CAS - Chirp Air Station

Antonio Rotundo
antonio.rotundo2@studio.unibo.it

Aprile 2024

Il seguente progetto mira a implementare un servizio di stazioni metereologiche per monitorare temperatura, pressione, umidità e indice di qualità dell'aria in ambienti urbani o in diverse città. Utilizzando un sistema a microcontrollore ESP32 con moduli di comunicazione LoRa e Wi-Fi, assieme al sensore BME680, i dati vengono trasmessi ai Gateway mediante connessione radio LoRa. I Gateway ricevono così le informazioni da tutti i sensori nel raggio di copertura, inviando i dati pre-elaborati tramite interfaccia Wi-Fi e rete Internet a un Broker MQTT. Infine, un servizio MQTT Reader, sfruttando un servizio database inserirà e aggregherà i dati all'interno di quest'ultimo, il quale verrà utilizzato per fornire, mediante Web Service, opportune API o Client Web per il monitoraggio e l'interazione con il sistema.

1 Obiettivi e Requisiti

L'obbiettivo del progetto è quello di fornire un servizio di monitoraggio ambientale localizzato e scalabile, avente la capacità di raccogliere le informazioni inerenti alla temperatura, umidità, pressione e qualità dell'aria dell'ambiente, specialmente in contesti urbani. Un requisito chiave è l'adozione di un protocollo a basso consumo energetico per la comunicazione tra i dispositivi sensori, garantendo così una maggiore autonomia. Inoltre, il protocollo proposto dovrà fornire tutte le funzionalità e possibilità di un architettura scalabile e distribuibile, a livello end-device.

I Gateway devono comunicare mediante lo stesso protocollo dei sensori e devono essere dotati anche di connettività IP (come Wi-Fi, Ethernet, ecc.), attraverso la quale invieranno i dati provenienti dai sensori attraverso Internet. L'architettura proposta, in funzione dei Gateway, deve essere progettata per una scalabilità orizzontale, consentendo l'ampliamento della rete da parte di qualsiasi utente.

I dati raccolti dai Gateway devono essere inviati attraverso Internet utilizzando un protocollo applicativo che permetta lo scambio di informazioni, indipendentemente dalla posizione del servizio principale. Questo protocollo deve supportare la ricezione e la gestione in tempo reale dei dati, strutturati attraverso un protocollo interno scalabile e accessibile a tutti i dispositivi Gateway, facilitando la comunicazione bidirezionale. La

compatibilità del protocollo deve essere garantita indipendentemente dalle reti IP in cui si trovano i dispositivi Gateway e il servizio applicativo (ad esempio, reti sotto NAT o Firewall).

Le informazioni ricevute tramite protocollo applicativo devono essere conservate in un database appositamente scelto, e in grado di memorizzare dati non omogenei e supportare l'espansione delle funzionalità del servizio. Il database deve essere progettato per una scalabilità orizzontale, in modo tale da poter garantire una gestione di numero crescente di sensori e utenti. Tale servizio, oltre ad essere incaricato di memorizzare tali informazioni, dovrà considerare anche i tempi di arrivo e gestire eventuali duplicati.

Infine, è richiesto un servizio con interfacce applicative ed utente, attraverso le quali gli utenti possano visualizzare e interagire con il sistema. Queste operazioni devono essere protette da un sistema di autenticazione utente e permessi basati sui ruoli, tra cui almeno utente normale ed amministratore di sistema. L'amministratore di sistema avrà accesso a tutte le informazioni di tutti gli utenti e sensori, mentre l'utente normale potrà accedere a solo i propri dati. La piattaforma deve consentire la visualizzazione dello storico dei dati, la visualizzazione in tempo reale e la rappresentazione grafica delle informazioni ricevute su qualsiasi dispositivo multimediale moderno.

Considerando i precedenti requisiti ed obiettivi, ci si aspettano i seguenti risultati:

- un dispositivo in grado di percepire temperatura ed altri fattori esterni mediante un sensore appropriato, in grado di poter comunicare mediante un protocollo a basso livello;
- il precedente dispositivo ed un firmware in grado di poter gestire un protocollo di comunicazione progettato ad-hoc, che possa essere scalabile e di garantire un risparmio energetico;
- un dispositivo in grado di poter utilizzare lo stesso protocollo di comunicazione dei sensori ed un protocollo di tipo IP;
- firmware che possa rendere tali dispositivi dei Gateway per la rete sensoristica, garantendo la scalabilità e l'espandibilità della rete;
- un protocollo progettato ad-hoc che permetta ai Gateway ed ai servizi applicativi, di poter comunicare a prescindere dall'architettura di rete in qui essi si trovino;
- un servizio applicativo in grado di poter ricevere le informazioni dai Gateway, conservando tali informazioni all'interno di un database esterno e di gestire eventuali repliche;
- un servizio database in grado di poter scalare orizzontalmente e di garantire la gestione di dati non-omogenei, garantendo una scalabilità verticale del servizio nel futuro e di carico di lavoro;
- un Web Service in grado di poter offrire delle API ed un'interfaccia utente con i quali sia possibile l'interazione da parte degli utenti, garantendo sicurezza mediante autenticazione e ruoli;

- una soluzione semplice da deployare mediante strumenti specifici e che ne semplifichi la distribuzione ed in controllo.

1.1 Scenari D'uso

Il sistema predisposto come da requisiti, potrebbe incontrare i seguenti scenari d'uso:

- **Registrazione ed Autenticazione**
  Un utente si registra nel sistema, fornendo le informazioni necessarie e successivamente accede utilizzando le proprie credenziali. L'utente può essere un normale utilizzatore del servizio o un amministratore di sistema.
- **Registrazione di un Dispositivo**
  Una volta avuto accesso al sistema, un utente o amministrazione può aggiungere un nuovo dispositivo sensore alla rete ed al proprio profilo. Durante il processo, configura le informazioni del sensore, come la posizione geografica e alcuni parametri di rilevamento, registrandolo utilizzando l'identificatore univoco mostrato dal dispositivo in forma visiva.
- **Visualizzazione Dati in Tempo Reale**
  L'utente accede al sistema e visualizza, in tempo reale, i dati provenienti dai sensori. Può selezionare specifici parametri ambientali o dispositivi connessi e monitorare le variazioni nel corso del tempo.
- **Analisi Storica dei Dati**
  L'utente esegue un'analisi storica dei dati, visualizzando le tendenze di temperatura, umidità, pressione e qualità dell'aria nel corso di giorni, settimane o mesi.
- **Gestione Utenti**
  L'amministratore può modificare o revocare privilegi o ruoli agli autenti, assicurandosi che solo utenti autorizzati possano accedere a determinate informazioni o eseguire determinate azioni.
- **Espansione di Copertura**
  Un qualsiasi utente o amministratore può installare, a proprio piacimento o in maniera strategica, i propri Gateway in modo da ampliare la copertura del servizio, rendendolo scalabile da chiunque.
- **Integrazione con altre Piattaforme**
  Gli utenti posso integrare i dati ricevuti sulla piattaforma con altre piattaforme o sistemi esterni, come sistemi di gestione edifici o monitoraggio del traffico, mediante opportune interfacce applicative.

2 Design

In base ai requisiti specificati nella sezione 1, la prima fase ha coinvolto la definizione di aree operative, all'interno delle quali sono stati identificati i servizi, i dispositivi e i

protocolli associati. Nell'architettura di progetto, sono state individuate diverse aree, tra cui Edge Field, IP Local Field, Internet Field e Service Field. Questa suddivisione e da considerarsi principalmente concettuale, finalizzata a comprendere le diverse componenti di ciascuna area.

La seguente sezione fornisce una descrizione dell'architettura progettuale, tenendo conto delle aree precedentemente menzionate e delle relative componenti

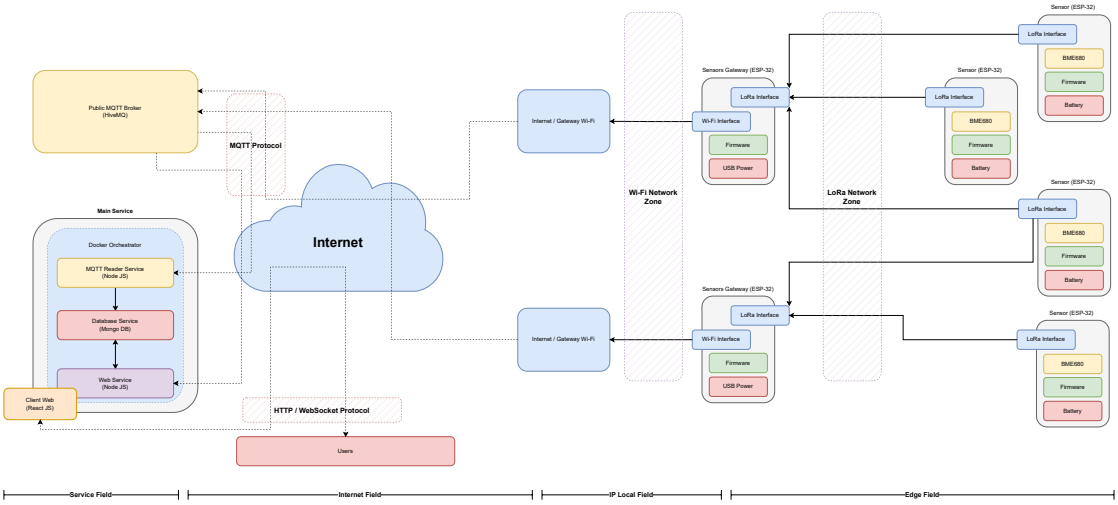

Figure 1: Architettura del progetto

Inizialmente, l'area Edge Field è stata progettata, definendo i suoi componenti (Sensori e Gateway) e optando per il protocollo a basso consumo energetico più adatto al progetto. Tenendo conto della necessità di scalabilità orizzontale, del requisito dei dispositivi sensoristici a basso consumo energetico e della richiesta di comunicazione a lunga distanza con un Gateway, ampliando l'area di copertura del servizio, la scelta del protocollo di comunicazione è caduta sul protocollo LoRa. Il protocollo LoRa (Long Range) si basa sulla modulazione di frequenza a spettro espanso con il Chirp Spread Spectrum (CSS), sviluppato dalla Semthec[21]. Utilizzando bande di radiofrequenza libere, il protocollo LoRa consente trasmissioni a lungo raggio, con una copertura superiore ai 10 chilometri in zone rurali e circa 5 chilometri in ambienti densamente urbanizzati, il tutto mantenendo bassi i consumi energetici. Questa scelta permette al sistema di adattarsi efficacemente indipendentemente dalla portata desiderata.

