

UNIVERSITÀ TELEMATICA INTERNAZIONALE UNINETTUNO

FACOLTÀ DI INGEGNERIA

Corso di Laurea in

Ingegneria Informatica

ELABORATO FINALE

in

Ingegneria del software e programmazione ad oggetti

Monitoraggio di saturazione sanguigna e frequenza cardiaca, in ambito di Telemedicina, attraverso dispositivi IoT

RELATORE

Prof. Patrizia Grifoni

CANDIDATO

Francesco Fabio

CORRELATORE

Prof. Mauro Mazzei

ANNO ACCADEMICO 2024/25

Desidero esprimere la mia sincera gratitudine a tutte le persone che hanno reso possibile il raggiungimento di questo importante traguardo.

Ringrazio i docenti, che con pazienza e professionalità mi hanno permesso durante tutto il percorso di ampliare la mia conoscenza e il mio bagaglio culturale.

Ringrazio la mia famiglia, per il sostegno costante, la pazienza e l'incoraggiamento in ogni momento di questo cammino.

Infine, ringrazio gli amici e i colleghi di corso che hanno condiviso con me questi anni di studio, tra fatiche e soddisfazioni.

Indice

1	Introduzione.....	1
1.1	Telemedicina	2
1.1.1	Telemedicina nel mondo	3
1.1.2	Telemedicina in Italia	3
1.2	Web Application.....	4
1.3	Microservizi.....	5
1.4	Protocollo http	6
1.5	WebSocket	7
1.6	Client-Server	7
1.7	Internet of Things	8
1.8	Internet of Medical Things	8
2	Panoramica Software e Framework.....	9
2.1	Thonny IDE.....	9
2.2	MicroPython.....	10
2.3	Node JS	10
2.4	MySQL.....	12
2.5	Chart JS	12
2.6	GitHub.....	13
3	Caso di Studio	15
3.1	Scenario d'uso	15
3.1.1	Caratteristiche del progetto.....	15
3.1.2	Focus del progetto	16
3.2	Panoramica del progetto.....	16
3.2.1	Struttura del progetto.....	17
4	Client Hardware	19
4.1	Schema elettrico	19
4.1.1	Alimentazione	20
4.1.2	MAX30102.....	20
4.2	ESP32.....	22
4.2.1	Principali caratteristiche	22
4.3	Firmware	23
4.3.1	Funzionalità.....	25
4.3.2	Connessione al WiFi	25
4.3.3	Connessione al WebSocket	26
4.3.4	Lettura dei dati del sensore.....	27
4.3.5	Ricezione dei messaggi	28

4.3.6	Invio dei messaggi	29
4.3.7	Creazione e avvio delle coroutine	29
5	Server Node JS	30
5.1	Moduli principali utilizzati	30
5.1.1	WS	30
5.1.2	Express	30
5.1.3	MySQL2.....	31
5.2	Codice.....	31
5.2.1	Servizio 1.....	31
5.2.2	Gestione della comunicazione	31
5.2.3	Gestione della registrazione	34
5.2.4	Inserimento dei dati nel database	36
5.2.5	Servizio 2.....	39
5.2.6	Server.....	40
5.2.7	Routers.....	40
5.2.8	Interrogazione del database	41
6	Database MySQL	44
6.1	Tool utilizzato nel progetto	44
6.2	Creazione utente	44
6.3	Creazione del database	47
6.4	Creazione della tabella	47
6.5	Visualizzazione dei dati.....	48
7	Front-End	50
7.1	Struttura del Front-End.....	50
7.2	Interfaccia Principale.....	51
7.3	Comunicazione con il server	52
7.4	Visualizzazione dei dati.....	54
7.5	Recupero dei dati dal database	56
8	Conclusione e sviluppi futuri	60
9	Bibliografia.....	62

Sommario

In poco meno di un secolo lo sviluppo di discipline come l'informatica, l'elettronica e le telecomunicazioni hanno iniziato a rivestire un ruolo sempre più importante nella società moderna. A beneficiare di questi progressi sono praticamente tutti i settori e non solo quelli più direttamente coinvolti nel mondo ingegneristico. Tra i principali è possibile trovare il settore della medicina.

Il monitoraggio di alcuni parametri vitali come la saturazione di ossigeno nel sangue, la frequenza cardiaca e la pressione sanguigna è ormai alla portata di tutti grazie ai numerosi strumenti, anche a basso costo, facilmente reperibili sul mercato.

Il progresso nel campo delle telecomunicazioni, la miniaturizzazione e l'efficientamento dei circuiti elettronici hanno permesso il collegamento in rete di dispositivi di uso comune anche di piccole dimensioni. L'**IoT** (Internet of Things) o Internet delle Cose, è un termine utilizzato per riferirsi ad oggetti connessi in rete che comunicano tra loro.

Il monitoraggio dei parametri vitali si è enormemente diffuso grazie alle **tecnologie wearable**, ovvero, grazie a particolari sensori incorporati all'interno di strumenti di utilizzo quotidiano come orologi, bracciali e anelli. Oggetti indossabili che rendono possibile il monitoraggio dei parametri vitali durante tutta la giornata e in base al tipo di attività svolta (rilevamento di attività sportiva, misurazione della qualità del sonno, ecc...).

Altro importantissimo traguardo che la tecnologia ha permesso di raggiungere è quello della **telemedicina** ovvero quella branca della medicina che permette l'interazione da remoto tra pazienti e medici. Attività come la gestione remota di visite, la condivisione di documenti e il monitoraggio dei parametri vitali permettono di alleggerire il carico sulle strutture ospedaliere consentendo ai pazienti di ricevere le cure necessarie da casa restando sempre in contatto con gli operatori sanitari.

Il progetto prevede la realizzazione di un dispositivo hardware e di una **web app**. Il dispositivo hardware è in grado di rilevare saturazione del sangue e frequenza cardiaca. Mentre la web app, formata da **back-end** e **front-end**, gestisce la comunicazione con il **microcontrollore**, la visualizzazione dei dati in tempo reale e la registrazione e visualizzazione dello storico delle misurazioni.

Il seguente elaborato ha lo scopo di analizzare nel dettaglio gli aspetti pratici e teorici del progetto oltre a dare un supporto per la comprensione e riproduzione dello stesso.

Nel secondo capitolo vengono trattate le tecnologie software utilizzate per la realizzazione del progetto.

Thonny IDE è un ambiente di sviluppo integrato (IDE) open source e multiplatforma realizzato per la programmazione in codice Python ma anche per il codice MicroPython. La semplicità di utilizzo e la compatibilità con MicroPython lo rendono ideale per programmare microcontrollori.

MicroPython è un compilatore e runtime Python open source che contiene un sottoinsieme delle librerie di Python3 ed è pensato e ottimizzato per essere eseguito su microcontrollori.

NodeJS è un runtime JavaScript molto popolare che sfrutta il motore V8 di Chrome permettendo di eseguire codice JavaScript al di fuori del browser. È stato usato nel progetto per realizzare il back-end che si divide in due servizi separati. Il primo servizio gestisce la comunicazione tra front-end e microcontrollore. Il secondo servizio si occupa di esporre REST API per far interagire il front-end con il database.

MySQL è un noto DBMS (Database Management System) di tipo relazionale open source e multiplatforma utilizzato in questo progetto per memorizzare i dati provenienti dal sensore in modo da renderli disponibili al front-end per costruire i grafici.

