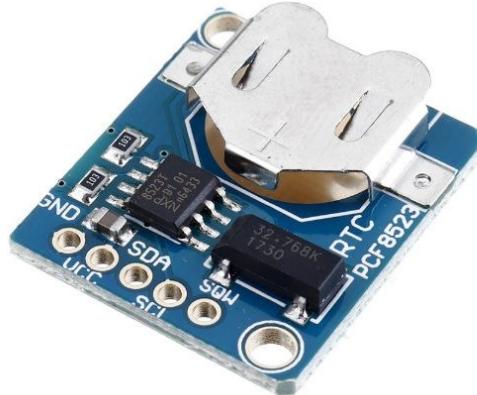


Figura 2.5: RTC PCF8523

Il modulo RTC (Real-Time Clock) **PCF8523** è un dispositivo utilizzato per tenere traccia del tempo in applicazioni embedded e sistemi IoT. Offre un'accurata funzionalità di orologio in tempo reale, che può essere utilizzata per sincronizzare eventi, registrare timestamp e gestire calendari.

Il modulo PCF8523 integra un oscillatore al quarzo e un circuito di gestione dell'orologio che consente la memorizzazione di data e ora anche in assenza di alimentazione. È in grado di mantenere l'orologio sincronizzato grazie alla compensazione automatica di eventuali derivate di frequenza.

Il modulo RTC PCF8523 è dotato di un'interfaccia di comunicazione I2C, che consente una facile integrazione con microcontrollori e sistemi embedded come Arduino e Raspberry Pi. Sono disponibili librerie software che semplificano la configurazione e l'uso del modulo RTC.

Utilizzando il modulo RTC PCF8523, è possibile aggiungere funzionalità di data e ora precise, consentendo la programmazione di eventi basati sul tempo, l'acquisizione di dati sincronizzati e molto altro ancora.

Grazie alle sue dimensioni compatte e alla facilità di utilizzo, il modulo RTC PCF8523 è ampiamente utilizzato in applicazioni IoT, domotica, sistemi di logistica e in tutti quei contesti in cui è necessario gestire il tempo in modo accurato.

2.6 OS Raspbian GNU/Linux (bullseye)

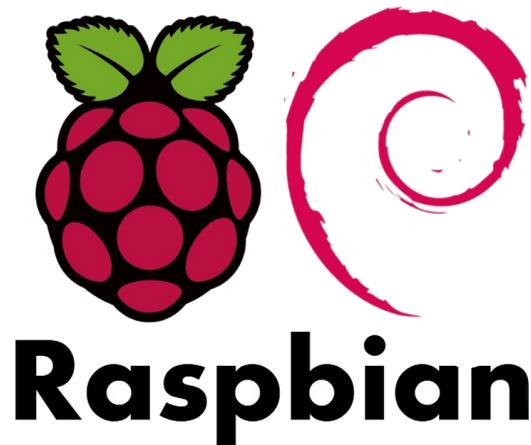


Figura 2.6: Logo Raspbian

Il sistema Linux installato su RPi ha le seguenti caratteristiche:

```
PRETTY_NAME="Raspbian GNU/Linux 11 (bullseye)"
NAME="Raspbian GNU/Linux"
VERSION_ID="11"
```

```

VERSION="11 (bullseye)"

VERSION_CODENAME=bullseye
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"

```

2.7 Node-Red

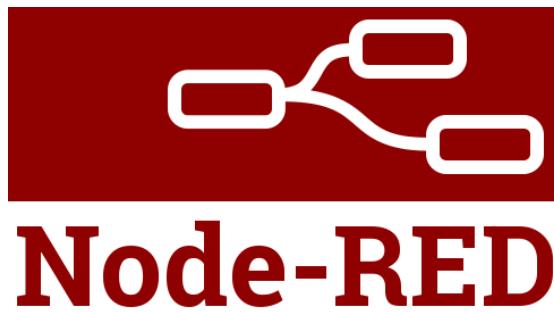


Figura 2.7: Logo Node-Red

Node-RED è un ambiente di sviluppo visuale per la creazione di applicazioni IoT (Internet delle cose) basate su flussi di dati. È basato su Node.js e offre un’interfaccia grafica intuitiva che consente agli utenti di collegare insieme i nodi per creare logicamente le applicazioni.

Node-RED utilizza un approccio a blocchi per la creazione di flussi di dati. Gli utenti possono trascinare e rilasciare nodi predefiniti o personalizzati sulla pagina di progettazione e collegarli insieme per definire il flusso dei dati e le relative azioni. Questo approccio visuale semplifica il processo di sviluppo e permette agli utenti di concentrarsi sulla logica dell’applicazione, senza dover scrivere codice da zero.

Node-RED supporta una vasta gamma di nodi predefiniti che offrono funzionalità per l’interazione con dispositivi IoT, l’elaborazione dei dati, l’accesso a servizi Web, la visualizzazione dei dati e molto altro ancora. Inoltre, gli utenti possono creare i propri nodi personalizzati per estendere ulteriormente le funzionalità di Node-RED.

Node-RED offre un modo potente e intuitivo per sviluppare applicazioni IoT su Raspberry Pi, consentendo agli utenti di creare facilmente flussi di dati complessi e connettere

dispositivi in modo efficace.

2.8 Database InfluxDB



Figura 2.8: Logo InfluxDB

InfluxDB è un database open-source progettato per la memorizzazione e l'elaborazione di dati in tempo reale ad alta velocità. È appositamente progettato per gestire dati di serie temporali, come ad esempio misurazioni di sensori, log di eventi, dati di monitoraggio e altro ancora.

InfluxDB offre una struttura di dati flessibile e scalabile, in grado di gestire volumi elevati di dati di serie temporali in modo efficiente. Supporta la memorizzazione e la consultazione dei dati basati su timestamp, consentendo ricerche rapide e analisi in tempo reale.

Per rendere sicuro InfluxDB, è possibile configurare la comunicazione tramite TLS/SSL. Questo consente di crittografare la connessione tra i client e il server InfluxDB, proteggendo i dati sensibili durante la trasmissione. È possibile generare e utilizzare certificati SSL/TLS per garantire l'autenticazione e la cifratura dei dati.

L'installazione di InfluxDB su Raspbian può essere effettuata seguendo i seguenti passaggi:

1. Apri il terminale sul tuo Raspberry Pi.
2. Aggiorna i pacchetti del sistema eseguendo il comando:

```
sudo apt update
```

3. Installa InfluxDB eseguendo il comando:

```
sudo apt install influxdb
```

4. Avvia il servizio InfluxDB eseguendo il comando:

```
sudo systemctl start influxdb
```

5. Verifica lo stato del servizio InfluxDB con il comando:

```
sudo systemctl status influxdb
```

Assicurati che il servizio sia in esecuzione 'active (running)' correttamente come nell'esempio sotto:

Figura 2.9: Esempio active(running) InfluxDB dal terminale

Si può avviare in automatico al boot di sistema con il comando:

```
sudo systemctl enable influxdb.service
```

6. Per abilitare la comunicazione TLS/SSL, sarà necessario utilizzare dei certificati SSL/TLS. Per generare i suddetti certificati si può utilizzare OpenSSL. Di seguito viene riportata la procedura per la generazione del certificato e della chiave privata:

```
openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout key.pem  
-out cert.pem -subj "/CN=pi" -days 365
```

Questo comando 'openssl' genera un certificato autofirmato (self-signed certificate) in formato X.509 utilizzando una chiave privata RSA a 4096 bit. Il certificato avrà una validità di 365 giorni (1 anno) e sarà identificato dal campo "Common Name" (CN) impostato a "pi".

Ecco una spiegazione passo per passo del comando:

1. **openssl:** È un'utility open-source per la crittografia e la gestione di certificati.

2. **req**: Specifica che l'azione da eseguire è la creazione di una richiesta di certificato (certificate request).
3. **-x509**: Specifica che vogliamo generare un certificato X.509 (auto-firmato).
4. **-newkey rsa:4096**: Genera una nuova coppia di chiavi RSA con una dimensione di 4096 bit. Questa coppia di chiavi verrà utilizzata per firmare il certificato.
5. **-sha256**: Specifica che l'algoritmo di hashing SHA-256 verrà utilizzato per firmare il certificato.
6. **-nodes**: Questa opzione indica che la chiave privata generata non verrà crittografata con una passphrase. In altre parole, il certificato non richiederà una password per essere utilizzato.
7. **-keyout key.pem**: Specifica il nome del file in cui verrà salvata la chiave privata (key.pem). Il file conterrà la chiave privata generata.
8. **-out cert.pem**: Specifica il nome del file in cui verrà salvato il certificato (cert.pem). Il file conterrà il certificato autofirmato.
9. **-subj "/CN=pi"**: Imposta il soggetto (subject) del certificato. In questo caso, viene impostato il campo "Common Name" (CN) a "pi". Il campo "Common Name" identifica il nome del dominio o dell'entità per cui il certificato è valido.
10. **-days 365**: Specifica la durata in giorni del certificato. In questo caso, il certificato sarà valido per 365 giorni (1 anno) dalla data di creazione.