Il dispositivo sensoristico deve essere equipaggiato, non solo con un modulo di comunicazione LoRa, ma anche con un sensore in grado di rilevare temperatura, umidità, pressione atmosferica e resistenza dell'aria. La scelta del sensore ricade sul BME680 della Bosch Sensortec, il quale supporta il cablaggio attraverso i protocolli seriali I2C ed SPI. Questo sensore offre la flessibilità allo sviluppatore di regolare le velocità di campionamento per ciascuna componente del sensore. Il BME680 può misurare temperature nell'intervallo da -40°C a 85°C con una precisione di 0,5°C ed una risoluzione di 0,01°C.

Per quanto riguarda l'umidità, il sensore fornisce la lettura in percentuale, invece per quanto riguarda la pressione atmosferica, quest'ultima è misurata nell'intervallo compreso tra 300hPa e 1100hPa. Per ridurre l'utilizzo di periferiche superflue si è scelto di interfacciare il sensore utilizzando il protocollo I2C.

La scelta del microcontrollore utilizzato è ricaduta sull'ESP32, in microcontrollore di nuova generazione sviluppato da Espressif Systems[7]. Questo microcontrollore offre una serie di funzionalità avanzate, tra cui l'elaborazione parallela che consente la gestione simultanea di diversi processi. Questa caratteristica ottimizza la gestione della connettività e l'elaborazione dei dati provenienti dal sensore BME680, riducendo significativamente il tempo di attività del dispositivo. L'ESP32 dispone inoltre di una modalità deep-sleep, che permette di entrare in uno stato di risparmio energetico e di risvegliarsi dopo un periodo di tempo configurabile. Per sfruttare appieno le capacità di questo microcontrollore è stata selezionata una scheda di sviluppo che incorpora l'ESP32, un modulo LoRa con relativa antenna e un display LCD. Poiché l'ESP32 dispone anche di connettività Wi-Fi, questa stessa scheda di sviluppo è stata adottata per realizzare il dispositivo Gateway.

Figure 2: Foto del dispositivo Gateway

Sia nell'Edge Field che nell'IP Local Field sono presenti i dispositivi Gateway, i quali operano in due modalità distinte. La prima è la modalità di configurazione, in cui gli utenti possono regolare la connettività del Gateway attraverso un interfaccia grafica web, fornita dal dispositivo stesso. Una volta completata la configurazione, il dispositivo passa alla modalità operativa. Durante la prima fase della modalità operativa, il Gateway cerca di connettersi alla rete configurata e di stabilire una connessione Internet. In caso di errore di connessione il dispositivo si riavvia nella modalità di configurazione, altrimenti, prosegue con la modalità operativa. Nella seconda fase della modalità operativa il Gateway è responsabile di ricevere i messaggi provenienti dai dispositivi sensoristici attraverso la connettività LoRa. Successivamente, il dispositivo ristruttura i dati in arrivo in un formato JSON progettato per il servizio e li invia mediante un protocollo di rete applicativo. E' importante sottolineare che la scelta della connettività Wi-Fi è motivata dalla presenza del microcontrollore ESP32 che la supporta. Tuttavia, è fondamentale notare che è possibile utilizzare qualsiasi forma di connettività IP che consenta l'accesso ad Internet come Ethernet, fibra ottica, connessione cellulare e così via.

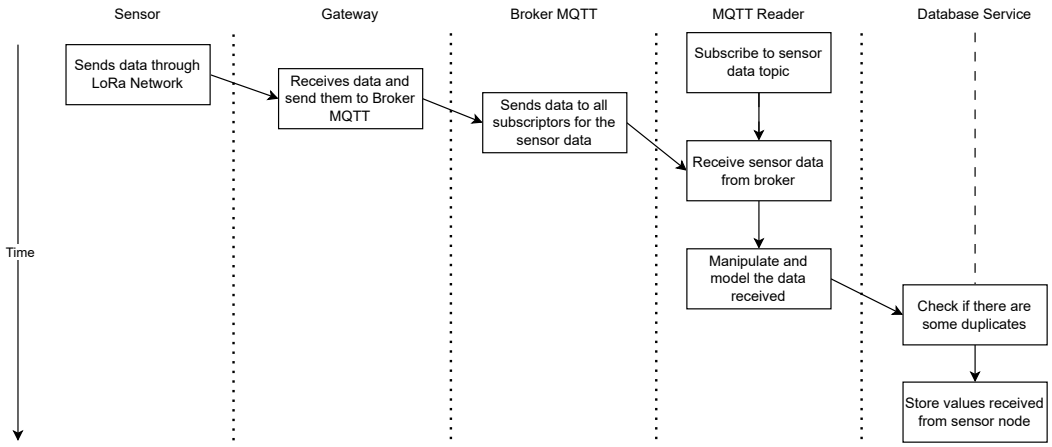

Figure 3: Diagramma temporale del servizio MQTT Reader

Il protocollo applicativo selezionato per la trasmissione dei dati attraverso la connessione Internet è MQTT (Message Queuing Telemetry Transport)[11]. MQTT, un protocollo di messaggistica basato su standard, è ampiamente utilizzato nelle comunicazioni tra macchine, in particolare nell'IoT, dove spesso si affrontano limitazioni di risorse e larghezza di banda limitata. MQTT si configura come un protocollo di tipo Publisher/Subscriber che consente la separazione tra il mittente del messaggio (editore) e il destinatario del messaggio (sottoscrittore). In aggiunta, un terzo elemento, chiamato Broker, assume la responsabilità di filtrare i messaggi inviati dai Publisher e distribuirli correttamente a tutti i Subscriber. La scelta di MQTT offre la possibilità ai Gateway e al servizio applicativo di comunicare indipendentemente dallo stato e dalla architettura della propria rete.

Nell'ambito del Service Field, identifichiamo tutte le componenti a microservizio del progetto. Tra i microservizi definiti durante la progettazione troviamo il microservizio MQTT Reader. Questo specifico microservizio ha l'incarico di sottoscriversi ad uno specifico topic del progetto, sul quale i Gateway invieranno i dati provenienti dai dispositivi sensoristici. Il microservizio, oltre a ricevere questi dati, ha l'onere di gestire possibili duplicazione dei messaggi e di archiviarli in modo appropriato all'interno del servizio database.

Come è possibile vedere dalla Figura 3, una volta che il Sensore invia i dati mediante LoRa ed un Gateway dovesse riceverlo, quest'ultimo invierà i dati ricevuti al Broker predisposto per il servizio. Questo re-inoltrerà a tutti i sottoscrittori del topic designato per la ricezione dei dati sensoristici i dati ricevuti. Nel nostro primo caso, il servizio MQTT Reader riceverà i dati da parte dei sensori, manipolandoli ed inviandoli al Servizio Database controllando se ne possano esistere dei duplicati e conservandoli all'interno della collezione.

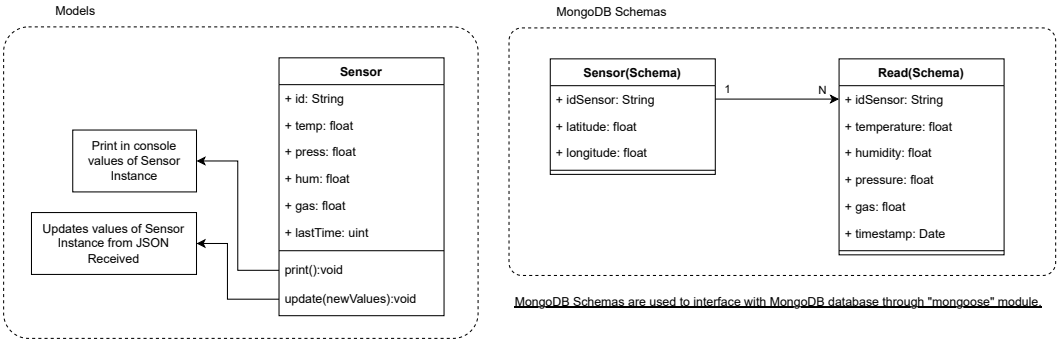

Figure 4: Diagramma strutturale del servizio MQTT Reader

Il servizio MQTT Reader, come rappresentato in Figura 4, è composto da una classe Sensor, che ne rappresenta un modello di dati in ricezione dal broker MQTT la quale rappresenta, in forma logica, i dati ricevuti da un Sensore. Questo modello viene utilizzato dai due modelli Schematici per il servizio Database, ovvero lo schema Sensor e lo Schema Read. Lo schema Sensor viene utilizzato per poter gestire le entità Sensor all'interno del Database mentre, lo schema Read viene utilizzato per rappresentare e gestire le varie letture, da parte di un sensore, all'interno del Database. Infine, il servizio sfrutta un file di configurazione esterno, caricato all'avvio, tramite il quale è possibile definire: l'URL di connessione al MQTT Broker (in demo "mqtt://broker.hivemq.com:1883"), l'URL di connessione al servizio Databse (in demo "mongodb://cas-db;27017/cas-db") e il topic MQTT designato per la ricezione dei dati da parte della rete sensoristica (in demo "cas/sensor").

Un altro servizio delineato durante la fase di progettazione è il Web Service. Questo servizio è responsabile di mettere a disposizione adeguate interfacce applicative (API) attraverso il protocollo HTTP e opportune interfacce utente. Queste interfacce con-

sentono agli utenti di interagire con i dati conservati all'interno del servizio database, provenienti dalla rete sensoristica. Le interazioni sono gestite e protette in modo appropriato da un sistema di autenticazione interno al progetto, il quale può essere esteso tramite altri sistemi di autenticazione. Il Web Service ha anche l'incarico di consentire agli utenti di visualizzare in tempo reale i dati provenienti da un sensore specifico, offrendo un'interazione più diretta con i propri dispositivi sensoristici.

