Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Realizzazione di un sistema embedded per la misurazione di dati atmosferisci e la previsione di pioggia

Docenti

Prof. Dragoni Franco Aldo

Componenti del gruppo

Bellante Luca
Zazzarini MIcol

Anno Accademico 2024-2025

Contents

1 Introduzione

Il presente progetto ha come obiettivo la realizzazione di un sistema embedded completo ("verticale") in grado di raccogliere dati atmosferici — nello specifico temperatura, umidità e pressione — e utilizzarli per prevedere in modo binario (sì/no), attraverso un modello di Machine-Learning, la possibilità di pioggia. Tali dati, una volta acquisiti e processati, vengono inviati a un server remoto (ThingSpeak), rendendoli accessibili da remoto tramite browser web o dispositivi mobili (Android/iOS). Inoltre, l'infrastruttura cloud consente l'integrazione con strumenti di analisi esterni, come Matlab, per ulteriori elaborazioni o rappresentazioni grafiche avanzate.

Il sistema, completamente autonomo, si basa sull'integrazione di più tecnologie hardware e software, ed è pensato per essere replicabile e adattabile ad altri contesti simili. Un aspetto fondamentale del progetto è l'impiego di un modello di Machine Learning (ML) direttamente eseguito a bordo della scheda STM32, rendendo il sistema indipendente da connessioni continue a un server per l'inferenza.

Il progetto trova applicazione in diversi ambiti. Ad esempio:

- **Monitoraggio ambientale in aree rurali**: un sistema a basso consumo che effettua previsioni meteo localizzate può fornire un supporto importante per l'agricoltura di precisione.
- **Stazioni meteo domestiche intelligenti**: può essere integrato in una smart home per attivare automaticamente tapparelle o coperture in caso di previsione di pioggia.
- **Sistemi di allerta urbana**: utile come nodo di una rete distribuita di sensori per il rilevamento di condizioni atmosferiche potenzialmente critiche in città.
- **Didattica e prototipazione**: data la semplicità del sistema e l'utilizzo di tecnologie open source, può essere usato come base per progetti didattici nelle scuole superiori o università.

1.1 Tecnologie utilizzate

In questa sezione vengono descritte le tecnologie utilizzate, sia hardware che software, e il modo in cui esse sono state integrate per ottenere un sistema funzionale e ottimizzato.

1.1.1 Hardware

STM32-F446RE La scheda STM32-F446RE, mostrata in Figura 1, è stata scelta per le sue buone prestazioni, il supporto a librerie avanzate come CMSIS e per l'elevata compatibilità con il sistema operativo real-time Zephyr. Si tratta di una scheda economica

ma dotata di un microcontrollore ARM Cortex-M4 a 180 MHz, con FPU integrata, utile per operazioni di inferenza su modelli leggeri di ML.

Questa scheda è spesso utilizzata in progetti embedded di tipo industriale, accademico e di prototipazione rapida grazie all'ecosistema ST e alla documentazione abbondante. Diversi lavori in letteratura e repository open-source ne dimostrano la validità in contesti simili a quello affrontato in questo progetto [3].

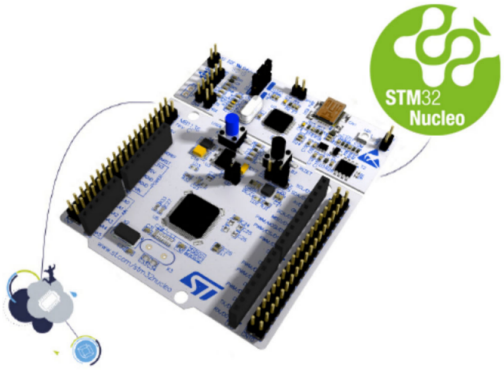


Figure 1: Microcontrollore STM32-F446RE.

ESP32-DevKitC-32E La scheda **ESP32-DevKitC-32E**, mostrata in Figura 2, è stata scelta per questo progetto in quanto rappresenta una soluzione economica, affidabile e largamente adottata nella prototipazione di dispositivi IoT grazie alla sua versatilità e al supporto di una vasta community.

Questo modulo è dotato del chip `ESP32-WROOM-32E`, che integra un microcontrollore dual-core Tensilica LX6 con frequenza di clock fino a 240 MHz, supporto Wi-Fi e Bluetooth a basso consumo energetico (BLE). Tali caratteristiche lo rendono particolarmente adatto per scenari in cui è necessario connettere dispositivi embedded a servizi cloud o reti locali in modalità wireless.

Nel nostro caso specifico, l'ESP32 è stato utilizzato per trasmettere i dati ambientali, elaborati dalla STM32, a un server remoto (*ThingSpeak*) attraverso una connessione Wi-Fi. Inoltre, la compatibilità con ambienti di sviluppo come *Arduino IDE* e *PlatformIO* ha reso l'integrazione rapida ed efficiente.

Per ulteriori dettagli tecnici sulle specifiche della scheda è possibile consultare il datasheet ufficiale disponibile in [2].

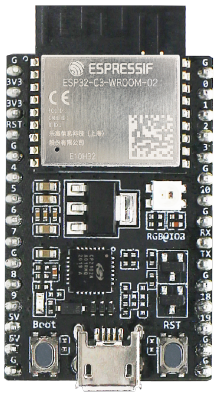


Figure 2: ESP32-DevKitC-32E.

GYBME280 Il modulo **GYBME280**, illustrato in Figura 3, integra il sensore Bosch BME280, il cui datasheet ufficiale è consultabile al riferimento [1]. Si tratta di un sensore digitale ad alta precisione, in grado di rilevare temperatura, pressione atmosferica e umidità relativa.

Il modulo GYBME280 include un regolatore di tensione integrato e un convertitore di livello logico (level shifter), che ne consente l'interfacciamento con microcontrollori a 3.3V o 5V tramite il protocollo di comunicazione I^2C[1].

Abbiamo scelto questo modulo per la sua compattezza, il basso consumo energetico e il costo contenuto, considerando anche che racchiude tre sensori in un unico dispositivo, facilitando l'integrazione nei sistemi embedded a risorse limitate.

Figure 3: Modulo sensore GYBME280.

[1] I^{2}C (Inter-Integrated Circuit) è un protocollo di comunicazione seriale sincrona master-slave a due fili, ampiamente utilizzato nei sistemi embedded per la trasmissione di dati tra microcontrollori e periferiche.

1.1.2 Software

Zephyr OS **Zephyr OS** è un sistema operativo real-time (RTOS)[2], open-source, scritto in linguaggio C, leggero ed estremamente modulare. Come mostrato in Figura 4, Zephyr è supportato da un'ampia community ed è compatibile con numerose piattaforme hardware, tra cui la `STM32-F446RE`, utilizzata nel nostro progetto.

In questo lavoro, Zephyr è stato impiegato per la gestione concorrente dei task di lettura dei sensori, esecuzione del modello di machine learning e comunicazione seriale con l'ESP32, assicurando reattività e gestione efficiente delle risorse.

Figure 4: Logo di *Zephyr OS*.

Arduino L'*IDE Arduino*, rappresentato in Figura 5, è stato utilizzato per la programmazione del modulo `ESP32-DevKitC-32E`. È stato scelto per la semplicità d'uso e per la vasta disponibilità di esempi e librerie che semplificano l'integrazione con componenti esterni, come il sensore descritto in 1.1.1. Il linguaggio, basato su C++, è facilmente comprensibile, anche per sviluppatori meno esperti.

Figure 5: Interfaccia grafica di Arduino IDE.

ThingSpeak *ThingSpeak* 6 è una piattaforma di IoT analytics basata su cloud, che consente di raccogliere, memorizzare, visualizzare ed elaborare dati provenienti da dispositivi connessi. Supporta la comunicazione tramite API RESTful (POST/GET) ed è direttamente integrabile con *Matlab*, il che permette una facile analisi e visualizzazione avanzata dei dati. Nel nostro progetto è stato utilizzato per inviare i dati ambientali elaborati dall'ESP32 e renderli consultabili da browser o dispositivi mobili.

Flutter **Flutter**, il cui logo è presente in Figura 7, è un framework open-source sviluppato da Google per la creazione di interfacce grafiche native cross-platform, cioè compi-

[2]RTOS: Real-Time Operating System, un sistema operativo progettato per garantire la prevedibilità temporale dell'esecuzione dei task, essenziale nei sistemi embedded.

Figure 6: Logo ThingSpeak.

Figure 7: Logo di Flutter

labili sia per Android che per iOS a partire dallo stesso codice sorgente (in linguaggio Dart). Nel nostro progetto è stato utilizzato per sviluppare un'applicazione mobile in grado di leggere e visualizzare i dati raccolti su ThingSpeak, fornendo così un'interfaccia utente intuitiva e accessibile da smartphone.

SPIN *SPIN* 8 è un formal model checker sviluppato per la verifica di sistemi concorrenti. Permette di modellare il comportamento di un sistema in linguaggio Promela e di verificarne proprietà come l'assenza di deadlock, race condition, correttezza dei protocolli di comunicazione, ecc. In questo progetto, SPIN è stato utilizzato per simulare e verificare alcune proprietà critiche legate al comportamento concorrente del sistema real-time basato su Zephyr, con particolare attenzione alla comunicazione tra task e alla sincronizzazione dei dati.

Figure 8: Logo di Spin.