ChartJS è una libreria JavaScript open source per la visualizzazione dei dati in modo semplice e pratico. È stata utilizzata nel progetto per la visualizzazione dei dati salvati nel database come la saturazione, la frequenza cardiaca e un riferimento temporale che indica il momento di acquisizione dei dati.

GitHub è tra le piattaforme più note che permettono agli sviluppatori di creare, salvare, gestire e condividere il codice. È compatibile con il software di versioning Git che permette il tracciamento delle modifiche del codice.

Nel terzo capitolo è stato affrontato l'aspetto progettuale descrivendo il caso d'uso e le specifiche di progetto. Il sistema è stato basato su un'architettura a microservizi dove sono presenti due servizi principali, uno che permette la comunicazione tra il microcontrollore e il front-end e l'altro che permette di accedere ai dati memorizzati sul database. Il progetto sposa l'approccio open source e l'intero codice è stato reso pubblico su GitHub in modo che chiunque voglia replicare o migliorare il progetto sia libero di farlo.

Gli scopi principali del progetto sono quelli di approfondire un argomento importante come quello della tecnologia applicata al benessere, mettendo a disposizione un dispositivo

economico (anche se di fatto non costituisce o sostituisce un'apparecchiatura medica a tutti gli effetti) che accoppiato con un comparto software intuitivo permettono all'utente di effettuare quello che è un monitoraggio di alcuni parametri vitali.

Nella restante parte dell'elaborato vengono analizzate delle porzioni significative di codice utili a descrivere il funzionamento del progetto nelle sue componenti principali che sono:

Componente hardware, formata dal microcontrollore e dal sensore in grado di rilevare i parametri dall'utente. Il microcontrollore si occupa di leggere e inviare i dati al server sotto forma di JSON quando richiesto.

Componente software, costituita da una parte server formata da due servizi principali. Il primo che si occupa di curare la comunicazione tra front-end e microcontrollore mettendo i due in comunicazione tramite un WebSocket e registrando (quando richiesto) il flusso dei dati su un database. E il secondo che si occupa di esporre delle API che permettono al front-end di recuperare i dati memorizzati nel database in modo da poter costruire i grafici per la visualizzazione dei dati stessi.

Il sistema realizzato comprende delle funzioni di base che possono essere estese ed arricchite in futuro ad esempio con la gestione dell'autenticazione per suddividere gli utenti in pazienti e medici e permettere (con la dovuta gestione delle autorizzazioni) ai medici di consultare i dati dei loro pazienti. È anche possibile implementare la registrazione e la gestione di più dispositivi per paziente.

1 Introduzione

Negli ultimi anni lo sviluppo tecnologico ha portato a modificare drasticamente il modo in cui la totalità dei settori opera. Questo anche grazie ad una diffusione delle reti (linee satellitari, 5G e fibra) che permette di essere sempre connessi e di scambiare una grossa mole di dati. Come riportato in [1], questa rivoluzione coinvolge ogni settore. Difatti anche i settori che sono tradizionalmente associati ad un lavoro che richiede l'interazione fisica di più parti sono stati coinvolti in questa trasformazione. Pensiamo alla rivoluzione introdotta dall'e-commerce nel commercio, o all'agricoltura e di come sia possibile il controllo da remoto dell'irrigazione dei campi o della temperatura e umidità nelle serre. Anche nel campo della medicina sta prendendo piede un approccio che permette di ridurre la presenza fisica del paziente nelle strutture ospedaliere.

A tale proposito il Ministero della Salute ha stanziato parte dei fondi del PNRR in ammodernamento tecnologico in ambito sanitario destinando i fondi ad assistenza domiciliare, centrali operative territoriali e telemedicina.

Questo balzo tecnologico ha reso più economico, e quindi alla portata di tutti, quei dispositivi connessi che rientrano nell'ambito dell'IoT. Con menzione particolare a tutte quelle tecnologie cosiddette wearable, ovvero dispositivi connessi che sono indossabili. Dispositivi economici di questo tipo, come lo può essere uno smartwatch, sono dotati di una sensoristica avanzata (accelerometro, barometro, sensore per la rilevazione della saturazione di ossigeno nel sangue e della frequenza cardiaca) e di enormi capacità di connessione disponendo di Bluetooth, WiFi o addirittura rete LTE. Questo gli permette di restare sempre connessi con altri device e alla rete internet. Questo approccio, come descritto in [2], può essere schematizzato in tre livelli. Il livello dei trasduttori, il livello gateway e il livello agent. Dove i dati vengono prima raccolti da un microcontrollore (livello dei trasduttori), inviati ad uno smartphone o router (livello gateway) e da qui raggiungono internet dove possono essere usati anche da dispositivi connessi dall'altro lato del mondo (livello agent).

L'IoT è entrato anche nel campo della medicina contribuendo ad una nuova branca che prende il nome di Telemedicina. Dispositivi wearable per il tracciamento dell'attività fisica possono essere utili in fase di prevenzione di diverse malattie (come quelle cardiovascolari) oltre che per il monitoraggio del peso corporeo o della qualità del sonno. Mentre in ambito più professionale troviamo un'ampia gamma di dispositivi medici che, installati a casa del paziente,

permettono un monitoraggio più accurato fornendo i dati direttamente alla struttura medica di riferimento.

Lo scopo del progetto è quello di mostrare un esempio di applicazione dell'IoT inerente alla telemedicina.

Il progetto prevede la realizzazione di un'infrastruttura che comprende un dispositivo per la raccolta e la trasmissione dei parametri vitali dell'utente e una piattaforma in cui i dati vengono raccolti, memorizzati e visualizzati.

Tramite il front-end, una volta che il dispositivo è collegato alla rete, è possibile avviare la raccolta e la trasmissione dei dati relativi a SpO2 e HR. La connessione tra microcontrollore e server (realizzato in tecnologia NodeJS) avviene tramite un WebSocket che consente un flusso continuo di dati.

La comunicazione tra front-end e microcontrollore avviene tramite scambio di messaggi JSON che passano per il server. È possibile tramite interfaccia grafica avviare la registrazione del flusso in tempo reale dei dati. Il server si occuperà di scrivere i dati su un file CSV che al termine della registrazione sarà caricato su un database MySQL.

Un servizio separato si occupa di esporre delle REST API per permettere l'estrazione dei dati dal database da parte del front-end in modo poi da visualizzarli sotto forma di grafici.

Per la parte hardware è stato utilizzato un ESP32 con MicroPython che controlla un sensore MAX30102.

Per la parte front-end sono stati utilizzati HTML, CSS e JavaScript puri senza l'uso di framework particolari. Per la visualizzazione grafica dei dati presenti nel database è stata utilizzata la libreria ChartJS.

1.1 Telemedicina

Oltre ad essere tra i temi principali di questo elaborato, la telemedicina è un argomento molto discusso in ambito medico per via delle potenziali implicazioni sulla vita delle persone. Non a caso gli investimenti in questo settore stanno aumentando.

Per dare una prima definizione di Telemedicina è possibile riportare quanto scritto sul sito del Ministero della Salute [3]:

“Con il termine telemedicina si indica tutto l’insieme di prestazioni sanitarie in cui, grazie all’utilizzo di tecnologie innovative, il professionista della salute e il paziente non si trovano nello stesso luogo”.