Dopo aver eseguito questo comando, otterrai due file:

key.pem: Contiene la chiave privata generata.

cert.pem: Contiene il certificato autofirmato.

- Copia i certificati generati nella directory corretta, ad esempio:

```
sudo cp cert.pem /etc/influxdb/  
sudo cp key.pem /etc/influxdb/
```

- Modifica il file di configurazione di InfluxDB eseguendo il comando:

```
sudo nano /etc/influxdb/influxdb.conf
```

-Aggiungi o modifica le seguenti opzioni nel file di configurazione:

```
[http]
https-enabled = true
https-certificate = "/etc/influxdb/cert.pem"
https-private-key = "/etc/influxdb/key.pem"
```

Assicurati di utilizzare il percorso corretto per i certificati e le chiavi.

7. Riavvia il servizio InfluxDB eseguendo il comando:

```
sudo systemctl restart influxdb
```

Ora InfluxDB è configurato per utilizzare la comunicazione TLS/SSL. Puoi connetterti al server InfluxDB utilizzando il protocollo HTTPS e la porta corrispondente.

InfluxDB fornisce ulteriori opzioni di configurazione per la sicurezza, come l'autenticazione degli utenti e la gestione dei ruoli.

3. Sviluppo del progetto

Lo sviluppo del progetto è avvenuto principalmente in un primo momento occupandoci della parte hardware e successivamente occupandoci della parte software.

Nello specifico le fasi sono state:

1. realizzazione del circuito elettrico;
2. installazione e configurazione del broker MQTT;
3. sviluppo del codice per la programmazione della Ameba AMB23 attraverso Arduino IDE;
4. installazione e configurazione di Node-RED per la realizzazione della dashboard;
5. installazione di InfluxDB per la creazione di un database contenente lo storico.

Nei paragrafi successivi verranno spiegati nel dettaglio le varie fasi

3.1 Realizzazione del circuito elettrico

La realizzazione del circuito elettrico è avvenuta consultando i datasheets, sui siti proprietari, dei vari componenti hardware utilizzati. I tre sensori (SGP30, MAX30102 e PCF8523) sono stati collegati in parallelo al microcontrollore AMEBA AMB23 attraverso il bus I2C. Questo è stato possibile in quanto gli indirizzi di memoria per l'I2C dei singoli sensori sono differenti e non si crea nessun conflitto tra di essi. In particolare il collegamento è avvenuto in questo modo:

- collegare il pin VCC dei sensori al pin di alimentazione 3.3V dell'Ameba AMB23;
- collegare il pin GND dei sensori al pin di terra dell'Ameba AMB23;

CAPITOLO 3: SVILUPPO DEL PROGETTO

- collegare il pin SDA dei sensori al pin corrispondente SDA dell’Ameba AMB23;
- collegare il pin SCL dei sensori al pin corrispondente SCL dell’Ameba AMB23.

Nella figura seguente è rappresentato, attraverso l’utilizzo del software Fritzing, lo schema elettrico semplificato per una migliore interpretazione e comprensione del circuito elettrico: Da notare bene che nello schema elettrico si utilizza al posto della scheda

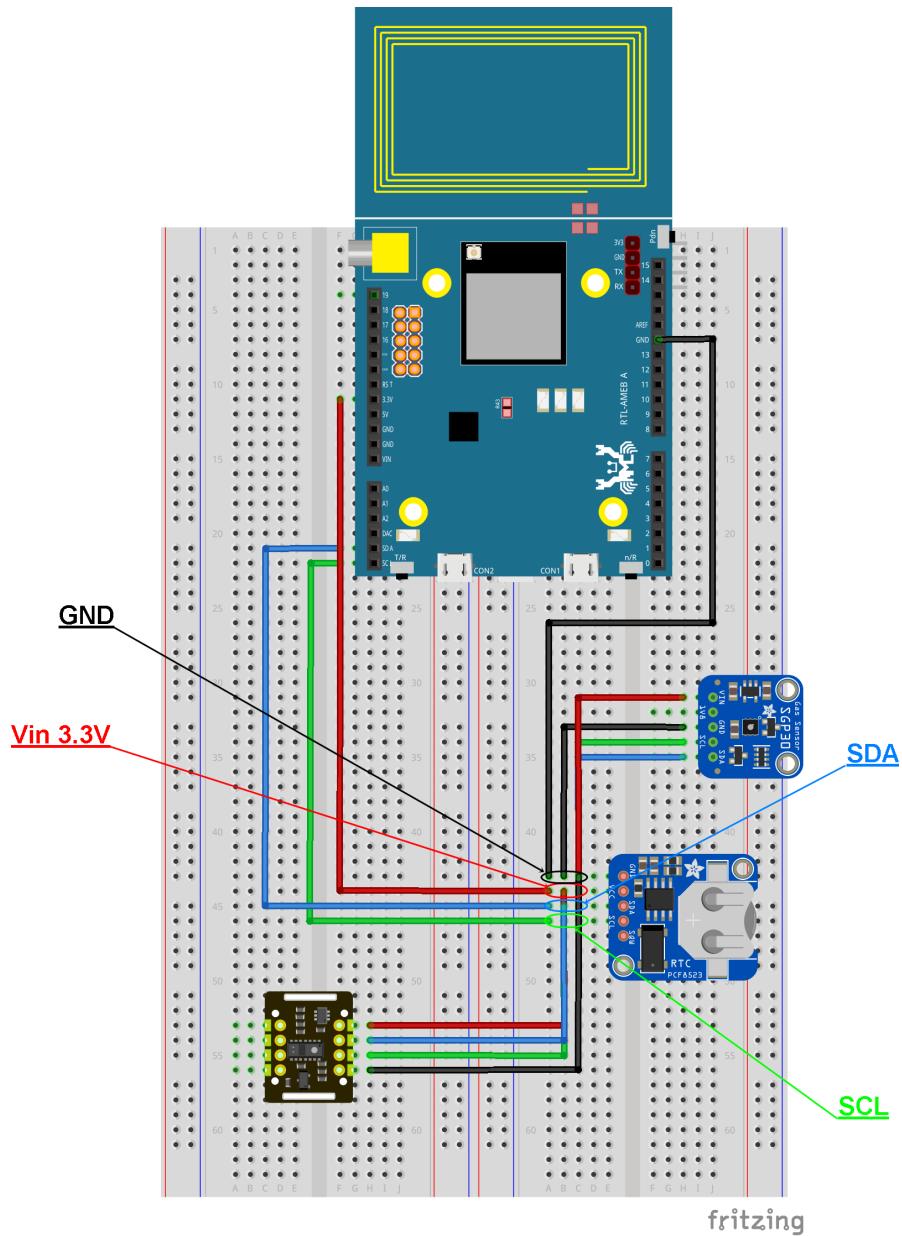


Figura 3.1: Schema elettrico realizzato con il software Fritzing

RTL8722DM un’altra versione simile non avendolo trovato per il software Fritzing. Lo stesso anche per quanto riguarda il sensore SGP30.

La figura seguente rappresenta il sistema reale.