2 Modello AI

La previsione meteorologica rappresenta una delle applicazioni più promettenti per i sistemi embedded intelligenti, specialmente in ambiti dove la connettività è limitata o l'elaborazione locale dei dati è preferibile. In questo progetto, abbiamo sviluppato un sistema capace di raccogliere dati atmosferici in tempo reale tramite un sensore BME280, e di utilizzare un modello di apprendimento automatico per prevedere la possibilità di pioggia nel giorno successivo. Il tutto è stato progettato per funzionare su un microcontrollore STM32 con sistema operativo Zephyr RTOS.
Descriviamo ora in dettaglio lo sviluppo e l'integrazione del modello di Machine Learning (ML) all'interno del sistema, per la previsione delle precipitazioni atmosferiche. Il modello scelto è un albero decisionale addestrato su un dataset meteorologico opportunamente pre-processato e bilanciato. La pipeline comprende l'esplorazione del dataset, il bilanciamento tramite SMOTE, l'addestramento e la validazione del modello, e la sua conversione in codice C compatibile con ambienti embedded. Il sistema è progettato per utilizzare i dati raccolti dal sensore BME280 (temperatura, umidità, pressione), e fornire in tempo reale una predizione sulla possibilità di pioggia.

2.1 USA Rainfall Prediction Dataset (2024–2025)

Il dataset utilizzato per l'addestramento del modello è denominato **USA Rainfall Prediction Dataset (2024–2025)**, reso disponibile tramite la piattaforma Kaggle (Dataset). Questo dataset raccoglie dati meteorologici giornalieri da 20 città principali degli Stati Uniti, coprendo un periodo di due anni, dal 2024 al 2025. Ogni riga del file rappresenta le misurazioni meteorologiche effettuate in una specifica città in una determinata data.

Le colonne principali incluse nel dataset sono:

- `Date` – data della misurazione, nel formato YYYY-MM-DD
- `Location` – città di riferimento
- `Temperature` – temperatura media giornaliera
- `Humidity` – umidità relativa media
- `Wind Speed` – velocità media del vento
- `Cloud Cover` – copertura nuvolosa espressa in percentuale
- `Pressure` – pressione atmosferica media

- `Precipitation` – quantità di precipitazioni giornaliere
- `RainTomorrow` – etichetta binaria, target del modello, che indica se il giorno successivo ha piovuto (1) o meno (0)

L'obiettivo principale del progetto è prevedere la variabile `RainTomorrow`, rendendo il problema una **classificazione binaria**. Il dataset si presta quindi bene a tecniche di Machine Learning supervisionato.

2.2 Preprocessing del Dataset

Prima dell'addestramento, è stato necessario effettuare una serie di controlli e operazioni di pulizia, quali:

- Rimozione dei valori nulli o anomali (outlier)
- Conversione dei dati da stringhe a tipi numerici, ove necessario
- Selezione delle features più rilevanti
- Verifica della distribuzione del target e sbilanciamento delle classi (pioggia vs. non pioggia)

Poiché l'obiettivo era integrare il modello in un sistema embedded a risorse limitate, è stato selezionato un sottoinsieme delle feature più rilevanti, cioè quelle disponibili anche via sensore fisico: **temperatura**, **umidità** e **pressione**. Il file CSV originale è stato escluso dal repository del progetto per motivi di peso e gestione tramite l'aggiunta a `.gitignore`. Il dataset preprocessato è stato salvato in un file denominato `cleaned_dataset.csv`.
In dettaglio, il file `preprocess_dataset.py` ha svolto le seguenti operazioni:

- Caricamento del dataset da CSV
- Selezione delle colonne: `Temperature`, `Humidity`, `Pressure`, e `Rain Tomorrow` come target
- Rimozione dei valori nulli e conversione in formato numerico
- Assicurazione che il target fosse binario (0 = no pioggia, 1 = pioggia)

Il dataset pulito è stato salvato poi nel file `cleaned_dataset.csv`, pronto per le fasi successive.

2.3 Analisi Esplorativa dei Dati (EDA)

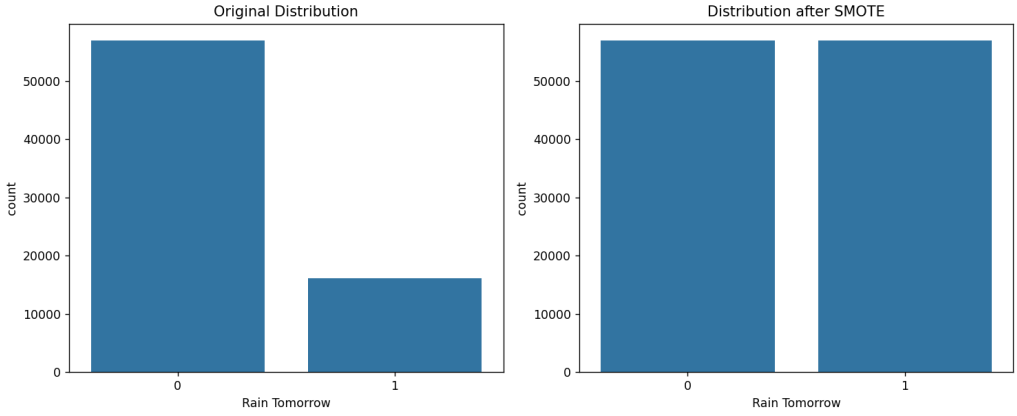
Il file `eda.py` ha permesso di analizzare statisticamente il dataset e visualizzarne le caratteristiche. In particolare:

- È stata evidenziata una forte sbilanciamento della classe `Rain Tomorrow`, con una predominanza di giorni senza pioggia.
- Sono state calcolate statistiche descrittive come media, deviazione standard, minimo e massimo per ciascuna feature.
- È stata generata una matrice di correlazione, utile a individuare relazioni tra le variabili.

Questa fase ha fornito le basi per migliorare la qualità del dataset e per decidere come affrontare lo sbilanciamento delle classi.

2.4 Bilanciamento con SMOTE

Lo squilibrio delle classi è stato dunque corretto con la tecnica *SMOTE (Synthetic Minority Over-sampling Technique)*, implementata nello script `rebalancing.py`. SMOTE genera nuovi esempi sintetici della classe minoritaria, aumentando l'equilibrio tra le classi. Dopo il riequilibrio, il numero di esempi della classe "pioggia" è stato portato allo stesso livello della classe "no pioggia". È stato quindi creato un nuovo dataset bilanciato, `cleaned_dataset_balanced.csv`, utilizzato per l'addestramento, e sono stati prodotti grafici di confronto tra la distribuzione delle classi prima e dopo SMOTE (Figure **??**)



2.5 Addestramento del Modello e Validazione

2.5.1 Configurazione e Scelta del Modello

Il file `train_model.py` è responsabile dell'addestramento. Il modello utilizzato è un albero decisionale (`DecisionTreeClassifier`) di Scikit-learn, addestrato sulle tre feature selezionate. È stato usato un valore elevato per la profondità massima dell'albero (`max_depth=60`) per consentire al modello di apprendere anche relazioni complesse.

In dettaglio, il dataset bilanciato è stato diviso in training e test set $80\% - 20\%$. Abbiamo dunque trainato il modello sul training set e valutato le performances sul test set, calcolando le seguenti metriche:

- Accuracy
- Precisione, Recall e F1-score per ciascuna classe
- Matrice di confusione

Questi risultati sono stati salvati nei file `model_info.txt` e `classification_report.txt` (Table 1) mentre la matrice di confusione (Figure 9 è stata esportata come immagine.

Table 1: Risultati della classificazione sul test set

	Precision	Recall	F1-score	Support
Classe 0	0.82	0.79	0.80	11400
Classe 1	0.80	0.83	0.81	11396
Accuracy		0.81		22796
Macro Avg	0.81	0.81	0.81	22796
Weighted Avg	0.81	0.81	0.81	22796

2.5.2 Conversione in Codice C e Integrazione su STM32 e Zephyr RTOS

Il modello è stato salvato nel formato `.joblib` per essere successivamente convertito in codice C.
Per rendere il modello utilizzabile nel microcontrollore STM32, è stato scritto lo script `export_tree_to_c.py`, che converte l'albero decisionale in una funzione C chiamata `predict_rain`. La struttura del codice generato è la seguente:

- `rain_model.c` contiene la funzione `predict_rain` che riceve in input una struttura `bme280_data_t` e restituisce 0 o 1
- `rain_model.h` definisce le intestazioni e include `bme280.h`

La funzione `predict_rain` contiene una serie di `if` e `else` annidati che replicano fedelmente la logica dell'albero decisionale.

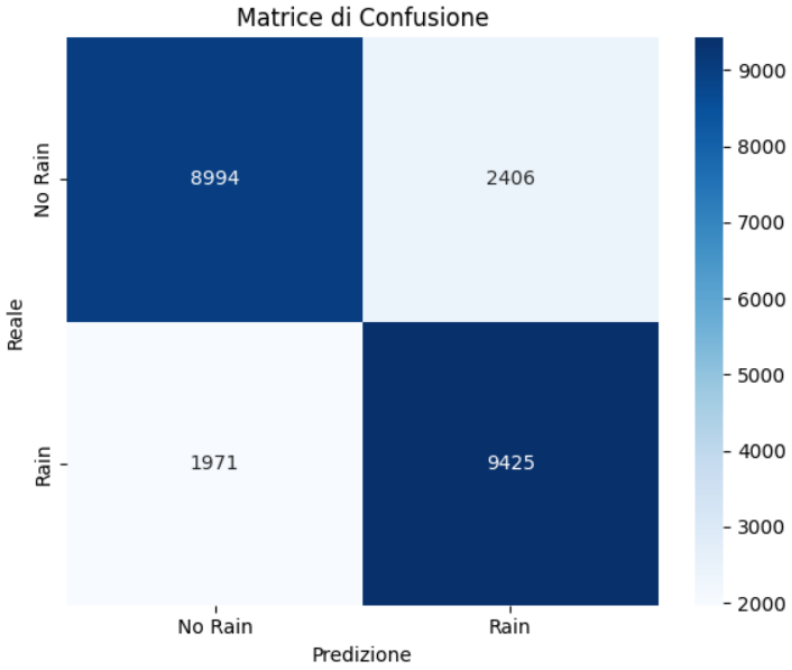


Figure 9: Matrice di Confusione

Il codice C generato è stato integrato nel firmware eseguito sul microcontrollore STM32 F446RE.