Sempre per il Ministero della Salute [3] la telemedicina deve consentire di:

- assistere e visitare un paziente a distanza;
- monitorare a distanza i parametri vitali di un paziente;
- mettere in comunicazione tra loro gli operatori sanitari in caso di necessità particolari;
- inviare e conservare documenti clinici.

1.1.1 Telemedicina nel mondo

Stando ai dati dell’OMS riportati in [4] l’adozione della teleassistenza (sotto forma di teleradiologia, telepsichiatria e telemedicina) è cresciuta notevolmente specialmente durante e dopo la pandemia di COVID-19. 40 nazioni europee aderenti all’OMS hanno adottato strategie di teleassistenza dirette o le hanno incluse in riforme sanitarie più ampie.

Tra gli esempi più virtuosi troviamo la Norvegia dove si fa ampio uso di teleradiologia (pratica tramite cui è possibile trasmettere le immagini radiologiche e richiedere ad esempio una seconda opinione), ma anche di telepsichiatria e telemedicina. In particolare dal 2023 sono state regolamentate le visite da remoto per ottenere ad esempio i certificati di malattia in ambito lavorativo.

La telemedicina è anche considerata una soluzione in supporto dei Paesi più svantaggiati dove aree remote possono essere supportate tramite servizi medici di base online [5]. Ma anche per zone con bassa densità di operatori sanitari per numero di abitanti.

Benché la telemedicina non possa sostituire in toto le visite e le prestazioni in presenza, costituisce un supporto in grado di ridurre i tempi di attesa, accorciare le distanze tra medico e paziente e alleggerire il carico sulle strutture sanitarie.

1.1.2 Telemedicina in Italia

In Italia una buona parte dei fondi del PNRR sono stati destinati alla sanità tramite la Missione Salute [6], fondi che ammontano ad un totale di € 15,63 miliardi. Di questi, tramite il PNRR/M6 [7] (Missione 6 Salute), 1,5 miliardi sono destinati alla telemedicina. La persona all’interno della sua comunità di riferimento viene messa al centro attraverso la filosofia “Casa come primo luogo di cura e telemedicina”.

Il governo si è posto tra i traguardi finali quelli (Figura 1.1) di:

- assistere almeno 300 mila persone attraverso strumenti di telemedicina entro la fine del 2025;
- almeno 842 mila persone over 65 trattate in assistenza domiciliare entro la fine del 2026.

I servizi minimi di telemedicina previsti nel PNRR sul territorio nazionale sono:

1. Televisita;
2. Teleconsulto;
3. Teleassistenza;
4. Telemonitoraggio.

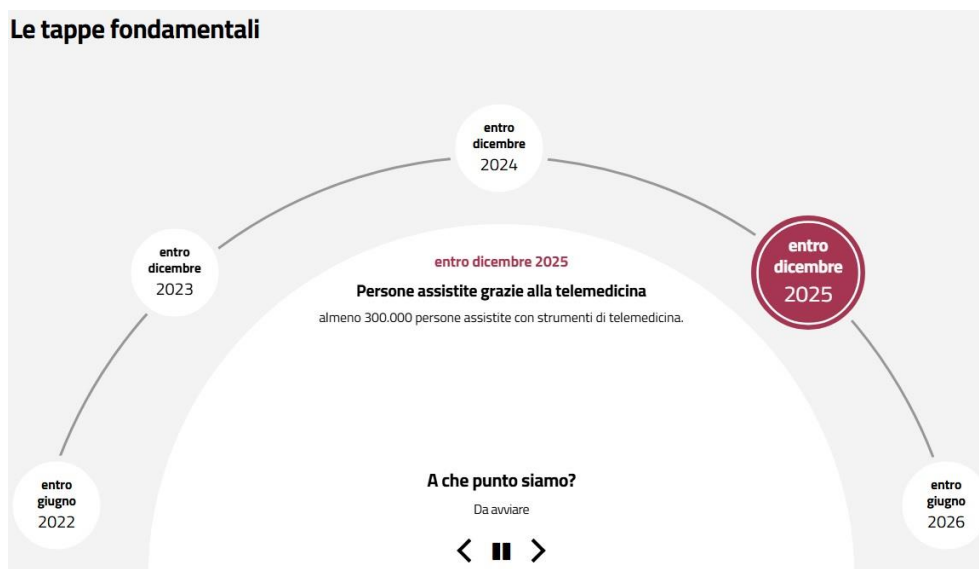


Figura 1.1: Schema riassuntivo presente nella home del portale "Italiadomani"

1.2 Web Application

Allo scopo pratico di trattare l'argomento della telemedicina e dell'IoT una parte dell'elaborato ruota attorno alla parte pratica e quindi la descrizione della web application per la gestione di un saturimetro smart.

Una web-app è una struttura software dinamica basata sui protocolli web e quindi lo stack TCP/IP. Sfruttano lato client l'utilizzo dei browser rendendo quindi queste applicazioni fruibili su una vasta gamma di dispositivi eterogenei.

Le web application moderne possono avere diverse architetture ma in generale è possibile notare un pattern formato da 3 componenti principali, ovvero come menzionato in [8]:

- Logica di presentazione: costituisce il front-end ovvero la parte grafica lato utente che funge da interfaccia permettendo la comunicazione tra l'utente finale e il server. Utilizza tipicamente codice scritto in HTML, CSS e JavaScript;
- Logica di business: rappresenta il back-end, il cuore dell'applicazione ovvero la parte di software che raccoglie, elabora e soddisfa le richieste del client. Può essere scritta con diversi linguaggi lato server come PHP, Java, NodeJS, ecc...;
- Strato dati: livello opzionale che racchiude tipicamente un database e quindi funge da luogo in cui vengono memorizzati i dati a lungo termine per poi essere recuperati all'occorrenza.

1.3 Microservizi

Esistono diversi approcci alle architetture software, tra i più diffusi troviamo l'architettura a microservizi.

L'approccio a microservizi consiste nel dividere la logica business dell'applicazione in più servizi indipendenti di dimensioni ridotte e che comunicano tra loro (Figura 1.2).

Tra i vantaggi principali abbiamo che:

- eventuali aggiornamenti o modifiche sono locali allo specifico servizio e non influenzano direttamente gli altri servizi;
- Le versioni e lo stack tecnologico sono indipendenti tra loro;
- maggiore flessibilità nella scalabilità in quanto è possibile allocare maggiori o minori risorse ai singoli servizi sulla base del loro carico.

La comunicazione tra microservizi può avvenire in diversi modi tra cui ad esempio delle API.

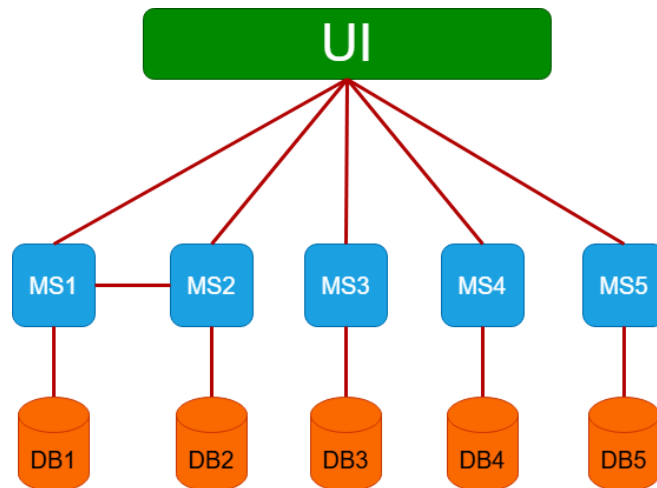


Figura 1.2: Struttura di un'architettura a microservizi

1.4 Protocollo http

HTTP (Hyper Text Transfer Protocol) è un protocollo unidirezionale in cui un client invia una richiesta (request) ad un server e il server invia una risposta (response) [9]. La connessione viene chiusa in seguito all'invio della risposta.