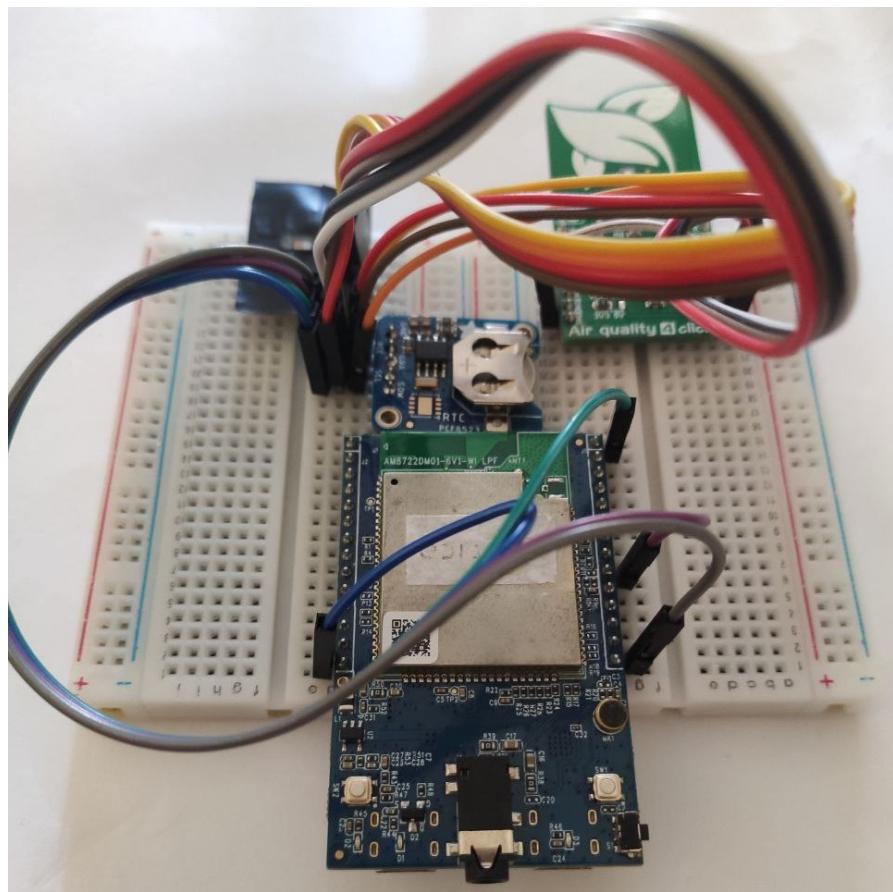


Figura 3.2: Sistema Reale con Collegamenti

3.2 Installazione e configurazione del broker MQTT

In questa sezione viene spiegata l'installazione del broker MQTT su RPi.

Passo 1: Aggiornare il sistema

Prima di iniziare, è consigliabile aggiornare il sistema operativo Raspberry Pi. Apri il terminale e esegui i seguenti comandi:

```
sudo apt update  
sudo apt dist-upgrade
```

Passo 2: Installare il broker MQTT (Mosquitto)

CAPITOLO 3: SVILUPPO DEL PROGETTO

Il broker MQTT più comune per Raspberry Pi è Mosquitto. Per installarlo, esegui il seguente comando:

```
sudo apt install mosquitto
```

Durante l'installazione, ti verrà chiesto di confermare l'installazione del software aggiuntivo necessario. Digita 'Y' per continuare.

Passo 3: Avviare il broker MQTT

Una volta installato Mosquitto, il broker verrà avviato automaticamente. Tuttavia, è sempre buona pratica verificare lo stato del servizio e assicurarsi che sia in esecuzione. Esegui il seguente comando:

```
sudo systemctl status mosquitto
```

Questo comando ti mostrerà lo stato del servizio Mosquitto. Assicurati che sia "active" e "running".

Passo 4: Abilitare l'avvio automatico del broker MQTT

Se il broker Mosquitto non viene avviato automaticamente all'avvio del Raspberry Pi, è possibile abilitare l'avvio automatico. Esegui il seguente comando:

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

Passo 5: Verificare il broker MQTT

Per verificare che il broker MQTT sia in esecuzione correttamente, puoi utilizzare il comando `mosquitto_sub` per sottoscriverti a un argomento (topic) e `mosquitto_pub` per pubblicare un messaggio su un argomento. Ecco un esempio:

Apri due finestre del terminale separate sul Raspberry Pi.

Nella prima finestra, esegui il seguente comando per sottoscriverti ad un topic di test chiamato 'testtopic':

```
mosquitto_sub -h localhost -t testtopic
```

Nella seconda finestra, esegui il seguente comando per pubblicare un messaggio sul topic "testtopic":

```
mosquitto_pub -h localhost -t testtopic -m "Ciao"
```

Se tutto funziona correttamente, dovrà vedere il messaggio "Ciao" apparire nella prima finestra del terminale dove ti sei sottoscritto al topic "testopic".

Successivamente si va a configurare la sicurezza SSL/TLS per MQTT e definendo nome utente e password di autenticazione.

3.2.1 Configurazione di Mosquitto con username, password e SSL/-TLS

Per rendere il broker MQTT (Mosquitto) sicuro e abilitare l'autenticazione con username e password, segui i passaggi seguenti:

Genera i certificati SSL/TLS

Innanzitutto, dovrà generare i certificati SSL/TLS per il server MQTT. Puoi farlo utilizzando lo strumento `openssl`. Assicurati di avere `openssl` installato sul tuo sistema, se nel caso non è stato installato allora si fa con il seguente comando:

```
sudo apt-get install openssl
```

Si crea una directory vuota

```
mkdir mqtt_tls
cd mqtt_tls

# Genera una chiave privata per il broker
openssl genrsa -out server.key 2048

# Genera una richiesta di certificato (CSR)
openssl req -new -key server.key -out server.csr

# Autofirma la CSR per creare un certificato autofirmato
openssl x509 -req -in server.csr -signkey
server.key -out server.crt

# Converti il certificato in un formato compatibile
# per Mosquitto
cat server.crt server.key > dhparam.pem
```

Configurazione di Mosquitto

Apri il file di configurazione di Mosquitto:

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

Aggiungi le seguenti righe al file di configurazione:

```
# Configurazione SSL/TLS
cafile /home/pi/mqtt_tls/server.crt
certfile /home/pi/mqtt_tls/server.crt
keyfile /home/pi/mqtt_tls/server.key

# Abilita l'autenticazione con username e password
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Salva il file e chiudi l'editor.

Aggiungi utenti e password

Mediante il comando 'mosquitto_passwd' si potranno creare gli utenti abilitati all'utilizzo del broker MQTT. Nel seguente esempio si crea l'utente 'ameba' e si salvano le credenziali nel file /etc/mosquitto/passwd

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd ameba
```

Verrà chiesto di inserire la password per ameba. Ripeti questo comando per aggiungere altri utenti.

Riavvia Mosquitto

Riavvia il servizio Mosquitto affinché le modifiche alla configurazione abbiano effetto.

```
sudo systemctl restart mosquitto.service
```

3.3 Installazione e Configurazione Node-Red

Di seguito si andrà a vedere l'installazione del Node-Red su RPi in modalità sicura.

Passo 1: Installazione di Node-RED

CAPITOLO 3: SVILUPPO DEL PROGETTO

Per prima cosa, assicurati che il tuo Raspberry Pi sia aggiornato con i comandi che si sono visti anche precedentemente.

Una volta completato l'aggiornamento, si procede con l'installazione di Node-RED. Il seguente comando esegue uno script di installazione per Node.js e Node-RED su sistemi operativi basati su Debian

```
bash < (curl -sL  
https://raw.githubusercontent.com/node-red/linux-installers/master/  
deb/update-nodejs-and-nodered)
```

Durante la procedura di installazione vi verranno poste le domande riportate in seguito. Rispondere in modo affermativo (y) ad entrambe.

```
Are you really sure you want to do this ? [y/N] ?  
Would you like to install the Pi-specific nodes ? [y/N] ?
```

Questo script farà le seguenti operazioni:

1. Rimuoverà la versione esistente di Node-RED se presente.
2. Se rileva che Node.js è già installato, verificherà che sia almeno la versione 14. Se la versione è inferiore a 14, lo script si fermerà e permetterà all'utente di decidere se mantenere la versione 1 di Node.js o eseguire l'aggiornamento a una versione più recente di Node.js con supporto a lungo termine (LTS). Se non viene trovata alcuna installazione di Node.js, installerà la versione 16 LTS utilizzando il pacchetto NodeSource.
3. Installa l'ultima versione di Node-RED utilizzando npm (Node Package Manager).
4. Opzionalmente, installerà una collezione di nodi specifici per il Raspberry Pi, che potrebbero essere utili per sfruttare le funzionalità specifiche di questo dispositivo.
5. Configurerà Node-RED per essere eseguito come servizio e fornirà un set di comandi per interagire con il servizio, come avviarlo, fermarlo e riavviarlo. In questo modo, Node-RED sarà eseguito automaticamente come servizio all'avvio del sistema.

Passo 2: Configurazione di Node-RED

CAPITOLO 3: SVILUPPO DEL PROGETTO

Dopo l'installazione, puoi avviare Node-RED eseguendo il seguente comando:

```
node-red-pi
```

Node-RED sarà ora accessibile all'indirizzo `http://<indirizzo_ip_raspberry>:1880` nel browser.

Passo 3: Generazione dei certificati e sicurezza SSL/TLS per Node-RED su Raspberry Pi

Generazione dei certificati SSL/TLS Per abilitare la sicurezza SSL/TLS su Node-RED, sarà necessario avere un certificato SSL e una chiave privata. Si può generare un certificato autofirmato (adatto per l'uso interno o di sviluppo) o ottenere un certificato firmato da un'autorità di certificazione (CA) per un utilizzo più sicuro.