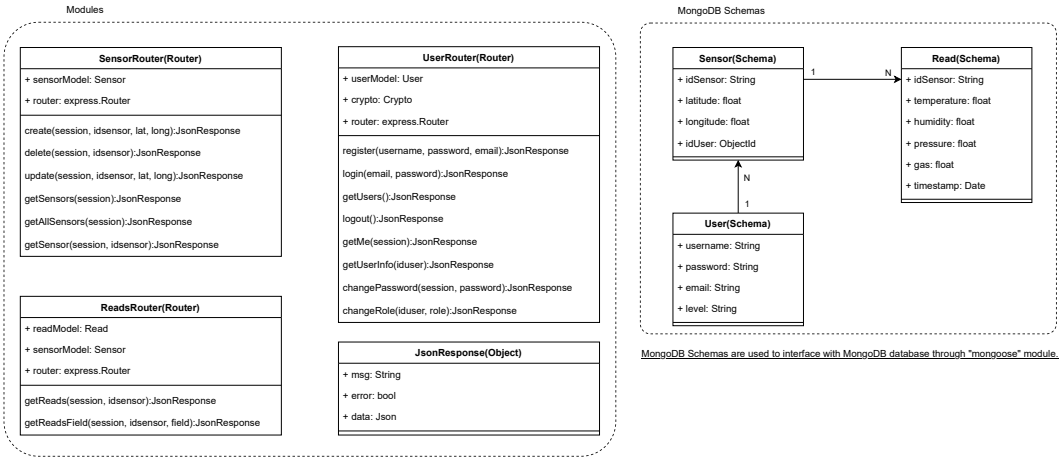

Figure 5: Diagramma strutturale del Web Service

Come è possibile vedere dalla Figura 5, il Web Service è composto dagli stessi modelli presenti anche nel MQTT Reader a differenza del modello User, il quale viene utilizzato per gestire gli account e la autenticazione degli utenti. Per gestire le varie routes HTTP sono stati progettati tre diversi router:

- **UserRouter**, router per la gestione dell'autenticazione, registrazione e gestione utenti. Lo stesso ha il compito di gestire la sessione utente.
- **SensorRouter**, router per la gestione dei sensori registrati.
- **ReadsRouter**, router per la gestione delle letture in arrivo dalla rete sensoristica.

Infine, un modulo JsonResponse per la gestione delle varie risposte HTTP, le quali possono notificare la presenza di un eventuale errore, un messaggio leggibile per gli eventuali errori ed un campo data nel caso sia necessaria appendere un ulteriore payload.

La Figura 6 rappresenta un meccanismo nel quale il Web Service avvia il proprio server HTTP e successivamente si sottoscrive allo stesso topic al quale si è sottoscritto l'MQTT Reader. Successivamente un client Web richiede allo stesso servizio il Web Client con il quale potrà interagire con il servizio stesso mediante apposite API. Il client dovrà effettuare l'autenticazione mediante l'operazioni di login o registrazione le quali verranno autenticate dal Web Service controllando mediante la Database Service le credenziali

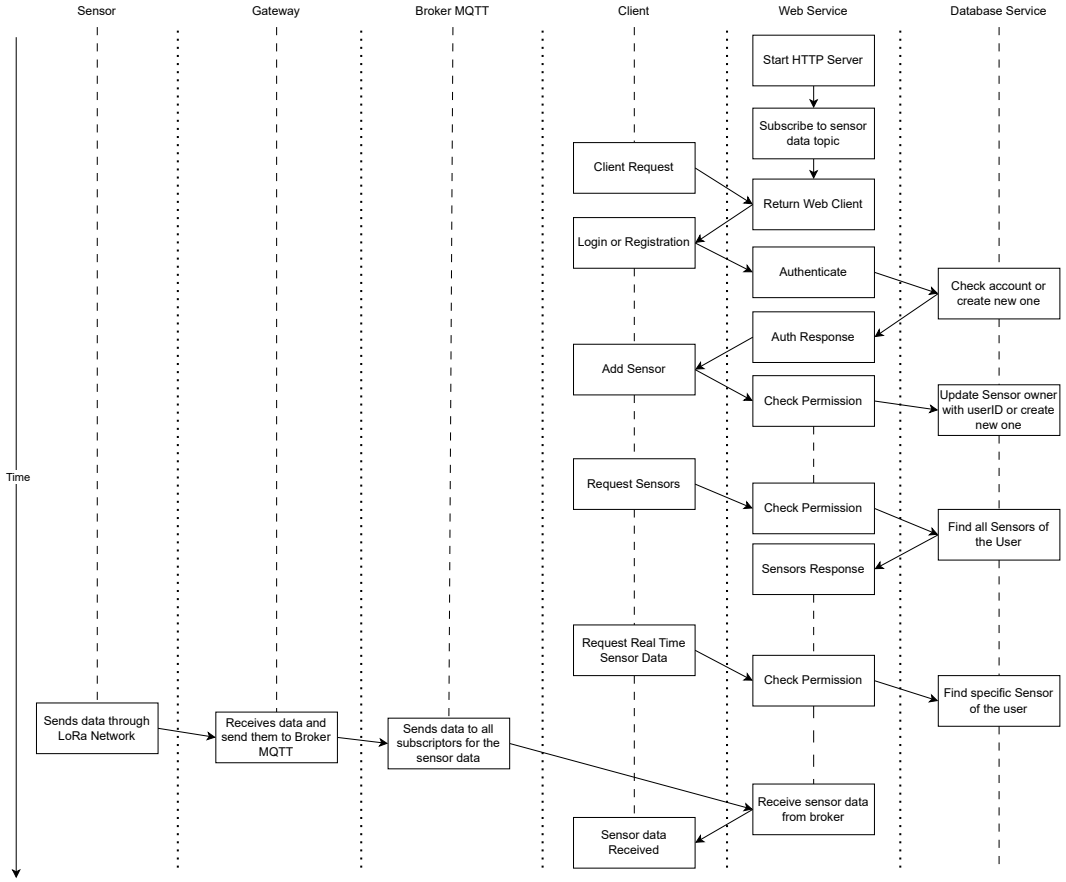

Figure 6: Diagramma temporale delle interazioni con il Web Service

utente. L'utente può quindi aggiungere sensori, specificando l'identificativo univoco del sensore che viene mostrato solo all'avvio di quest'ultimo ed il Web Service utilizzerà questo identificativo per controllare che già esistano delle letture da parte di questo dispositivo ma che non sia di proprietà di nessuno (in questo caso il dispositivo viene associato all'utente), oppure il dispositivo viene semplicemente creato. Una volta inseriti i sensori all'interno della piattaforma, l'utente può richiederne la propria lista di appartenenza e da quest'ultima può richiede le specifiche informazioni di un sensore ben specifico. A questo punto, il Web Service fornirà le informazioni specifiche del sensore richiesto e tutta la storia delle letture ricevute fino ad ora, inoltre fornirà, qualora dovessero arrivare, le letture in real-time provenienti dalla rete sensoristica.

E' importante sottolineare che il Web Service non ha il compito di conservare le letture provenienti da parte della rete sensoristica ma di consegnare il singolo messaggio ricevuto al Web Client connesso per quello specifico sensore. La manipolazione dei dati e la loro

conservazione spetta all'MQTT Reader.

La progettazione di questo servizio è orientata alla scalabilità orizzontale, permettendo la distribuzione del servizio mediante un bilanciatore di carico e anche verticalmente, ampliando le sue funzionalità. Infine, il servizio facilita un accesso multipiattaforma per gli utenti, consentendo loro di autenticarsi e interagire con le funzionalità senza la necessità di implementare alcuna interfaccia.

Il Web Service dispone di un client Web per una facile interazione da parte dell'utente, il quale però si basa sulle API HTTP che è lo stesso Web Service mette a disposizione. Queste API vengono definite da parte dei vari router (vedi Figura 5) e definite come segue:

- GET /

  Fornisce, in maniera statica, il Client Web.

- POST /user/register

  Permette la registrazione dell'utente. L'API richiede una username, un email ed una password.

- POST /user/login

  Permette l'autenticazione dell'utente. L'API richiede l'email e la password.

- ALL /user/logout

  Permette la disconnessione e della chiusura della sessione utente.

- GET /user/me [allowLogged]

  Permette di richiedere i dati dell'utente autenticato con sessione attiva. L'API è disponibile solo per le sessioni attive ed autenticate

- GET /user/:id [allowLogged, allowAdmin]

  Permette di richiedere i dati di uno specifico utente (parametro id). L'API è disponibile solo per amministratori con sessione attiva ed autenticata.

- PUT /user/password [allowLogged]

  Permette di aggiornare la propria password. L'API è disponibile solo per utenti con sessione attiva ed autenticata.

- PUT /user/:id/role [allowLogged, allowAdmin]

  Permette di modificare il livello di permessi (ruolo) di un utente specifico. L'API è disponibile solo per amministratori con sessione attiva ed autenticata.

- GET /sensor/ [allowLogged]

  Permette di richiedere tutti i sensori dell'utente. L'API è disponibile solo per utenti con sessione attiva ed autenticata.

- POST /sensor/ [allowLogged]

  Permette la creazione di un sensore e la registrazione di quest'ultimo all'utente che lo sta creando. L'API è disponibile solo per utenti con sessione attiva autenticata.

- GET /sensor/:id [allowLogged]

  Permette di ottenere le informazioni di un sensore specifico dell'utente. L'API è disponibile solo per utenti con sessione attiva autenticata.

- DELETE /sensor/:id [allowLogged]

  Permette l'eliminazione di un sensore specifico da parte del proprietario. L'API è disponibile solo per utenti con sessione attiva autenticata.

- PUT /sensor/:id [allowLogged]

  Permette l'aggiornamento delle informazioni riguardanti il sensore, come la posizione geografica. L'API è disponibile solo per utenti con sessione attiva autenticata.

- GET /sensor/all [allowLogged, allowAdmin]

  Permette di richiedere la lista di tutti i dispositivi registrati sulla piattaforma. L'API è disponibile solo per amministratori con sessione attiva autenticata.

- GET /sensor/user/:id [allowLogged, allowAdmin]

  Permette di ottenere la lista di tutti i dispositivi registrati di un utente specifico (parametro id). L'API è disponibile solo per amministratori con sessione attiva autenticata.

- GET /reads/:id [allowLogged]

  Permette di richiedere lo storico delle letture di un sensore specifico da parte dell'utente. L'API è disponibile solo per utenti con sessione attiva autenticata.

- GET /reads/:id/:field [allowLogged]

  Permette di richiedere lo storico delle letture di un sensore specifico, filtrando esclusivamente il campo richiesto. L'API è disponibile solo per utenti con sessione attiva autenticata.