Una richiesta HTTP è formata da:

method URI version
headers
body

Dove l'URI (Univoque Resource Identifier) identifica univocamente la risorsa.

Version identifica la versione del protocollo HTTP utilizzata.

Headers permette di passare informazioni aggiuntive tra client e server per coordinare la comunicazione come ad esempio nome dell'host, user-agent e il content-type.

Il body rappresenta il corpo della richiesta HTTP e quindi conterrà le informazioni.

Method permette di specificare il tipo di richiesta HTTP. Esistono diversi metodi, quelli di maggior rilievo sono:

- GET: usato per ottenere dati dal server;
- POST: usato per inviare dati al server (ad esempio dati da inserire nel database);
- PATCH: usato per aggiornare dei dati sul server;
- DELETE: utilizzato per eliminare dei dati dal server.

1.5 WebSocket

Come descritto in [10] WebSocket è un protocollo che fornisce un canale di comunicazione bidirezionale (full-duplex) appoggiandosi al protocollo TCP. È supportato dai principali browser (e anche lato server) dal 2010.

Differentemente dal protocollo HTTP dove alla richiesta del client viene aperto un canale di comunicazione che viene chiuso dopo che il server ha inviato una risposta, nei WebSocket il canale di comunicazione resta aperto finché almeno uno tra client e server chiude la connessione. Per una descrizione più completa è possibile consultare [11].

Per aprire il canale di comunicazione viene fatta una richiesta (handshake) utilizzando il protocollo HTTP, da quel momento in poi viene aperta una comunicazione bidirezionale.

I WebSocket risultano particolarmente utili nei casi di:

- Applicazioni Web in tempo reale: nei casi in cui serve uno scambio continuo di dati mantenere aperto il canale di comunicazione risulta più efficiente di aprirne e chiuderne uno per ogni invio di dati;
- Applicazioni di messaggistica: dopo aver aperto la connessione permette una gestione ottimizzata di ricezione e inoltro (anche in broadcast) di messaggi e materiale multimediale tipico della messaggistica moderna;
- Online gaming: fornisce un canale veloce e ottimizzato per contesti in cui vi è necessità di veloce scambio di informazioni.

1.6 Client-Server

L'approccio client-server è un approccio molto comune al giorno d'oggi. Questo tipo di architettura generalmente prevede la condivisione delle risorse dove un client richiede ad un server l'accesso alle risorse (Figura 1.3).

Ci sono due tipologie principali di accesso che caratterizzano il tipo di architettura [12]:

- 2-tier: i client accedono direttamente alle risorse. È possibile immaginare un'applicazione client richiedere direttamente al database presente sul server l'accesso ai dati;
- 3-tier: il client (chiamato thin client) non si occupa della business logic ma si limita ad operare come interfaccia tra l'utente e il server. In questo caso le richieste del client vanno al server (che si occupa della business logic) ed è il server stesso ad accedere alle

risorse condivise (come il database). Questo approccio comporta diversi vantaggi come una maggiore sicurezza e permette di utilizzare dei terminali con risorse modeste dato che le elaborazioni che consumano più risorse avvengono lato server.

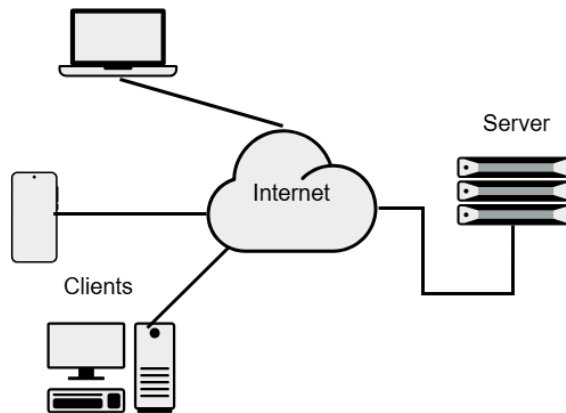


Figura 1.3: Struttura modello client-server

1.7 Internet of Things

L'IoT (Internet of Things) fa riferimento all'interconnessione in rete di oggetti diversi tra loro, da grossi macchinari ad oggetti di uso comune.

Questi dispositivi tipicamente scambiano dati con dispositivi vicini e lontani tramite tecnologie di comunicazione come Bluetooth e WiFi.

L'utilizzo di sensori e attuatori permette sia la raccolta dei dati che l'esecuzione di azioni basate sul risultato delle elaborazioni sui dati.

L'irrisorio costo di sensori e microcontrollori ha permesso una rapida diffusione dei dispositivi connessi arrivando anche alla realizzazione di dispositivi wearable ovvero dispositivi indossabili e connessi come orologi, bracciali e anelli.

1.8 Internet of Medical Things

In campo medico l'IoT è un mercato in rapida crescita con diversi dispositivi adibiti principalmente a scopo di monitoraggio dei parametri vitali dei pazienti.

Esistono dispositivi in grado di raccogliere e inviare dati come pressione sanguigna, frequenza cardiaca, saturazione dell'ossigeno nel sangue, livello della glicemia. Per una lista più completa di dispositivi medici in ambito IoT è possibile consultare l'articolo [13].

Macchinari come ad esempio i ventilatori (CPAP/BPAP) sono connessi in rete e oltre a raccogliere parametri sull'utilizzo da parte del paziente possono essere gestiti dai tecnici da remoto in modo da ridurre al minimo le operazioni di manutenzione diretta garantendo comunque un monitoraggio costante del paziente.

2 Panoramica Software e Framework

In questo capitolo verranno introdotti al livello teorico le principali tecnologie e framework utilizzati nel progetto. Mentre dal capitolo successivo si scenderà più in dettaglio nella parte pratica.

2.1 Thonny IDE

Come riportato in [14] Thonny è un IDE (Integrated Development Environment) open source e multiplatforma sviluppato nel 2014 all'università di Tartu, Estonia.

Tra le caratteristiche principali di questo IDE è possibile trovare l'interfaccia semplice e intuitiva (che lo rende ideale per studenti e neofiti), un debugger integrato, la sottolineatura degli errori e la compilazione automatica, la visualizzazione dello stack delle chiamate, la possibilità di gestire file su macchine remote tramite SSH.

Il motivo principale per cui è stato utilizzato in questo progetto, oltre alla sua semplicità di utilizzo, è la compatibilità con MicroPython che lo rende ideale per programmare microcontrollori.

Nell'utilizzo con i microcontrollori come l'ESP sono presenti 4 pannelli principali (come da Figura 2.1). Il primo box in alto a sinistra individua i file locali alla macchina che si sta utilizzando. Il box in basso a sinistra mostra i file presenti all'interno della memoria del microcontrollore. Il box principale costituisce l'editor di testo in cui scrivere e modificare i file sorgente. Infine in basso è possibile vedere una sezione che identifica la Shell, particolarmente utile per visualizzare informazioni provenienti dal microcontrollore e per eseguire direttamente dei comandi.