La funzione `predict_rain` viene chiamata dopo l'acquisizione dei dati da parte del sensore. Il risultato viene inoltrato tramite UART all'ESP32, che funge da nodo di comunicazione. Sul dispositivo ESP32, un'applicazione Zephyr riceve la stringa UART, la converte in un oggetto JSON e lo invia tramite HTTP POST a un server remoto per la memorizzazione o ulteriore analisi. L'intero flusso è gestito da un'applicazione dedicata che utilizza le API Zephyr per UART, parsing e invio di richieste HTTP.

Il nostro progetto ha dunque dimostrato la fattibilità dell'integrazione di un modello di Machine Learning in un microcontrollore a risorse limitate, grazie all'utilizzo di un modello interpretabile e convertibile come l'albero decisionale. I dati raccolti da un singolo sensore BME280 si sono rivelati sufficienti per ottenere buone prestazioni predittive. Il sistema è poi ovviamente espandibile con nuove fonti di dati o modelli più complessi, eventualmente tramite tecniche di quantizzazione o pruning.

3 Sviluppo del sistema

Il seguente capitolo si propone come obiettivo quello di mostrare l'architettura generale sia in ambito Hardware che software, illustrando come tutte le tecnologie viste nel capitolo 1.1 entrino effettivamente in gioco.

3.1 Schema di connessione

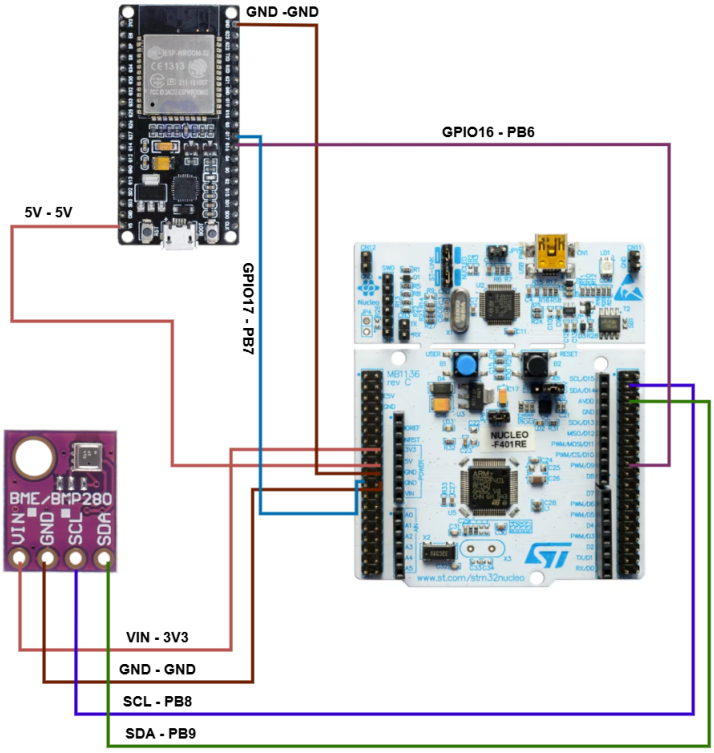


Figure 10: Schema di connessione generale

In Figura 10 è possibile osservare il collegamento tra i dispositivi hardware. I collegamenti al microcontrollore STM32-F446RE sono i seguenti:

- **GYBME280**
  - VIN →3V3[3]
  - GND →GND
  - SCL →PB8
  - SDA →PB9

[3] 3V3 è un'indicazione comune in elettronica che rappresenta 3.3 Volt.

- **ESP32-DevKitC-32E**
  - 5V →5V
  - GND →GND
  - GPIO16 →PB6
  - GPIO17 →PB7

Per quanto riguarda il sensore *GYBME280*, per una comunicazione I^2C più stabile ed efficiente, si consiglia generalmente l'uso di resistenze di pull-up, tipicamente da 4.7kΩ. Tuttavia, nel nostro caso non sono state aggiunte, poiché:

- Il collegamento tra sensore e microcontrollore è molto corto.
- Il modulo GYBME280 include già resistenze di pull-up da 10kΩ, come mostrato in Figura 11.

Lo schema elettrico del modulo GYBME280 con il chip BME280 è rappresentato in Figure 11.

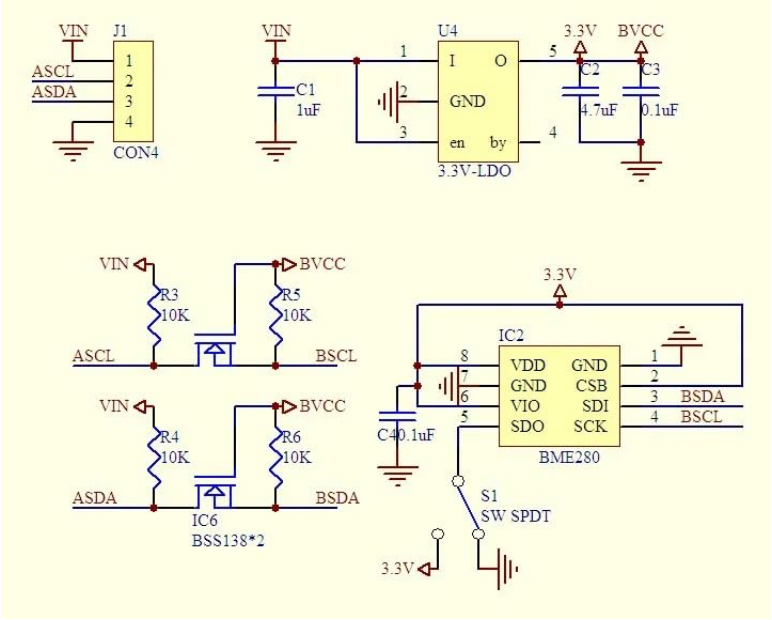
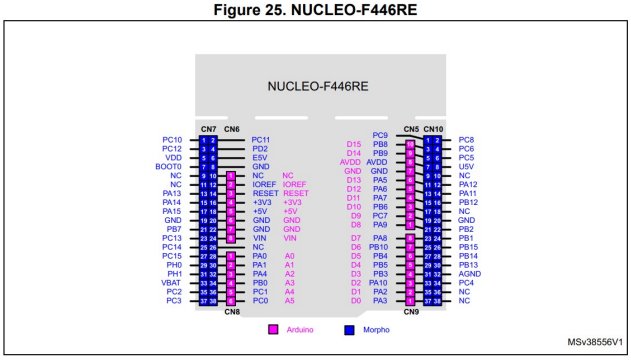


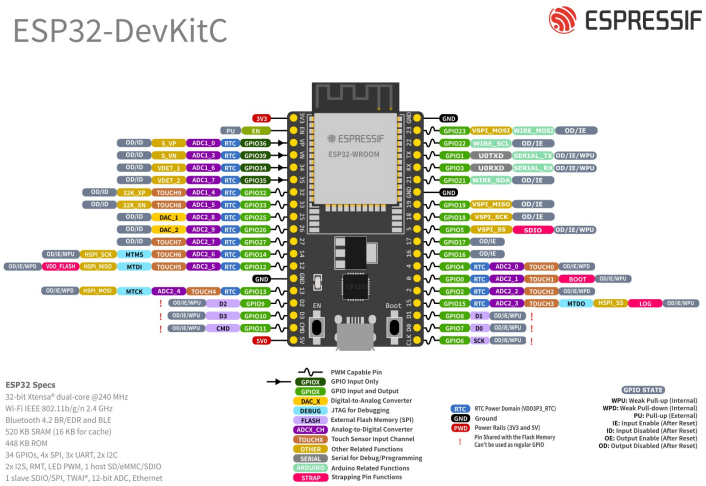
Figure 11: Schema elettrico del modulo GYBME280 con chip BME280.

Per individuare fisicamente i pin corretti per i collegamenti, si può invece far riferimento ai diagrammi di pinout delle due schede, mostrate in Figura 12, ovvero l'*ESP32-DevKitC-32E* e la *STM32-F446RE*.

Figure 25. NUCLEO-F446RE



(a) *STM32-F446RE*



(b) *ESP32-DevKitC-32E*

Figure 12: Configurazioni dei pin delle due schede utilizzate.

Per la comunicazione tra la scheda *ESP32* e il microcontrollore *STM32-F446RE* è stato adottato il protocollo UART[4]. Sebbene meno efficiente dell'I^2C in termini di velocità, UART consente trasmissioni stabili anche su distanze maggiori, il che risulta utile in un contesto dove il microcontrollore e il sensore possono essere posizionati in una zona remota, mentre la scheda ESP32 può essere collocata in una posizione favorevole per la trasmissione dei dati via Wi-Fi e/o Bluetooth.

[4]UART (Universal Asynchronous Receiver-Transmitter) è un protocollo di comunicazione seriale asincrono che permette lo scambio di dati tra dispositivi tramite due fili: uno per la trasmissione (TX) e uno per la ricezione (RX).

3.2 Progettazione

Il seguente paragrafo si propone come obiettivo quello di illustrare tutti i diagrammi di progettazione che sono stati utilizzati per implementare il software sul microcontrollore, ESP32 e Applicazione mobile.

3.2.1 Architettura STM32-F446RE

L'architettura software del microcontrollore *STM32-F446RE* è quella mostrata in Figura 13.