Un terzo ed ultimo servizio delineato durante la fase di progettazione è il database service. Questo servizio è incaricato della gestione di una struttura dati di tipo database, preferibilmente di natura non relazionale. La scelta di optare per questa tipologia di database deriva dalla possibilità di memorizzare dati appartenenti alla stessa categoria o collezione senza necessariamente conformarsi a uno schema omogeneo. Ciò consente ai sensori di inviare dati aggiuntivi rispetto a quelli attesi, permettendo l'introduzione di nuovi sensori. Inoltre, questa tipologia di database è facilmente scalabile orizzontalmente.

3 Implementazione

Nella fase di progettazione, sono state identificate diverse aree applicative in base alle specifiche sfide da affrontare e alla formulazione di soluzioni adeguate. La prima area selezionata per l'approfondimento è l'Edge Field. In tale contesto, come precedentemente delineato nella progettazione, sono stati concepiti e sviluppati i dispositivi sensoristici e i Gateway.

3.1 Dispositivo Sensoristico

Per la realizzazione dei dispositivi sensoristici, specialemente per creare un firmware replicabile su numerosi dispositivi con identici microcontrollori e configurazioni hardware (moduli di comunicazione, sensori, ecc.), è stata adottata la decisione di sviluppare il firmware utilizzando il framework Arduino. Arduino è una collezione di strumenti e interfacce applicative che consentono l'utilizzo delle librerie native di Espressif, seguendo però la struttura di codice caratteristica di Arduino. Inoltre, considerando il sensore scelto durante la fase di progettazione, le relative librerie sono state progettate per essere compatibili con il framework Arduino, agevolando così l'integrazione all'interno del firmware.

Un prima problema affrontato ha riguardato la generazione di un identificativo univoco per ciascun dispositivo sensoristico. Per ottenere tale risultato, è stato utilizzato l'indirizzo fisico MAC del modulo Wi-Fi integrato nel ESP32, prelevando le ultime quattro cifre da esso. Questo approccio potrebbe certamente benificiare di futuri miglioramenti. Al fine di semplificare la lettura e quindi l'individuazione del singolo dispositivo, l'identificativo univoco viene mostrato all'avvio del dispositivo sul display, il quale verrà successivamente spento, è attraverso la comunicazione UART. Questo ID sarà utilizzato dall'utente per registrare, sotto il proprio nome, il dispositivo sensoristico sulla piattaforma.

Un secondo problema affrontato è stato quello di individuare e gestire il ciclo operativo del dispositivo (duty cycle) e stabilire un tempo medio di deep-sleep. A tale scopo, è stato stabilito un tempo medio di funzionamento di circa 30/60 minuti in modalità deep-sleep (limitato a soli 5 secondi durante la fase di sviluppo e debug). Per implementare questa politica è stata definita una strategia di risveglio del dispositivo dalla modalità di deep-sleep, durante la quale tutti i moduli vengono disabilitati assieme alle interfacce di comunicazione.

Un terzo problema affrontato è stato quello di determinare un valore rappresentativo per la qualità dell'aria. Per raggiungere questo obbiettivo, è stato necessario sfruttare alcuni dati forniti dal sensore BME680, tra cui l'umidità e la resistenza dell'aria. Inoltre, per ottenere un indice di qualità dell'aria affidabile, il dispositivo effettua almeno 10 campionamenti. Questa fase si caratterizza per la sua lentezza, poiché il dispositivo esegue questi campionamenti a intervelli di tempo ben definiti, ma conserva solo gli ultimi 10 per calcolare successivamente l'indice di qualità dell'aria con un singolo campionamento.

Infine, l'ultimo problema affrontato è stato quello di elaborare un protocollo di comunicazione per facilitare lo scambio di informazioni tra i dispositivi sensoristici ed i Gateway.

A tal fine, è stata definita una struttura pacchettizzata denominata LoRaPacket, nella quale sono specificati la tipologia di applicazione (per consentire ai Gateway del medesimo progetto di identificare i pacchetti correlati), l'ID del dispositivo sensoristico che invia il pacchetto, la temperatura, la pressione, l'umidità e l'indice di qualità dell'aria, definendo un totale di 20 byte di pacchetto. Dopo l'invio del pacchetto, il dispositivo entra in modalità deep-sleep.

3.2 Dispositivo Gateway

Nell'implementazione dei dispositivi Gateway, è stato impiegato lo stesso framework Arduino, seguendo la stessa scelta adottata per i dispositivi sensoristici. Una delle sfide iniziali nello sviluppo del Gateway consisteva nella definizione di due modalità operative: configurazione e operativa. Durante il primo avvio, il Gateway entrerà in modalità configurazione, verificando la presenza di un configurazione Wi-Fi pre-esistente. In questa modalità, il Gateway fungerà da Access Point Wi-Fi e consentirà l'accesso ad una interfaccia Web servita dallo stesso dispositivo. Tramite questa interfaccia, sarà possibile configurare la rete Wi-Fi, salvarne le impostazioni e riavviare il dispositivo in modalità operativa. In modalità operativa, il dispositivo cercherà di connettersi alla rete Wi-Fi configurata in precedenza. Se la connessione non dovesse riuscire, il dispositivo si riavvierà nuovamente in modalità configurazione; in caso contrario, proseguirà in modalità operativa.

Un secondo ostacolo affrontato riguardava l'instaurare una connessione con un Broker MQTT, utilizzando una libreria che consentisse di collegarsi ad un Broker, pubblicare e sottoscrivere specifici argomenti. Dopo aver stabilito una connessione con il Broker, il dispositivo inizierà l'inizializzazione del proprio modulo LoRa per ricevere i pacchetti strutturati secondo il protocollo precedentemente definito.

L'ultimo dei problemi affrontati riguardava la gestione della ricezione dei pacchetti dei dispositivi sensoristici mediante il protocollo LoRa, il loro riconoscimento, estrarre le informazioni ed incapsularle in un formato MQTT adeguato per il servizio applicativo del progetto. Per identificare i pacchetti LoRa e assicurarci che provengano dai dispositivi sensoristici del progetto, vengono controllati i primi due byte del pacchetto ricevuto, in accordo con il tipo di progetto. Se questa verifica ha successo, il pacchetto noto viene estratto utilizzando la stessa struttura definita nel firmware sensoristici, altrimenti viene scartato. Successivamente, il pacchetto viene convertito in un formato JSON e pubblicato verso il Broker MQTT al quale è connesso, su un argomento specifico del progetto.

3.3 MQTT Reader Service

L'MQTT Reader Service è un servizio il cui compito è quello di memorizzare nel database le letture provenienti dei sensori. Per fare ciò il servizio deve innanzitutto stabilire una connessione nel database, definendo anche i modelli di informazione che andrà a utilizzare con il database mediante le API fornite da esso. Una volta stabilito la connessione al database, il servizio prova a collegarsi al Broker MQTT al quale si connetterano

anche i Gateway. Dopo essersi collegato con successo il servizio si sottoscrivere allo stesso argomento al quale i Gateway pubblicheranno i dati, permettendo così di ricevere le varie misurazioni dai sensori. Inoltre, lo stesso servizio filtrerà eventuali messaggi duplicati provenienti da più Gateway, poiché un dispositivo sensoristico potrebbe trovarsi all'interno dell'area di copertura di più di un Gateway.

3.4 Web Service

Il servizio Web Service definisce delle specifiche API HTTP per consentire l'interazione con il sistema, implementando un sistema di autenticazione utente. Inoltre, lo stesso servizio fornirà una comunicazione in tempo reale utilizzando il protocollo WebSocket (quando possibile) per la visualizzazione in tempo reale delle letture da parte dei dispositivi sensoristici. Infine, il servizio includerà un client Web per semplificare l'interazione con l'utente. Questo client presenta un interfaccia grafica attentamente progettata per garantire la piena reattività su una vasta gamma di dispositivi, sia desktop che mobile.

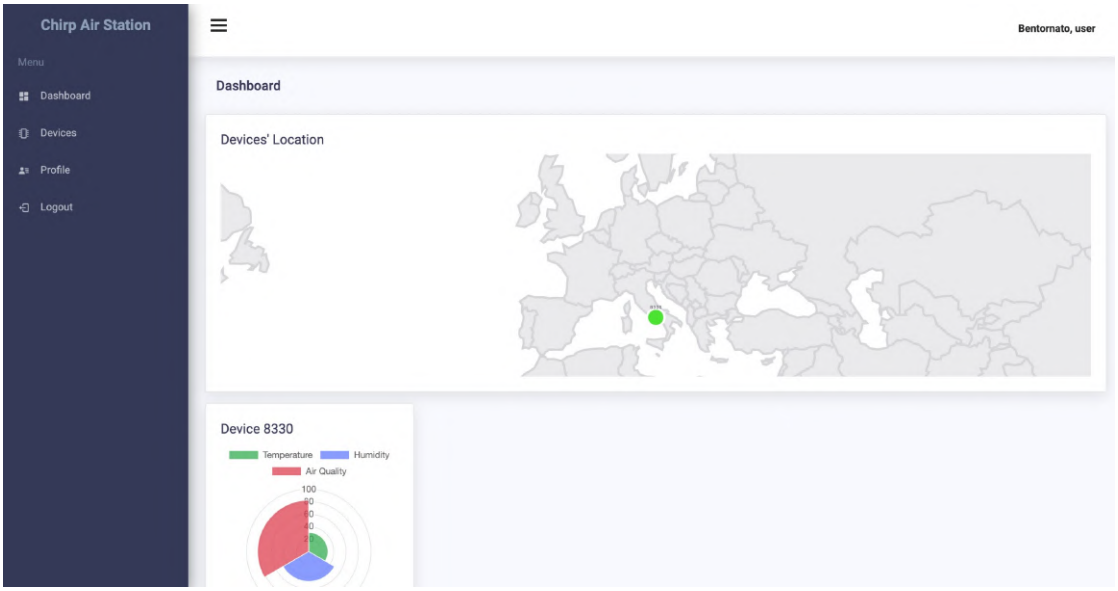

Figure 7: Screen della dashboard utente

L'interfaccia web è progettata per utilizzare un sistema di layout indipendentemente dalla view alla quale si vuole accedere. Tra i layout disponibili vi è quello pubblico, che offre una versione semplificata dell'interfaccia grafica accessibile a qualsiasi visitatore della piattaforma, ed il layout autenticato, che include il medesimo design aggiungendo però un menu di navigazione ed una barra utente, di solito posizionata in modo fisso dalla parte superiore dell'interfaccia.