Questo IDE ha ricevuto pareri positivi all'interno della comunità di Python, tanto che dal 2017 è incluso nelle versioni ufficiali di Raspberry Pi OS.

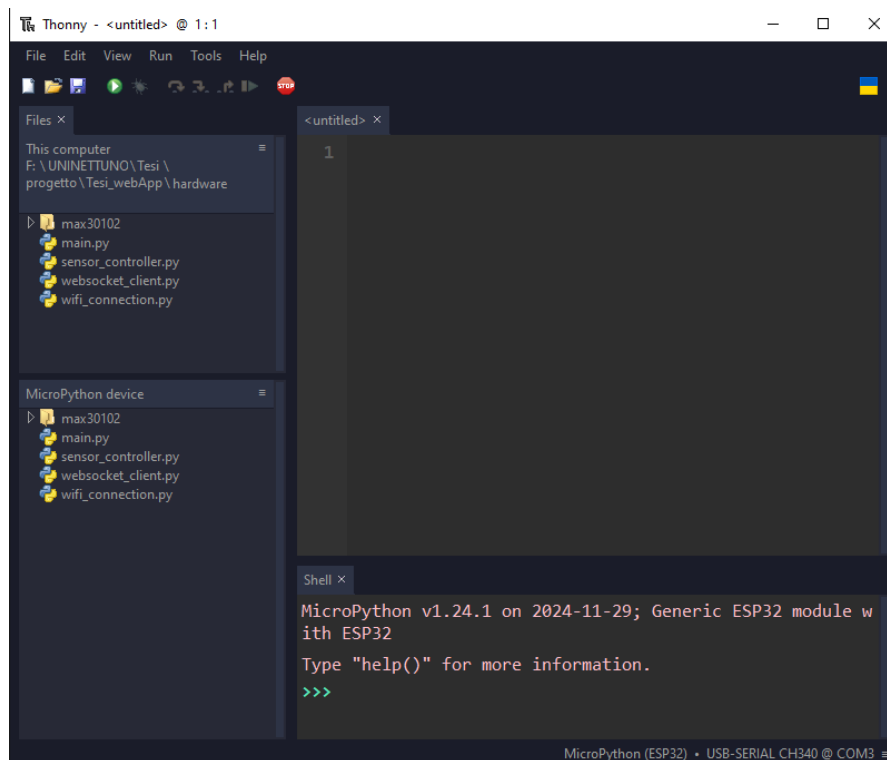


Figura 2.1: Schermata Thonny IDE

2.2 MicroPython

Come riportato in [15], nato da una campagna sostenuta su Kickstarter nel 2013, MicroPython è un compilatore Python completo e runtime scritto in linguaggio C e ottimizzato per l'esecuzione su microcontrollori.

L'utilizzo di MicroPython consente l'esecuzione delle principali librerie Python3 di base, ma esistono anche librerie per l'accesso a funzionalità hardware di basso livello.

L'utilizzo di MicroPython nei microcontrollori offre una valida alternativa a quello che era lo standard de-facto ovvero il linguaggio Wiring. Utilizzare codice in un linguaggio di alto livello come Python permette di semplificare il primo approccio alla programmazione dei microcontrollori da parte di utenti meno esperti.

2.3 Node JS

Node JS è un runtime environment JavaScript open source e multiplatforma. Node JS viene eseguito sul motore JavaScript V8 di Google Chrome. Ovvero un motore dalle ottime performance scritto in linguaggio C++ che permette di compilare ed eseguire codice JavaScript.

Node JS ha quindi permesso l'utilizzo di un linguaggio di alto livello, versatile, portabile e molto popolare come JavaScript anche in ambiente server realizzando il paradigma "JavaScript Everywhere" in cui appunto è possibile utilizzare lo stesso linguaggio sia per il front-end che

per il back-end. Per utilizzare Node JS è sufficiente installarlo sulla propria macchina (vedi figura 2.2).

Come argomentato in [16] tra le caratteristiche principali di Node JS è possibile trovare:

- filosofia asynchronous by default: l'architettura è di tipo asincrono e non bloccante. Questo permette l'esecuzione concorrente di operazioni multiple permettendo al server di gestire richieste multiple senza dover attendere il completamento di una o più delle stesse. È comunque possibile avere una gestione sincrona di alcune operazioni tramite i metodi `await/async`;
- Programmazione orientata agli eventi: questo approccio permette al server di operare ed eseguire le funzioni nel momento in cui avviene una richiesta da parte dell'utente o al verificarsi di un evento. Questo a beneficio della scalabilità dell'applicazione che può gestire simultaneamente un numero alto di connessioni;
- single-threaded ma altamente scalabile: la gestione in un unico thread riduce la complessità e migliora i tempi di risposta e quindi le performance;
- compatibilità cross-platform: l'ambiente Node è compatibile con tutti i maggiori sistemi operativi consentendo agli sviluppatori di scrivere codice che può essere eseguito su qualsiasi sistema;
- RESTful API: la creazione di API RESTful è semplificata dall'uso di framework come Express che in poche righe di codice permettono di creare API efficienti ed efficaci;
- Comunicazione in tempo reale: le ottime prestazioni e il supporto nativo ai WebSocket rendono Node l'ambiente ideale per sviluppare applicazioni che richiedono uno scambio di dati bidirezionale continuo.

Node può anche contare su una grossa community di supporto e sulla ricca presenza di moduli (sia open source che non) che grazie al gestore di pacchetti npm (Node Package Manager) sono facilmente integrabili all'interno dei progetti e consentendo anche una gestione automatica delle dipendenze.

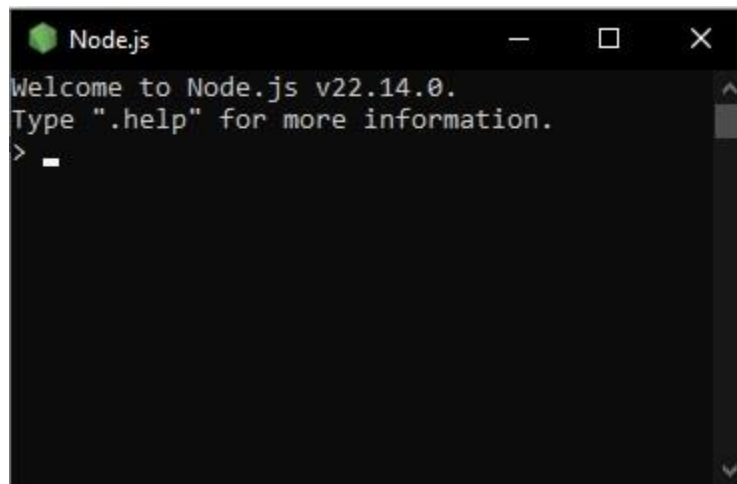


Figura 2.2: Schermata Node.js

2.4 MySQL

MySQL è un RDBMS (Relational Database Management System) open source tra i più usati al mondo.

Un DBMS è un software che consente la creazione e la gestione di database. Questo tipo di software mette in pratica tutta una serie di misure per garantire sicurezza e integrità dei dati oltre che a gestire le operazioni sui dati in modo efficiente.

MySQL è un database relazione, ovvero basato sui concetti dell'algebra relazione e i dati vengono immagazzinati in strutture tabellari (entità) dove ogni colonna rappresenta un attributo e ogni riga un'istanza di dato. Dev'essere possibile identificare in modo univoco ogni riga presente nelle tabelle e possono esistere relazioni tra le tabelle presenti nel database (modello E/R, entità/relazioni).

