ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE DI UN PURIFICATORE D'ARIA INTELLIGENTE E SVILUPPO DI UNA VERSIONE PROTOTIPALE DEL SOFTWARE DI CONTROLLO

Elaborato in
SISTEMI EMBEDDED E INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
LIVIA CARDACCIA

Prima Sessione di Laurea
Anno Accademico 2023 – 2024

A papà e all'importanza di respirare aria pulita

Indice

Capitolo 1

Introduzione

Respirare aria pulita è un diritto dell'uomo, che negli ultimi anni è stato riconosciuto dalle legislazioni di numerosi Stati di tutto il mondo. Un diritto che però, con il generale incremento dell'inquinamento, non è garantito ovunque in egual modo. Al contrario di quanto si possa pensare, il problema ci riguarda spaventosamente da vicino. I dati raccolti dall'inizio di quest'anno, 2024, segnalano la Pianura Padana come una delle zone più inquinate d'Europa, con particolare attenzione alla città di Milano, che a febbraio è addirittura risultata la terza città con la qualità dell'aria peggiore nel mondo.

Come riportato in un articolo pubblicato da *RaiNews*[1], i motivi per cui la Pianura Padana presenta un'aria così inquinata sono diversi, principalmente derivanti da questioni di conformazione geografica e demografica; infatti, la presenza di Alpi e Appennini chiude quest'area e impedisce una corretta ventilazione. Inoltre, la densità abitativa in Piemonte, Lombardia, Emilia-Romagna, Veneto e Friuli-Venezia Giulia è tra le più elevate in Europa, il che porta con sé alcune conseguenze, quali un elevato numero di veicoli circolanti e moltissime abitazioni che, soprattutto in inverno, emettono gas per il riscaldamento. Il problema dell'inquinamento dell'aria non si limita alle sostanze tossiche che respiriamo quando siamo all'aperto ma persiste nell'aria *indoor*, ovvero quella presente nelle nostre case, uffici e scuole. Nello specifico, i fattori più inquinanti dell'aria domestica sono il fumo che si libera dalle sigarette, la combustione di stufe a gas, stufe a legna, caminetti, caldaie, forni e fornelli.

Sebbene possano essere prese alcune accortezze per garantire una qualità dell'aria accettabile, la misura di prevenzione più efficace è il monitoraggio di quest'ultima. Monitorare la qualità dell'aria significa misurare in modo continuo le concentrazioni degli inquinanti nell'ambiente, attraverso strumenti e tecnologie apposite. Il passo successivo al monitoraggio è la purificazione, ovvero l'eliminazione delle particelle dannose, possibile attraverso dispositivi quali i purificatori d'aria.

L'obiettivo di questa tesi è sviluppare un prototipo di un purificatore d'aria "intelligente", progettando un sistema software di controllo, che permetta di regolare il funzionamento del purificatore a distanza e dia la possibilità di visualizzare i dati relativi agli inquinanti presenti nell'ambiente domestico. Ciò è volto a sensibilizzare sull'importanza della qualità dell'aria che respiriamo, per poter agire in modo efficace a protezione della nostra salute.

La tesi è articolata in cinque capitoli: il capitolo corrente, ovvero un primo capitolo che introduce il problema; il secondo capitolo, volto all'analisi delle caratteristiche principali che il sistema da sviluppare dovrà avere; il terzo capitolo, riservato alla progettazione e al design del software di controllo; il quarto capitolo, che illustra i passaggi seguiti per l'implementazione ed infine, il quinto ed ultimo capitolo che riporta alcune riflessioni finali, elaborate a progetto concluso. Da qui in avanti ci si riferirà al progetto con il nome di *Smart-Air-Purifier*.

Capitolo 2

Smart-Air-Purifier: Analisi dei Requisiti

La prima fase di sviluppo del progetto si articola in un'attenta analisi volta ad individuare le principali caratteristiche che il sistema in oggetto dovrà presentare. Un primo paragrafo introduttivo ha lo scopo di analizzare il modello del dominio applicativo, ovvero il contesto specifico all'interno del quale l'applicazione dovrà operare, inclusi i processi ed i problemi che essa deve affrontare e risolvere. Si vogliono, poi, elencare nello specifico sia i requisiti funzionali, che andranno a delineare l'effettivo comportamento dell'applicativo, sia i requisiti non funzionali, che quindi non riguarderanno direttamente gli aspetti più comportamentali, bensì i criteri di qualità e le proprietà operative del sistema. A seguire, viene dato spazio alla presentazione dei casi d'uso, che rappresentano una formalizzazione dei requisiti individuati, mettendo in evidenza come l'utente deve poter interagire con il sistema. Infine, per gli obiettivi che si pone il progetto, è importante riservare uno spazio anche all'analisi dei tipi di dato raccolti dai sensori.

2.1 Analisi e Modello del Dominio

Smart-Air-Purifier è un progetto volto a sensibilizzare sull'importanza del monitoraggio e la purificazione dell'aria all'interno delle abitazioni. Il sistema è composto da una parte software che interagisce con una componente hardware. Il modulo software si pone l'obiettivo di aggiungere funzionalità al purificatore fisico, contribuendo a renderlo un purificatore d'aria *intelligente*[1].

[1]Un dispositivo intelligente è un dispositivo elettronico, generalmente connesso ad altri dispositivi o reti tramite diversi protocolli wireless, che può funzionare in modo interattivo e autonomo[...]. I dispositivi intelligenti possono ricevere input vocali e tattili, elaborare le informazioni e comunicare con altri dispositivi e sistemi connessi.[2]

Smart-Air-Purifier integra dei sensori[2] e degli attuatori[3]: dai primi raccoglie misurazioni su parametri significativi in base ai quali regola il funzionamento dei secondi, mettendo così in funzione il purificatore fisico.

I dati raccolti vengono gestiti e rielaborati dal sistema, per poi essere messi a disposizione dell'utente finale attraverso un'interfaccia.

L'utente è l'attore principale del sistema: egli vi interagisce controllando a distanza la modalità di funzionamento del purificatore e monitorando le variazioni della qualità dell'aria, nella stanza in cui è posizionato il dispositivo fisico. *Smart-Air-Purifier* garantisce una visualizzazione dei dati in tempo reale ed un funzionamento fluido ed intuitivo, rendendo la gestione della qualità dell'aria domestica semplice e accessibile.

2.2 Requisiti Funzionali

Come già accennato in precedenza, questa parte di analisi è volta ad individuare e presentare gli aspetti relativi al comportamento generale del sistema, descrivendo a parole ciò che il software dovrà fare e spiegando con chiarezza gli obiettivi che ci si è posti.

1. **Monitoraggio dei dati raccolti**

   Al centro dello sviluppo di questo progetto sta l'importanza di dare la possibilità all'utente di visualizzare costantemente i dati di monitoraggio raccolti dai sensori; si vuole, infatti, permettere di avere un'idea chiara e in tempo reale della qualità dell'aria della stanza in cui il purificatore è posizionato.

   Per garantire un monitoraggio efficace, sarà cruciale concentrarsi sulla logica di lettura dei dati dai sensori; in particolare dovranno essere implementate metodologie robuste e corrette per garantire la continua raccolta dei dati, assicurandosi che i suddetti siano accurati e privi di anomalie.

2. **Controllo del funzionamento a distanza**

   Invece che limitare l'utilizzo del purificatore ad un naturale ciclo di accensione/spegnimento gestito tramite hardware, si vuole incorporare una logica di funzionamento più estesa, che permetta all'utente finale di controllare il dispositivo a distanza.

[2]Un sensore è un dispositivo che permette di misurare un fenomeno fisico (ad esempio la temperatura) o di rilevare e quantificare una concentrazione chimica (ad esempio un gas) e trasforma tale misurazione in un segnale leggibile, analogico o digitale.

[3]Un attuatore è un dispositivo che produce un effetto sull'ambiente: riceve un segnale di controllo e lo trasforma in un'azione fisica, come movimento, forza o cambiamento di stato.

Ciò vuol dire andare oltre la semplice visualizzazione dei dati raccolti, dando maggiore controllo e possibilità di personalizzazione all'utilizzatore. Sarebbe opportuno individuare modalità aggiuntive che differiscano da "acceso" o "spento", in modo da permettere un'ulteriore regolazione della velocità di lavoro del purificatore.

Questo approccio consentirà una gestione più precisa e flessibile del dispositivo, migliorando sia l'efficienza che l'efficacia del processo di purificazione dell'aria.

3. **Individuazione e conseguente avviso di situazioni anomale**

   Dato che il sistema monitora un aspetto molto delicato, ovvero la qualità dell'aria in un ambiente domestico, è essenziale che, se la logica di lettura dei dati dai sensori rileva qualche misurazione sopra la soglia di normalità, come un livello troppo alto di un determinato composto nell'aria, questa variazione venga opportunamente segnalata all'utente.

   È importante che il sistema disponga di un meccanismo di allarme efficace, che possa avvisare immediatamente l'utente attraverso notifiche visive. Questo garantirà che l'utente possa prendere tempestivamente le misure necessarie per mitigare eventuali rischi legati alla qualità dell'aria, proteggendo così la salute e il benessere degli abitanti della casa.

4. **Creazione di una connessione stabile con il dispositivo**

   È stato già posto l'accento sulla rilevanza di un costante monitoraggio dei dati in un sistema come questo; per garantire ciò, alla base dovrà esserci una connessione stabile e robusta fra dispositivo hardware e ambiente software.

   Questa connessione potrà essere fornita attraverso varie tecnologie, purché sia veloce e affidabile, in modo da garantire un corretto funzionamento del sistema nella sua interezza.

   È cruciale che il sistema di gestione e la comunicazione tra i sensori sia efficiente e priva di interruzioni, per evitare perdita di dati e garantire un monitoraggio continuo e preciso.

   Implementare protocolli di comunicazione sicuri e resilienti contribuirà a mantenere l'integrità dei dati e a prevenire possibili malfunzionamenti, assicurando una risposta tempestiva alle condizioni rilevate.

2.3 Requisiti non Funzionali

Rispetto alla sezione precedente, in questa si vogliono evidenziare i requisiti non funzionali, vale a dire quelle caratteristiche che si discostano dall'aspetto prettamente comportamentale del sistema ma che sono comunque aspetti importanti da incorporare.

1. **Interfaccia utente intuitiva**

   La visualizzazione dei dati deve essere intuitiva e accessibile, in modo da facilitare l'interpretazione rapida delle informazioni critiche relative alla qualità dell'aria.

   Un design ben studiato contribuirà a migliorare l'esperienza dell'utente, mostrandogli in modo chiaro tutte le informazioni rilevanti e permettendogli di interagire con i dati in modo semplice ed efficace.

2. **Sicurezza del sistema**

   Trattandosi di un sistema che veicola dati sensibili, e più in generale per garantire un servizio ottimale agli utenti finali, è importante considerare gli aspetti critici legati alla sicurezza delle informazioni raccolte.

   Un aspetto da tenere in considerazione sarà quindi quello di adottare delle tecnologie adeguate affinché possa essere garantita la protezione dei dati e delle operazioni del sistema da attacchi o accessi non autorizzati.

3. **Raccolta dei dati per elaborazioni statistiche**

   I dati letti dai sensori, destinati ad essere visualizzati dall'utente finale, potrebbero essere sfruttati ulteriormente.

   Potrebbe risultare interessante effettuare delle elaborazioni statistiche, per capire come la concentrazione di un certo composto nell'aria varia durante il giorno e con il susseguirsi del tempo, per cercare di individuare eventuali pattern o variazioni patologiche che potrebbero rivelare una situazione anomala, relativa alle abitudini degli abitanti della casa.

2.4 Casi d'Uso

Per *caso d'uso* si intende la metodologia usata nei processi di ingegneria del software volta ad effettuare in maniera esaustiva e non ambigua la raccolta dei requisiti di un sistema, focalizzandosi sugli attori che vi interagiscono e valutandone le varie interazioni: vengono quindi descritti gli scenari elementari di utilizzo del software da parte degli attori che si interfacciano con esso.

Un attore, in un caso d'uso, è una qualsiasi entità che svolge un ruolo e interagisce con il sistema; egli può essere un umano, un altro sistema o un dispositivo hardware. Nel contesto dei casi d'uso, si possono identificare attori primari, ovvero le entità principali che avviano un'interazione con il sistema mosse da uno specifico obiettivo, e attori secondari, che aiutano gli attori primari a raggiungere il loro scopo.

Nel caso di *Smart-Air-Purifier*, l'attore principale è l'utente finale mentre il dispositivo fisico può essere considerato un attore secondario.