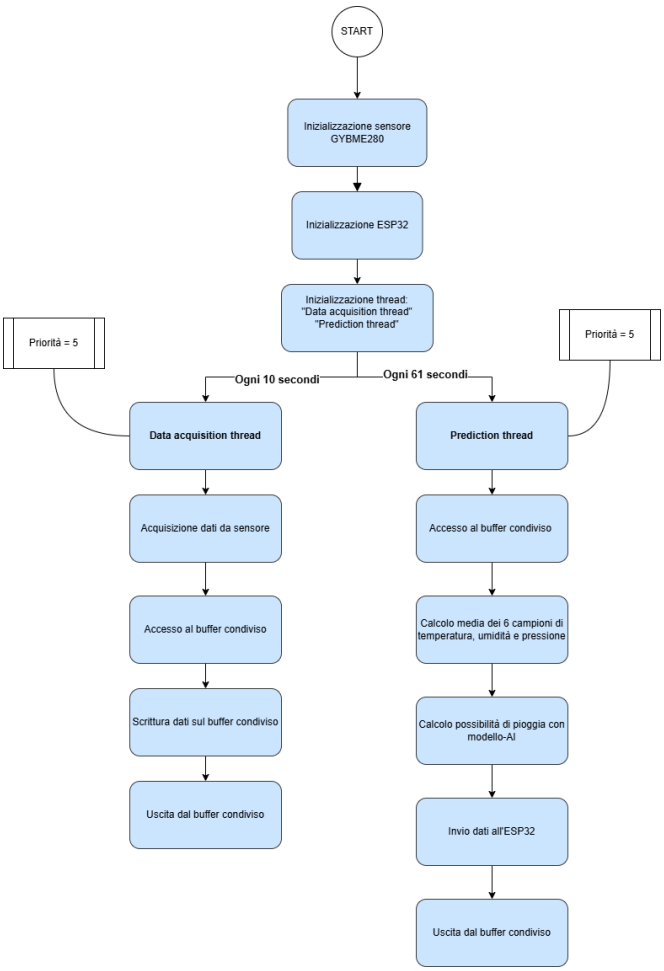


Figure 13: Architettura software *STM32-F446RE*

L'implementazione del sistema è basata sul sistema operativo real-time *Zephyr-OS*. La struttura prevede l'uso di due thread indipendenti ma coordinati, entrambi con uguale priorità[5], che operano in modo asincrono grazie alla temporizzazione e alla sincroniz-

[5]L'assegnazione della stessa priorità ai due thread consente una gestione simmetrica dell'accesso al

zazione tramite semafori.

Il primo thread, denominato *data acquisition thread*, è responsabile della lettura periodica dei dati dal sensore ambientale *GY-BME280*. Ogni 10 secondi, questo thread acquisisce temperatura, umidità e pressione, e salva tali misure in un buffer condiviso. Per garantire la coerenza dei dati e prevenire condizioni di race, l'accesso al buffer è protetto mediante le primitive `k_mutex_lock` e `k_mutex_unlock` offerte da Zephyr. Queste permettono di acquisire e rilasciare un mutex in modo sicuro, evitando accessi concorrenti da parte del secondo thread.

Il secondo thread, chiamato *prediction thread*, viene attivato ogni 61 secondi. Il suo compito è accedere in mutua esclusione al buffer condiviso, calcolare la media dei sei campioni più recenti di temperatura, umidità e pressione, ed effettuare una predizione tramite un modello di machine learning (AI) riguardo la possibilità di pioggia. Una volta ottenuto il risultato, il thread si occupa di inviarlo al modulo *ESP32* tramite comunicazione seriale UART. Al termine dell'elaborazione, il buffer viene svuotato, in modo da accogliere nuovi dati per il ciclo successivo.

Per sincronizzare correttamente l'inizio dell'esecuzione dei due thread, viene utilizzato un semaforo binario. Questo garantisce che entrambi i thread inizino la loro attività solo dopo l'avvenuta inizializzazione di tutti i dispositivi hardware (sensore, UART, ecc.) e del modello AI. Il sistema sfrutta inoltre le funzioni di temporizzazione fornite da Zephyr per scandire l'attivazione dei due thread in modo preciso e deterministico, evitando conflitti o accessi simultanei al buffer.

Grazie a questa architettura, il sistema garantisce un'elevata affidabilità nell'acquisizione e nell'elaborazione dei dati ambientali, evitando fenomeni di *buffer overflow* o *deadlock*, di cui verrà fornita analisi dettagliata nel capitolo **??**.

3.2.2 Architettura ESP32-DevKitC-32E

In Figura 14 è mostrato il diagramma di flusso del codice *Arduino* implementato e caricato sull'*ESP32*.

buffer condiviso, riducendo il rischio di starvation o preemption indesiderata durante sezioni critiche. Questo approccio garantisce una cooperazione più prevedibile tra thread concorrenti.

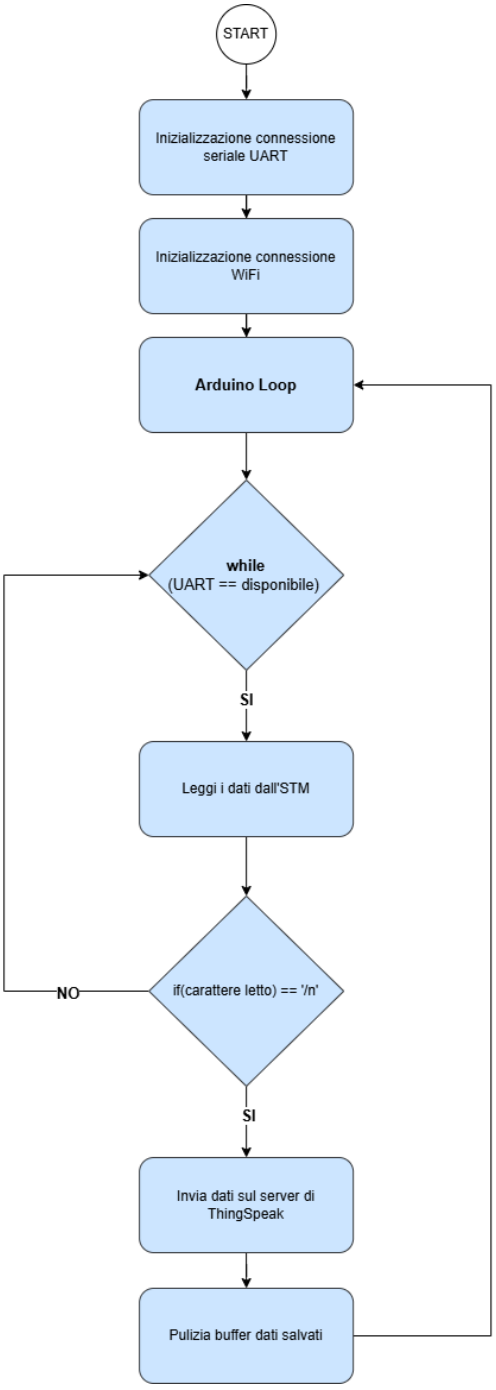


Figure 14: Architettura software *ESP32-DevKitC-32E*

Dopo una fase di *setup*, in cui vengono inizializzate la connessione seriale UART e quella Wi-Fi, si entra nel loop principale di *Arduino*. In questo ciclo continuo, l'ESP32

verifica la disponibilità di dati provenienti dal microcontrollore STM32 tramite UART.

Durante la progettazione e successivamente nella fase di implementazione, è stato deciso di utilizzare come carattere di terminazione del pacchetto di dati (contenente umidità, pressione, temperatura e possibilità di pioggia) il carattere `\n`[6].

Una volta ricevuto l'intero campione (terminato con il carattere `\n`), i dati vengono salvati temporaneamente e quindi inviati al server *ThingSpeak*. Infine, il buffer contenente i dati viene svuotato in attesa del successivo pacchetto.

3.2.3 Architettura applicazione mobile

La Figura 15 mostra il diagramma di flusso dell'applicazione mobile, con particolare enfasi sul meccanismo di aggiornamento e rendering automatico dei grafici. Si precisa inoltre che l'applicazione è stata dotata di filtri che permettono di selezionare e visualizzare solo alcune misurazioni (da/a), con diverse granularità temporali (ore, giorni, mesi).

[6]Questo carattere rappresenta un "a capo".

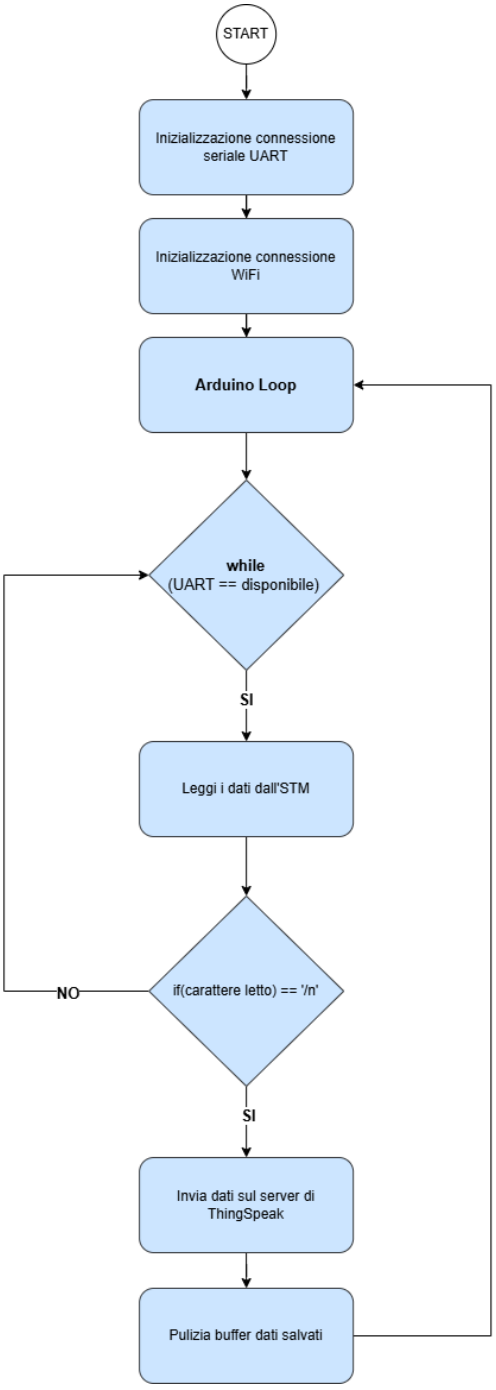


Figure 15: Architettura software *Applicazione mobile*

3.2.4 Architettura complessiva

La Figura 16 mostra, in forma schematica, l'intera struttura del sistema, suddivisa per livello tecnologico (basso e alto livello) e per tecnologie impiegate. Scorrendo la figura dall'alto verso il basso e da sinistra verso destra, è possibile osservare come i dati raccolti dal sensore vengano trasmessi al microcontrollore *STM32-F446RE*, successivamente inviati al modulo *ESP32*, e infine raggiungano il server *ThingSpeak* e l'applicazione mobile sviluppata in *Flutter*.