Per generare un certificato autofirmato, è possibile utilizzare lo strumento `openssl`. Si crea una directory sulla home di raspberry con il comando:

```
mkdir node-red_ssl  
#Posizionarsi sulla directory appena creata  
cd node-red_ssl
```

Con i due comandi seguenti si genera una chiave privata e un certificato autofirmato

```
# Generazione della chiave privata  
openssl genrsa -out key.pem 2048  
# Generazione del certificato autofirmato  
openssl req -new -x509 -sha256 -key key.pem -out cert.pem -days  
365
```

Durante il processo di generazione del certificato, verranno richieste alcune informazioni, come il nome del proprio Raspberry Pi, paese e l'indirizzo e-mail. Si può fornire valori appropriati o lasciarli vuoti per scopi di sviluppo.

Configurazione di Node-RED per utilizzare SSL/TLS Una volta generati i certificati, sarà necessario configurare Node-RED per utilizzarli. Aprire il file di configurazione di Node-RED (solitamente chiamato `settings.js`) con il seguente comando:

```
nano ~/.node-red/settings.js
```

CAPITOLO 3: SVILUPPO DEL PROGETTO

Cercare la sezione che inizia con // The ‘https’ setting requires the ‘fs’ module e seguire le istruzioni per abilitare SSL/TLS.

Configurare i percorsi dei file chiave e certificato nella sezione https come segue:

```
https: {  
    key: require('fs').readFileSync('/home/pi/node-red_ssl/key.pem'),  
    cert: require('fs').readFileSync('/home/pi/node-red_ssl/cert.pem')  
},
```

Assicurarsi di scrivere bene i percorsi completi dei file `key.pem` e `cert.pem` che sono stati generati in precedenza, nel nostro caso `/home/pi/node-red_ssl`.

Riavvio di Node-RED Salvare le modifiche apportate al file di configurazione e riavviare Node-RED affinché le nuove impostazioni siano applicate:

```
node-red-restart
```

Accesso a Node-RED tramite HTTPS Ora Node-RED sarà accessibile tramite HTTPS. Si può accedere utilizzando un browser Web all’indirizzo `https://indirizzo_del_tuo_raspberry:1880`. Durante la connessione, il browser potrebbe avvisare che il certificato è autogenerato e non firmato da una CA. Questo è normale con i certificati autofirmati, quindi si può procedere accettando l’eccezione di sicurezza.

Ora si è abilitata la sicurezza SSL/TLS per Node-RED sul proprio Raspberry Pi. Assicurarsi di proteggere i certificati e la chiave privata, poiché sono critici per garantire una comunicazione sicura. Se si prevede di utilizzare Node-RED in un ambiente di produzione o pubblico, si consiglia di ottenere un certificato firmato da una CA affidabile per garantire la massima sicurezza.

Oltre alle librerie con i nodi pre-installate su Node-Red vengono installate le seguenti librerie manualmente:

- node-red-contrib-influxdb
- node-red-contrib-toggle
- node-red-contrib-ui-digital-clock

CAPITOLO 3: SVILUPPO DEL PROGETTO

- node-red-dashboard
- node-red-node-ui-table
- node-red-contrib-influxdb

Per installare librerie in Node-RED, puoi seguire questa procedura:

1. Avvia Node-RED.
2. Apri l'editor di Node-RED nel tuo browser, accedendo all'indirizzo <https://localhost:1880>.
3. Nel menu in alto a destra, clicca sull'icona con tre righe orizzontali per aprire il menu principale.
4. Seleziona "Manage palette" dal menu.
5. Verrà aperta la finestra "Palette Manager", che ti permette di gestire le librerie installate su Node-RED.
6. Nella scheda "Install", puoi cercare le librerie disponibili. Puoi digitare il nome della libreria nella casella di ricerca o sfogliare le librerie disponibili.
7. Una volta trovata la libreria che vuoi installare, clicca sul pulsante "Install" accanto ad essa.
8. Verrà visualizzato un avviso di installazione. Conferma l'installazione cliccando sul pulsante "Install" nella finestra di dialogo.
9. Node-RED scaricherà e installerà automaticamente la libreria selezionata. Potrebbe essere richiesto di riavviare Node-RED per completare l'installazione.
10. Dopo l'installazione, la libreria sarà disponibile per l'uso su Node-RED.

Dopo aver configurato Node-Red si va sull'indirizzo precedentemente specificato e vediamo l'UI admin di Node-Red. Per accedere invece alla dashboard bisogna aggiungere /ui alla fine dell'indirizzo completo. Di seguito si mostra una screenshot di tutti i nodi implementati:

Si hanno le seguenti funzionalità:

CAPITOLO 3: SVILUPPO DEL PROGETTO

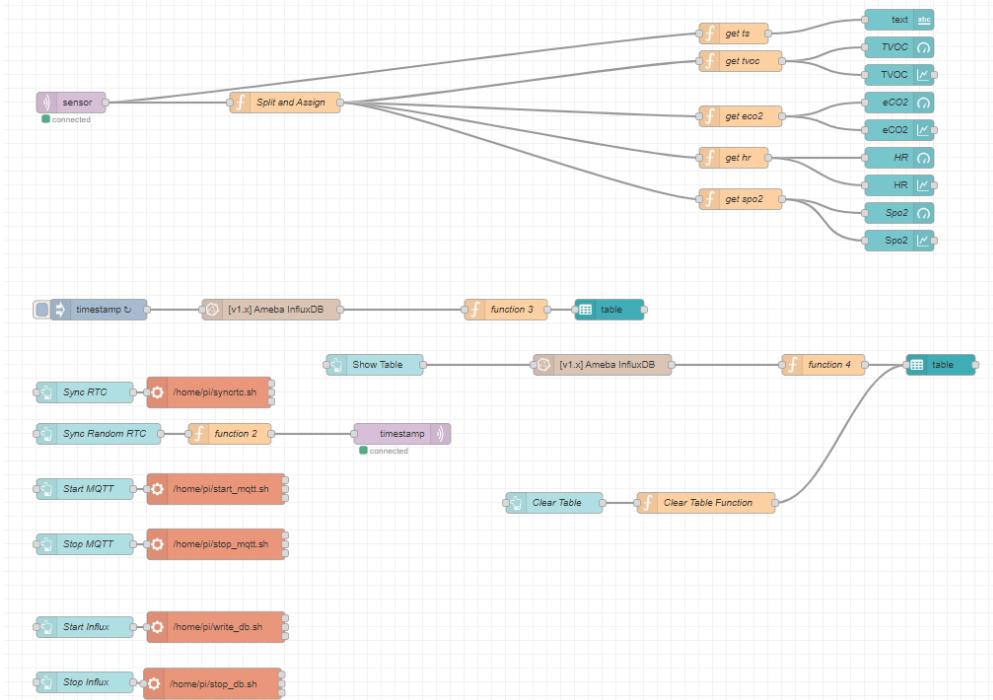


Figura 3.3: Schema Node-Red

- Flow che legge dal topic sensor per i dati in real-time
- Tasti con script per attivare/disattivare Mqtt
- Flow che fa vedere i dati in una tabella
- Flow che fa il parsing di tutti i dati nella database
- Flow con i button per sincronizzare RTC, dati tabella...
- Tasti con script per attivare/disattivare la scrittura su InfluxDB

Di seguito si vede la dashboard completa:

CAPITOLO 3: SVILUPPO DEL PROGETTO

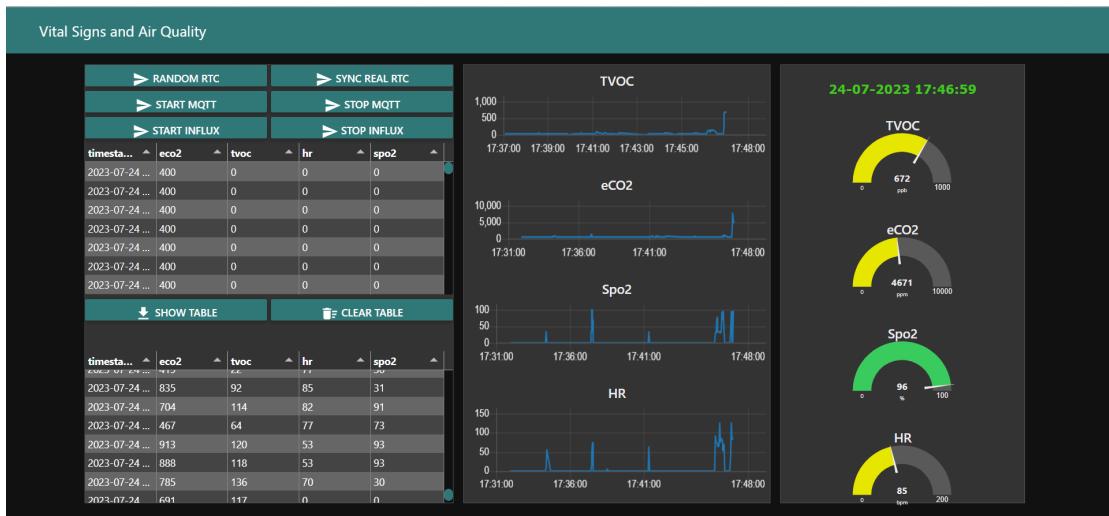


Figura 3.4: Dashboard

3.4 Programmazione dell'Ameba AMB23

Di seguito viene descritta l'implementazione software del microcontrollore Ameba AMB23.