L'utente, con un determinato obiettivo o scopo, pone in essere un'azione o una richiesta che viene soddisfatta dal sistema con l'ausilio del purificatore d'aria fisico. Per una spiegazione più efficace ed approfondita, vengono riportati di seguito un diagramma dei casi d'uso, disegnato seguendo il formalismo UML (Unified Modeling Language) in figura 2.1, e alcuni esempi di possibili casi d'uso del sistema. In quest'ultimi, quello che viene chiamato *Main Success Scenario* è lo scenario principale che può però subire alcune variazioni, in caso di errore o comportamento diverso da quello standard, indicate nelle estensioni.

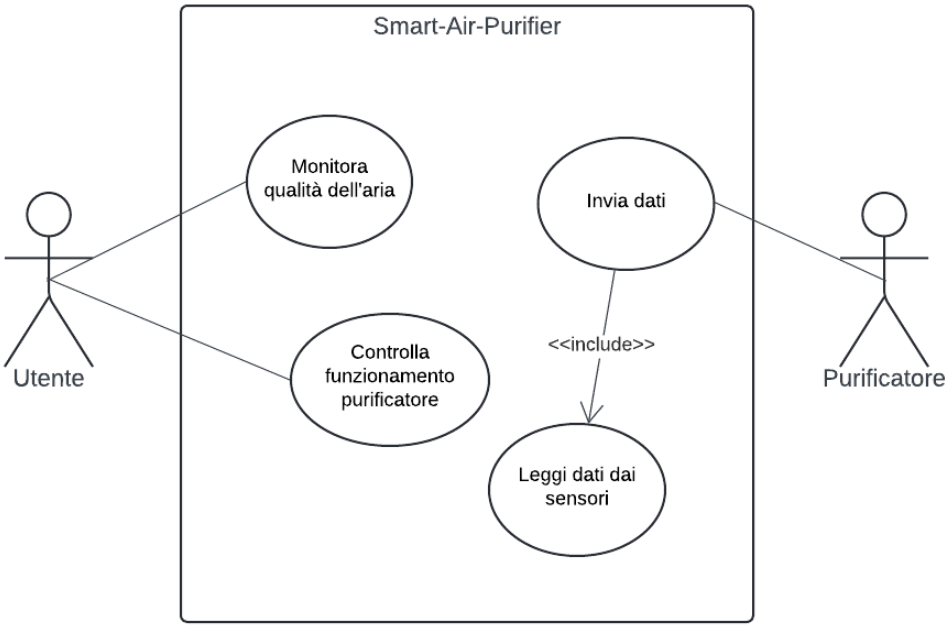

Figura 2.1: Diagramma dei casi d'uso

Monitora qualità dell'aria.

Main Success Scenario:

1. L'utente collega il purificatore alla corrente.
2. Il purificatore si connette alla rete Wi-Fi.
3. Il purificatore legge i dati dai sensori.
4. Il purificatore invia i dati letti al server.
5. L'utente si connette per conoscere la qualità dell'aria della stanza.
6. L'interfaccia richiede e mostra i dati relativi alle misurazioni dei sensori.
7. L'utente visualizza i dati di interesse e apprende la qualità dell'aria.

Estensioni:

2a. Il purificatore non riesce a connettersi alla rete.

.3: L'utente scollega il purificatore e risolve eventuali problemi con la rete, si torna poi allo step 1.

6a. L'interfaccia non riceve i dati e visualizza un messaggio di errore.

.7: L'utente ricarica la pagina.

Controlla funzionamento purificatore.

Main Success Scenario:

1. L'utente collega il purificatore alla corrente.
2. Il purificatore si connette alla rete Wi-Fi.
3. L'utente seleziona la modalità di funzionamento del purificatore attraverso l'interfaccia.
4. Il server riceve l'informazione e la inoltra al purificatore.
5. Il purificatore cambia la sua modalità di funzionamento in quella scelta dall'utente.

Estensioni:

4a. Il server non riceve correttamente l'informazione o non riesce ad inoltrarla al purificatore.

.5: L'interfaccia non mostra l'aggiornamento di modalità, perciò si torna allo step 3.

2.5 Stato dell'Arte dei Purificatori d'Aria Intelligenti

Per capire quali siano le sostanze inquinanti più rilevanti da monitorare in un ambiente interno e, di conseguenza, quali siano i sensori più adatti da integrare nel progetto, si vuole fare una panoramica sullo stato dell'arte dei purificatori d'aria intelligenti, evidenziando le loro principali funzionalità.

Rispetto ad un purificatore d'aria classico che è limitatamente progettato per rimuovere le particelle inquinanti dall'aria all'interno di uno spazio chiuso, un purificatore d'aria *intelligente* integra funzionalità aggiuntive, quali ad esempio: monitoraggio in tempo reale della qualità dell'aria, controllo da remoto, automazione, notifiche agli utenti e integrazione con assistenti vocali.

Le più recenti tecnologie sviluppate in ambito di monitoraggio e purificazione dell'aria mettono a disposizione filtri in grado di rimuovere numerosi agenti inquinanti: le polveri sottili, quindi il PM2.5 ed il PM10; allergeni, come polline e acari della polvere; fumo di sigaretta e altri tipi di fumo nell'aria.

Solo alcuni purificatori, però, sono dotati di tecnologie specifiche che possano efficacemente rilevare e rimuovere gas nocivi come il CO, monossido di carbonio, il CO2, anidride carbonica, ed il NO2, biossido di azoto.

Alcuni sistemi di purificazione dell'aria, per offrire una panoramica più completa, integrano anche sensori di umidità e temperatura, che sono ritenuti parametri rilevanti da monitorare: questo perché, ad esempio, un ambiente interno particolarmente umido risulta propenso alla formazione di muffe, che abbassano drasticamente la qualità dell'aria interna di una casa, visti i rischi per la salute che ne derivano.

2.6 Fattori Critici per il Controllo della Qualità dell'Aria

Alla luce delle informazioni raccolte in merito allo stato attuale delle tecnologie per la purificazione dell'aria, il sistema dovrebbe essere in grado di effettuare misurazioni relative a quattro tipi di dato: umidità, temperatura, anidride carbonica e monossido di carbonio.

I seguenti paragrafi sono volti ad illustrare e discutere in che maniera questi tipi di dato siano significativi per un corretto monitoraggio della qualità dell'aria domestica.

2.6.1 Umidità

Secondo il Ministero della Salute, uno degli aspetti più fastidiosi e dannosi per il comfort abitativo è quello legato all'umidità; la presenza di acqua nelle murature può provocare inconvenienti come la diminuzione del comfort termico, il degrado dei materiali a causa di reazioni chimiche distruttive e la comparsa di muffe. Inoltre, è stato dimostrato come all'esposizione all'umidità domestica si associno numerosi problemi respiratori [3].

Perciò, pur non essendo l'umidità un fattore di per sé inquinante, è un agente che può profondamente influire sulla variazione della qualità dell'aria che respiriamo in ambienti chiusi; monitorare la percentuale di umidità è importante per capire se si è in condizioni normali o d'allarme.

L'Organizzazione Mondiale della Sanità (OMS) suggerisce di mantenere i livelli di umidità in un valore compreso fra il 30% ed il 60%, per essere sicuri di garantire un ambiente né troppo secco né propenso alla formazione di muffe. È quindi rilevante che il sistema integri un monitoraggio del livello di umidità, per dare maggiore consapevolezza all'utente.

2.6.2 Temperatura

La temperatura interna di un'abitazione chiaramente non è un agente inquinante della stessa, ma situazioni di caldo o freddo prolungato possono alterare il benessere degli abitanti della casa; grandi sbalzi di temperatura potrebbero, ad esempio, favorire la formazione della muffa.

Allo stesso tempo, vivere in una casa fredda può contribuire all'insorgenza di malattie: secondo le direttive dell'OMS, in inverno la temperatura nelle nostre case non dovrebbe scendere sotto una minima di 18°C, per ridurre gli eventuali rischi.

A fronte di queste considerazioni, è importante integrare nel sistema il monitoraggio della temperatura della stanza in cui è posizionato il purificatore.

2.6.3 Anidride Carbonica

L'anidride carbonica (CO_2) è un gas incolore, inodore e non infiammabile, composto da un atomo di carbonio e due atomi di ossigeno.

La formazione di anidride carbonica nelle nostre case avviene principalmente in maniera naturale: gli esseri umani, infatti, durante il processo di respirazione inalano ossigeno e rilasciano anidride carbonica.

Perciò, basse concentrazioni di questo gas nell'aria non sono allarmanti; la situazione cambia quando ci sono altri fattori che contribuiscono alla sua formazione. Banalmente, anche la presenza di più persone in una stanza porta

ad un aumento di anidride carbonica; altri elementi impattanti possono essere la combustione legata al riscaldamento o l'anidride carbonica dovuta alla circolazione delle auto che, inevitabilmente, entra anche nelle nostre case. Se il livello di inquinamento atmosferico legato alla presenza di questo gas raggiunge livelli medio-alti, si possono avere riscontri negativi sulla salute, come emicranie e difficoltà respiratorie.

Data la frequenza con cui questo gas si forma in ambienti *indoor*, la progettazione del sistema deve includere la rilevazione della concentrazione di anidride carbonica.

2.6.4 Monossido di Carbonio

Il monossido di carbonio (CO) è un gas tossico, inodore, incolore, insapore e non irritante, composto da un atomo di carbonio e uno di ossigeno legati con triplo legame. Questo gas è prodotto dalla combustione incompleta di qualsiasi materiale di tipo organico, in concomitanza con una scarsa concentrazione di ossigeno nell'ambiente.

La pericolosità di questo gas sta nel fatto che può essere inalato senza che l'individuo se ne accorga e può arrivare a raggiungere concentrazioni nell'organismo che lo rendono letale.

Secondo il Ministero della Salute nelle abitazioni, in condizioni normali, i livelli sono compresi tra 1,5 e 4,5 mg/m3; invece, in presenza di processi di combustione, quali sistemi di riscaldamento e di cottura o di fumo di tabacco e inadeguata ventilazione, le concentrazioni interne possono superare quelle esterne e raggiungere livelli sino a 60 mg/m3[4].

Poiché la formazione di monossido di carbonio nelle nostre case non è inusuale e considerando la pericolosità del gas, è fondamentale includere un monitoraggio della sostanza nel sistema, così da poter segnalare opportunamente situazioni di allarme.

Capitolo 3

Progettazione

La seconda fase di sviluppo del progetto è incentrata sulla progettazione del sistema. Per progettazione si intende il processo che, a partire dai requisiti definiti in fase di analisi, stabilisce come è strutturata l'architettura del software, ne individua i diversi componenti e, successivamente, distribuisce le varie funzionalità fra di essi.

L'obiettivo di questo capitolo è illustrare le strategie adottate per soddisfare i requisiti identificati in precedenza, spiegando le scelte architetturali e di design che sono state ritenute più idonee per raggiungerli.

3.1 Panoramica sull'Architettura del Sistema

Per poter fornire una panoramica sull'architettura di *Smart-Air-Purifier*, è propedeutico stabilire cosa si intende per "architettura di un software".

Secondo lo standard 1471-2000 dell'IEEE (Institute of Electrical and Electronics Engineers), l'architettura di un software è definita come "L'organizzazione fondamentale di un sistema che si concretizza nei suoi componenti, nelle loro relazioni reciproche e con l'ambiente, e i principi che ne guidano la progettazione e l'evoluzione"[5].

Lo scopo principale di questa prima parte della fase di progettazione è, quindi, definire nel dettaglio l'architettura del sistema: per fare ciò, si è partiti dall'analisi dei pattern architetturali[1] esistenti, per poterne individuare uno che potesse rappresentare opportunamente il sistema in oggetto, e che facilitasse il raggiungimento degli obiettivi posti nella precedente fase di analisi.

[1]Un pattern architetturale è una soluzione di progettazione standardizzata e ben documentata che rappresenta un insieme di decisioni volte a risolvere problemi ricorrenti nell'ambito dello sviluppo software. Un pattern offre regole e principi per organizzare le interazioni tra sottosistemi predefiniti e le loro rispettive funzioni, definendo le componenti principali, le loro responsabilità e le relazioni tra di esse.

In ingegneria del software, uno dei pattern architetturali più comuni è quello chiamato *Layered Architecture*, in italiano "Architettura a Strati": i componenti del modello di architettura a strati sono organizzati in livelli orizzontali, come mostrato in figura 3.1, ciascuno dei quali svolge un ruolo specifico all'interno dell'applicazione.

Non ci sono restrizioni riguardo al numero di livelli in cui è possibile suddividere il sistema; in generale, tre strati sono sempre individuabili in quasi tutti i software, e in particolare anche in quello in oggetto: lo strato di presentazione, costituito dai componenti dell'interfaccia utente; lo strato di logica (o di business), formato dai componenti che gestiscono le operazioni fondamentali del sistema; lo strato dei dati, che raccoglie, organizza e immagazzina i dati presenti.