MySQL si compone di una parte server e una parte client. Data la sua grande popolarità questo DBMS è compatibile con tutti i principali linguaggi di programmazione oltre che software di data analysis.

2.5 Chart JS

Come descritto in [17] Chart.js è una libreria JavaScript open source creata nel 2013 per la rappresentazione grafica dei dati.

Chart.js mette a disposizione 8 tipi di grafici tra cui diagrammi a barre, linee, torta e radar (per alcuni esempi vedi Figura 2.3).

Oltre al fatto di essere open source e di avere una buona community di supporto, tra le principali caratteristiche di questa libreria vi è il fatto di renderizzare i grafici nelle canvas di HTML5 e soprattutto la capacità di adattare automaticamente i grafici alle dimensioni dello schermo.

Benché non sia tra le librerie più complete in termini di funzionalità, il vantaggio di Chart.js sta nella semplicità e immediatezza d'uso che l'hanno resa tra le opzioni più popolari per la visualizzazione di dati.



Figura 2.3: Tipi di grafici renderizzabili con ChartJS

2.6 GitHub

GitHub è una piattaforma di hosting dov'è possibile caricare progetti software (sia interi che parziali). Il nome deriva dal noto software di versionamento Git. GitHub può essere quindi visto come un'evoluzione di Git volta alla creazione e condivisione in rete di software.

L'utilizzo di piattaforme come GitHub agevola il lavoro di gruppo su uno stesso software e questo favorisce lo sviluppo di progetti open source dove sviluppatori di tutto il mondo possono contribuire ad un progetto open source senza necessariamente conoscersi a vicenda.

GitHub utilizza tutta una serie di strumenti e di pratiche che permettono di semplificare al massimo la cooperazione e la condivisione di codice. In pochi click è possibile scaricare un progetto, apportare delle modifiche ed effettuare dei fork o delle pull-request. Ma è anche

possibile segnalare dei bug o fare delle richieste agli sviluppatori per implementare nuove funzionalità al programma stesso.

Ad ogni progetto sono in genere associati dei file README in cui è possibile fornire informazioni aggiuntive sul progetto oltre che eventualmente una guida su come replicare ed eseguire quel software sul terminale dell'utente.

Questo progetto è stato reso pubblico su GitHub all'indirizzo:

https://github.com/francesco-fabio/Tesi_webapp.git

3 Caso di Studio

I saturimetri sono dispositivi al giorno d'oggi comuni da trovare che hanno lo scopo di rilevare i valori di saturazione di ossigeno nel sangue e di frequenza cardiaca. Ne esistono svariati modelli. Dai più semplici che sono in grado di effettuare letture in tempo reale, a quelli più avanzati (che permettono la raccolta dei dati su un dispositivo come uno smartphone) a quelli di uso professionale a cui viene richiesto di visualizzare e raccogliere i dati anche per delle ore e di fornire un report dettagliato dei valori raccolti.

Il caso di studio consiste nella realizzazione di un'infrastruttura (composta da un dispositivo hardware e una piattaforma software) che sia in grado di raccogliere, immagazzinare e visualizzare dei parametri vitali di un utente.

Nello specifico viene preso in esempio il caso delle rilevazioni di SpO2 e frequenza cardiaca di un utente.

3.1 Scenario d'uso

Lo scenario d'uso di questo progetto consiste nel monitoraggio in tempo reale dei dati provenienti dal saturimetro e nella possibilità da parte dell'utente di effettuare delle registrazioni e di visualizzare per mezzo di grafici i valori raccolti durante le misurazioni.

Queste funzionalità si prestano bene ad usi che possono andare da quello sportivo, al monitoraggio della qualità del sonno fino al monitoraggio continuo dei parametri di individui con insufficienze respiratorie.

Si ricorda infine che lo scopo del progetto non è quello di realizzare un dispositivo medico, ma solo di mostrare come una tale infrastruttura possa essere realizzata.

3.1.1 Caratteristiche del progetto

Le caratteristiche del progetto sono:

- **Open-Source:** il codice sorgente è reso disponibile pubblicamente su GitHub. Le tecnologie software e i framework utilizzati sono a loro volta open source. Per cui c'è libertà di utilizzo;
- **Costi ridotti:** essendo la parte software open source non prevede canoni o costi di acquisto licenze. Per la parte hardware si è utilizzato un microcontrollore e un sensore molto economico il cui costo complessivo, al momento della realizzazione di questo elaborato, non supera i 10€;

- Modularità: la separazione tra front-end e back-end oltre che l'architettura a microservizi consente di disaccoppiare buona parte delle funzionalità dell'applicazione. È quindi possibile estendere i moduli già presenti così come aggiungerne di nuovi per includere nuove funzionalità;
- Semplicità di utilizzo: dal punto di vista utente l'interfaccia grafica è di semplice utilizzo e consente di controllare la ricezione e la registrazione dei dati rilevati dal sensore oltre che la loro visualizzazione;
- Multiplatforma: la scelta dei linguaggi di programmazione utilizzati consente l'esecuzione del codice su qualsiasi piattaforma oltre che la possibilità di self-host del servizio svincolando l'applicazione da qualsiasi vincolo sulla scelta del sistema operativo o dell'hosting da utilizzare.

3.1.2 Focus del progetto

Molti degli strumenti per la rilevazione della SpO2 risultano essere spesso vincolati fortemente al produttore. Accade quindi frequentemente di avere buoni dispositivi hardware che però lato software sono carenti di funzionalità o compatibilità con dispositivi o piattaforme di altre aziende. Altri casi sono quelli dei dispositivi professionali dove i costi sono esorbitanti e spesso il dispositivo non viene venduto assieme al software di gestione dei dati (comportando una spesa ancora maggiore).

Questo progetto vuole concentrarsi su:

- Inclusività: mettendo a disposizione una base open source è possibile aggiungere la compatibilità del software con hardware diversi. Dando quindi la possibilità a più utenti di utilizzare una piattaforma comune in cui è comunque possibile aggiungere funzionalità;
- Risparmio: mettere a disposizione degli utenti una piattaforma software e un dispositivo hardware a basso costo.

3.2 Panoramica del progetto

In questa sezione si andranno ad analizzare le scelte progettuali mentre a partire dal prossimo capitolo verranno approfondite varie componenti del progetto.