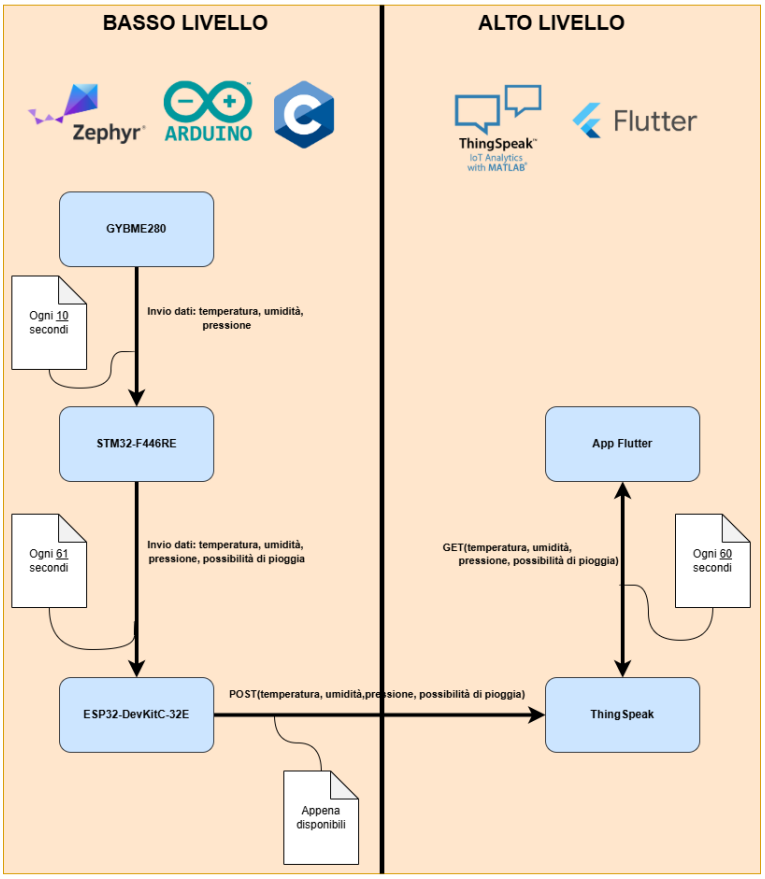


Figure 16: Architettura complessiva

Sempre osservando la Figura 16, nella parte inferiore è possibile notare che i dati vengono inviati a *ThingSpeak* non a intervalli regolari, ma **non appena disponibili**. In particolare, come descritto nella sezione 3.2.2, l'invio avviene ogni volta che viene rilevato il carattere '\n', il quale segnala la ricezione di una tupla completa composta da: temperatura, umidità, pressione e possibilità di pioggia.

Tenendo conto della frequenza di acquisizione dei dati e dei tempi di elaborazione

descritti in precedenza, si può stimare che l'invio dei dati da parte del modulo *ESP32* al server *ThingSpeak* avvenga approssimativamente ogni 62–64 secondi.

Si tenga presente che l'aggiornamento dei dati (*GET request*) da parte dell'applicazione mobile avviene automaticamente ogni 60 secondi, ma può essere effettuato anche manualmente. In particolare, l'utente ha la possibilità di premere un apposito pulsante di aggiornamento in qualsiasi momento, forzando così una nuova *GET request* e aggiornando immediatamente le schermate dell'app.

3.3 Implementazione

Il seguente paragrafo si propone come obiettivo quello di illustrare, l'implementazione di quanto progettato al punto 3.2 al fine di adempiere agli obiettivi descritti al punto 1.

3.3.1 STM32-F446RE

Di seguito saranno descritti tutti i codici sorgenti basati su *Zephyr*.

CMakeLists.txt Il file `CMakeLists.txt` rappresenta il punto di configurazione iniziale per il progetto basato su *Zephyr OS*. In esso viene specificata la versione minima richiesta di `CMake` (3.20.0) e viene definito il percorso del sistema operativo Zephyr tramite la variabile `ZEPHYR_BASE`. Successivamente, il progetto viene denominato `weather_prediction` e vengono elencati i file sorgenti che compongono l'applicazione: `main.c`, `bme280.c`, `rain_model.c` e `esp32_comm.c`, tutti situati nella cartella `src`. Infine, si specifica la directory dei file header da includere, in modo che il compilatore possa accedere correttamente alle definizioni durante la fase di build. Questo file è fondamentale per l'integrazione corretta dei moduli dell'applicazione all'interno dell'infrastruttura di build di Zephyr.

nucleo_f446.overlay Il file `nucleo_f446.overlay` è un file di overlay utilizzato per configurare specificamente l'hardware della scheda *STM32-F446RE*. Questo file permette di personalizzare la configurazione delle periferiche e la mappatura delle risorse hardware, sovrascrivendo le impostazioni di default.

In particolare, tramite questo file è possibile:

- Abilitare o disabilitare periferiche come UART, SPI o I2C;
- Configurare i pin associati alle periferiche, impostandone funzione, velocità e resistenze di pull-up o pull-down;
- Definire la mappatura delle interruzioni o dei canali DMA;
- Adattare la configurazione hardware alle caratteristiche specifiche della scheda Nucleo F446RE.

L'utilizzo di un file `overlay` in ZephyrOS è fondamentale per garantire un corretto funzionamento del kernel e dei driver in relazione all'hardware effettivo della scheda embedded.

bme280.c e bme280.h Il driver `bme280.c` e la sua intestazione `bme280.h` implementano la gestione del sensore ambientale BME280, utilizzando l'interfaccia I2C su ZephyrOS.

Descrizione generale:

- Il file `bme280.h` definisce la struttura dati `bme280_data_t` che contiene i valori compensati di temperatura (°C), pressione (hPa) e umidità relativa (%).
- Sono dichiarate le funzioni principali:
  - `bool bme280_init(const struct device *i2c_dev);` per l'inizializzazione del sensore.
  - `bool bme280_read_data(const struct device *i2c_dev, bme280_data_t *data);` per la lettura dei dati ambientali.

bme280.c - Dettagli di implementazione:

- Definisce i registri interni del BME280 e le costanti necessarie.
- Implementa la lettura e scrittura dei registri tramite funzioni `bme280_read_reg()` e `bme280_write_reg()` che utilizzano le API I2C di Zephyr.
- Gestisce la lettura dei parametri di calibrazione dal sensore, necessari per la compensazione delle misure grezze.
- Calcola i valori compensati di temperatura, pressione e umidità tramite funzioni dedicate che applicano gli algoritmi di calibrazione forniti dal datasheet del BME280.
- La funzione `bme280_init()` verifica la presenza del sensore su entrambi gli indirizzi I2C (0x76 e 0x77), configura il sensore con parametri standard di oversampling e filtro, e legge i dati di calibrazione.
- La funzione `bme280_read_data()` legge i dati grezzi da registri multipli e restituisce i valori compensati attraverso la struttura `bme280_data_t`.

Nota: Il driver è progettato per integrarsi facilmente in un'applicazione basata su ZephyrOS, utilizzando le API standard per l'I2C e la gestione dei dispositivi.

esp32_comm.c e esp32_comm.h Il file `esp32_comm.c` contiene l'implementazione della comunicazione UART con il modulo ESP32 all'interno di Zephyr OS. Viene definito un buffer di dimensione `UART_BUF_SIZE` per la trasmissione dei dati.

Le due funzioni principali sono:

- `esp32_comm_init`: verifica se il dispositivo UART passato come parametro `uart_dev` è pronto e quindi inizializza la comunicazione. L'inizializzazione specifica della UART è generalmente gestita dal Device Tree di Zephyr. La funzione restituisce `true` se il dispositivo è pronto, `false` altrimenti.
- `esp32_send_data`: prende come parametri il device UART, i dati del sensore BME280 e la probabilità di pioggia. Formatta i dati in una stringa JSON nel buffer `tx_buf`, utilizzando la funzione `snprintf` con il formato JSON contenente temperatura, pressione, umidità e probabilità di pioggia. Se la formattazione ha successo, i caratteri vengono inviati uno a uno sulla UART con `uart_poll_out`. Ritorna `true` se l'invio va a buon fine, `false` in caso di errore di formattazione o buffer overflow.

Il file header, invece, definisce l'interfaccia della comunicazione con il modulo ESP32. Entrambi i file utilizzano le API di Zephyr per la gestione UART e la struttura dati `bme280_data_t` per rappresentare i dati del sensore. La comunicazione avviene tramite trasmissione di stringhe JSON formattate per semplicità di parsing lato ESP32.

File `main.c` Il file principale implementa la logica dell'applicazione con *Zephyr OS* per la raccolta dati da un sensore BME280, la predizione della probabilità di pioggia tramite modello ML e la comunicazione con il modulo *ESP32* via UART.

Di seguito vengono descritti gli elementi principali del programma `main.c`, quali costanti, variabili globali, thread e funzione principale.