Ogni livello dell'architettura forma un'astrazione intorno al lavoro che deve essere svolto per soddisfare una particolare richiesta. Ad esempio, il livello di presentazione non deve sapere o preoccuparsi di come ottenere i dati dei clienti, deve solo visualizzare le informazioni su uno schermo in un determinato formato. Allo stesso modo, il livello di business non deve preoccuparsi di come formattare i dati dei clienti per visualizzarli su uno schermo e nemmeno da dove provengono questi dati, deve solo ottenerli, eseguire la logica di business su di essi (ad esempio, calcolare valori o elaborare nuovi dati) e passare le informazioni al livello di presentazione[6].

Il concetto fondamentale che è alla base di questo pattern architetturale è quello della "layer isolation", ovvero l'isolamento di ogni livello: gli strati sono organizzati in una gerarchia verticale in cui ogni layer comunica solamente con il livello sopra di esso o direttamente sottostante, attraverso delle interfacce o delle API[2] ben definite, favorendo la modularità e riducendo le dipendenze.

L'adozione del pattern appena descritto risulta ottimale per il sistema in oggetto; questo perché, a partire dai requisiti individuati in precedenza, è possibile riconoscere tre macro elementi che dovranno interagire fra di loro: un elemento che gestirà la raccolta dei dati, un elemento che incorporerà la logica centrale del sistema ed un elemento che si occuperà di presentare i dati raccolti all'utente finale.

Questi tre macro elementi possono costituire, in ottica *Layered Architecture*, tre livelli distinti: *Presentation Layer*, il livello di presentazione, *Business Logic Layer*, il livello della logica, *Data Layer*, il livello della raccolta dati.

[2]Con il termine API (Application Programming Interface) si intende un insieme di definizioni e protocolli che consentono alle diverse applicazioni software di comunicare tra loro. Un API definisce i metodi e le strutture dati che gli sviluppatori possono utilizzare per interagire con un componente software, un servizio o un sistema, senza doverne comprendere il funzionamento interno.

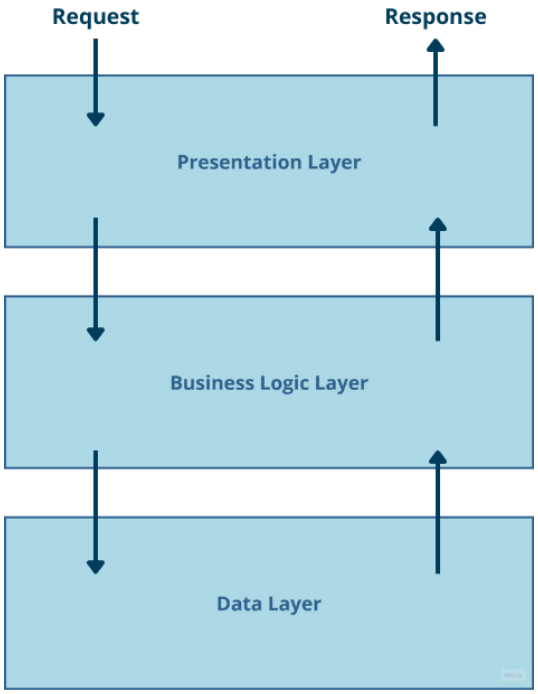

Figura 3.1: Layered Architecture

3.1.1 Obiettivi Architetturali

La scelta del pattern architetturale *Layered Architecture* come base per l'architettura di *Smart-Air-Purifier* è legata anche alle numerose proprietà che esso attribuisce al software.

Il primo beneficio è la **modularità** del sistema: la separazione delle funzionalità e delle responsabilità per ogni livello consente di ottenere una struttura modulare che permetta di apportare modifiche o sostituire un singolo strato senza dover agire su tutto il sistema; ciò aumenta anche la **facilità di manutenzione** di esso.

Un altro grande vantaggio è la **scalabilità**, che permette di ampliare e ottimizzare ogni livello in maniera flessibile e indipendente.

Infine, è importante anche la **testabilità** del sistema, ovvero la facilità di testing[3]: ogni livello può essere testato separatamente, agevolando l'individuazione e la correzione degli errori, così da garantire la robustezza e l'affidabilità dei singoli componenti.

[3]Il testing è il processo di valutazione e verifica di un software, volto ad accertare che il sistema porti al risultato atteso senza incombere in errori o malfunzionamenti.

3.2 Decomposizione del Sistema

Sul piano logico sono già stati individuati, nei precedenti paragrafi, i tre livelli che sono alla base dell'architettura del sistema: Presentation Layer, Business Logic Layer e Data Layer.

A partire da questa suddivisione, si vuole passare ad un piano prettamente strutturale, scendendo nel dettaglio e delineando con più accuratezza il risultato che questa ripartizione ha sul sistema. Il software nella sua interezza può essere scorporato in tre moduli distinti, ad ognuno dei quali viene assegnato uno specifico ruolo, vale a dire vengono ripartite fra di essi le funzionalità che dovranno poi essere implementate.

I tre moduli individuati rispecchiano la stratificazione logica iniziale: il *modulo front-end*, che implementa le funzionalità relative al Presentation Layer, il *modulo back-end*, che implementa quelle relative al Business Logic Layer, ed infine il *modulo embedded*, che implementa le funzionalità afferenti al Data Layer. Di seguito viene descritto singolarmente ogni modulo, spiegando con chiarezza qual è il suo meccanismo di funzionamento.

3.2.1 Modulo Front-end

Il modulo front-end è il sottosistema che si occupa di gestire tutti gli aspetti relativi all'ambito dell'interfaccia utente e alla gestione delle interazioni con esso. Questo modulo implementa le funzionalità del Presentation Layer e ha il compito di presentare i dati raccolti e elaborati dal sistema in modo chiaro e intuitivo. Per fare ciò, si rende necessaria la progettazione di un'interfaccia user-friendly e accessibile che consenta agli utenti di visualizzare i dati sulla qualità dell'aria e di interagire con il purificatore.

È importante tenere in considerazione la responsività, ovvero la capacità dell'interfaccia di adattarsi a vari dispositivi, inclusi desktop, tablet e smartphone, garantendo un'esperienza ottimale indipendentemente dalla dimensione dello schermo utilizzato.

3.2.2 Modulo Back-end

Il modulo back-end è il sottosistema il cui ruolo principale è fare da tramite fra modulo front-end e modulo embedded. Esso racchiude la logica di gestione delle comunicazioni nel sistema; infatti, riceve e gestisce le richieste provenienti dal front-end, come l'ottenimento di dati aggiornati o comandi per controllare il purificatore e, per soddisfare queste richieste, si interfaccia con il modulo embedded.

3.2.3 Modulo Embedded

Il modulo embedded è il sottosistema che ha il compito di interagire direttamente con l'hardware del dispositivo fisico. Questo modulo è il responsabile della raccolta dei dati dai sensori e della trasmissione di essi al modulo back-end. Inoltre, controlla il funzionamento degli attuatori in base ad una logica di calcolo interna o ai comandi ricevuti dal modulo back-end.

3.3 Diagramma dei Componenti

Per visualizzare con maggiore chiarezza gli elementi che compongono i moduli appena descritti, viene riportato un diagramma dei componenti realizzato seguendo il formalismo UML, figura 3.2.

Il diagramma illustra in modo chiaro come i vari componenti del sistema collaborano per monitorare e controllare la qualità dell'aria, assicurando una gestione efficiente delle risorse e una comunicazione efficace tra i diversi livelli. Nel dettaglio, sono presenti i seguenti elementi:

- **Web Interface**: rappresenta l'interfaccia utente accessibile via web, attraverso un browser.
- **Mobile Interface**: rappresenta l'interfaccia utente adattata per la fruizione attraverso un dispositivo mobile.
- **Front-end Connection Handler**: gestisce le richieste provenienti dalle interfacce.
- **Embedded Connection Handler**: gestisce la comunicazione tra il back-end ed il modulo embedded.
- **Smart-Air-Purifier Business Logic**: elabora i dati e gestisce lo scambio di informazioni.
- **Purifier Controller**: gestisce la logica interna e le operazioni principali del modulo embedded.
- **Sensors Controller**: raccoglie i dati dei sensori sulla qualità dell'aria.
- **Actuators**: regola il funzionamento degli attuatori, che eseguono operazioni per modificare le condizioni ambientali.

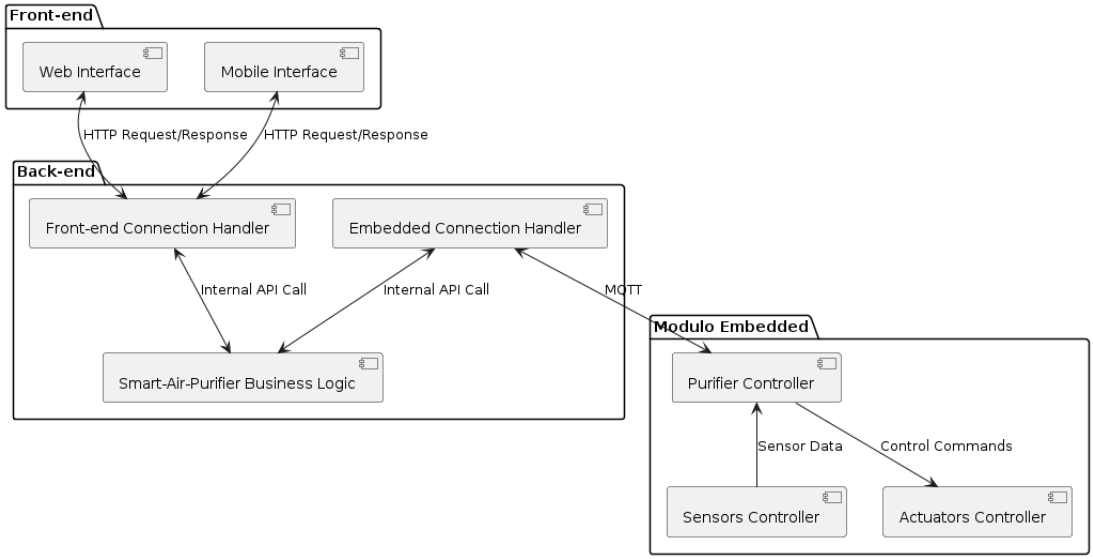

Figura 3.2: Diagramma dei componenti

3.4 Comunicazione tra i Moduli

Un aspetto su cui porre particolare attenzione per garantire un corretto funzionamento del sistema è quello della comunicazione fra i vari moduli.

I flussi di comunicazione nel sistema sono due: uno fra modulo front-end e modulo back-end ed uno fra modulo back-end e modulo embedded.

Come già spiegato in precedenza, la comunicazione è principalmente volta ad ottenere dati relativi al monitoraggio dell'aria o comandi per controllare il purificatore; è importante garantire che questa sia stabile e continuativa, per permettere un aggiornamento in tempo reale.

Per rappresentare visivamente come avviene la comunicazione fra i moduli viene riportato, in figura 3.3, un diagramma di sequenza realizzato con il formalismo UML. Il diagramma visualizza come i messaggi vengono scambiati tra i vari componenti del sistema durante un processo tipico di monitoraggio della qualità dell'aria e controllo del purificatore.

L'attore è l'utente, mentre i componenti sono `Front-end` (abbreviazione di modulo front-end), `Back-end` (abbreviazione di modulo back-end) e `Modulo Embedded`.

Il flusso di comunicazione si articola nel seguente modo:

- **Utente**: Avvia l'interazione collegandosi all'interfaccia del front-end.
- **Front-end**: Invia richieste al back-end per ottenere i dati della qualità dell'aria e inviare comandi.

- **Back-end**: Elabora le richieste, comunica con il modulo embedded per ottenere i dati dai sensori e inviare comandi.
- **Modulo Embedded**: Rileva i dati dai sensori e invia le letture al back-end, riceve comandi dal back-end e agisce di conseguenza.