Possiamo suddividere l'implementazione in varie fasi:

1. Installazione e configurazione dell'IDE Arduino;
2. Installazione delle librerie
3. Implementazione del codice

3.4.1 Installazione e configurazione Arduino IDE

Per prima cosa è necessario collegarsi nella sezione "download" della pagina ufficiale di Arduino e scaricare il software in base al sistema operativo utilizzato. Nel nostro caso abbiamo installato la versione 2.1.1 per Windows 10. Dopo aver installato il software bisogna configurare l'ambiente di sviluppo per poter lavorare con la scheda Ameba AMB23. Per prima cosa andiamo su "File/Preferences" e nel campo "Additional boards manager URLs" inseriamo il seguente link: https://github.com/ambiot/ambd_arduino/raw/master/Arduino_package/package_realtek.com_amebad_index.json.

In seguito andiamo su "Tools/Boards/Boards Manager" e installiamo "Realtek Ameba Boards (32-bits ARM Cortex-M33 @200 MHz)". A questo punto abbiamo installato e configurato l'ambiente di sviluppo per Ameba AMB23, possiamo passare all'installazione delle librerie e in seguito all'implementazione del codice.

3.4.2 Installazione delle librerie

Le librerie utilizzate nel nostro progetto sono:

- FreeRTOS
- WiFi
- PubSubClient
- Adafruit SGP30
- Vega MAX30102
- RTCLib

Nei prossimi paragrafi andiamo a descrivere la funzione di ogni singola libreria utilizzata nel nostro progetto.

FreeRTOS

FreeRTOS (Real-Time Operating System) è una libreria open-source che fornisce un sistema operativo a tempo reale per microcontrollori e microprocessori. Questa libreria è stata sviluppata per consentire la gestione efficiente e affidabile dei thread, permettendo l'esecuzione concorrente di più attività su un singolo microcontrollore. Essa è progettata per essere portabile e può essere utilizzata con diverse architetture hardware, tra cui Ameba AMB23.

Ecco una spiegazione di alcuni concetti chiave e funzionalità di FreeRTOS:

- Task

Un task rappresenta un'attività o un thread indipendente che viene eseguito contemporaneamente con altre attività nel sistema. FreeRTOS consente di creare più task e specificare la loro priorità. Il kernel dell'RTOS si occupa della gestione del cambio di contesto tra i task e dell'assegnazione del tempo di esecuzione in base alle priorità.

- Scheduler

Lo scheduler è una parte fondamentale del sistema operativo e gestisce la selezione del task da eseguire in un dato momento. Esistono diversi algoritmi di scheduling,

ma FreeRTOS utilizza un approccio preemption-based, dove i task con priorità più alta possono interrompere quelli con priorità inferiore.

- **Semafori**

I semafori sono meccanismi di sincronizzazione utilizzati per gestire l'accesso corrente a risorse condivise tra task. Essi aiutano a prevenire situazioni di race condition e a garantire l'accesso esclusivo alle risorse.

- **Code di Messaggi**

Le code di messaggi consentono la comunicazione tra i task. Un task può inviare un messaggio a una coda, e un altro task può ricevere e elaborare il messaggio dalla stessa coda. Questo meccanismo è utile per scambiare dati tra task in modo sincrono.

WiFi

La libreria WiFi fornisce una serie di funzioni che consentono ai dispositivi di connettersi a reti Wi-Fi e di comunicare tramite il protocollo di rete TCP/IP. Questa libreria è fondamentale per abilitare la connettività Wi-Fi e permettere al microcontrollore di interagire con altri componenti o servizi online.

PubSubClient

La libreria PubSubClient è utilizzata per implementare la comunicazione tramite il protocollo MQTT (Message Queuing Telemetry Transport). Questa libreria permette al microcontrollore di agire come client MQTT, inviando e ricevendo messaggi da un broker MQTT.

Adafruit SGP30

La libreria Adafruit SGP30 è sviluppata per supportare il sensore di gas digitale SGP30. Questo sensore è in grado di rilevare concentrazioni di anidride carbonica (CO₂) e composti organici volatili (COV) nell'ambiente circostante. La libreria semplifica l'interfacciamento con il sensore e permette agli sviluppatori di ottenere rapidamente i dati ambientali forniti dal SGP30.

Vega MAX30102

La libreria Vega MAX30102 è utilizzata per accedere alle funzionalità di lettura dei dati di frequenza cardiaca e ossigenazione del sangue (SpO2) offerte dal sensore MAX30102.

RTCLib

La libreria "RTCLib" (Real Time Clock Library) è sviluppata per fornire funzionalità di gestione del tempo e del calendario. Essa consente di interagire con i moduli RTC (Real Time Clock) e utilizzare il microcontrollore per tenere traccia del tempo in modo preciso, anche quando il microcontrollore è spento o viene riavviato.

3.4.3 Implementazione del codice

Di seguito è descritta l'implementazione del codice con le principali funzionalità.

Librerie incluse Le prime righe di codice includono le librerie necessarie per il progetto. Queste librerie forniscono le funzioni e le definizioni necessarie per gestire i sensori, la connessione WiFi e MQTT e per l'utilizzo del sistema operativo FreeRTOS.

```
// Include libraries
#include <FreeRTOS.h>
#include <queue.h>
#include <task.h>
#include <Wire.h>
#include <Adafruit_SGP30.h>
#include "MAX30102_PulseOximeter.h"
#include <RTCLib.h>
#include <WiFi.h>
#include <PubSubClient.h>
```

Definizione delle credenziali di rete e del broker MQTT In questa sezione, vengono definite le credenziali per la connessione WiFi e i parametri per la connessione al broker MQTT. Questi parametri vengono utilizzati successivamente per stabilire la connessione con la rete e il broker MQTT.

```
// WiFi credentials
```

```

char* ssid = "****";
char* password = "****";

// MQTT broker information
char* mqttBroker = "****";
int mqttPort = 1883;
char* mqttClientId = "amebaClient";
char* mqttUsername = "****";
char* mqttPassword = "****";

```

Definizione dei topic MQTT Vengono definiti i topic MQTT che saranno utilizzati per inviare e ricevere i dati dal broker MQTT. I topic specificano i canali di comunicazione in cui i dati saranno pubblicati o da cui saranno sottoscritti.

```

// Topic definition
char* mqttSensorTopic = "sensor";
char* mqttSyncTopic = "timestamp";
char* mqttHistoryTopic = "history";

```

Inizializzazione e creazione delle code e del semaforo Vengono create due code, sensorQueue e historyQueue, utilizzate per memorizzare i dati acquisiti dai sensori e i dati storici. Viene anche creato un semaforo binario, binarySemaphore, utilizzato per garantire l'accesso esclusivo ai dati dei sensori tra i vari task.

```

// Queues
QueueHandle_t sensorQueue;
QueueHandle_t historyQueue;

// Binary semaphore
SemaphoreHandle_t binarySemaphore;

// Create queues
sensorQueue = xQueueCreate(1, sizeof(SensorData));
historyQueue = xQueueCreate(100, sizeof(SensorData));

// Create binary semaphore
binarySemaphore = xSemaphoreCreateBinary();

```

Inizializzazione dei sensori La funzione sensorInit() è responsabile per l'inizializzazione dei sensori MAX30102 e SGP30. Se i sensori non vengono inizializzati correttamente (ad esempio, se ci sono problemi di comunicazione con i sensori), il programma si bloccherà con un messaggio di errore.

```
// Sensors initialization procedure
void sensorInit() {
    // Initialize MAX30102
    if (!pox.begin()) {
        Serial.println("Failed to initialize MAX30102!");
        while (1);
    } else {
        Serial.println("MAX30102 initialized!");
    }
    pox.setIRLedCurrent(MAX30102_LED_CURR_7_6MA);

    // Initialize SGP30
    if (!sgp30.begin()) {
        Serial.println("Failed to initialize SGP30!");
        while (1);
    } else {
        Serial.println("SGP30 initialized!");
    }
}
```