Tra le view pubbliche vi sono la pagina di accesso e quella di registrazione utente, con la prima che funge da pagina principale della piattaforma. La view di accesso consente

all'utente di autenticarsi utilizzando l'indirizzo e-mail e la password, considerando che l'e-mail verrà utilizzata come identificativo unico dell'utente nel database. La view di registrazione consente agli utenti di creare un nuovo account fornendo le loro informazioni insieme all'indirizzo e-mail e la loro password. In caso di errori derivanti dalla validazione dei dati di accesso o dalla creazione dell'account, l'utente riceverà delle notifiche grafiche appropriate attraverso messaggi di errori gestiti in modo adeguato. Dopo la creazione dell'account o dopo il login, l'utente sarà autenticato dal server, avviando così la sua sessione e reindirizzandolo automaticamente alla prima pagina dell'layout autenticato.

4 Tecnologie utilizzate

Per quanto riguarda lo sviluppo del firmware del dispositivo sensoristico e del Gateway è stato utilizzato il framework Arduino il quale, come già detto in precedenza, permette una rapida implementazione logica sfruttando le API del framework Espressif. Per implementare la seralizzazione e la deserializzazione dei pacchetti LoRa in pacchetti JSON è stata utilizzata la libreria ArduinoJson[1], una libreria C++ che consente di manipolare i dati JSON in modo semplice ed efficiente su dispositivi a microcontrollore. La libreria offre un set di funzionalità che permettono di creare, analizzare e modificare dati JSON su dispositivi con risorse limitate, gestendo in maniera trasparente la memoria da utilizzare. Per la connessione e la pubblicazione MQTT è stata scelta la libreria PubSubClient[8], una libreria di pubblicazione/sottoscrizione per i microcontrollori. Per implementare l'interfaccia grafica web sul dispositivo, per permettere all'utente di configurare alla prima accensione il dispositivo, è stata sviluppata un interfaccia web semplificata e caricata all'interno del filesystem del firmware. Per poter servire tale interfaccia mediante HTTP è stata utilizzata la libreria ESPAsyncWebServer[9], la quale permette di definire un server HTTP asincrono su un microcontrollore. Infine, sono state utilizzate ulteriori librerie di comune utilizzo per la gestione del display e della connettività.

Il Web Service è stato sviluppato utilizzando NodeJS[12], un interprete/framework Javascript che abilita l'utilizzo di Javascript a livello back-end. A differenza degli interpreti Javascript, tipicamente utilizzati nei browser Web, NodeJS dispone di un set complete di chiamate di sistema che consentono un interazione completa con il sistema operativo sottostante. NodeJS si basa sull'utilizzo di moduli Javascript, ovvero librerie esterne o create dallo stesso sviluppatore, offrendo la possibilità di esportare e configurare funzionalità aggiuntive. Questi moduli e librerie possono essere gestiti ed installati tramite il Node Package Manager (NPM)[13], insieme al file package.json, che semplifica la gestione delle dipendenze del progetto. Questo approccio semplifica il processo di distribuzione del progetto e l'installazione delle sue dipendenze. NodeJS utilizza un modello di gestione degli eventi, principalmente attraverso la classe EventEmitter, che rappresenta il fondamento per tutti i moduli con comportamenti asincroni. Inoltre, NodeJS gestisce questi eventi in un contesto mono-thread, agevolando la scalabilità orizzontale delle applicazioni.

Per la comunicazione in tempo reale dei dati sensoristici, è stata utilizzata la libreria

Socket.IO[19]. Socket.IO è un modulo di NodeJS progettato per facilitare le comunicazioni in tempo reale attraverso gli eventi. Utilizzando Socket.IO, è possibile implementare la comunicazione in tempo reale mediante due protocolli a livello applicativo: HTTP e WebSocket[20]. La selezione del protocollo da utilizzare dipende dalla compatibilità del client. Socket.IO utilizza principalmente HTTP per simulare una comunicazione in tempo reale attraverso il Polling, ma se il client dovesse supportare anche il protocollo WebSocket, la libreria utilizzerà HTTP per richiedere l'upgrade del protocollo. Questo processo è gestito in maniera trasparente per lo sviluppatore da una componente chiamata Engine.IO[6], inclusa in Socket.IO. Il modulo consente lo scambio di messaggi tra client e server, mediante l'attivazione di eventi personalizzati. Inoltre, Socket.IO offre funzionalità avanzate come la gestione di namespace e stanze, oltre alla possibilità di trasmettere eventi in modalità broadcast.

Per la connessione al database è stato utilizzato Mongoose[10], un modulo NodeJS che semplifica l'interfacciamento con il database MongoDB attraverso operazioni asincrone. Questo modulo consente di definire modelli di dati e le loro proprietà all'interno dell'applicazione. Utilizzando questi modelli, è possibile eseguire operazioni comuni come la ricerca, aggiornamento, creazione, eliminazione e aggregazione dei dati utilizzando le operazioni a catena native di MongoDB. Inoltre, Mongoose offre funzionalità avanzate come la validazione dei dati e la definizione di relazioni tra i modelli, facilitando lo sviluppo e la gestione dei dati nel contesto dell'applicazione.

Come database, già citato sopra, è stato scelto MongoDB, un database orientato ai documenti e alle collezioni, caratterizzato dalla flessibilità nel conservare informazioni non omogenee all'interno delle stesse collezioni. Questa natura dinamica offre un sistema che può adattarsi a varie strutture di dati, consentendo una maggiore agilità nello sviluppo delle applicazioni. Tuttavia, le operazioni di ricerca possono essere più costose alle operazioni di creazione e aggiornamento, specialmente quando si richiedono query complesse. Per gestire questi query, MongoDB offre il supporto per le operazioni di aggregazione, che consentono di elaborare ed analizzare i dati in modi diversi. Inoltre, MongoDB è progettato per essere distribuito su più server, consentendo una scalabilità orizzontale per gestire carichi di lavoro crescenti in modo efficiente. Questa architettura distribuita rende MongoDB la scelta ideale per applicazioni che richiedono flessibilità e prestazioni scalabili.

Per migliorare il processo di deploy dei diversi servizi, anche su più macchine con sistemi operativi diversi, è stata adottata la tecnologia Docker[4]. Docker è una piattaforma di containerizzazione dei servizi, sfruttata nativamente nei sistemi Unix, che consente la creazione di ambienti isolati, chiamati container, all'interno dello stesso sistema. A differenza della virtualizzazione tradizionale, che emula l'intero stack hardware, Docker utilizza direttamente le risorse hardware del sistema host, garantendo prestazioni native ed un utilizzo efficiente delle risorse. Utilizzando Docker, è possibile creare immagini contenenti l'intero ambiente di esecuzione dei servizi, le quali possono essere eseguite come container indipendenti. Queste immagini possono essere scaricate dal registro ufficiale di Docker, da un registro privato o create utilizzando un file di specifica chiamato Dockerfile. Per gestire e coordinare il deploy dei servizi, comprese le loro dipendenze reciproche, è stato impiegato Docker Compose[5], uno strumento che permette di definire i servizi

e le relative dipendenze all'interno di un file YAML. Docker Compose si occupa quindi del building delle immagini e dell'avvio dei container, semplificando e automatizzando il processo di deploy dei servizi.

MQTT (Message Queueing Telemetry Transport) è un protocollo di comunicazione a livello appplicativo ampiamente utilizzando nel campo dell'Internet of Things. Basato sul modello Publish/Subscriber, MQTT consente la comunicazione tra dispositivi indipendentemente dall'infrastruttura di rete utilizzata, superando le limitazioni imposte da configurazioni NAT o Firewall. Il protocollo si basa sulla presenza di un Broker MQTT, il quale gestisce le connessioni e filtra i messaggi scambiati tra i vari nodi della rete IoT. Le comunicazioni avvengono attraverso l'uso di topic, ovvero stringhe di testo utilizzate per pubblicare o sottoscrivere messaggi. I topic possono essere strutturati in diversi livelli, utilizzando caratteri speciali per definire la gerarchia dei sottotopics.

Per il client web è stato utilizzato ReactJS[18], un framework per lo sviluppo frontend compatibile con Javascript e Typescript. Rispetto agli altri framework, React si concentra sulla creazione di microcomponenti, facilitando la progettazione e lo sviluppo di interfacce utente modulari. Un elemento distintivo di React è l'utilizzo del Virtual DOM, una rappresentazione virtuale del DOM (Document Object Model), che ottimizza le operazioni di modifica e aggiornamento, consentendo una maggiore velocità di rendering. Questo approccio, sebbene richieda un maggiore utilizzo di memoria, porta ad una migliore performance complessiva dell'applicazione. Inoltre, React offre la possibilità di integrare librerie esterne per ampliare le funzionalità del framework.

Tra le librerie impiegate, Axios[2] si distingue come una libreria client che semplifica le gestione delle richieste HTTP e la gestione degli errori, offrendo un alternativa all'uso di fetch. Un altra risorsa preziosa è stata la libreria React table component[16], che consente di creare in maniera asincrona tabelle dinamiche con funzionalità di ricerca e filtraggio. Per la rappresentazione delle informazioni geografiche, è stata utilizzata la libreria React Simple Maps[17], che facilita la creazione di mappe vettoriali utilizzando file di mappa, mentre per la rappresentazione dei grafici è stata impiegata React Chart.js[15]. Infine, per la progettazione dell'interfaccia utente è stata utilizzata la rinomata libreria Bootstrap[3], un framework CSS/JS che offre una vasta gamma di componenti predefiniti e altamente personalizzati tramite CSS.

Infine, per quanto concerne la progettazione e lo sviluppo dei dispositivi e del loro firmware, è stato adottato il framework Arduino, integrato con l'estensione PlatformIO[14] per Visual Studio Code. Questa combinazione offre un ambiente di sviluppo altamente efficiente e versatile, consentendo una programmazione agevole ed una gestione ottimizzata delle risorse hardware. Grazie a PlatformIO è possibile gestire facilmente librerie, eseguire il debug del codice e caricare il firmware sui dispositivi in modo rapido ed affidabile. Tale approccio si è dimostrato particolarmente efficace nell'assicurare la stabilità e le prestazioni ottimali dei dispositivi durante tutto il processo di sviluppo e test.