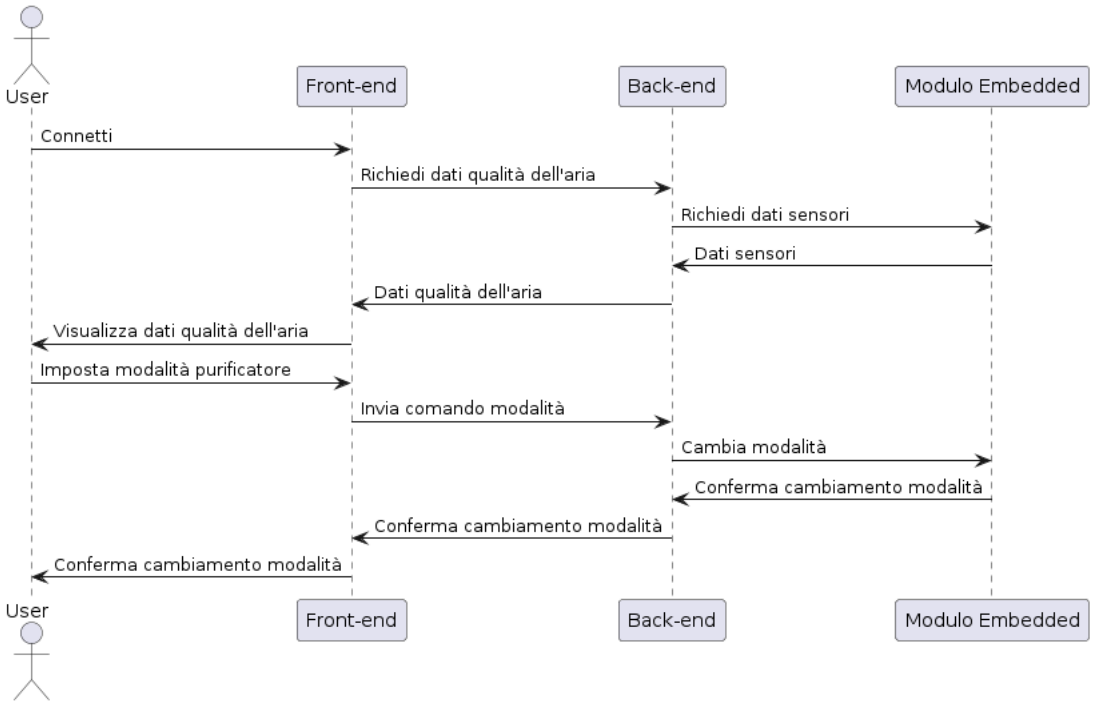

Figura 3.3: Diagramma di sequenza

3.5 Design Dettagliato dei Componenti

Ogni modulo del sistema è soggetto ad un processo di design differente: i ragionamenti e le riflessioni fatte in fase di progettazione variano a seconda delle funzionalità che il modulo deve implementare.

Per documentare e rendere più chiaro il processo creativo che ha poi condotto all'effettiva implementazione dei componenti, vengono di seguito riportati e descritti gli schemi e i diagrammi utilizzati a supporto di questa ultima fase della progettazione.

3.5.1 Design del Modulo Front-end

Il modulo front-end è relativo alla presentazione del sistema, perciò i suoi componenti costituiscono l'aspetto visivo. Per progettare una buona inter-

faccia utente, che soddisfi i requisiti elencati in precedenza, esistono diverse tecniche nell'ambito del Web Design, ovvero la disciplina che si occupa della progettazione e realizzazione di siti web.

Una di queste tecniche è l'utilizzo dei "mockup": in generale, un mockup è la riproduzione di un oggetto o di un dispositivo, utilizzato per l'insegnamento, la dimostrazione, la valutazione di un progetto, la promozione e altri scopi. In contesto Web, un mockup è una rappresentazione grafica della schermata di interazione con il sistema che serve a mostrare come sarà un prodotto prima ancora della sua realizzazione.

Per capire come strutturare l'interfaccia di *Smart-Air-Purifier* e decidere come organizzarne gli elementi, è stato realizzato il mockup rappresentato nella figura 3.4. La dimensione scelta fa riferimento ad uno schermo piccolo, come ad esempio quello di un dispositivo mobile; per un dispositivo più grande, come quello di un computer, la disposizione degli elementi andrà adattata in fase di implementazione.

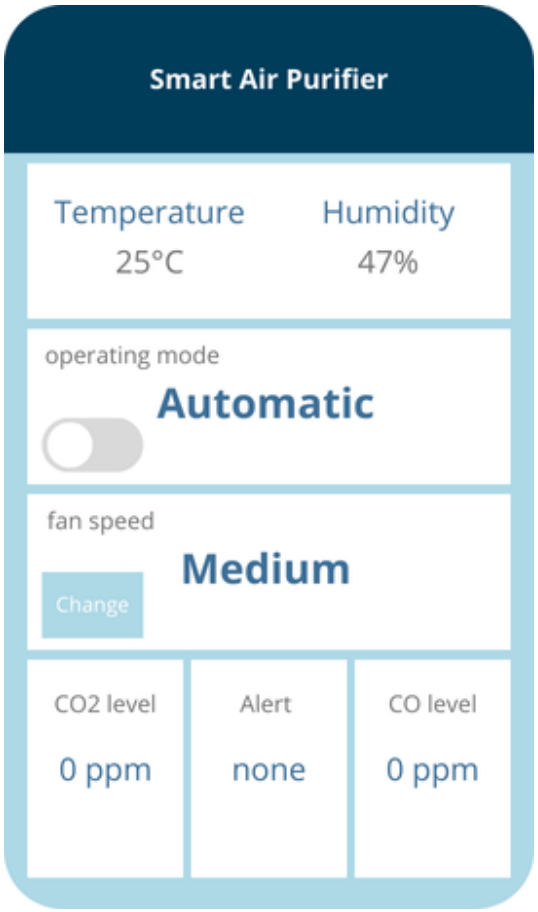

Figura 3.4: Mockup dell'interfaccia utente

3.5.2 Design del Modulo Back-end

Come spiegato nei precedenti paragrafi, il modulo back-end è il centro della comunicazione fra i moduli del sistema; ai fini di questo progetto, il design di questo modulo non risulta essere particolarmente laborioso.

Sarà più cruciale la parte finale relativa all'implementazione, volta ad individuare le tecnologie e metodologie più adatte per soddisfare i requisiti posti in fase di analisi.

3.5.3 Design del Modulo Embedded

Il modulo embedded racchiude la logica per interfacciarsi con la parte hardware. Per effettuare una progettazione completa e approfondita, è importante visualizzare quali sono i componenti del sottosistema; uno schema spesso utilizzato per rappresentare questo tipo di sistemi è il "block diagram", che permette di dare una panoramica ad alto livello.

Un block diagram, o diagramma a blocchi, è un tipo di rappresentazione grafica che descrive un sistema o un processo, suddividendo le sue funzioni principali in blocchi interconnessi. In questo tipo di diagramma comunemente utilizzato in ingegneria e in elettronica, ogni blocco rappresenta un componente, una funzione o un processo, e le linee di collegamento indicano il flusso di dati, le informazioni o i segnali tra questi blocchi.

I componenti del modulo embedded di *Smart-Air-Purifier* sono rappresentati nel diagramma in figura 3.5; nel dettaglio, sono i seguenti:

- **Program Control**: è il cuore del sottosistema. Esso gestisce le operazioni di controllo del dispositivo, ricevendo i dati dai sensori, elaborando le informazioni e inviando i comandi agli attuatori (in questo caso, la ventola).

- **Temperature sensor**: sensore che monitora la temperatura dell'ambiente e invia le informazioni raccolte al modulo di controllo.

- **Humidity sensor**: sensore che lavora similmente al sensore di temperatura, raccogliendo però dati relativi all'umidità.

- **Carbon Monoxide sensor**: sensore di monossido di carbonio, il cui funzionamento è simile a quello degli altri sensori.

- **Carbon Dioxide sensor**: sensore di anidride carbonica che lavora in maniera assimilabile agli altri.

- **Fan**: è l'unico vero attuatore del sistema, ovvero una ventola che regola la sua velocità o accensione/spegnimento in base ai dati ricevuti dai sensori o ai comandi dell'utente.

- **Connectivity**: è il componente che permette al modulo di comunicare all'esterno attraverso diverse tecnologie.

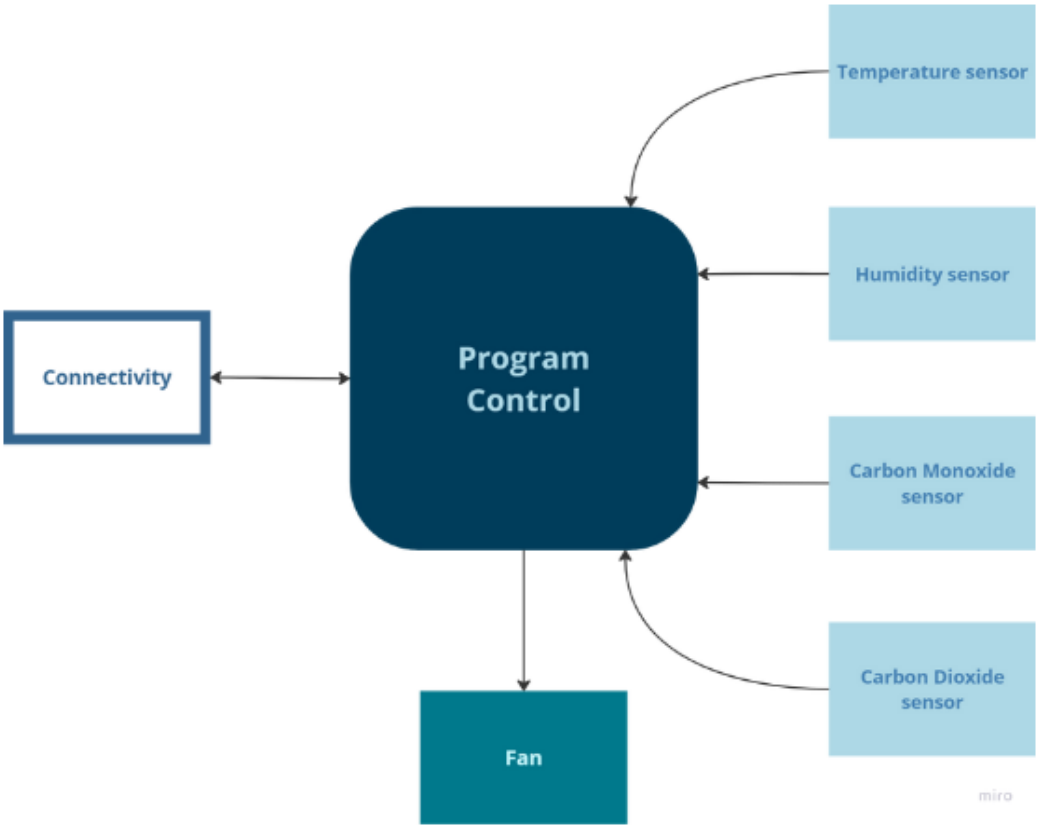

Figura 3.5: Block diagram del modulo embedded

Oltre a capire quali sono gli elementi che compongono il modulo, un altro aspetto importante da considerare in fase di progettazione è fornire una rappresentazione chiara del comportamento dinamico e, perciò, del meccanismo di funzionamento del sottosistema; ciò è fondamentale per ottenere un software di controllo reattivo ed efficiente.

Ad accompagnamento di questa fase di design è stato, quindi, realizzato un diagramma degli stati, seguendo il formalismo UML.

Dal diagramma rappresentato in figura 3.6 si evince che il sottosistema può operare in tre diverse modalità: "Automatic", modalità automatica; "Manual", modalità manuale; "Alarm", modalità d'allarme.

Il funzionamento ha inizio quando il dispositivo fisico viene accesso, si connette e calibra i sensori; all'accensione, la modalità indicata è quella automatica.

In questa modalità, viene costantemente controllato lo stato della connessione, viene calcolata e impostata la velocità della ventola e vengono letti e inviati i valori dai sensori.

Quando viene ricevuto dall'esterno il comando di cambio modalità e si passa in quella manuale, il funzionamento è simile a quello della modalità precedente; la differenza sostanziale è che la velocità della ventola viene impostata manualmente dall'utente.

Se viene superata una soglia di allarme relativa alla qualità dell'aria rilevata, si passa nell'ultima modalità, quella di allarme appunto, in cui il dispositivo riprende il controllo; si rimane in questa modalità fino a che non si scende nuovamente sotto la soglia limite.