Inizializzazione di WiFi e sincronizzazione del tempo La funzione wifiInit() si occupa di connettersi alla rete WiFi specificata. Continua a tentare la connessione fino a quando non riesce a connettersi con successo. La sincronizzazione del tempo avviene attraverso la funzione syncFT(), la quale utilizza il modulo RTC per sincronizzare il tempo ricevuto attraverso MQTT quando il dispositivo si connette per la prima volta.

```
// WiFi initialization procedure
void wifiInit() {
    // Connect to WiFi
    while (WiFi.status() != WL_CONNECTED) {
        Serial.println("\nConnecting to WiFi...");
        WiFi.begin(ssid, password);
        delay(10000);
    }
    Serial.println("\n\nConnected to WiFi!\n");
```

```

}

void syncFT() {
    // Initialize PCF8523
    if (!rtc.begin()) {
        Serial.println("Failed to initialize PCF8523!");
        while (1);
    } else {
        Serial.println("PCF8523 initialized!");
        rtc.start();
    }

    while (isFirst) {
        if (WiFi.status() != WL_CONNECTED) {
            WiFi.begin(ssid, password);
            delay(5000);
        }

        if (!reconnectMQTT()) { delay(5000); }
        mqttClient.loop();
    }
}

```

Task per i sensori Il task sensorTask acquisisce i dati dai sensori e li invia alla coda sensorQueue. In questa funzione, vengono letti i valori della frequenza cardiaca, della saturazione di ossigeno nel sangue, delle concentrazioni di gas TVOC e CO2, e viene ottenuto il timestamp corrente dal modulo RTC. I dati acquisiti vengono quindi inviati alla coda sensorQueue.

```

// Sensor task: read data from sensors and send them through queue to MQTT task
void sensorTask(void* params) {

    (void) params;
    SensorData sensorData;
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();

    while (1) {

```

CAPITOLO 3: SVILUPPO DEL PROGETTO

```
// Take semaphore
if (xQueueSemaphoreTake(binarySemaphore, portMAX_DELAY) == pdTRUE) {

    // Read sensors data
    pox.update();
    sensorData.hr = pox.getHeartRate();
    sensorData.spo2 = pox.getSpO2();

    if (!sgp30.IAQmeasure()) {
        Serial.println("Failed to measure IAQ!");
        while (1);

    } else {
        sensorData.tvoc = sgp30.TVOC;
        sensorData.eco2 = sgp30.eCO2;
    }

    sensorData.currentTime = rtc.now().unixtime();

    // Give semaphore
    xSemaphoreGive(binarySemaphore);
}

// Send sensors data through queue
xQueueOverwrite(sensorQueue, &sensorData);

// Block task for a fixed period
vTaskDelayUntil(&xLastWakeTime, delaySensor);
}
```

Task MQTT Il task mqttTask gestisce la connessione MQTT e invia i dati acquisiti dal task dei sensori ai relativi topic MQTT. Se la connessione MQTT è attiva, i dati vengono prelevati dalla coda sensorQueue e inviati tramite messaggi MQTT ai rispettivi topic. Se la connessione MQTT non è attiva, i dati vengono comunque letti dalla coda sensorQueue e memorizzati temporaneamente nella coda historyQueue per essere inviati successivamente quando la connessione verrà ripristinata.

CAPITOLO 3: SVILUPPO DEL PROGETTO

```
/*
MQTT task:
1. Check if WiFi & MQTT are connected
2. Receive timestamp from MQTT topic and sync time on PCF8523
3. Receive sensors data through queue from Sensor task and send it as message
to both MQTT topic
4. If MQTT is disconnected save data into queue
*/
void mqttTask(void* params) {

(void) params;
SensorData sensorData;
unsigned long currentMills;
TickType_t interval;
TickType_t xLastWakeTime = xTaskGetTickCount();

while (1) {

// If MQTT is disconnected
if (!mqttClient.connected()) {

reconnectMQTT();

// Save data from sensor queue to history queue, if history queue is full
// overwrite the oldest value
if (xQueueReceive(sensorQueue, &sensorData, 0) == pdPASS) {
if (xQueueSend(historyQueue, &sensorData, 0) == errQUEUE_FULL) {
xQueueReceive(historyQueue, &sensorData, 0);
}
}
}

interval = delayMqttDc;

// If MQTT is connected
} else {

unsigned long previousMills = 0;
// Empty history queue as fast as possible
while (uxQueueMessagesWaiting(historyQueue) > 0) {
currentMills = millis();
```

CAPITOLO 3: SVILUPPO DEL PROGETTO

```
    if (currentMills - previousMills >= delayEmpty) {
        previousMills = currentMills;
        if (xQueueReceive(historyQueue, &sensorData, 0) == pdPASS) {
            char historyMess[100];
            sprintf(historyMess, sizeof(historyMess), "%d,%d,%d,%d,%d",
                    sensorData.tvoc, sensorData.eco2, (int)sensorData.hr, sensorData.spo2,
                    sensorData.currentTime);
            mqttClient.publish(mqttHistoryTopic, historyMess);
        }
    }
}

// Publish data on both topic
if (xQueueReceive(sensorQueue, &sensorData, 0) == pdPASS) {
    char sensorMess[100];
    sprintf(sensorMess, sizeof(sensorMess), "%d,%d,%d,%d,%d",
            sensorData.tvoc, sensorData.eco2, (int)sensorData.hr, sensorData.spo2, sensorData.
            currentTime);
    mqttClient.publish(mqttSensorTopic, sensorMess);
    mqttClient.publish(mqttHistoryTopic, sensorMess);
}

interval = delayMqttC;

}

vTaskDelayUntil(&xLastWakeTime, interval);
}
}
```

Callback MQTT La funzione callback è utilizzata per ricevere i messaggi di sincronizzazione dal topic MQTT. In particolare, questa funzione è responsabile di ricevere il timestamp dal topic MQTT e sincronizzare il modulo RTC con il timestamp ricevuto.

```
// Callback function: receive timestamp as message from topic through MQTT and
// sync time on PCF8523
void callback(char* topic, byte* payload, unsigned int length) {

    if (strcmp(topic, mqttSyncTopic) == 0) {
```

CAPITOLO 3: SVILUPPO DEL PROGETTO

```
char mqttMess[100];
uint32_t timestamp;
memcpy(mqttMess, payload, length);
mqttMess[length] = '\0';
String mess = String(mqttMess);
timestamp = mess.toInt();
DateTime dt = DateTime(timestamp);

// Run only first time sync
if (isFirst) {
    isFirst = false;
    rtc.adjust(dt);

// Run other times
} else {
    if (xQueueSemaphoreTake(binarySemaphore, portMAX_DELAY) == pdTRUE) {
        rtc.adjust(dt);
        xSemaphoreGive(binarySemaphore);
    }
}
}
```

Setup e loop La funzione setup() è responsabile per l'inizializzazione di tutte le configurazioni iniziali del sistema. Questo include l'inizializzazione delle connessioni WiFi e MQTT, l'inizializzazione dei sensori e la creazione dei task. La funzione loop() non è utilizzata in questo caso poiché il codice è stato progettato per eseguire i task specifici all'interno del sistema operativo FreeRTOS.

```
void setup() {

    Serial.begin(115200);
    isFirst = true;

    mqttClient.setServer(mqttBroker, mqttPort);
    mqttClient.setCallback(callback);

    // Connect to WiFi
    wifiInit();
```

```
Serial.println("\nWait for timestamp...\n");
syncFT();

// Initialize sensors
sensorInit();

// Create queues
sensorQueue = xQueueCreate(1, sizeof(SensorData));
historyQueue = xQueueCreate(10, sizeof(SensorData));

// Create binary semaphore
binarySemaphore = xSemaphoreCreateBinary();

// Create tasks with their priorities
xTaskCreate(sensorTask, "SensorTask", 8192, NULL, 1, NULL);
xTaskCreate(mqttTask, "MqttTask", 8192, NULL, 1, NULL);

// Give semaphore
xSemaphoreGive(binarySemaphore);
}

// Idle task used for call mqtt loop
void loop() {
    mqttClient.loop();
}
```

4. Conclusioni

Durante lo sviluppo del progetto, sono stati affrontati alcuni problemi e sono stati fatti dei progressi significativi nell'implementazione di un sistema IoT con l'utilizzo del protocollo MQTT. Di seguito vengono elencati i problemi riscontrati e i possibili sviluppi futuri per il progetto.

4.1 Problemi Riscontrati

- **Connessione MQTT Instabile**

Si è riscontrato un problema con la stabilità della connessione MQTT. Durante la fase di test, è stato necessario implementare una funzione di riconnessione per gestire situazioni di disconnessione improvvisa. Tuttavia, il problema potrebbe essere dovuto a una cattiva qualità del segnale WiFi o a un flusso eccessivo di dati, il che richiede ulteriori indagini e ottimizzazioni.