5 Deployment

Il progetto è stato sviluppato sfruttando, per il controllo di versione distribuito, il software Git, utilizzando il servizio offerto dalla piattaforma GitHub. Git è stato un elemento fondamentale nel processo di gestione del codice sorgente, offrendo un sistema di controllo delle versioni robusto e flessibile. Durante lo sviluppo del progetto, è stato utilizzato in combinazione con submoduli per gestire in modo efficiente ogni singola repository, in modo da organizzare il progetto in maniera modulare. Grazie all'uso strategico di Git e dei submoduli, è stato possibile garantire una maggiore coerenza e coesione nel codice migliorando la tracciabilità delle modifiche e agevolando il processo di deployment. Innanzitutto, è necessario installare Git nel sistema ove si vuole effettuare il deploy di tutto il servizio. E' anche possibile effettuare il deploy di ogni microservizio anche su macchine diverse, clonando le repository di ogni submodulo di ciascun servizio. Per semplificare il deploy di tutte le componenti necessarie, consigliamo di effettuare il deploy dalla repository che comprenderà tutti i submoduli[1]. Dopo avere installato Git e Make, basterà eseguire i seguenti comandi bash da linea di comando:

```
#!/bin/bash
git clone https://github.com/antoniorotundo2/chirp-air-station
cd chirp-air-station
make install
make update
```

In sintesi, questi comandi cloneranno la repository principale del progetto e inizializzeranno tutti i submoduli del progetto. Una volta preparati tutti i submoduli, sarà necessario istanziare i servizi necessari, per fare ciò utilizzeremo Docker.

Docker è una piattaforma che sfrutta la tecnologia di containerizzazione per creare, distribuire e gestire applicazioni in ambienti isolati chiamati container. Un container Docker include tutto il necessario per eseguire un applicazione, compresi il codice, le librerie e le dipendenze, garantendo che l'applicazione funzioni in modo coerente e affidabile su qualsiasi sistema in cui viene eseguita. I container Docker sono leggeri, portabili e autosufficienti, rendendoli ideali per la distribuzione di applicazioni in ambiente di sviluppo, test e produzione. Docker permette agli sviluppatori di poter creare, condividere e distribuire facilmente le proprie applicazioni insieme a tutto l'ambiente necessario, eliminando così i problemi di compatibilità e configurazione che possono sorgere su sistemi diversi. E' possibile installare Docker su qualsiasi piattaforma, ciò nonostante solo su piattaforme Linux non avviene alcuna virtualizzazione, poiché lo stesso Docker sfrutterà il kernel di sistema per gestire i vari container. Per installare Docker, nel caso di Linux basterà eseguire i seguenti comandi da terminale:

```
#!/bin/bash
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

[1] https://github.com/antoniorotundo2/chirp-air-station

Docker dà disposizione uno script di convenienza per installare l'engine all'interno del proprio sistema. Questa modalità di installazione non è però consigliata per ambienti di produzione, ma è utile per installare Docker in ambiente di test e sviluppo. Lo script proverà a capire anche la tipologia di distribuzione Linux del sistema e di configurare il package manager per poter installare Docker. Di default, lo script installerà sempre l'ultima versione di Docker stabile, assieme a tutte le sue dipendenze. Un volta installato Docker, il sistema avvierà un service demone che sfrutterà le socket Unix per poter gestire i vari comandi, tuttavia solo l'utente root ha accesso a queste tipologie di socket pertanto, per installare ed usare Docker sarà necessario avere i privilegi di amministratore di sistema. Se, per ovvie comodità si voglia utilizzare Docker senza i permessi di amministratore, operazione non consigliata, sarà necessario aggiungere il proprio utente al gruppo di utenti Docker:

```
#!/bin/bash
sudo groupadd docker
sudo usermod -aG docker $USER
```

Per testare la corretta installazione di Docker, basterà eseguire un container di test mediante il seguente comando:

```
#!/bin/bash
docker run hello-world
```

Il seguente comando scaricherà un immagine di test e avvierà un container utilizzando quest'ultima, stampando un messaggio di Hello World e chiudendo il container. Come per l'immagine di test, anche il progetto fa largo uso delle immagini Docker.

Le immagini Docker sono gli elementi fondamentali che consentono di creare, distribuire e gestire i container Docker in maniera efficace. Un immagine Docker non è altro che un file di sola lettura che contiene tutto il necessario per eseguire un applicazione all'interno di un container Docker. Questo include il codice dell'applicazione, le librerie di sistema, le dipendenze e le configuraizoni necessarie. Le immagini Docker vengono organizzate in layer, ognuno dei quali rappresenta una modifica apportata all'immagine Docker di base. Le immagini vengono create utilizzando un file di configurazione chiamato Dockerfile e possono essere distribuite e riutilizzate semplicemente trasferendo lo stesso Dockerfile. Un Dockerfile è un file di testo che contiene una serie di istruzione per la creazione di un immagine Docker. Le istruzioni nel Dockerfile sono eseguite sequenzialmente dall'alto verso il basso e vengono utilizzate per creare una serie di layer che comporranno l'immagine finale. Ogni istruzione aggiunge o modifica un layer esistente, consentendo di creare immagini efficienti e leggere. Un esempio di Dockerfile del progetto è quello che crea e gestisce l'immagine per il Web Service:

```
FROM ubuntu:22.04
SHELL ["/bin/bash","--login","-c"]
RUN apt-get update -y
RUN apt-get install curl -y
```

```
RUN adduser --disabled-password --gecos "" cas-user
RUN adduser cas-user sudo
RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
USER cas-user
WORKDIR /home/cas-user/
RUN touch ~/.bashrc && chmod +x ~/.bashrc
RUN curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
ENV NVM_DIR /home/cas-user/.nvm
RUN . $NVM_DIR/nvm.sh && nvm install 18.14.2
ENV NODE_PATH $NVM_DIR/v18.14.2/lib/node_modules
ENV PATH $NVM_DIR/versions/node/v18.14.2/bin:$PATH
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm","start"]
```

Il seguente Dockerfile crea un immagine a partire da un immagine di base di Ubuntu versione 22.04, esegue l'aggiornamento delle repositories, installa la dipendenza curl e crea un utente denominato cas-user che verrà utilizzato dal servizio all'interno del container. Una volta creato l'utente e la sua directory di lavoro, verrà installata la libreria nvm per poter installare successivamente la versione di NodeJS 18.14. Infine, verrà copiata tutto il contenuto della cartella ove si trova lo stesso Dockerfile all'interno della home dell'utente creato all'interno del container, verranno installate le dipendenze e librerie per il servizio, si esporrà la porta relativa al servizio da avviare e si definirà il comando da eseguire dopo l'esecuzione del container, comando che avvierà il servizio web.

Come per il servizio web, anche il servizio MQTT ed il servizio database avranno il loro Dockerfile, permettendo così la build delle tre immagini e di poter avviare singolarmente i servizi. Per semplificare ed automatizzare tale operazione, occorre però un altro strumento che permetta di gestire la build e l'avvio dei container in maniera intelligente, considerando anche le dipendenze tra servizio e servizio. Infatti, il servizio web ed il servizio MQTT non potrebbero essere avviati prima che un servizio database sia avviato, cosa di cui l'amministratore deve tenere conto. Per automatizzare tali processi è stato utilizzato Docker Compose.

Docker Compose è uno strumento che consente di definire e gestire applicazioni multi-container Docker, in maniera intuitiva. Docker Compose sfrutta un singolo file di configurazione denominato docker-compose.yaml. Questo file contiene una serie di definizioni che descrivono i container, le reti ed i volumi utilizzati dall'applicazione, nonché le loro interazioni e dipendenze. Con Docker Compose è quindi possibile avviare, fermare e gestire l'intera applicazione utilizzando un singolo comando, semplificando così il processo di sviluppo e test dell'applicazione. Inoltre, fornisce funzionalità avanzate per la gestione delle variabili d'ambiente, la scalabilità dei servizi e la gestione dei volumi di dati. All'interno della repository del progetto, contenente anche tutti i submoduli, è stato definito un file docker-compose.yaml per gestire l'avvio di tutta l'applicazione e di

tutti i microservizi:

```
version: '3'
services:
    cas-mqtt-service:
        build:
            context: ./cas-mqtt-service
            dockerfile: Dockerfile
        image: cas-mqtt-service
        depends_on:
            - cas-database
    cas-web-service:
        build:
            context: ./cas-web-service
            dockerfile: Dockerfile
        image: cas-web-service
        depends_on:
            - cas-database
        ports:
            - 80:8080
    cas-database:
        container_name: cas-db
        image: mongo
        ports:
            - 27017:27017
```

Questo file consente a Docker Compose di buildare le immagini cas-web-service e cas-mqtt-service, utilizzando come contesto i submoduli e il Docker file di ogni submodulo. Inoltre, permette di definire le porte di rete da utilizzare o rimappare (il web service internamente utilizza la porta 8080 che verrà rimappata nel sistema host sulla porta 80). Infine, permette di definire per entrambi i servizi, una dipendenza sul servizio cas-database, il quale, a differenza dei due servizi che dovranno essere buildati per creare le nuove immagini, sfrutta l'immagine già esistente all'interno del registro di Docker pubblico di MongoDB. Per poter avviare o fermare i microservizi basterà eseguire questi comandi all'interno della cartella del progetto:

```
#!/bin/bash
#avvio dei servizi:
docker-compose up

#stop dei servizi
docker-compose down
```

Infine per semplificare ulteriori comandi per la gestione dei microservizi e altre operazioni di gestione, è stato utilizzato Make. Make è un tool che legge un file di configurazione chiamato Makefile per eseguire una serie di comandi, utilizzando regole e

dipendenze definite al suo interno. Make utilizza un linguaggio di scripting semplice e leggibile per definire le regole e le dipendenze. Ogni regola è costituita da un target, da una lista di dipendenze ed un insieme di comando da seguire per generare il target. Nel progetto è stato utilizzato Make per semplificare ed automatizzare alcune operazioni comuni, come l'avvio e lo stop dei servizi, l'aggiornamento dei submoduli e la rimozione delle immagini e dei container del servizio. All'interno del progetto vi è il seguente Makefile:

```
install:
        git submodule init
        git submodule update --init
update:
        git submodule foreach git pull origin main
        git submodule update --remote
run:
        docker-compose up -d
stop:
        docker-compose down
stop-clear:
        docker-compose down --rmi all
```

Il seguente Makefile definisce alcuni comandi utili per le operazioni più comuni:

```
#!/bin/bash
#installazione dei submoduli
make install

#aggiornamento dei subumoduli
make update

#avvio dei servizi
make run

#stop dei servizi
make stop

#stop dei servizi e disinstallazione
make stop-clear
```

E' possibile quindi utilizzare i comandi definiti nel Makefile per gestire l'avvio o lo stop dei microservizi del progetto.

Per quanto riguarda i submoduli dei due dispositivi, ovvero il Gateway e il dispositivo sensoristico, dobbiamo fare una piccola introduzione alla piattaforma PlatformIO. PlatformIO è un ecosistema per lo sviluppo e l'automazione del processo di sviluppo di software embedded. Offre un ambiente unificato per lo sviluppo di firmware per

una vasta gamma di microcontrollori e microprocessori consentendo agli sviluppatori di scrivere, compilare, caricare e debuggare il codice in modo semplice ed intituitivo. PlatformIO supporta anche una vasta gamma di piattaforme hardware e framework software, rendendolo quindi una scelta appropriata per il progetto. In particolare, oltre a supportare le piattaforme hardware Arduino, supporta anche le piattaforme ESP32. Inoltre, gestisce automaticamente le dipendenze del progetto, inclusi i framework software e le librerie esterne. Tutto ciò è possibile grazie al file di configurazione denominato platformio.ini dove vengono definite le dipendenze e le varie configurazioni per potere compilare il progetto finale, processo di compilazione che viene ampiamente semplificato in maniera dinamica in funzione del framework utilizzato e dal microcontrollore scelto. PlatformIO è un ambiente completamente integrato all'interno di VisualStudio Code, offrendo un esperienza di sviluppo moderna e ricca di funzionalità, dando a disposizione la stessa interfaccia utente di VSCode per poter progettare e sviluppare il firmware del dispositivo.

Una volta inizializzati i submoduli del Gateway e del dispositivo sensoristico, basterà aprire la cartella del submodulo con VisualStudio Code, avendo PlatformIO installato mediante le estensioni di VSCode, ed il progetto verrà automaticamente identificato come progetto di PlatformIO, permettendo la compilazione e l'upload del firmware direttamente sul dispositivo connesso al proprio computer.

6 Esempio di Utilizzo

Ipotizziamo di voler avviare l'intero servizio e di avere a disposizione due dispositivi, uno che fungerà da Gateway e uno da dispositivo sensoristico, compreso un sensore di temperatura, pressione ed umidità.

In primo luogo, come già spiegato nella sezione precedente, avviamo i servizi necessari per il funzionamento del progetto mediante il Docker Compose. Una volta eseguiti correttamente tutti i servizi sarà possibile accedere alla schermata principale del servizio.

La prima schermata che il servizio ci offre sarà la schermata di login (Figura 8), tramite la quale sarà possibile eseguire l'accesso con i propri dati, a prescindere dal proprio ruolo. Essendo la prima volta che si esegue il servizio, è chiaro che non ci saranno utenti registrati, sarà quindi necessario effettuare una prima registrazione del primo utente della piattaforma attraverso la schermata di registrazione, accessibile direttamente da quella di login.

Una volta aggiunti alla schermata di registrazione (Figura 9), sarà possibile inserire pochi dei propri dati per potere creare l'account utente. E' importante precisare che l'esimio numero di dati richiesti dipende semplicemente dalla progettazione semplificato del modello utente, il quale potrà essere ampliata semplicemente modificando il modello utente. Ogni utente registrato avrà ruolo di utente normale e solo dopo potrà essere promosso ad amministratore dall'amministratore di sistema. Una volta eseguita la registrazione, sarà eseguito l'accesso automatico al sistema e l'utente verrà riportato alla sua dashboard.

Nella dashboard (Figura 10) è possibile visionare i propri dispositivi registrati ed un

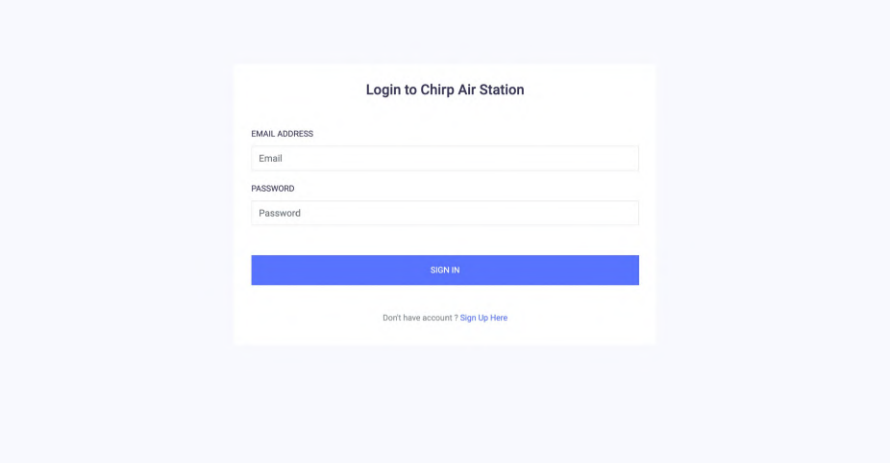

Figure 8: Login del servizio

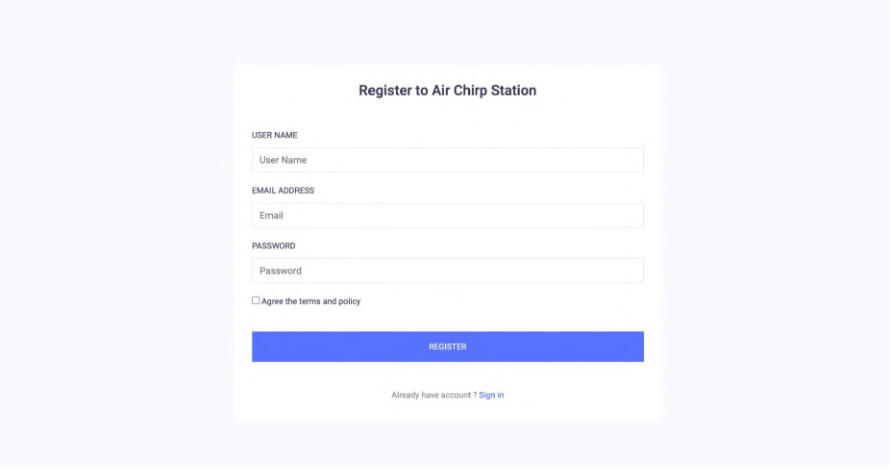

Figure 9: Registrazione al servizio

menù a sinistra con il quale interagire, che ci porterà a diverse schermate. Prima di procedere, sarà necessario programmare e avviare un dispositivo Gateway.

Una volta avviato un Gateway per la prima volta entrerà in modalità configurazione, permettendo all'utente di configurare la connessione Internet dello stesso Gateway. Il funzionamento è semplice, in questa modalità il Gateway creerà un Access Point Wi-Fi e servirà un captive portal tramite il quale l'utente potrà inserire i dati della propria connessione Wi-Fi e configurare il proprio Gateway. Al salvataggio di queste informazioni, il Gateway si riavvierà e proverà a connettersi con i dati inseriti dall'utente, se la connessione dovesse riuscire il Gateway sarà configurato correttamente altrimenti, il Gateway tornerà nuovamente in modalità configurazione. Questa operazione è necessaria poiché non ci sono altri Gateway attivi nella zona ma è importante precisare che l'attivazione

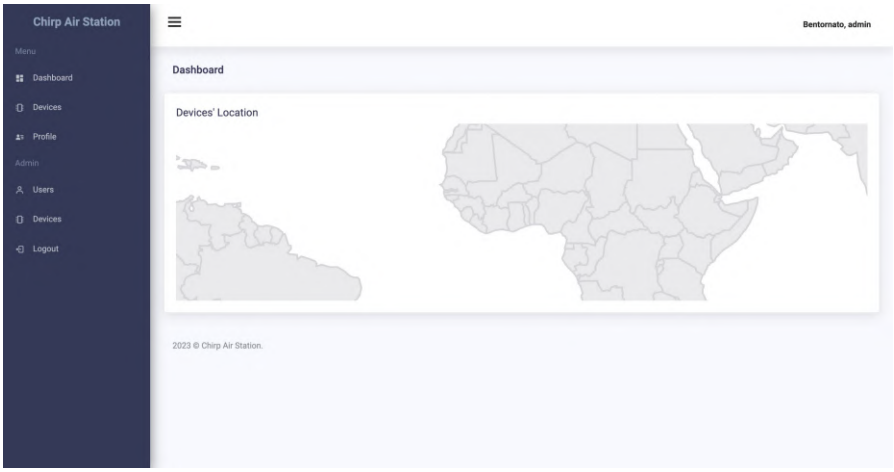

Figure 10: Dashboard dell'utente

di altri Gateway, anche non personali, contribuisce all'estensione del servizio. Una volta configurato correttamente il proprio Gateway e programmato il dispositivo sensoristico, siamo pronti a registrare il nostro dispositivo sulla piattaforma.

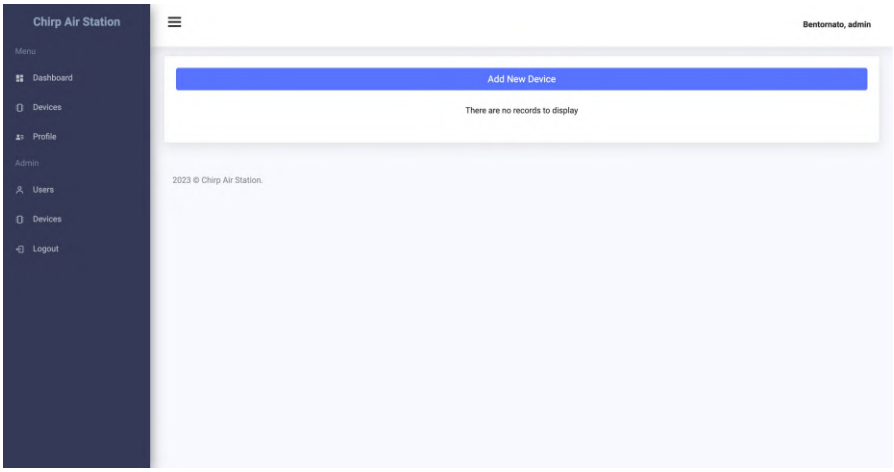

Figure 11: Schermata dei device registrati

Dal menù della piattaforma (Figura 11) è possibile accedere alla lista dei propri device (o ai device di tutti gli utenti se si è amministratore) e anche di poterne aggiungere di nuovi. Per fare ciò è necessario cliccare sul pulsante "Add new device" e allo stesso tempo accendere il proprio device.

Una volta acceso il proprio dispositivo (Figura 12), quest'ultimo mostrerà su schermo brevemente il proprio identificativo univoco che l'utente dovrà inserire nella piattaforma per poter registrare il dispositivo a proprio nome.