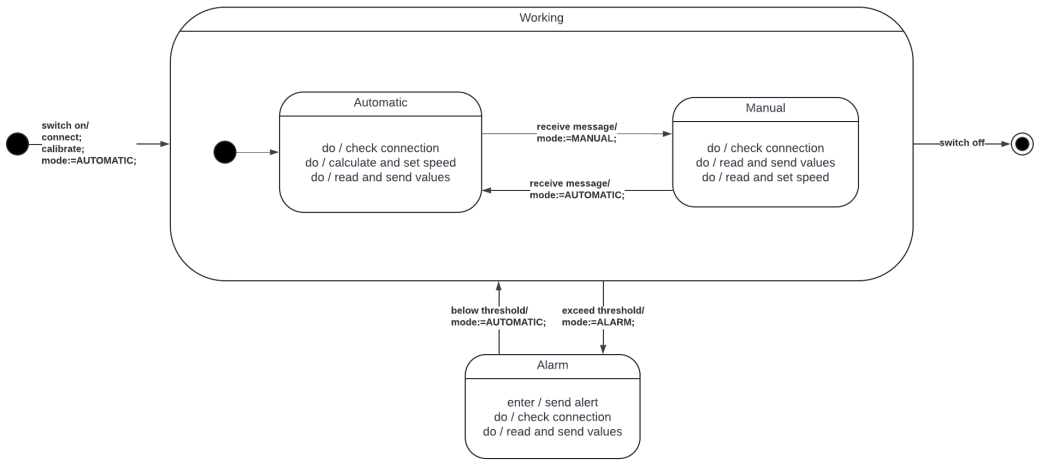

Figura 3.6: Diagramma degli stati

Capitolo 4

Implementazione di un Prototipo

La terza fase di sviluppo del progetto è l'effettiva implementazione. In questo capitolo vengono descritte le soluzioni messe in campo per raggiungere gli obiettivi posti nelle precedenti fasi. Inoltre, vengono spiegate le tecnologie che sono state utilizzate e vengono citate le librerie sfruttate a supporto dell'implementazione.

4.1 Implementazione del Modulo Front-end

Per l'implementazione del modulo front-end si è deciso di sfruttare le tecnologie alla base del Web Design, senza l'utilizzo di particolari framework.

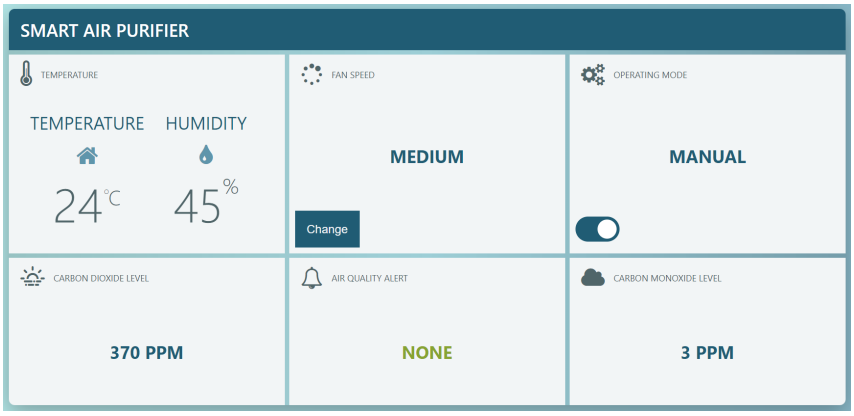

Figura 4.1: Dashboard Web di Smart-Air-Purifier

Ciò vuol dire che per la struttura della pagina web è stato utilizzato il linguaggio di markup HTML; la personalizzazione dell'aspetto presentazionale e

decorativo dell'interfaccia è stata fatta attraverso CSS, che ha inoltre permesso l'utilizzo delle media query[1] per adattare la pagina a schermi di dimensioni ridotte, così da rendere il sito fruibile attraverso sia un PC che un dispositivo mobile. Infine, per interagire con il back-end e per ottenere i dati sulla qualità dell'aria, è stato utilizzato il linguaggio di programmazione JavaScript, che ha permesso anche di curare l'aspetto interattivo della pagina.

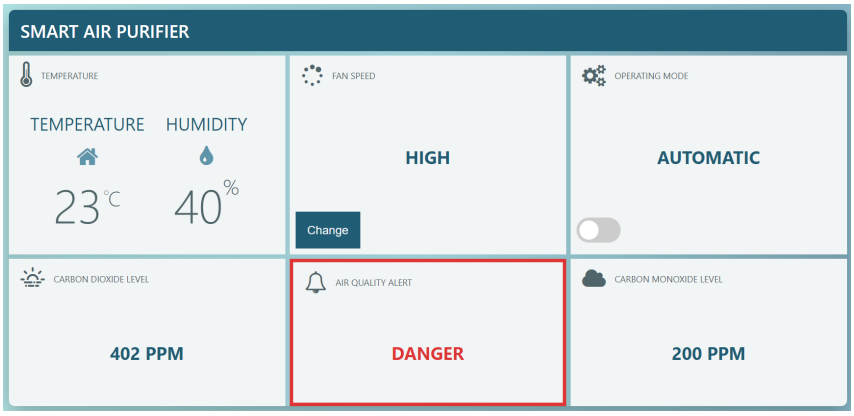

Figura 4.2: Dashboard Web di Smart-Air-Purifier in stato di allarme

HTTP

Per gestire la comunicazione tra modulo front-end e modulo back-end è stato utilizzato il protocollo HTTP.

L'HyperText Transfer Protocol (HTTP) è un protocollo di comunicazione largamente utilizzato per la diffusione ed il trasferimento di informazioni nel World Wide Web. Il meccanismo di funzionamento è richiesta-risposta (request-response) tra un client, generalmente un browser che invia richieste per ottenere le pagine web, ed un server, che ospita le pagine richieste dal client. È un protocollo *stateless*, cioè senza stato, il che vuol dire che ogni richiesta dal client al server è indipendente e non conserva alcuna informazione di stato. HTTP è alla base della comunicazione su Internet e del funzionamento del web: rende possibile l'accesso alle risorse e l'interazione tra client e server in modo standardizzato e interoperabile.

[1]Le media query sono elementi introdotti in CSS3 fondamentali per lo sviluppo web *responsive*: servono a gestire comportamenti diversi e modificare la strutturazione del sito in base ad alcune condizioni come, ad esempio, risoluzioni differenti.

4.1.1 Accessibilità e Validazione del Codice HTML

Nell'ambito del Web design, il concetto di *accessibilità* riguarda la progettazione di siti e applicazioni in modo che siano facilmente fruibili anche da persone con disabilità, assicurando che tutti, indipendentemente dalle loro capacità fisiche o cognitive, possano accedere ai contenuti e interagire con le funzionalità. Volendo implementare un sistema che sia apprezzabile da diverse categorie di utenti, l'accessibilità è un aspetto che non può essere trascurato.

Esistono online diversi strumenti per verificare l'accessibilità di un sito web, conformi alle WCAG (Web Content Accessibility Guidelines), ovvero una serie di linee guida pubblicate dal Web Accessibility Initiative che, a sua volta, fa parte del World Wide Web Consortium. Per verificare l'accessibilità del codice HTML, in questo caso, è stato utilizzato il tool di un sito web, consultabile al link `https://websiteaccessibilitychecker.com/checker/index.php`.

Inoltre, sono stati controllati i contrasti fra i colori utilizzati nel foglio di stile dell'interfaccia, per rendere ottimale anche l'esperienza di utenti con disabilità visive; ciò è stato fatto utilizzando il tool che si può trovare al link `https://webaim.org/resources/contrastchecker/`.

4.2 Implementazione del Modulo Back-end

Il modulo back-end racchiude la logica di comunicazione e scambio dei messaggi fra modulo front-end e modulo embedded, come spiegato nel precedente capitolo.

Per l'implementazione, si è scelto di realizzare due classi, scritte in Java: una prima classe, `MQTTConnectionHandler` per gestire la connessione con il modulo embedded attraverso MQTT, un protocollo di messaggistica il cui funzionamento verrà approfondito in seguito; una seconda classe, `HTTPConnectionHandler` che si occupa, invece, della gestione della comunicazione con il modulo front-end attraverso HTTP, il cui funzionamento è stato già spiegato in precedenza.

Per gestire lo scambio di informazioni interno fra le due classi, si è fatto uso di Vert.x.

Vert.x

Eclipse Vert.x è un tool-kit di per la creazione di applicazioni *reactive* sulla JVM (Java Virtual Machine). Le applicazioni *reactive* sono sia scalabili quando i carichi di lavoro crescono, sia resilienti quando si verificano guasti. Un'applicazione *reactive* è "reattiva" perché tiene sotto controllo la latenza, utilizzando in modo efficiente le risorse di sistema e proteggendosi dagli errori[7].

Vert.x supporta la programmazione asincrona e mette a disposizione delle API per diversi servizi, tra cui lo scambio di messaggi via HTTP e MQTT.

Verticle
Un Verticle è un'unità di distribuzione di codice che può essere eseguita da Vert.x, simile ad un thread ma con alcune differenze: lo si può pensare come un componente di un'applicazione Vert.x, che a sua volta può essere composta da molti Verticle che lavorano insieme. Ogni Verticle ha un ciclo di vita e può essere distribuito, eseguito e ritirato indipendentemente.

In particolare, un Verticle può essere scritto come una classe Java che estende la classe `AbstractVerticle` di Vert.x e sovrascrive il metodo `start()`, esattamente ciò che è stato fatto per `MQTTConnectionHandler` e `HTTPConnectionHandler`; questo metodo viene chiamato quando il Verticle viene distribuito.

Per comunicare tra loro, i Verticle utilizzano l'`EventBus` di Vert.x, che supporta la pubblicazione/sottoscrizione, la messaggistica punto-punto e la richiesta/risposta.

MQTTConnectionHandler
Come già specificato, la classe `MQTTConnectionHandler` è un Verticle Vert.x che gestisce la connessione con un broker MQTT. Il suo funzionamento è strutturato come segue:

- All'avvio, viene creato un client MQTT che tenta di connettersi ad un broker su "broker.hivemq.com" (un broker gratuito disponibile online, `https://www.hivemq.com/mqtt/public-mqtt-broker/`);
- Se la connessione ha successo, si iscrive a tre topic: "Smart-air-purifier/sensor/data", "Smart-air-purifier/fan/speed" e "Smart-air-purifier/control/mode";
- Quando riceve un messaggio su uno di questi tre topic, lo elabora come segue: se si tratta del topic "Smart-air-purifier/sensor/data", elabora i dati dei sensori, converte il messaggio in un oggetto JSON e pubblica i valori di temperatura, umidità, livello di CO e livello di CO2 sull'`EventBus` Vert.x. Se il messaggio è ricevuto su uno degli altri due topic, viene semplicemente pubblicato sul bus;
- Ascolta anche i messaggi sull'`EventBus` Vert.x, sugli argomenti "http/Smart-air-purifier/fan/speed" e "http/Smart-air-purifier/control/mode". Quando riceve un messaggio su uno di questi argomenti, pubblica il messaggio sull'argomento MQTT corrispondente.

```java
public class MQTTConnectionHandler extends AbstractVerticle {

    private MqttClient client;

    @Override
    public void start() {
        client = MqttClient.create(vertx);
        client.connect(1883, "broker.hivemq.com", s -> {
            if (s.succeeded()) {
                Logger.success("Connected to a server");
                subscribeToTopics();
            } else {
                Logger.error("Failed to connect to a server");
            }
        });
        vertx.eventBus().consumer("http/Smart-air-purifier/fan/speed",
    message -> {
            String speed = (String) message.body();
            publishMessage("Smart-air-purifier/fan/speed", speed);
        });
        vertx.eventBus().consumer("http/Smart-air-purifier/control/mode",
    message -> {
            String mode = (String) message.body();
            publishMessage("Smart-air-purifier/control/mode", mode);
        });
    }
```

Listato 4.1: Codice della classe MQTTConnectionHandler, metodo `start()`

HTTPConnectionHandler

Anche la classe `HTTPConnectionHandler` è un Verticle Vert.x che, specularmente alla classe precedente, gestisce la connessione HTTP.

Il suo funzionamento è il seguente:

- Inizialmente viene creato un router Vert.x;
- Il router è usato per configurare un percorso GET a "/data", che restituisce un oggetto JSON con i dati dei sensori, la velocità della ventola e la modalità corrente;
- Inoltre, configura due percorsi POST a "/fan-speed" e "/mode" che permettono di impostare, rispettivamente, la velocità della ventola e la modalità. Quando riceve una richiesta su uno di questi percorsi, legge il corpo della richiesta, lo usa per aggiornare la velocità della ventola o la modalità, pubblica il nuovo valore sull'`EventBus` Vert.x e risponde con un messaggio di conferma.

- Il router è usato anche per ascoltare i messaggi sull'`EventBus` Vert.x sugli argomenti "sensor/temperature", "sensor/humidity", "sensor/CO-level", "sensor/CO2level", "Smart-air-purifier/control/mode" e "Smart-air-purifier/fan/speed". Quando riceve un messaggio su uno di questi argomenti, aggiorna il valore corrispondente.