- **Dimensione della Coda di History**

La coda di history ha una dimensione limitata, il che potrebbe portare alla perdita di dati importanti in caso di un elevato flusso di dati in entrata. È necessario studiare la possibilità di ridimensionare la coda in base alle esigenze del progetto.

- **Gestione dei Timestamp**

Il sistema attualmente si basa sull'invio dei timestamp attraverso MQTT per sincronizzare il tempo sul sensore RTC. Tuttavia, questa implementazione potrebbe non essere la più affidabile, in quanto dipende dalla stabilità della connessione MQTT e da eventuali ritardi nella trasmissione dei messaggi. Sarebbe opportuno considerare altre soluzioni per garantire la precisione del tempo.

4.2 Sviluppi Futuri

- **Ottimizzazione della Connessione MQTT**

Per migliorare la stabilità della connessione MQTT, sarebbe utile indagare ulteriormente sulla qualità del segnale WiFi e implementare strategie per gestire connessioni instabili o congestioni della rete. L'implementazione di tecniche di riconnessione migliorate e la gestione degli errori potrebbero ridurre i problemi di perdita di dati.

- **Implementazione di un Buffer di Salvataggio**

Per affrontare il problema della dimensione limitata della coda di history, si potrebbe considerare l'implementazione di un buffer di salvataggio su una memoria esterna, come ad esempio una scheda SD. Ciò permetterebbe di archiviare e conservare i dati raccolti in modo più affidabile e consentirebbe anche di analizzarli successivamente.

- **Sincronizzazione NTP**

Per garantire un'accurata sincronizzazione temporale, si potrebbe utilizzare il protocollo di sincronizzazione di rete NTP (*Network Time Protocol*) anziché dipendere dalla trasmissione dei timestamp attraverso MQTT. NTP è noto per fornire un'alta precisione nella sincronizzazione dei tempi su dispositivi IoT.

In conclusione, il progetto ha dimostrato la fattibilità dell'implementazione di un sistema di monitoraggio dati IoT. Tuttavia, sono necessari ulteriori sforzi per ottimizzare la connessione MQTT, gestire in modo efficiente i dati raccolti e garantire la precisione del tempo. Con gli sviluppi futuri proposti, il sistema potrebbe diventare una soluzione completa e affidabile per il monitoraggio dati e trovare applicazioni in diversi contesti.

5. Appendice

5.1 Codici Script Linux

In questa appendice verranno elencati gli script in bash collegati alla dashboard Node-Red. Tutti gli script si trovano nella directory di root, come nella figura sotto

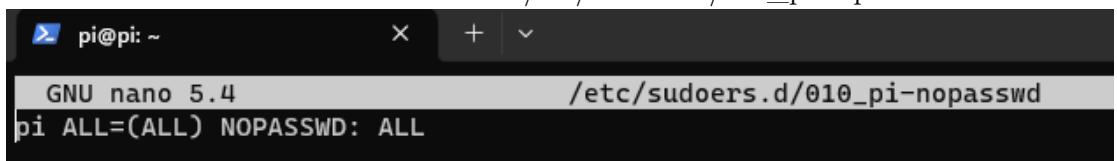


```
pi@pi:~ $ ls
Bookshelf    Documents  Music        Public       stop_mqtt.sh  Videos
commandi.txt  Downloads  node-red_ssl  start_mqtt.sh syncrtc.sh   write_db.sh
Desktop      mqtt_tls  Pictures     stop_db.sh    Templates    write_db.sh.save
```

Figura 5.1: Scripts Directory

Gli script start_mqtt.sh e stop_mqtt.sh contengono il comando 'sudo', ciò vuol dire che viene richiesta la password che abbiamo impostato quando viene flashato la SD, ma su Raspberry Pi OS per semplicità hanno aggiunto una direttiva per fare in modo che sudo non chieda la password dell'utente che sta eseguendo il comando.

La direttiva si trova nel file /etc/sudoers.d/010_pi-nopasswd.



```
pi@pi: ~          * + *
GNU nano 5.4          /etc/sudoers.d/010_pi-nopasswd
pi ALL=(ALL) NOPASSWD: ALL
```

Figura 5.2: Screenshot di /etc/sudoers.d/010_pi-nopasswd

start_mqtt.sh

```
#!/bin/bash
# Start Mosquitto Service
sudo systemctl start mosquitto.service
```

CAPITOLO 5: APPENDICE

stop_mqtt.sh

```
#!/bin/bash
# Start Mosquitto Service
sudo systemctl stop mosquitto.service
```

write_db.sh

```
#!/bin/bash

# Get the IP address of the current Linux system
IP_ADDRESS=$(hostname -I | awk '{print $1}')
# MQTT Broker settings
MQTT_BROKER=$(hostname -I | awk '{print $1}')
MQTT_TOPIC="history"

# InfluxDB settings
INFLUXDB_SERVER=$(hostname -I | awk '{print $1}')
INFLUXDB_DB="ameba"
INFLUXDB_URL="http://$INFLUXDB_SERVER:8086/write?db=$INFLUXDB_DB"
INFLUXDB_USER=""
INFLUXDB_PASSWORD=""

# Function to process and write the MQTT message to InfluxDB
write_to_influxdb() {
    local message=$1
    local tvoc
    local eco2
    local hr
    local spo2
    local timestamp

    IFS=', ' read -ra values <<< "$message"

    if [[ "${values[@]}" -eq 5 ]]; then
        tvoc=${values[0]}
        eco2=${values[1]}
        hr=${values[2]}
```

CAPITOLO 5: APPENDICE

```
spo2=${values[3]}

timestamp=${values[4]}

# Format the data as InfluxDB line protocol with
# measurement name "data"
data="data

eco2=$eco2,hr=$hr,spo2=$spo2,timestamp=$timestamp,tvoc=$tvoc"

# Perform HTTP POST to InfluxDB
curl -i -XPOST "$INFLUXDB_URL" --data-binary "$data"
--user "$INFLUXDB_USER:$INFLUXDB_PASSWORD"

else
    echo "Invalid message format: $message"
fi

}

# Subscribe to MQTT topic and process messages
mosquitto_sub -h "$MQTT_BROKER" -t "$MQTT_TOPIC" | while read
-r line; do
    write_to_influxdb "$line"
done
```

stop_db.sh

```
#!/bin/bash

# Check if the write_db.sh script is running
if pgrep -x "write_db.sh" > /dev/null; then
    # Stop the write_db.sh script
    pkill -f "write_db.sh"
    echo "write_db.sh script has been stopped."
else
    echo "write_db.sh script is not running."
fi
```

syncrtc.sh

```
#!/bin/bash

# MQTT broker details
```

CAPITOLO 5: APPENDICE

```
USERNAME="ameba"
PASSWORD="ameba"
TOPIC="timestamp"

# Get the IP address of the current Linux system
IP_ADDRESS=$(hostname -I | awk '{print $1}')

# Get the current date and time in the desired format
DATE_TIME=$(date +%s)

# Publish the date and time with IP address via MQTT
mosquitto_pub -h "$IP_ADDRESS" -t "$TOPIC" -u "$USERNAME" -P "$PASSWORD"
-m "$DATE_TIME"i
```

5.2 Codici Script Arduino

```

#include <FreeRTOS.h>
#include <queue.h>
#include <task.h>
#include <Wire.h>
#include <Adafruit_SGP30.h>
#include "MAX30102_PulseOximeter.h"
#include <RTClib.h>
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi credentials
char* ssid = "*****";
char* password = "*****";

// MQTT broker information
char* mqttBroker = "*****";
int mqttPort = 1883;
char* mqttClientId = "*****";
char* mqttUsername = "*****";
char* mqttPassword = "*****";

// Topic definition
char* mqttSensorTopic = "sensor";
char* mqttSyncTopic = "timestamp";
char* mqttHistoryTopic = "history";

// Sensor objects
Adafruit_SGP30 sgp30;
PulseOximeter pox;
RTC_PCF8523 rtc;

// Wifi & Mqtt objects
WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

// Struct to contain sensor data
typedef struct {
    uint16_t tvoc;