Costanti principali:

- `STACK_SIZE` definisce la dimensione dello stack dei thread (2048 byte).
- `THREAD_PRIORITY` imposta la priorità dei thread (valore 5).
- `SAMPLING_INTERVAL_MS` indica l'intervallo di campionamento del sensore (10 secondi).
- `PREDICTION_INTERVAL_MS` indica l'intervallo tra predizioni (circa 1 minuto e 1 secondo).
- `MAX_SAMPLES` è la dimensione massima del buffer dati (7 campioni).

Variabili globali:

- `i2c_dev` e `uart_dev` sono i device pointer per I2C e UART, ottenuti tramite `DEVICE_DT_GET` da alias definiti nel Device Tree.
- `init_sem` è un semaforo binario per la sincronizzazione dell'inizializzazione.
- `sensor_buffer` è una struttura che contiene array per temperatura, pressione e umidità, più un contatore `count` per i campioni acquisiti.
- `buffer_mutex` è un mutex per proteggere l'accesso concorrente a `sensor_buffer`.
- Stack e dati dei thread sono allocati con `K_THREAD_STACK_DEFINE` e strutture `k_thread`.

Thread di acquisizione dati (`data_acquisition_thread`):

- Attende il semaforo di inizializzazione con `k_sem_take`.
- In un ciclo infinito, legge i dati dal sensore BME280 tramite `bme280_read_data`.
- Se la lettura ha successo, acquisisce il mutex, salva i valori (temperatura, pressione, umidità) nel buffer finché non raggiunge `MAX_SAMPLES`, quindi rilascia il mutex.
- Attende per `SAMPLING_INTERVAL_MS` prima di leggere di nuovo.

Thread di predizione (`prediction_thread`):

- Attende il semaforo di inizializzazione.
- In un ciclo continuo, acquisisce il mutex e calcola la media dei valori nel buffer.
- Resetta il contatore del buffer a zero per nuovi dati.
- Utilizza la funzione `predict_rain` per ottenere la probabilità di pioggia basata sui dati medi.
- Invia i dati medi e la probabilità tramite la funzione `esp32_send_data` su UART.
- Rilascia il mutex e attende `PREDICTION_INTERVAL_MS` prima della successiva predizione.

Funzione `main`:

- Verifica che i dispositivi I2C e UART siano pronti usando `device_is_ready`, loggando un errore e terminando se non lo sono.
- Inizializza il sensore BME280 tramite `bme280_init`.
- Inizializza il modello ML chiamando `ml_model_init`.

- Inizializza la comunicazione con ESP32 tramite `esp32_comm_init`.
- Rilascia due volte il semaforo `init_sem` per sbloccare i thread.
- Crea e avvia i thread di acquisizione dati e predizione con `k_thread_create` e assegna loro un nome descrittivo con `k_thread_name_set`.

3.3.2 ESP32_DevKitC32e

Di seguito viene descritto il codice *Arduino* necessario per far funzionare l'*ESP32*.

thingspeak.ino Il file `thingspeak.ino` contiene il firmware sviluppato per l'ESP32 che funge da stazione meteorologica connessa. Il suo scopo principale è ricevere dati ambientali da un microcontrollore STM32 tramite la porta UART, interpretarli e inviarli in cloud su ThingSpeak, una piattaforma per l'IoT che permette di raccogliere e visualizzare dati in tempo reale.

All'avvio, nella funzione `setup()`, il programma inizializza la comunicazione seriale sia per il debug (porta seriale principale) sia per la comunicazione con il STM32 (utilizzando una UART secondaria, `Serial2`, sui pin 16 e 17). Subito dopo tenta la connessione alla rete WiFi usando le credenziali definite nel file `secrets.h`. Durante questa fase, il codice mostra all'utente tramite la seriale di debug lo stato della connessione fino al successo, indicando anche l'indirizzo IP assegnato al dispositivo.

La parte principale del programma è nel ciclo `loop()`, dove l'ESP32 monitora continuamente la porta UART secondaria alla ricerca di nuovi dati provenienti dal STM32. Questi dati arrivano come stringhe JSON separate da un carattere di newline. Il codice accumula i caratteri ricevuti fino a riconoscere questo terminatore, dopodiché procede a interpretare la stringa come JSON grazie alla libreria `ArduinoJson`. Se la deserializzazione va a buon fine, estrae i valori di temperatura, pressione, umidità e probabilità di pioggia.

Questi valori vengono poi inviati a *ThingSpeak* tramite una richiesta `HTTP GET`. La funzione dedicata `sendToThingSpeak` costruisce dinamicamente l'URL contenente la chiave API e i valori da aggiornare, quindi invia la richiesta al server di ThingSpeak. Nel caso in cui la connessione WiFi venga persa, il codice tenta di ristabilirla prima di inviare i dati. Se la richiesta ha successo, il dato viene aggiornato correttamente; altrimenti, viene segnalato un errore tramite la seriale di debug.

Il firmware è progettato per gestire in modo robusto sia eventuali errori di parsing JSON sia problemi di rete, fornendo informazioni diagnostiche utili durante lo sviluppo e il funzionamento. In questo modo, l'ESP32 agisce come un ponte affidabile tra il microcontrollore STM32, che si occupa della raccolta dati dai sensori, e la piattaforma cloud, consentendo di monitorare da remoto le condizioni meteorologiche in tempo reale.

3.4 Applicazione mobile

Di seguito sono descritti i file necessari per l'implementazione dell'app Flutter, la quale, come ribadito nella fase di progettazione 3.2.3, ha lo scopo di rendere fruibili, tramite un qualsiasi smartphone basato su *Android* o *iOS*, i grafici visionabili dalla piattaforma online di *ThingSpeak*.

pubspec.yaml File di configurazione del progetto Flutter. Specifica nome, versione, ambiente SDK e dipendenze utilizzate, tra cui `http` per le richieste API, `fl_chart` per i grafici, `provider` per lo stato, e `shared_preferences` per il salvataggio locale.

chart_data.dart Contiene le classi `ChartData` e `ThingSpeakFeed`. La prima rappresenta i dati da visualizzare nei grafici, convertiti dal formato JSON; la seconda modella la risposta JSON di ThingSpeak, separando canale e misurazioni.

home_screen.dart Costituisce la schermata principale dell'app. Visualizza i grafici di temperatura, pressione e umidità, lo stato del meteo (pioggia o bel tempo) e fornisce un'interfaccia per applicare filtri temporali e di granularità sui dati. Si appoggia al servizio `ThingSpeakService` tramite `Provider` per accedere e aggiornare i dati.

thingspeak_service.dart Questo file gestisce il recupero e la gestione dei dati da un canale ThingSpeak. Utilizza l'API REST per scaricare i feed dei campi `field1`(temperatura), `field2`(pressione), `field3`(umidità) e `field4`(possibilità di pioggia) da un canale specifico (`channelId`) utilizzando una chiave API. I dati vengono convertiti in oggetti `ChartData` e memorizzati in liste per l'uso nei grafici. Il servizio supporta filtri di data (`startDate`, `endDate`) e granularità (`hours`, `days`, `months`), ed è dotato di refresh automatico ogni minuto tramite un `Timer`. La funzione `fetchData()` si occupa di costruire l'URL con i parametri, effettuare la richiesta HTTP, parsare la risposta JSON, gestire errori HTTP (inclusi i codici 404 e 406), e notificare eventuali aggiornamenti. Sono presenti funzioni ausiliarie per validare l'intervallo temporale, impostare i filtri e resettarli.

indicator_widget.dart Questo widget *Flutter stateless* rappresenta un indicatore visuale circolare per mostrare lo stato attivo o inattivo di un'entità. Accetta tre parametri: `isActive` (booleano che determina lo stato), `color` (colore dell'indicatore), e `label` (etichetta descrittiva). Quando attivo, il cerchio ha un colore pieno con un'ombra; quando inattivo, appare con trasparenza e senza ombra.

status_widget.dart Questo widget *Flutter stateful* mostra lo stato della connessione al servizio ThingSpeak. Aggiorna in tempo reale l'orario corrente e visualizza un indicatore colorato: verde per connessione attiva, rosso per errore, e arancione per aggiornamento in corso. Il widget mostra l'ora dell'ultimo aggiornamento e, se presente,

un messaggio di errore. L'aggiornamento dell'orologio avviene ogni secondo tramite un `Timer`, mentre il servizio ThingSpeak si aggiorna automaticamente ogni minuto.

main.dart Il file `main.dart` avvia l'app Flutter utilizzando `runApp` con il widget radice `MyApp`. Quest'ultimo configura un `MultiProvider` per la gestione dello stato tramite `Provider`, in particolare istanziando il servizio `ThingSpeakService` come `ChangeNotifier`. Viene definito il tema dell'app con `Material3` e impostata la schermata iniziale su `HomeScreen`.

4 Verifiche e test

Viene descritta qui la fase di verifica formale effettuata sul modello Promela del sistema embedded sviluppato, tramite l'uso dello strumento *SPIN Model Checker*. Lo scopo della verifica è stato quello di analizzare e validare le proprietà di correttezza del sistema, quali la mutua esclusione nell'accesso alle risorse condivise, l'assenza di deadlock e il rispetto delle specifiche temporali definite.

4.1 Ambiente di verifica

L'ambiente di verifica è stato realizzato utilizzando un container Docker appositamente configurato per contenere lo strumento SPIN nella versione 6.5.2. Il `Dockerfile` utilizzato per la creazione dell'immagine è riportato nel file `TestMain/Dockerfile` e include i seguenti passaggi fondamentali:

- Installazione delle dipendenze necessarie quali `gcc`, `make`, `bison`, `flex`, e `graphviz`;
- Download e compilazione di SPIN a partire dal codice sorgente ufficiale;
- Configurazione della directory di lavoro per il montaggio dei file di modello Promela.