- Infine, viene avviato un server HTTP che utilizza il router configurato per gestire le richieste.

```
public class HTTPConnectionHandler extends AbstractVerticle {

    private Float temperature;
    private Float humidity;
    private Float COlevel;
    private Float CO2level;
    private String fanSpeed;
    private String mode;

    @Override
    public void start() {
        Router router = Router.router(vertx);

        router.route("/*").handler(StaticHandler.create("../frontend-module
    "));

        router.get("/data").handler(ctx -> {
            JsonObject response = new JsonObject()
                    .put("temperature", temperature)
                    .put("humidity", humidity)
                    .put("COlevel", COlevel)
                    .put("CO2level", CO2level)
                    .put("fanSpeed", fanSpeed)
                    .put("mode", mode);
            ctx.response().putHeader("Content-Type", "application/json").
    end(response.encode());
        });

        router.post("/fan-speed").handler(ctx -> ctx.request().bodyHandler(
    body -> {
            String speed = body.toString();
            this.fanSpeed = speed;
            vertx.eventBus().publish("Smart-air-purifier/fan/speed", speed)
    ;
            ctx.response().end("Fan speed set to " + speed);
        }));

        router.post("/mode").handler(ctx -> ctx.request().bodyHandler(body
    -> {
```

```
            String mode = body.toString();
            this.mode = mode;
            vertx.eventBus().publish("Smart-air-purifier/control/mode",
    mode);
            ctx.response().end("Mode set to " + mode);
        }));

        vertx.eventBus().consumer("sensor/temperature", message ->
    temperature = (Float) message.body());
        vertx.eventBus().consumer("sensor/humidity", message -> humidity =
    (Float) message.body());
        vertx.eventBus().consumer("sensor/COlevel", message -> COlevel = (
    Float) message.body());
        vertx.eventBus().consumer("sensor/CO2level", message -> CO2level =
    (Float) message.body());
        vertx.eventBus().consumer("Smart-air-purifier/fan/speed", message
    -> fanSpeed = (String) message.body());
        vertx.eventBus().consumer("Smart-air-purifier/control/mode",
    message -> mode = (String) message.body());

        vertx.createHttpServer().requestHandler(router).listen(8000, res ->
     {
            if (res.succeeded()) {
                Logger.success("HTTP server started on port 8000");
            } else {
                Logger.error("Failed to start HTTP server");
            }
        });
    }
}
```

Listato 4.2: Codice della classe HTTPConnectionHandler

4.3 Implementazione del Modulo Embedded

L'implementazione del modulo embedded può essere divisa in due parti: una relativa all'aspetto software, l'altra relativa all'aspetto hardware.

Questo perché la fase di sviluppo del progetto ha incluso non solo l'effettiva implementazione del sistema, ma anche l'assemblaggio del circuito fisico.

Essendo due piani di sviluppo differenti, che hanno richiesto tecnologie e conoscenze diverse, è bene che siano separati, per poter approfondire entrambi nel modo corretto.

Software

Per l'implementazione del codice di controllo del dispositivo, è stato utilizzato PlatformIO, un ambiente di sviluppo integrato facile da usare che offre una serie di strumenti professionali, funzioni moderne e potenti per accelerare e semplificare la creazione di prodotti embedded[8]. Il codice è stato scritto utilizzando C++.

L'organizzazione del codice è la seguente:

- `SensorHandler`: classe che gestisce i sensori, vale a dire la loro calibrazione, la raccolta delle misurazioni e la codifica di esse in formato JSON;

```
String SensorHandler::encode()
{
  JsonDocument jsonDoc;

  jsonDoc["temperature"] = temperature;
  jsonDoc["humidity"] = humidity;
  jsonDoc["COlevel"] = MQ7CO;
  jsonDoc["CO2level"] = CO2 + PollutionOffeset;

  String jsonString;
  serializeJson(jsonDoc, jsonString);

  return jsonString;
}
```

Listato 4.3: Codifica dei dati in formato JSON

- `WIFIConnection`: classe che si occupa della connessione Wi-Fi dell'ESP32 e che offre metodi per controllare periodicamente che questa sia stabile;

```
void WIFIConnection::connect()
{
    delay(100);
    if (WiFi.status() != WL_CONNECTED)
    {
        Serial.println("Connecting to WiFi...");
        WiFi.mode(WIFI_STA);
        WiFi.begin(ssid, password);
        while (WiFi.status() != WL_CONNECTED)
        {
            delay(500);
            Serial.print(".");
        }
        greenLed->switchOn();
        redLed->switchOff();
```

```
        Serial.println("Connected to WiFi");
        Serial.println("IP address: ");
        Serial.print(WiFi.localIP());
    }
}
```

Listato 4.4: Codice per la connessione alla rete Wi-Fi

- `MQTTConnection`: classe con funzionamento simile alla precedente ma relativa alla connessione MQTT;

```
void MQTTConnection::connect()
{
    delay(100);
    if (!client.connected())
    {
        Serial.println("Connecting to MQTT...");
        String clientID = "ESP32Client-";
        clientID += String(random(0xffff), HEX);
        while (!client.connect(clientID.c_str(), mqttUsername,
    mqttPassword))
        {
            delay(500);
            Serial.print(".");
        }
        greenLed->switchOn();
        redLed->switchOff();
        Serial.println("Connected to MQTT");
    }
}
```

Listato 4.5: Codice per la connessione ad MQTT

- `FanController`: classe che imposta e gestisce la velocità della ventola;

```
int FanController::calculateFanSpeed(float temp, float coLevel, float
    co2Level)
{
    float weightedScore = temp * tWeight + co2Level * co2Weight +
    coLevel * coWeight;
    auto bound = speedTable.lower_bound(weightedScore);

    if (bound == speedTable.end())
    {
        return speedTable.rbegin()->second;
    }
    return bound->second;
}
```

Listato 4.6: Metodo per calcolare la velocità della ventola

- `PurifierStateMachine`: classe che gestisce il cambio della modalità di funzionamento del purificatore, secondo la logica mostrata nello schema 3.6.

Chiaramente, è presente un `main` che richiama ciclicamente i metodi delle classi sopra citate, per garantire un funzionamento fluido.

Hardware
Con le risorse e le conoscenze disponibili, non sarebbe stato possibile realizzare un vero e proprio dispositivo per la purificazione dell'aria, con caratteristiche simili a quelli presenti sul mercato. Ai fini di questa tesi, si è scelto di realizzare un prototipo che integrasse i sensori che potessero rilevare i tipi di dato individuati in fase di analisi e che, attraverso una logica interna, controllasse una ventola, a scopo dimostrativo dell'azione purificante. È stato necessario capire come collegare i sensori al microcontrollore nel modo corretto e come gestire il controllo della velocità della ventola, per far sì che tutti gli elementi del circuito cooperassero fluidamente.

4.3.1 Circuito

Per la creazione del dispositivo è stata utilizzata una breadboard, ovvero una piattaforma riutilizzabile per prototipazione elettronica che permette di costruire circuiti senza saldature. In essa, sono stati integrati i seguenti elementi, visualizzabili in figura 4.3:

- **ESP32-S3-DevKitC-1**: microcontrollore, cuore del sistema;
- **DHT11**: sensore di umidità e temperatura;
- **MQ-135**: sensore per l'anidride carbonica;
- **MQ-7**: sensore per il monossido di carbonio;
- Un **led rosso** e un **led verde** che segnalano lo stato della connessione del microcontrollore alla rete Wi-Fi;
- **Resistenze da 220 Ohm**, per limitare la corrente che scorre attraverso i led;
- **Resistenze da 10k Ohm**: quella utilizzata per il sensore DHT11 funge da pull-up per il pin dati, ovvero garantisce che questo pin sia in uno stato definito ed evita che fluttui tra alto e basso, il che potrebbe causare letture errate. Quelle utilizzate per i sensori MQ-135 ed MQ-7, invece, sono resistenze di polarizzazione che permettono di stabilizzare la tensione di uscita del sensore, per garantire letture precise.

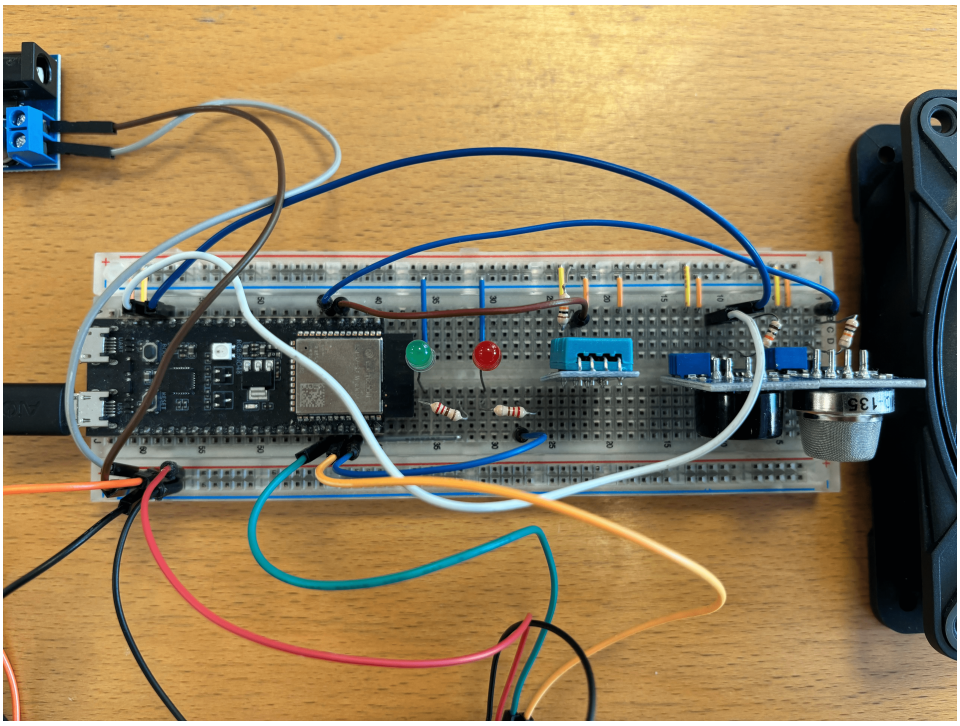

Figura 4.3: Immagine della board

- Vari jumper per collegare tutti i componenti;

Oltre ai componenti citati in precedenza, il dispositivo dispone anche di una ventola per PC: il prototipo completo è presentato in figura 4.4. Per caricare il codice da eseguire sull'ESP32 è stato utilizzato un cavo USB/MicroUSB; per il funzionamento autonomo del purificatore, invece, il dispositivo viene collegato alla corrente con un cavo con output 12V (necessari per alimentare la ventola), che, attraverso un modulo di regolazione della tensione, vengono ridotti a 5V per alimentare il microcontrollore.

4.3.2 Tecnologie Utilizzate

Di seguito viene fatta una panoramica sulle tecnologie utilizzate per l'implementazione del modulo embedded e per la costruzione effettiva del purificatore d'aria, descrivendone il funzionamento e le caratteristiche principali.

ESP32
ESP32 è una famiglia di microcontrollori (MCU - Micro Controller Unit) sviluppata da *Espressif Systems* e progettata per un vasto numero di applicazioni. In particolare, *Espressif* ha sviluppato tre categorie di ESP32: ESP32 SoC

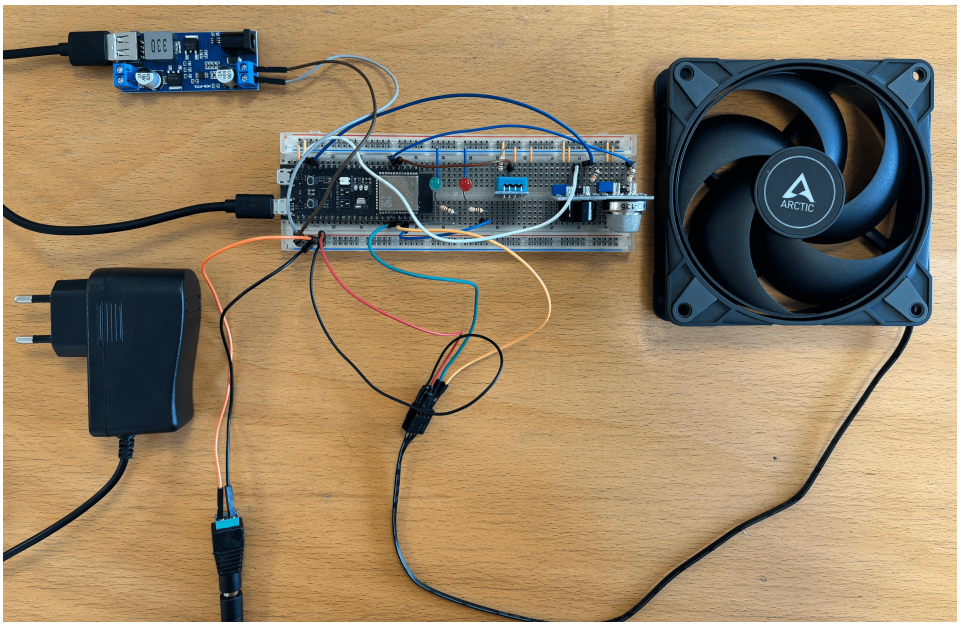

Figura 4.4: Immagine del prototipo completo

(System on a Chip), ESP32 Modules ed ESP32 DevKits, le quali hanno specifiche e utilizzi differenti. Per questo progetto, è stata utilizzata una scheda ESP32-S3-DevKitC-1 rappresentata in figura 4.5.