Una volta registrato il proprio dispositivo (Figura 13) sarà possibile modificarne la

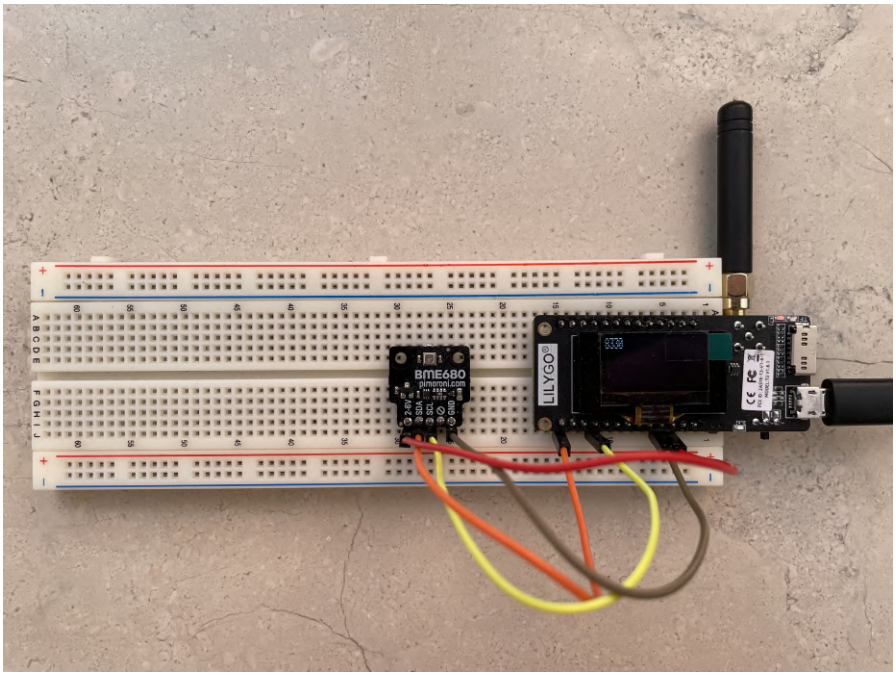

Figure 12: Dispositivo sensoristico acceso

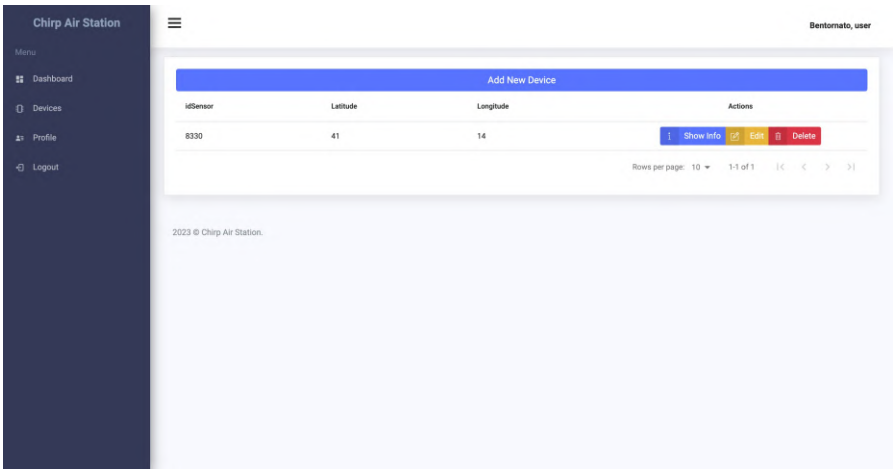

Figure 13: Schermata dispositivi dopo l'aggiunta

posizione geografica e visionare i dati in tempo reale provenienti dal dispositivo.

Dalla schermata del dispositivo (Figura 14) è possibile monitorare in tempo reale i dati proveniente da essa e esaminarne anche lo storico dei dati.

Questo rappresenta un utilizzo d'esempio della piattaforma e dei dispositivi. E' chiaro che il processo di aggiunta dei dispositivi ed il loro monitoraggio sia del tutto simile ai passaggi eseguiti finora.

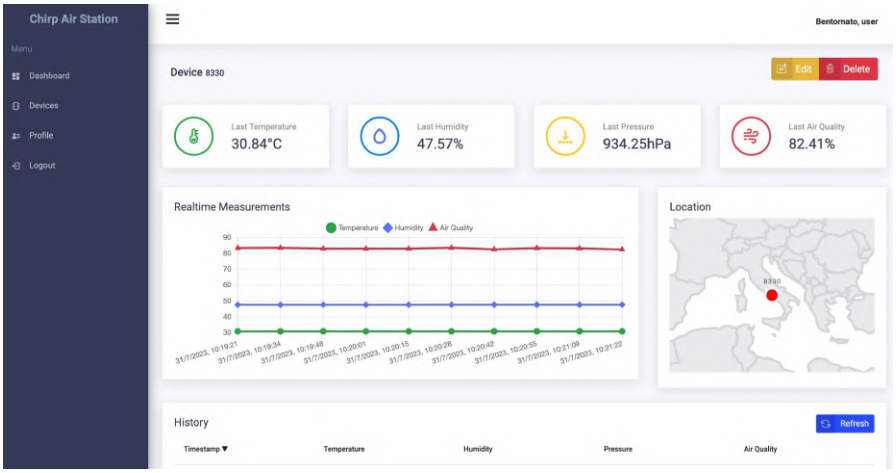

Figure 14: Schermata del dispositivo

7 Conclusioni

Il progetto rappresenta una piattaforma distribuita per l'analisi metereologica della qualità dell'aria in un contesto metropolitano. Il progetto si presenta espandibile a livello geografico mediante l'installazione di nodi sensoristici e Gateway in opportune locazione della metropoli. Le informazioni provenienti dalla rete sensoristica vengono trasmesse mediante protocollo opportuno e adattabile a qualsiasi infrastruttura di rete che il servizio andrà ad utilizzare e conservate all'interno di opportuni database.

Oltre alla rete sensoristica espandibile, il progetto si compone anche di una serie microservizi, distribuibili e deployabili attraverso la tecnologia Docker. Tra i servizi è stato progettato opportunatamente un servizio che abbia il compito di ottenere le informazioni dalla rete sensoristica e di conservarle all'interno di un servizio database. Il servizio principale, il quale mette a disposizione una serie di API con le quali è possibile interagire con l'intero sistema, sfruttando meccaniche di autenticazione ed è stato sviluppato con le tecnologie necessarie affinché lo stesso servizio potesse essere scalabile, orizzontalmente e verticalmente, e distribuibile anche su più host.

Infine, lo stesso servizio principale dà a disposizione un client Web, dotato di opportuna interfaccia grafica, sviluppata mediante tecnologie e framework moderni, la quale permette l'interazione completa con il sistema, compresa la possibilità di monitorare in tempo reale i propri dispositivi.

7.1 Lavori Futuri

Il progetto così come sviluppato, permetterà in futuro di poter essere ampliato di nuove funzionalità, come ad esempio l'aggiunta di nuovi dati sensoristici da parte della rete sensoristica, senza dover modificare ulteriormente il progetto. Inoltre, la struttura stessa del progetto ne permette la distribuzione su qualsiasi infrastruttura, che sia cloud pubblico o privato.

7.2 Cosa ho imparato

Durante la progettazione del seguente progetto, ho avuto modo di imparare a programmare sistemi a microcontrollori e capire quanto in sistemi del genere sia importante la buona programmazione, sia in termini di performance che gestione della memoria. Inoltre, cosa più importante, è stato chiaro come un sistema distribuito potesse non essere solo software ma anche hardware, in questo caso come una rete di sensori. In merito a questo, è stato proficuo scoprire e studiare protocolli di comunicazione a lungo raggio e a basso consumo energetico come LoRa e di come implementare, per quanto semplice, un proprio protocollo di comunicazione sfruttando questo canale fisico, realizzando una rete di sistemi distribuita che potesse funzionare anche su lunghe distanze.

Un aspetto fondamentale è stato quello di realizzare un integrazione tra protocolli e sistemi totalmente diversi tra loro, in quanto, non solo a basso livello sfruttando protocolli IoT ma anche sull'alto livello sfruttando, in maniera ottimale, più protocolli applicativi, tra i quali MQTT, HTTP e WebSocket.

Infine, ho avuto modo di sperimentare nuove tecnologie pensate appositamente per un facile sviluppo e distribuzione come NodeJS, il quale mi ha permesso di progettare e sviluppare una soluzione funzionante in poco tempo; Docker, che mi ha permesso di poter rendere la mia soluzione a microservizi più semplice da portare ed eseguire.

References

[1] Arduinojson: Efficient json serialization for embedded c++. https://arduinojson.org/.

[2] Axios, promise based http client for the browser and node.js. https://www.npmjs.com/package/axios.

[3] Bootstrap · the most popular html, css, and js library in the world. https://getbootstrap.com/.

[4] Docker: Accelerated container application development. https://www.docker.com/.

[5] Docker compose. https://docs.docker.com/compose/.

[6] The engine.io protocol — socket.io. https://socket.io/docs/v4/engine-io-protocol/.

[7] Esp32 wi-fi & bluetooth soc — espressif systems. https://www.espressif.com/en/products/socs/esp32.

[8] knolleary/pubsubclient: A client library for the arduino ethernet shield that provides support for mqtt. https://github.com/knolleary/pubsubclient.

[9] me-no-dev/espasyncwebserver: Async web server for esp8266 and esp32. https://github.com/me-no-dev/ESPAsyncWebServer.

[10] Mongoose, a mongodb object modeling tool. https://www.npmjs.com/package/mongoose.

[11] Mqtt - the standard for iot messaging. https://mqtt.org/.

[12] Node.js. https://nodejs.org/en.

[13] npm — home. https://www.npmjs.com/.

[14] Platformio, your gateway to embedded software development excellence. https://platformio.org/.

[15] React chartjs. https://www.npmjs.com/package/react-chartjs-2.

[16] React data table component. https://www.npmjs.com/package/react-data-table-component.

[17] React simple maps. https://www.npmjs.com/package/react-simple-maps.

[18] React, the library for web and native user interfaces. https://react.dev/.

[19] Socket.io, bidirectional and low-latency communication for every platform. https://socket.io/.

[20] The websocket api (websockets). https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

[21] Semtech. Lora and lorawan: Technical overview. https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/.