L'uso di Docker consente di eseguire SPIN in un ambiente isolato e replicabile, facilitando la riproducibilità dei test su diversi sistemi.

4.2 Modello Promela

Il modello Promela `test_main.pml` rappresenta una simulazione semplificata del sistema embedded basato su Zephyr RTOS con sensore BME280 e modulo di previsione meteo. I principali elementi del modello sono:

- Due thread principali: *data_acquisition_thread* e *prediction_thread*, che condividono un buffer protetto da un mutex;
- Canali di comunicazione per I2C e UART che simulano le interfacce hardware;
- Variabili di stato e flag per la gestione di errori, sincronizzazione e inizializzazione;
- Proprietà di correttezza espresse in Linear Temporal Logic (LTL), quali:
  - **Nessun overflow del buffer** (il buffer non supera mai la capacità massima);
  - **Assenza di deadlock** tra i thread di acquisizione dati e previsione;
  - **Mutua esclusione** garantita nell'accesso al buffer condiviso.

Il modello è stato sviluppato in modo da rispecchiare fedelmente il comportamento concorrente del sistema reale, inclusa la gestione delle risorse condivise e le condizioni di sincronizzazione.

4.3 Procedura di verifica

Per effettuare la verifica, sono stati utilizzati i seguenti comandi:

1. **Build del container Docker:**

   ```
   docker build -t spin .
   ```

   Questo comando crea l'immagine Docker con SPIN installato e pronto all'uso.

2. **Esecuzione del container con montaggio della directory di lavoro:**

   ```
   docker run -it --rm -v "%cd%:/work" spin
   ```

   In questo modo si accede a una shell all'interno del container con i file del progetto disponibili in `/work`.

3. **Verifica proprietà Deadlock:**

   ```
   spin -run -ltl thread_deadlock_free test_main.pml
   ```

   Questa proprietà verifica che il sistema non vada mai in uno stato di deadlock tra i due thread principali (acquisizione dati e previsione). In particolare, si assicura che se uno dei thread è in attesa di acquisire il mutex (bloccato sull'accesso al buffer condiviso), allora prima o poi riuscirà a ottenerlo. Questo evita situazioni in cui entrambi i thread restano bloccati indefinitamente aspettando risorse occupate dall'altro.

4. **Verifica proprietà no-buffer-overflow:**

   ```
   spin -run -ltl no_buffer_overflow test_main.pml
   ```

   Questa proprietà controlla che durante tutta l'esecuzione il conteggio dei campioni nel buffer condiviso non superi mai la capacità massima definita (`MAX_SAMPLES`). Garantisce quindi che il sistema non tenti di scrivere più dati di quanti ne possa memorizzare, evitando overflow e conseguenti corruzioni o perdite di dati.

5. **Verifica proprietà di mutua esclusione:**

   ```
   spin -run -ltl mutual_exclusion test_main.pml
   ```

Questa proprietà assicura che i due thread (acquisizione dati e previsione) non entrino mai simultaneamente nella sezione critica protetta dal mutex. In altre parole, garantisce la mutua esclusione nell'accesso al buffer condiviso, impedendo condizioni di gara e possibili inconsistenze dovute ad accessi concorrenti non sincronizzati.

4.4 Risultati

I test effettuati mediante model checking con SPIN hanno confermato le proprietà critiche del modello:

- **Assenza di Deadlock**: Nessuna situazione di stallo tra i thread è stata rilevata.
- **Mutua Esclusione**: Corretto accesso al buffer condiviso, senza conflitti critici.
- **Gestione del Buffer**: Prevenzione completa di overflow, mantenendo il buffer entro i limiti definiti.

Analisi Statistica dell'Esplorazione degli Stati

Durante la verifica formale, SPIN ha esplorato:

- *Verifica Thread Deadlock Free*:
  - **Stati totali**: 901
  - **Stati visitati**: 1.209
  - **Transizioni**: 2.373
- *Verifica Mutual Exclusion*:
  - **Stati totali**: 593
  - **Stati visitati**: 359
  - **Transizioni**: 952
- *Verifica No Buffer Overflow*:
  - **Stati totali**: 593
  - **Stati visitati**: 359
  - **Transizioni**: 952

Tali risultati confermano la *correttezza* e la *robustezza* della progettazione concorrente del sistema embedded[1][2][3].

La verifica formale con SPIN ha fornito un supporto metodologico fondamentale per l'analisi delle proprietà critiche del sistema. L'utilizzo di un container Docker ha

semplificato l'integrazione del processo di verifica nel ciclo di sviluppo, garantendo un ambiente di test stabile e riproducibile. Il modello Promela ha consentito di esplorare efficacemente gli scenari di concorrenza e sincronizzazione, rilevando potenziali criticità e validando le assunzioni progettuali.

[1]Gli *stati totali* rappresentano il numero di stati unici generati e memorizzati durante l'esplorazione del modello.

[2]Gli *stati visitati* indicano il numero complessivo di stati esaminati, compresi quelli ripetuti durante l'analisi.

[3]Le *transizioni* rappresentano i passaggi tra gli stati che descrivono l'evoluzione del sistema modello.

5 Preparazione e Avvio del Sistema

Descriviamo ora in dettaglio la procedura necessaria per avviare il sistema di monitoraggio ambientale. Come già spiegato, il sistema è composto da due microcontrollori (STM32 e ESP32), un sensore ambientale (BME280) e un servizio cloud (ThingSpeak) per la raccolta e visualizzazione dei dati. I microcontrollori comunicano tra loro tramite UART, mentre l'ESP32 invia i dati al cloud via WiFi.

5.1 Flash del firmware sulla ESP32

Il primo passo consiste nel caricare il firmware sull'ESP32. Questo programma è responsabile della ricezione dei dati meteorologici dalla STM32 tramite UART, della loro decodifica da formato JSON e dell'invio dei valori letti a ThingSpeak mediante richieste HTTP GET.

Per caricare il firmware:

1. Collegare la scheda ESP32 al computer tramite cavo USB.
2. Aprire l'ambiente di sviluppo `Arduino IDE` (o equivalente).
3. Importare lo sketch `thinkspeak.ino`.
4. Inserire le credenziali della rete WiFi e la chiave API di ThingSpeak in un file separato `secrets.h`.
5. Selezionare la scheda ESP32 corretta e la relativa porta COM.
6. Compilare e caricare il codice sulla scheda.

5.2 Flash del firmware sulla STM32

Il firmware per la scheda STM32 è sviluppato utilizzando Zephyr RTOS e si occupa di acquisire i dati dal sensore BME280 e trasmetterli in formato JSON alla ESP32 tramite UART. E' possibile trovare la documentazione relativa all'installazione del sistema operativo Zephyr RTOS al link[7] (Getting Started Guide). Una volta installato il sistema operativo, per compilare e caricare il firmware:

1. Aprire Visual Studio Code nella cartella del progetto Zephyr.
2. Aprire il terminale integrato e digitare i seguenti comandi:

[7] `https://docs.zephyrproject.org/latest/develop/getting_started/index.html`

```
C:\zephyrproject\venv\Scripts\activate.bat
zephyr-env.cmd
west build -b nucleo_f446re
west flash
```

Il percorso in cui si trova il file `activate.bat` potrebbe variare da un dispositivo all'altro. Inoltre, prima di eseguire il comando `zephyr-env.cmd`, è necessario impostare le variabili `ZEPHYR_BASE` e `ZEPHYR_SDK_INSTALL_DIR` rispettivamente con il percorso base di Zephyr e con il percorso della SDK.

3. Verificare che la scheda STM32 sia collegata correttamente via USB e che il *sensore* e l'*ESP32* siano alimentati.

5.3 Installazione ed avvio dell'applicazione mobile

Per avviare il progetto, è necessario aver installato correttamente *Flutter*. La guida ufficiale è disponibile al seguente link[8]. Per generare il file *APK*[9] e far girare l'applicazione su dispositivi *Android*, occorre eseguire i seguenti comandi all'interno della cartella di progetto `thingspeak_app`:

- **Aggiornare le dipendenze del progetto**:

  ```
  flutter pub get
  ```

- **Costruire il file APK** in modalità release:

  ```
  flutter build apk --release
  ```

- **Installare l'APK su un dispositivo Android collegato via USB**:

  ```
  flutter install
  ```

  Prima di lanciare il comando, è necessario consentire l'installazione da *Sorgenti sconosciute* e attivare il *Debug USB* dal menù *Sviluppatore*[10].

[8] `https://docs.flutter.dev/install`
[9] Un file *Android Package* (.apk) è il formato usato per distribuire e installare applicazioni Android.
[10] Il modo per accedere al menù sviluppatore varia da dispositivo a dispositivo.

5.4 Avvio e monitoraggio del sistema

Una volta completata la configurazione, il sistema è pronto all'uso. Le fasi operative sono le seguenti:

- L'ESP32 si connette automaticamente alla rete WiFi specificata.
- La STM32 acquisisce periodicamente i dati ambientali dal sensore BME280.
- I dati vengono formattati in JSON e trasmessi via UART all'ESP32.
- L'ESP32 riceve i dati, li decodifica e li invia a ThingSpeak.

5.5 Monitoraggio dei dati

Di seguito vengono illustrate le due principali modalità di visualizzazione dei dati raccolti dall'hardware.