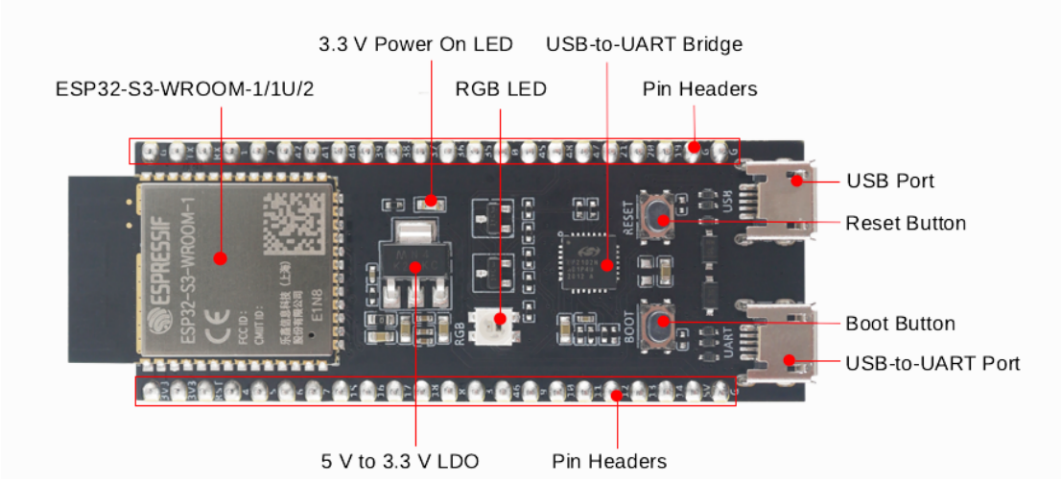

Figura 4.5: Immagine che mostra i componenti dell'ESP32-S3-DevKitC-1[9]

ESP32-S3-DevKitC-1 è una scheda di sviluppo entry-level dotata di un modulo ESP32-S3-WROOM-1, in grado di fornire funzionalità avanzate di

connettività Wi-Fi e Bluetooth. Inoltre, è dotata di pin di I/O distribuiti su entrambi i lati della scheda e può essere utilizzata attraverso una breadboard.

La scelta di utilizzare questa scheda è dovuta al fatto che le caratteristiche che presenta sono ottimali per le funzionalità implementate in questo progetto; inoltre, l'ESP32 è uno dei microcontrollori più diffusi nei progetti IoT (Internet of Things).

Sensore di Umidità e Temperatura DHT11
Il sensore DHT11 è un dispositivo digitale utilizzato per misurare l'umidità e la temperatura. È composto da un sensore di umidità capacitivo e un termistore, insieme a un chip per la conversione dei segnali analogici in digitali.

Le misurazioni, nello specifico, avvengono nei seguenti modi:

- **Misurazione dell'umidità**: il sensore di umidità capacitivo è in grado di misurare l'umidità relativa basandosi sulla variazione della capacità elettrica. Il sensore è costituito da due piastre conduttive separate da un materiale dielettrico, sensibile all'umidità; quando quest'ultima, nell'aria circostante, cambia, l'umidità assorbita dal materiale dielettrico modifica la sua permittività[2], causando una variazione della capacità tra le piastre. Questa variazione viene poi convertita in un segnale elettrico che può essere letto e interpretato da un microcontrollore.

- **Misurazione della temperatura**: il termistore[3] è di tipo NTC (Negative Temperature Coefficient), il che significa che la sua resistenza diminuisce all'aumentare della temperatura.

  Quando la temperatura ambientale cambia, la variazione di resistenza del termistore viene misurata e convertita analogamente a quanto avviene per la misurazione dell'umidità.

Le caratteristiche del sensore DHT11 più rilevanti vengono elencate di seguito:

- **Precisione**: ha una precisione di ±2°C per la temperatura e ±5% per l'umidità.

- **Intervallo di misurazione**: la temperatura può essere misurata tra 0°C e 50°C, mentre l'umidità può essere misurata tra 20% e 90% RH (Relative Humidity).

[2]La permittività elettrica è una grandezza fisica che quantifica la tendenza di un materiale a contrastare l'intensità del campo elettrico presente al suo interno. Descrive quindi il comportamento di un materiale in presenza di un campo elettrico.

[3]Un termistore è un resistore il cui valore di resistenza varia in maniera significativa in base alla temperatura.

- **Comunicazione**: la comunicazione avviene attraverso *single-bus*, ovvero è presente un unico filo che gestisce sia i segnali di alimentazione che quelli dei dati[10].

Sensore di Monossido di Carbonio MQ-7
Il sensore di monossido di carbonio MQ-7 è un componente elettronico progettato per rilevare la presenza di monossido di carbonio (CO) nell'aria.

MQ-7 utilizza un sensore elettrochimico che è capace di interagire chimicamente con il CO ed ha quindi un'alta sensibilità ad esso. MQ-7 è dotato di un elemento riscaldante interno che porta il sensore alla temperatura di lavoro ottimale: questo riscaldamento è necessario per garantire una risposta stabile e accurata. Quando il monossido di carbonio entra in contatto con il materiale sensibile all'interno del sensore, avviene una reazione chimica che modifica la resistenza elettrica del materiale.

La variazione della resistenza del materiale sensibile è proporzionale alla concentrazione di CO nell'aria circostante: questa variazione di resistenza viene, poi, convertita in un segnale elettrico ed inviata al microcontrollore.

Sensore di Anidride Carbonica MQ-135
Il sensore MQ-135 è progettato per rilevare la presenza di una vasta gamma di gas nocivi; perciò e grazie alla sua alta sensibilità, è comunemente utilizzato in applicazioni di monitoraggio della qualità dell'aria.

In questo progetto, il sensore è stato utilizzato per ottenere la capacità di anidride carbonica (CO_2) nell'aria, la cui misurazione avviene nel seguente modo: il principio di funzionamento del sensore MQ-135 si basa sulla variazione della resistenza elettrica del materiale sensibile, ovvero diossido di stagno (SnO_2), in presenza del gas target, in questo caso anidride carbonica.

Quando quest'ultima viene assorbita dal materiale, la resistenza elettrica cambia; questa variazione di resistenza è proporzionale alla concentrazione del gas nell'ambiente. Il sensore richiede, inizialmente, un periodo di riscaldamento per raggiungere una temperatura operativa stabile; dopodiché emette un segnale elettrico analogico che può essere letto dal microcontrollore per determinare la concentrazione di anidride carbonica.

Pulse Width Modulation
Il *Pulse Width Modulation* (PWM), in italiano modulazione a larghezza di impulso, è una tecnica per emulare un segnale di uscita analogico utilizzando pin digitali, generando un'onda quadra periodica che alterna stati HIGH e LOW: l'energia del segnale viene distribuita attraverso una serie di impulsi anziché attraverso un segnale a variazione continua.

La tensione di uscita emulata è determinata dal *duty cycle*, ciclo di lavoro, del segnale, che rappresenta la percentuale di tempo in cui il segnale è HIGH rispetto al tempo totale. Il valore del *duty cycle* può essere calcolato con la seguente formula:

$$\text{Duty Cycle} = \frac{\text{Turn On Time}}{\text{Turn On Time} + \text{Turn Off Time}} \tag{4.1}$$

La frequenza del PWM determina quanto velocemente viene completato un periodo, ovvero tempo in cui il segnale è HIGH più tempo in cui il segnale è LOW, e si calcola come segue:

$$\text{Frequency} = \frac{1}{\text{Time Period}} \tag{4.2}$$

$$\text{Time Period} = \text{Turn On Time} + \text{Turn Off Time} \tag{4.3}$$

Il valore medio della tensione è uguale al ciclo di lavoro moltiplicato per la tensione di alimentazione (VCC). Ad esempio, un ciclo di lavoro del 50% su una VCC di 5V produce una tensione media di 2.5V.

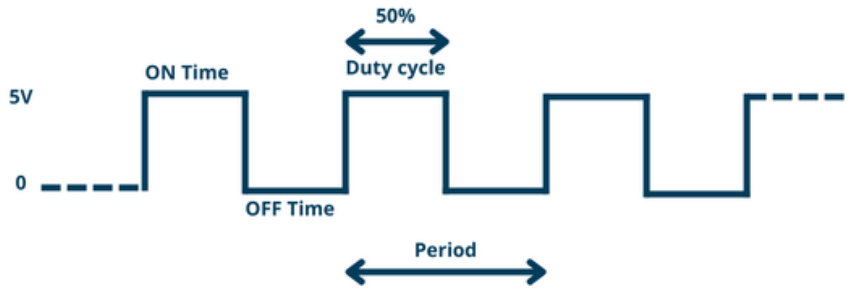

Figura 4.6: Rappresentazione grafica del PWM

In questo progetto, la tecnica del *Pulse Width Modulation* viene usata per controllare la velocità della ventola, che rappresenta il funzionamento del purificatore. In particolare, ciò viene effettuato nella classe `FanController` attraverso il metodo `analogWrite()`.

MQTT

MQTT è un protocollo per lo scambio di messaggi, progettato per l'Internet of Things. Si basa su un modello publish/subscribe, è molto leggero ed è pensato per connettere dispositivi remoti, anche se essi dispongono di risorse limitate; gli headers dei messaggi MQTT hanno dimensione limitata e ciò rende questo protocollo ideale anche per reti con larghezza di banda ridotta.

MQTT consente una comunicazione efficiente e scalabile, particolarmente adatta per applicazioni IoT, dove i dispositivi devono scambiare piccoli volumi di dati in modo affidabile.

La comunicazione in MQTT funziona attraverso un broker centrale che gestisce la distribuzione delle informazioni tra client; quest'ultimi possono pubblicare messaggi su un determinato topic e iscriversi a topic di interesse per ricevere aggiornamenti.

I client MQTT richiedono l'impiego di poche risorse perciò possono essere facilmente usati anche nei microcontrollori.

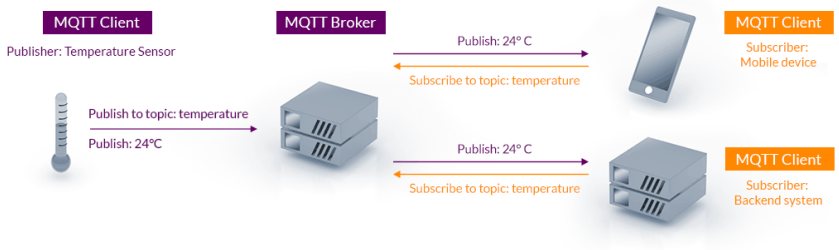

Figura 4.7: Schema di esempio del funzionamento di MQTT[11]

In questo progetto, MQTT è usato per la comunicazione tra modulo embedded e modulo back-end. La scelta dell'utilizzo di questo protocollo è dovuta alle caratteristiche che presenta:

- **Leggerezza**: riduce l'overhead di comunicazione, fondamentale per dispositivi con risorse limitate;
- **Efficienza energetica**: la sua architettura publish/subscribe minimizza il consumo energetico, poiché i dispositivi inviano dati solo quando necessario;
- **Affidabilità**: offre livelli di qualità del servizio (QoS - Quality of Service) che garantiscono la consegna affidabile dei messaggi, essenziale per lo scambio di dati critici relativi al monitoraggio dell'aria;
- **Semplicità di implementazione**: è facile da implementare e integrare con altre tecnologie e piattaforme IoT.

4.3.3 Librerie Utilizzate

Per l'implementazione del modulo embedded si è fatto uso di alcune librerie volte ad agevolare l'interfacciamento con i sensori presenti nel progetto e a facilitare la connessione dell'ESP32 ad MQTT. Di seguito vengono citate, fornendo alcuni esempi di dove siano state utilizzate nel codice.

PubSubClient
PubSubClient[4] è una libreria che permette di creare un client attraverso il quale connettersi ad un broker MQTT per effettuare operazioni di subscribe e publish su un topic MQTT.

Nella classe del modulo embedded dedicata alla gestione della comunicazione MQTT, `MQTTConnection`, i metodi di questa libreria sono stati ampiamente sfruttati.