3.2.1 Struttura del progetto

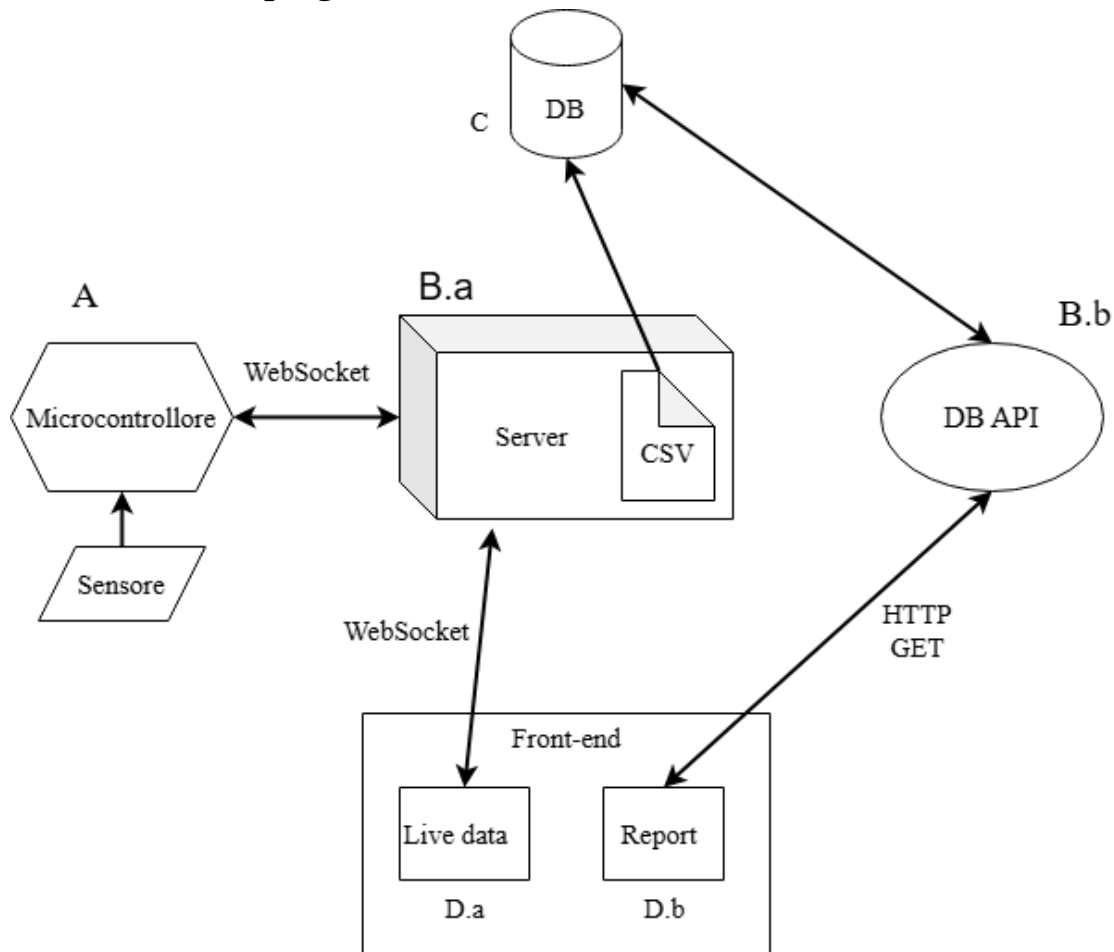


Figura 3.1: Struttura del progetto

Il progetto si compone di 4 sezioni principali (come illustrato in Figura 3.1):

- A. Client Hardware: costituito da un ESP32 collegato ad un sensore MAX30102 in grado di rilevare SpO2 e HR. Una volta acceso il microcontrollore, tramite il wifi, si collega alla rete e apre un canale WebSocket con il server. Una volta registrato sul server il microcontrollore resta in attesa del comando per avviare la lettura e l'invio dei dati. Il canale bidirezionale viene utilizzato sia per inviare i dati al server che per ricevere i messaggi dal front-end che gestiscono l'avvio/arresto del flusso dei dati;
- B. Server Node JS: è formato da due servizi distinti:
 - a. servizio 1: si occupa di creare il WebSocket server a cui microcontrollore e front-end si connettono. Il server gestisce l'inoltro dei messaggi inviando il flusso di dati dal microcontrollore al front-end e i messaggi di start/stop dal front-end al microcontrollore.

Quando viene avviata la registrazione dal front-end, il server crea un file csv e memorizza il flusso di dati sul file. Alla fine della registrazione il contenuto del file csv viene caricato dal server sul database;

- b. servizio 2: si occupa di esporre delle REST API per accedere ai dati memorizzati nel database. Il front-end utilizza le API per recuperare i dati da utilizzare nei grafici;
- C. Database MySQL: costituito da una semplice tabella che contiene SpO2, HR, un timestamp che indica in quale momento i dati sono stati ricevuti dal server e un identificativo al file csv. L'identificativo viene utilizzato per indicare a quale registrazione fanno riferimento i dati memorizzati;
- D. Front-End: è diviso in due componenti rappresentati da due differenti pagine HTML:
- a. homepage: costituisce la pagina principale. In questa pagina sono presenti i comandi start/stop per avviare il flusso di dati in diretta dal microcontrollore, dati che sono visualizzati in tempo reale. Tramite il tasto "rec" è possibile avviare e arrestare la registrazione dei valori;
 - b. report: Questa pagina è costituita da un menu a tendina che permette di selezionare una tra le registrazioni presenti in memoria sul database. Selezionando una registrazione si vanno a popolare i grafici presenti sulla pagina che permettono appunto di visualizzare i dati relativi alla specifica registrazione.

Nei capitoli successivi verrà analizzato più nel dettaglio ognuno di questi punti.

4 Client Hardware

La scelta dell'hardware, come menzionato precedentemente, è ricaduta su di un ESP32 WROOM e un sensore MAX30102.

Entrambi i componenti sono facilmente reperibili e dai costi ridotti. La scelta dell'ESP32 è dovuta appunto all'ottimo rapporto qualità/prezzo oltre che alla semplicità di utilizzo.

4.1 Schema elettrico

Dal punto di vista dei collegamenti è stato sufficiente utilizzare 4 degli 8 contatti presenti sul sensore (Figura 4.1).

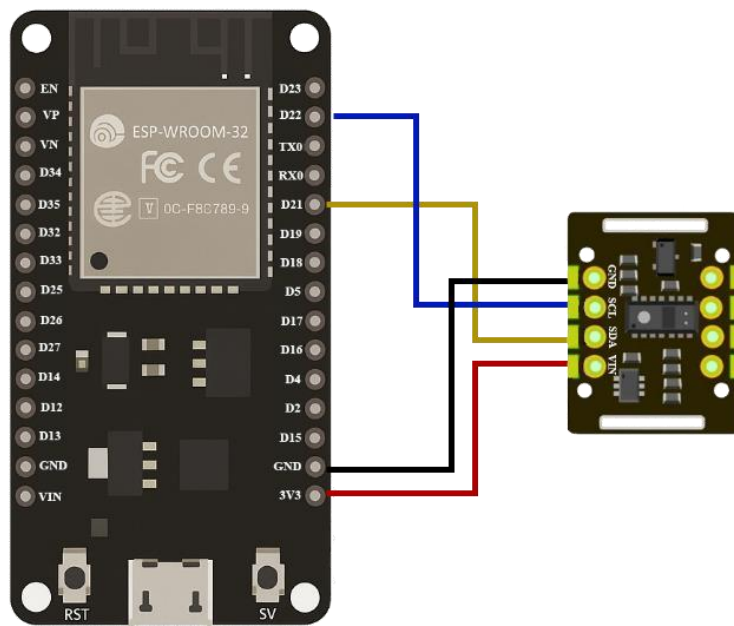


Figura 4.1: Diagramma di connessione

Il collegamento tra il sensore e l'ESP32 segue lo schema della Tabella 4.1:

MAX30102	ESP32
V _{IN}	3.3V
GND	GND
SCL	GPIO22
SDA	GPIO21

Tabella 4.1: Schema collegamento

Dove:

- SCL → I2C clock input
- SDA → I2C data, bidirezionale (open drain)

4.1.1 Alimentazione

Il microcontrollore viene alimentato tramite la porta USB con la possibilità di fornire una tensione di 3.3V o 5V alle periferiche collegate.