5.5.1 Visualizzazione mediante interfaccia Web

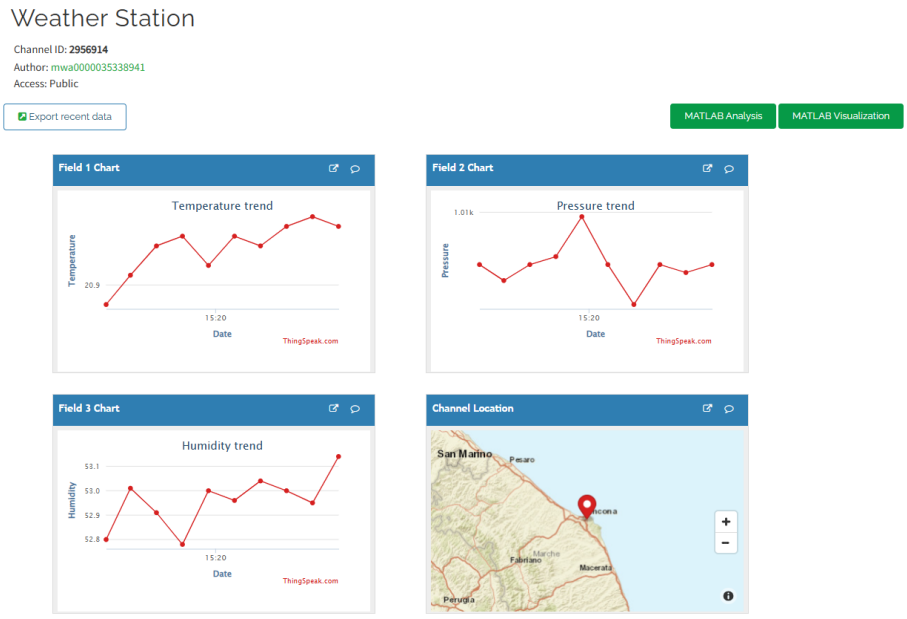
Per monitorare il funzionamento del sistema tramite l'interfaccia web, è sufficiente cliccare sul seguente link[11]. Una volta cliccato, verrà visualizzata la schermata mostrata in Figura 17.

5.5.2 Visualizzazione mediante applicazione mobile

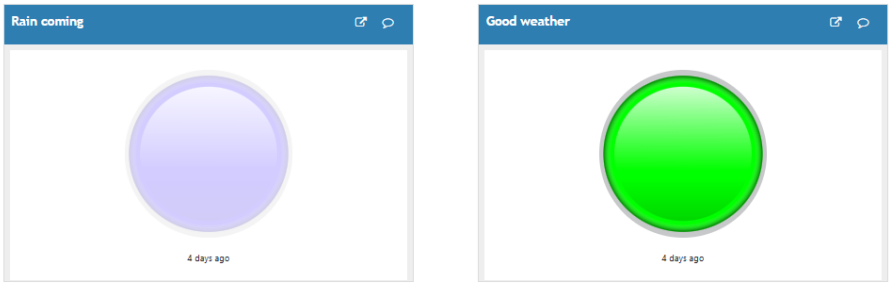
Una volta installata e avviata l'applicazione, come descritto al punto 5.3, l'*Home screen* sarà quella mostrata in Figura 18.

Inoltre, come descritto al punto 3.2.3, l'app consente di filtrare i dati per giorno e ora, e di modificare la granularità della visualizzazione da ore a mesi, passando per i giorni. La Figura 19 illustra le possibili interazioni relative a quanto appena descritto.

[11] https://thingspeak.mathworks.com/channels/2956914

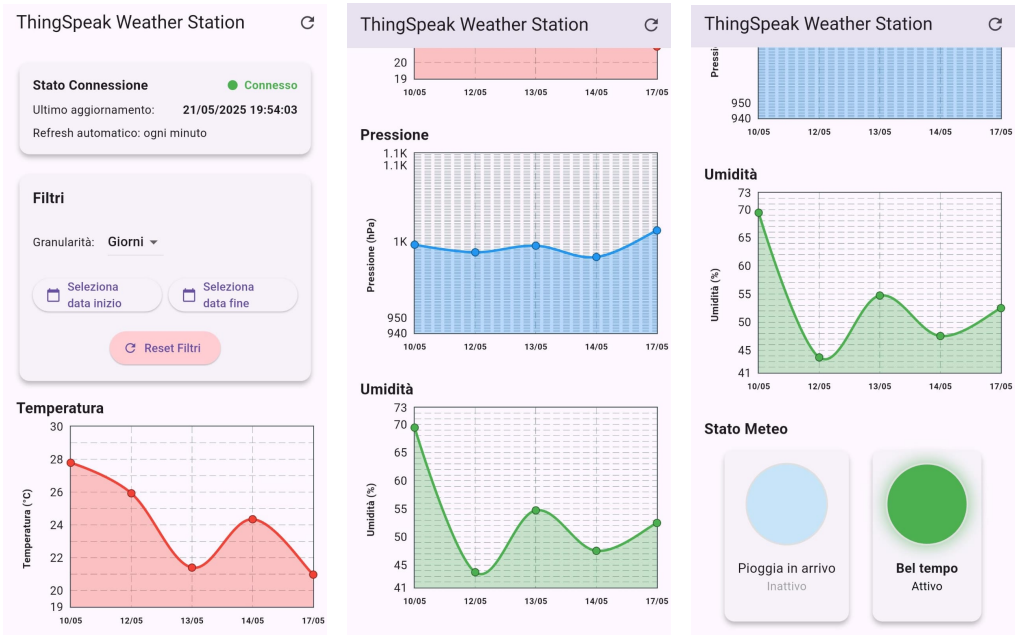


(a) Interfaccia parte 1



(b) Interfaccia parte 2

Figure 17: Interfaccia di *ThingSpeak*

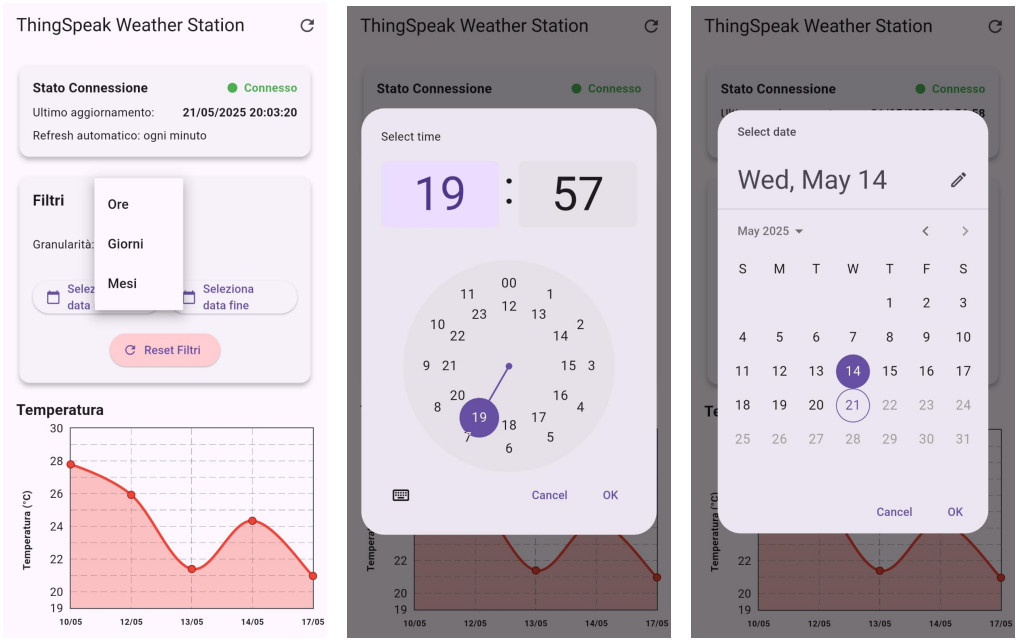


(a) Stato connessione, filtri e grafico della temperatura

(b) Grafico pressione e umidità

(c) Grafico umidità e widget per la previsione del tempo

Figure 18: Home screen dell'applicazione "*ThingSpeak Weather station*"



(a) Selezione della risoluzione di visualizzazione

(b) Selezione del giorno per visualizzare i dati

(c) Selezione dell'ora per visualizzare i dati

Figure 19: Opzioni per filtrare e visualizzare i dati

6 Conclusioni e sviluppi futuri

Il progetto realizzato ha portato alla creazione di un sistema in grado di monitorare l'ambiente e prevedere in modo semplice e automatico la possibilità di pioggia, grazie all'integrazione di sensori fisici e modelli di apprendimento automatico. La combinazione tra l'utilizzo di microcontrollori a basso consumo, il sistema operativo Zephyr RTOS e un'app mobile per la visualizzazione dei dati ha permesso di costruire una soluzione completa, flessibile e replicabile.

Nonostante i buoni risultati ottenuti, il lavoro può essere ulteriormente migliorato in diverse direzioni. Un primo sviluppo futuro riguarda l'applicazione Flutter, che potrebbe essere resa più interattiva e completa, ad esempio migliorando la rappresentazione dei grafici, inserendo notifiche personalizzate e una gestione più avanzata dell'utente. Un'altra direzione importante è quella dell'efficientamento energetico: ottimizzare i consumi dei dispositivi, sfruttando modalità di risparmio energetico e strategie intelligenti per la raccolta e trasmissione dei dati, permetterebbe al sistema di funzionare più a lungo anche in contesti isolati o alimentati a batteria. Infine, una possibile evoluzione tecnica consiste nel portare il sistema operativo Zephyr anche sulla scheda ESP32, sostituendo l'attuale gestione con Arduino. In questo modo si otterrebbe una maggiore coerenza e robustezza a livello software, semplificando la manutenzione e migliorando le performance complessive del sistema.

Queste prospettive aprono la strada a nuove sperimentazioni e applicazioni, rendendo il progetto ancora più utile in contesti reali, quali ad esempio l'agricoltura di precisione, il monitoraggio urbano e i sistemi di allerta ambientale.

List of Figures

References

[1] Bosch. *BME280 Datasheet*. 2023. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf.

[2] Espressif Systems. *ESP32-WROOM-32E Datasheet*. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf.

[3] STMicroelectronics. *STM32F446RE Datasheet*. 2023. URL: https://www.st.com/resource/en/datasheet/stm32f446re.pdf.