DHT-sensor-library
DHT-sensor-library[5] è una libreria curata da *Adafruit*, una società che produce e distribuisce componenti elettronici. La libreria offre diverse funzioni per la gestione dei sensori della serie DHT (ovvero *Digital Humidity and Temperature*).

Per raccogliere i dati relativi a temperatura e umidità sono stati utilizzati, in particolare, i metodi `readTemperature()` e `readHumidity()`.

ArduinoJson
ArduinoJson[6] è una libreria che offre metodi per serializzare e deserializzare oggetti JSON, progettata per Arduino ma compatibile anche con un ESP32.

Nel codice, i metodi di questa libreria sono stati utilizzati per codificare in formato JSON i valori di umidità, temperatura, monossido di carbonio e anidride carbonica, in modo da trasferirli al modulo back-end via MQTT.

MQUnifiedsensor
MQUnifiedsensor[7] è una libreria progetta per interfacciarsi con sensori della famiglia MQ e leggere i valori misurati da essi.

In questo progetto, è stata usata con il sensore MQ-7 ed il sensore MQ-135. Nello specifico, è stato inizialmente usato un metodo per calibrare i sensori e, a seguire, una serie di altri metodi per raccogliere le misurazioni da essi.

Essendo che sia il sensore MQ-7 che il sensore MQ-135 sono sensibili a diversi tipi di gas, per ottenere le misurazioni specifiche relative a monossido di carbonio ed anidride carbonica è opportuno settare il corretto valore dei coefficienti A e B, ovvero i coefficienti della curva di calibrazione dei sensori.

Questi coefficienti definiscono la relazione tra la tensione di uscita del sensore e la concentrazione del gas. La curva di calibrazione è tipicamente

[4] `https://github.com/knolleary/pubsubclient`
[5] `https://github.com/adafruit/DHT-sensor-library`
[6] `https://github.com/bblanchon/ArduinoJson`
[7] `https://www.arduinolibraries.info/libraries/mq-unifiedsensor`

un'equazione di potenza del tipo:

$$y = A(x^B) \tag{4.4}$$

dove A è il coefficiente impostato da `setA()`, B è il coefficiente impostato da `setB()`, x è la tensione di uscita del sensore e y è la concentrazione del gas che si vuole misurare.

Capitolo 5

Discussione dei Risultati e Considerazioni Finali

Terminata la fase di sviluppo è importante riservare un capitolo per discutere i risultati ottenuti e per esporre alcune considerazioni finali relative al progetto completo.

Partendo dai requisiti individuati nella prima fase, ovvero la fase di analisi, è fondamentale valutare quali di questi siano stati soddisfatti pienamente e quali, al contrario, non siano stati raggiunti; questo per permettere di comprendere se gli obiettivi iniziali sono stati rispettati e quali aspetti, invece, necessitano di ulteriori miglioramenti.

I casi d'uso presentati in precedenza (pagina 6) possono essere sfruttati come scenari per il testing del software. Questo approccio consente di verificare il funzionamento del sistema in situazioni reali e di individuare eventuali criticità. Oltre a ciò, il testing basato sui casi d'uso è particolarmente utile per evidenziare problemi che potrebbero non essere stati previsti durante lo sviluppo e per individuare funzionalità aggiuntive che potrebbero essere incluse in un futuro ampliamento del progetto.

5.1 Valutazione Critica dei Requisiti Soddisfatti e non Soddisfatti

In primo luogo, sono stati soddisfatti i requisiti funzionali di monitoraggio dei dati raccolti, controllo del funzionamento a distanza, individuazione e conseguente avviso di situazioni anomale (si rimanda a pagina 4 per l'elenco completo).

La scelta dei parametri da analizzare e conseguentemente dei sensori da utilizzare si è rivelata efficace per fornire un quadro completo relativo alla qualità dell'aria.

In più, è stata garantita la creazione di una connessione stabile con il dispositivo, che assicura un'interazione fluida e continua fra utente e sistema.

Il protocollo MQTT ha giocato un ruolo cruciale in questo: le sue funzionalità sono state sfruttate al meglio per ottenere uno scambio di messaggi efficace.

Un altro importante risultato è stato il raggiungimento del requisito relativo alla progettazione di un'interfaccia utente intuitiva: *Smart-Air-Purifier* può essere usato con facilità anche da utenti non esperti o con disabilità di varia natura.

Tuttavia, emergono alcune criticità su cui porre attenzione, perché potrebbero rivelarsi una buona base per migliorare ulteriormente il progetto.

Sebbene il sistema presenti molte delle funzionalità previste, la sicurezza potrebbe essere irrobustita, per garantire una protezione maggiore.

Inoltre, non è stato soddisfatto il requisito relativo alla raccolta dei dati per elaborazioni statistiche, a causa della mancata creazione di un database; ciò limita la capacità di analizzare le misurazioni in modo approfondito e di estrarre da esse ulteriori informazioni (ad esempio, nel caso del monossido di carbonio, si potrebbe individuare in quali momenti della giornata la concentrazione è più alta, per indagare quale sia l'effettiva causa di formazione del gas).

Nonostante il progetto abbia soddisfatto molti dei requisiti fondamentali e offra una base solida per il funzionamento previsto, vi sono aree specifiche che necessitano di ulteriori accorgimenti per completare pienamente tutti gli obiettivi prefissati.

È stata, però, progettata una solida e scalabile architettura del sistema, che permette di poter apportare modifiche e aggiungere nuove funzionalità in maniera agevole.

5.2 Possibili Miglioramenti e Sviluppi Futuri

Ci si vuole ora concentrare sui possibili ambiti in cui *Smart-Air-Purifier* può essere ulteriormente migliorato.

Come accennato nel precedente paragrafo, una delle principali aree di intervento riguarda il rafforzamento della sicurezza, per garantire una maggiore protezione delle informazioni veicolate.

Sempre facendo riferimento agli obiettivi non ancora raggiunti, sarebbe ottimale l'effettiva implementazione di un database dedicato, per permettere di

archiviare in modo strutturato i dati raccolti, facilitando l'analisi e l'estrazione di informazioni utili. Ciò potrebbe condurre all'integrazione di funzionalità interessanti, come ad esempio l'analisi predittiva, ovvero la formulazione di previsioni attraverso algoritmi di *machine learning*.

Per quanto riguarda l'interfaccia utente, si potrebbe considerare di ampliare le funzionalità messe a disposizione, introducendo strumenti di visualizzazione avanzati, come ad esempio report personalizzati che riguardino i cambiamenti della qualità dell'aria nell'arco della giornata, per fornire una comprensione più approfondita e permettere di individuare soluzioni mirate, volte a migliorare il comfort dell'ambiente domestico.

A livello hardware, sono due i possibili miglioramenti che sarebbe interessante mettere in atto: da una parte, si potrebbe valutare di integrare altri tipi di sensori, per ottenere dati relativi a differenti gas presenti nell'aria domestica. Dall'altra, con ulteriori risorse, si potrebbe costruire un prototipo più completo del purificatore, che non abbia solo una ventola a scopo dimostrativo ma anche dei filtri che realizzino l'effettiva azione purificante.

Un'altra direzione interessante per lo sviluppo futuro potrebbe essere il deployment in cloud dell'applicazione; considerando che quello realizzato è un software prototipale, attualmente risiede in locale ma, per renderlo un'applicazione *real-world*, sarebbe opportuno valutare i possibili strumenti per la messa in esercizio di *Smart-Air-Purifier*.

Conclusioni

Smart-Air-Purifier è un progetto che, come obiettivo principale, ha quello di sensibilizzare le persone sull'importanza della qualità dell'aria e la necessità di nuovi mezzi e soluzioni innovative per monitorarla e migliorarla. Arrivati alla conclusione del progetto, si può affermare che esso riesce nel suo intento, facendo un passo importante verso l'obiettivo di rendere i nostri ambienti interni più sicuri e salubri.

Le tecnologie e i protocolli usati sono un punto di riferimento nello sviluppo di applicazioni in ottica Internet of Things e permettono di considerare un'eventuale integrazione del progetto in sistemi più ampi e complessi; il purificatore creato potrebbe essere integrato in smart-home dotate di altri dispositivi intelligenti e scambiare informazioni con essi, dando il suo importante contributo nella transizione tecnologica delle nostre abitazioni.

Il lavoro svolto in questa tesi evidenzia che il purificatore d'aria intelligente ha un impatto significativo su diversi livelli.

In primo luogo, la capacità di monitorare e purificare l'aria in tempo reale affronta direttamente i problemi dell'inquinamento *indoor*, offrendo un ambiente più sano e sicuro nei luoghi in cui il dispositivo viene posizionato, come ad esempio nelle abitazioni e nei luoghi di lavoro.

Questo è particolarmente rilevante in aree ad alta densità abitativa, come la Pianura Padana, dove l'inquinamento è un problema quotidiano e la qualità dell'aria spesso scende a livelli pericolosi.

In secondo luogo, il progetto ha il potenziale di sensibilizzare maggiormente l'opinione pubblica sull'importanza di una buona qualità dell'aria. Attraverso l'uso di tecnologie di monitoraggio avanzate e un'interfaccia utente accessibile, gli utenti possono essere informati in tempo reale sulle condizioni dell'aria che respirano, rendendoli più consapevoli dei rischi e delle misure che possono adottare per mitigare l'inquinamento.

Inoltre, l'approccio modulare e scalabile del sistema sviluppato permette di estendere con facilità le sue funzionalità, aprendo la possibilità ad integrazioni e miglioramenti futuri.

In conclusione, il purificatore d'aria intelligente progettato in questa tesi non è solo un prototipo base per sviluppi futuri, ma un elemento chiave nella

lotta contro l'inquinamento dell'aria *indoor*. Esso rappresenta un passo avanti verso un futuro in cui la qualità dell'aria possa essere monitorata e migliorata costantemente, proteggendo così la salute delle persone e migliorando la qualità della vita. Lo sviluppo di questo progetto evidenzia l'importanza di continuare a investire in soluzioni innovative e sostenibili per affrontare le sfide ambientali del nostro tempo.

Ringraziamenti

Arrivata alla conclusione di questo percorso, vorrei ringraziare la mia famiglia per l'affetto e per il sostegno costante.

Vorrei ringraziare tutti gli amici conosciuti durante questo percorso, che mi hanno fatto sorridere anche nei momenti più difficili.

Infine, vorrei riservare un ringraziamento speciale al mio relatore, il professor Alessandro Ricci, per avermi seguita durante lo sviluppo di questo progetto.

Bibliografia

[1] RaiNews.it. Smog, i dati shock della pianura padana. in lombardia polveri sottili ancora sopra il limite. *Sito online di RaiNews*, 2024. Disponibile online all'indirizzo www.rainews.it.

[2] Fiona Kuan. Cos'è un dispositivo intelligente? i gadget connessi che plasmano le nostre vite. *MOKOSmart*, 2017. Disponibile online all'indirizzo: `https://www.mokosmart.com/it/what-is-a-smart-device/`.

[3] Ministero della Salute. Umidità e muffe. Opuscolo, Ministero della Salute, 2015. Disponibile online all'indirizzo: `https://www.salute.gov.it/imgs/C_17_opuscoliPoster_283_ulterioriallegati_ulteriореallegato_14_alleg.pdf`.

[4] Ministero della Salute. Monossido di carbonio. Opuscolo, Ministero della Salute, 2015. Disponibile online all'indirizzo: `https://www.salute.gov.it/imgs/C_17_opuscoliPoster_283_ulterioriallegati_ulteriореallegato_2_alleg.pdf`.

[5] IEEE Computer Society. Ieee recommended practice for architectural description of software-intensive systems. IEEE Std 1471-2000, 2000. Disponibile online all'indirizzo: `https://standards.ieee.org/ieee/1471/2187/`.

[6] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Inc. Disponibile online all'indirizzo: `https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html`.

[7] Descrizione presa dall'introduzione al tool-kit Vert.x, disponibile all'indirizzo `https://vertx.io/introduction-to-vertx-and-reactive/`.

[8] Descrizione presa dal sito ufficiale `https://platformio.org`.

[9] Immagine presa da ESP-IDF Programming Guide - ESP32-S3-DevKitC-1 v1.1, disponibile online all'indirizzo: `https:`

//docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html#id5.

[10] Tutte le informazioni citate, relative al sensore, provengono dal datasheet fornito dal produttore ed eventualmente consultabile al seguente link https://cdn.shopify.com/s/files/1/1509/1638/files/DHT_11_Temperatursensor_Modul_Datenblatt_a59ef62a-ee56-4c72-918f-00cb97f71f64.pdf?16953870400002276923.

[11] Immagine presa dal sito ufficiale di MQTT, https://mqtt.org.