```

CAPITOLO 5: APPENDICE

```
uint16_t eco2;
float hr;
uint8_t spo2;
uint32_t currentTime;
} SensorData;

// Control variable
boolean isFirst;

// Predefined delay time
TickType_t delaySensor = pdMS_TO_TICKS(100);
TickType_t delayMqttC = pdMS_TO_TICKS(1000);
TickType_t delayMqttDc = pdMS_TO_TICKS(1000);
unsigned long delayEmpty = 50;

// Queues
QueueHandle_t sensorQueue;
QueueHandle_t historyQueue;

// Binary semaphore
SemaphoreHandle_t binarySemaphore;

// Reconnect function for MQTT
boolean reconnectMQTT() {

    if (mqttClient.connect(mqttClientId, mqttUsername, mqttPassword)) {
        mqttClient.subscribe(mqttSyncTopic);
    }

    return mqttClient.connected();
}

// Callback function: receive timestamp as message from topic through MQTT and
// sync time on PCF8523
void callback(char* topic, byte* payload, unsigned int length) {

    if (strcmp(topic, mqttSyncTopic) == 0) {

        char mqttMess[100];
        uint32_t timestamp;
        memcpy(mqttMess, payload, length);
```

CAPITOLO 5: APPENDICE

```
mqttMess[length] = '\0';
String mess = String(mqttMess);
timestamp = mess.toInt();
DateTime dt = DateTime(timestamp);

// Run only first time sync
if (isFirst) {
    isFirst = false;
    rtc.adjust(dt);

// Run other times
} else {
    if (xQueueSemaphoreTake(binarySemaphore, portMAX_DELAY) == pdTRUE) {
        rtc.adjust(dt);
        xSemaphoreGive(binarySemaphore);
    }
}
}

// WiFi initialization procedure
void wifiInit() {

// Connect to WiFi
while (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nConnecting to WiFi...");
    WiFi.begin(ssid, password);
    delay(10000);
}
Serial.println("\n\nConnected to WiFi!\n");
}

// Sensors initialization procedure
void sensorInit() {

// Initialize MAX30102
if (!pox.begin()) {
    Serial.println("Failed to initialize MAX30102!");
    while (1);
} else {
    Serial.println("MAX30102 initialized!");
}
```

CAPITOLO 5: APPENDICE

```
}

pox.setIRLedCurrent(MAX30102_LED_CURR_7_6MA);

// Initialize SGP30
if (!sgp30.begin()) {
    Serial.println("Failed to initialize SGP30!");
    while (1);
} else {
    Serial.println("SGP30 initialized!");
}
}

// Function that receive timestamp and set rtc sensor for the first time
void syncFT() {

    // Initialize PCF8523
    if (!rtc.begin()) {
        Serial.println("Failed to initialize PCF8523!");
        while (1);
    } else {
        Serial.println("PCF8523 initialized!");
        rtc.start();
    }
}

// Loop until first timestamp is received and meanwhile check if WiFi and MQTT
// are connected
while (isFirst) {
    if (WiFi.status() != WL_CONNECTED) {
        WiFi.begin(ssid, password);
        delay(5000);
    }
    if (!reconnectMQTT()) { delay(5000); }
    mqttClient.loop();
}
}

// Sensor task: read data from sensors and send them through queue to MQTT task
void sensorTask(void* params) {

    (void) params;
    SensorData sensorData;
```

CAPITOLO 5: APPENDICE

```
TickType_t xLastWakeTime;
xLastWakeTime = xTaskGetTickCount();

while (1) {

    // Take semaphore
    if (xQueueSemaphoreTake(binarySemaphore, portMAX_DELAY) == pdTRUE) {

        // Read sensors data
        pox.update();
        sensorData.hr = pox.getHeartRate();
        sensorData.spo2 = pox.getSpO2();

        if (!sgp30.IAQmeasure()) {
            Serial.println("Failed to measure IAQ!");
            while (1);

        } else {
            sensorData.tvoc = sgp30.TVOC;
            sensorData.eco2 = sgp30.eCO2;
        }

        sensorData.currentTime = rtc.now().unixtime();

        // Give semaphore
        xSemaphoreGive(binarySemaphore);
    }

    // Send sensors data through queue
    xQueueOverwrite(sensorQueue, &sensorData);

    // Block task for a fixed period
    vTaskDelayUntil(&xLastWakeTime, delaySensor);
}

/*
MQTT task:
1. Check if WiFi & MQTT are connected
2. Receive timestamp from MQTT topic and sync time on PCF8523
3. Receive sensors data through queue from Sensor task and send it as message
*/
```

CAPITOLO 5: APPENDICE

```
    to both MQTT topic
4. If MQTT is disconnected save data into queue
*/
void mqttTask(void* params) {

    (void) params;
    SensorData sensorData;
    unsigned long currentMills;
    TickType_t interval;
    TickType_t xLastWakeTime = xTaskGetTickCount();

    while (1) {

        // If MQTT is disconnected
        if (!mqttClient.connected()) {

            reconnectMQTT();

            // Save data from sensor queue to history queue, if history queue is full
            // overwrite the oldest value
            if (xQueueReceive(sensorQueue, &sensorData, 0) == pdPASS) {
                if (xQueueSend(historyQueue, &sensorData, 0) == errQUEUE_FULL) {
                    xQueueReceive(historyQueue, &sensorData, 0);
                }
            }

            interval = delayMqttDc;

            // If MQTT is connected
        } else {

            unsigned long previousMills = 0;
            // Empty history queue as fast as possible
            while (uxQueueMessagesWaiting(historyQueue) > 0) {
                currentMills = millis();
                if (currentMills - previousMills >= delayEmpty) {
                    previousMills = currentMills;
                    if (xQueueReceive(historyQueue, &sensorData, 0) == pdPASS) {
                        char historyMess[100];
                        sprintf(historyMess, sizeof(historyMess), "%d,%d,%d,%d",

```

CAPITOLO 5: APPENDICE

```
    sensorData.tvoc, sensorData.eco2, (int)sensorData.hr, sensorData.spo2,
    sensorData.currentTime);

    mqttClient.publish(mqttHistoryTopic, historyMess);
}
}

// Publish data on both topic
if (xQueueReceive(sensorQueue, &sensorData, 0) == pdPASS) {
    char sensorMess[100];
    sprintf(sensorMess, sizeof(sensorMess), "%d,%d,%d,%d,%d", sensorData.
tvoc, sensorData.eco2, (int)sensorData.hr, sensorData.spo2, sensorData.
currentTime);

    mqttClient.publish(mqttSensorTopic, sensorMess);
    mqttClient.publish(mqttHistoryTopic, sensorMess);
}

interval = delayMqttC;

}

vTaskDelayUntil(&xLastWakeTime, interval);
}

}

void setup() {

    Serial.begin(115200);
    isFirst = true;

    mqttClient.setServer(mqttBroker, mqttPort);
    mqttClient.setCallback(callback);

    // Connect to WiFi
    wifiInit();

    Serial.println("\nWait for timestamp... \n");
    syncFT();

    // Initialize sensors
    sensorInit();
```

CAPITOLO 5: APPENDICE

```
// Create queues
sensorQueue = xQueueCreate(1, sizeof(SensorData));
historyQueue = xQueueCreate(10, sizeof(SensorData));

// Create binary semaphore
binarySemaphore = xSemaphoreCreateBinary();

// Create tasks with their priorities
xTaskCreate(sensorTask, "SensorTask", 8192, NULL, 1, NULL);
xTaskCreate(mqttTask, "MqttTask", 8192, NULL, 1, NULL);

// Give semaphore
xSemaphoreGive(binarySemaphore);
}

// Idle task used for call mqtt loop
void loop() {
    mqttClient.loop();
}
```

Sitografia

- [1] ADAFRUIT INDUSTRIES, LLC: *PCF8523.* – URL <https://www.adafruit.com/product/3295>
- [2] AMAZON WEB SERVICES, Inc.: *FreeRTOS.* – URL https://www.freertos.org/Documentation/RTOS_book.html
- [3] ANALOG DEVICES, Inc.: *MAX30102.* – URL <https://www.analog.com/en/products/max30102.html>
- [4] ARDUINO ©: *Arduino.* – URL <https://docs.arduino.cc/>
- [5] ECLIPSE FOUNDATION, Inc.: *Mosquitto.* – URL <https://mosquitto.org/documentation/>
- [6] FOUNDATION, OpenJS: *Node-RED.* – URL <https://nodered.org/docs/>
- [7] FOUNDATION, Raspberry Pi.: *Raspberry Pi.* – URL <https://www.raspberrypi.com/documentation/>
- [8] INFLUXDATA ®: *InfluxDB.* – URL <https://docs.influxdata.com/>
- [9] MQTT.ORG: *MQTT.* – URL <https://mqtt.org/>
- [10] SENSIRION AG ©: *SGP30.* – URL <https://sensirion.com/products/catalog/SGP30/>
- [11] SOLUTIONS, Realtek IoT/Wi-Fi M.: *Ameba AMB23.* – URL <https://www.amebaiot.com/en/amebad-mini-arduino-getting-started/>