Per il sensore l'alimentazione di 3.3V viene fornita direttamente dal microcontrollore.

4.1.2 MAX30102

Il MAX30102 (mostrato in Figura 4.2) è un modulo integrato compatto in grado di fornire i valori di SpO2 e frequenza cardiaca. Le dimensioni ridotte, che non sacrificano le prestazioni, e il fatto di non richiedere un particolare hardware aggiuntivo lo rendono ideale anche per applicazioni in campo wearable. Il MAX30102 è interfacciabile tramite I2C e permette di accedere direttamente ai propri registri, il suo output è di tipo digitale per cui si presta bene ad essere utilizzato con i più comuni microcontrollori come arduino, raspberry pi e ESP32. Per maggiori informazioni consultare il documento di specifiche [18].

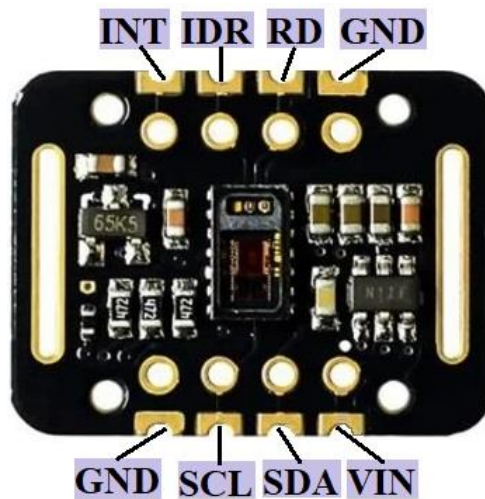


Figura 4.2: Sensore MAX30102

Diagramma di Sistema

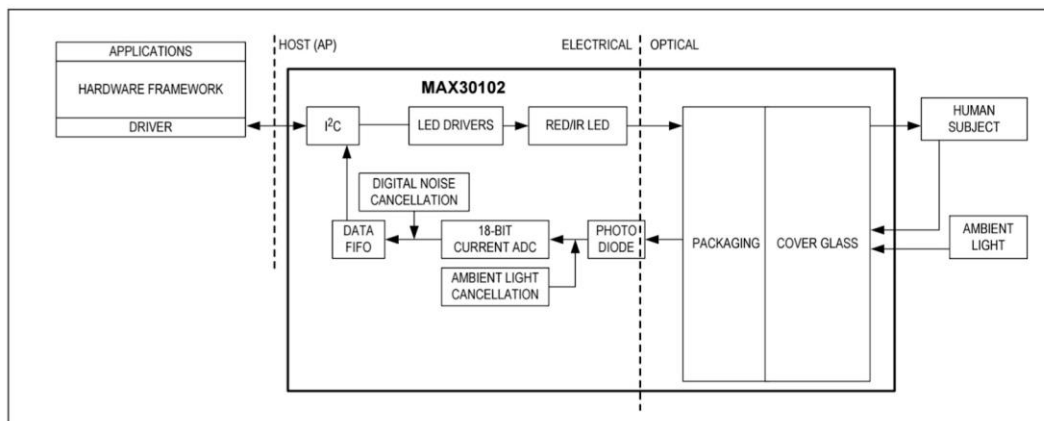


Figura 4.3: Diagramma di sistema del MAX30102

Com'è possibile vedere nello schema in Figura 4.3 il sensore è composto da due LED che emettono luce RED e IR, un fotodiodo che cattura la luce riflessa, un filtro che attenua le interferenze della luce ambiente e un filtro che riduce il rumore. Il segnale digitale viene infine salvato sui registri FIFO del sensore.

I LED emettono luce RED e IR, luce che viene in parte assorbita (da tessuti, vasi sanguigni e ossa) e in parte riflessa. Il fotodiodo si occupa di catturare la luce RED e IR riflessa. Il processo è brevemente illustrato nella Figura 4.4.

Come discusso in [19] il principio con cui vengono determinate HR e SpO2 è la PPG (Photoplethysmography) ovvero una tecnica ottica non invasiva che permette di misurare i cambiamenti di pressione all'interno del letto microvascolare dei tessuti (insieme di piccoli vasi sanguigni sia venosi che arteriosi).

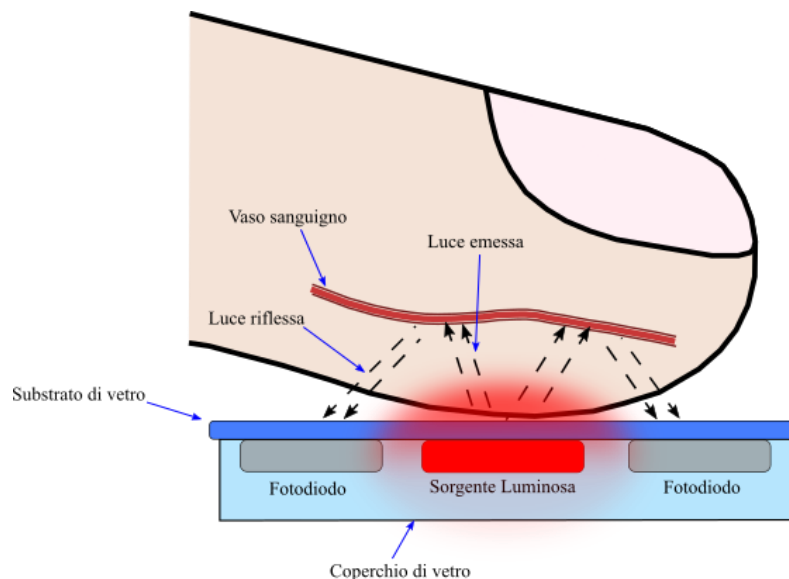


Figura 4.4: Schema funzionamento sensore

Per determinare la frequenza cardiaca (HR) si usa la PPG ottenuta dal LED IR. Una volta ottenuti i campioni (opportunamente filtrati) si determinano i picchi (massimi locali) nel segnale e si calcola la distanza tra i picchi. Dividendo questo valore per 60 si ottengono i BPM ovvero i battiti per minuto.

La SpO2, come descritto in [20], si ottiene usando il rapporto tra l'assorbimento dei LED RED e IR.

$$R = \frac{(AC_{RED}/DC_{RED})}{(AC_{IR}/DC_{IR})}$$

$$SpO_2 = 110 - 25 \cdot R$$

Dove:

- AC → segnale che rappresenta l'assorbimento della luce nel sangue arterioso;
- DC → segnale che rappresenta l'assorbimento in altre sostanze come la pigmentazione dei tessuti, le vene, i capillari, le ossa, ecc...;
- R → è il rapporto tra le componenti RED e IR.

Per un valore più preciso di SpO2 bisogna calibrare il valore empirico R sulla base dello specifico dispositivo utilizzato.

4.2 ESP32

La scheda utilizzata in questo progetto è una ESP32-DevKitC che monta un modulo ESP32-WROOM-32 prodotto dalla Espressif Systems. Il chip di controllo è un ESP32-D0WDQ6-V3 dual-core 32bit con WiFi e Bluetooth integrati.

Tramite i pin è possibile la connessione con interfacce SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2C, IR (vedi Figura 4.5). Ai fini di questo progetto i più rilevanti sono i pin I2C.

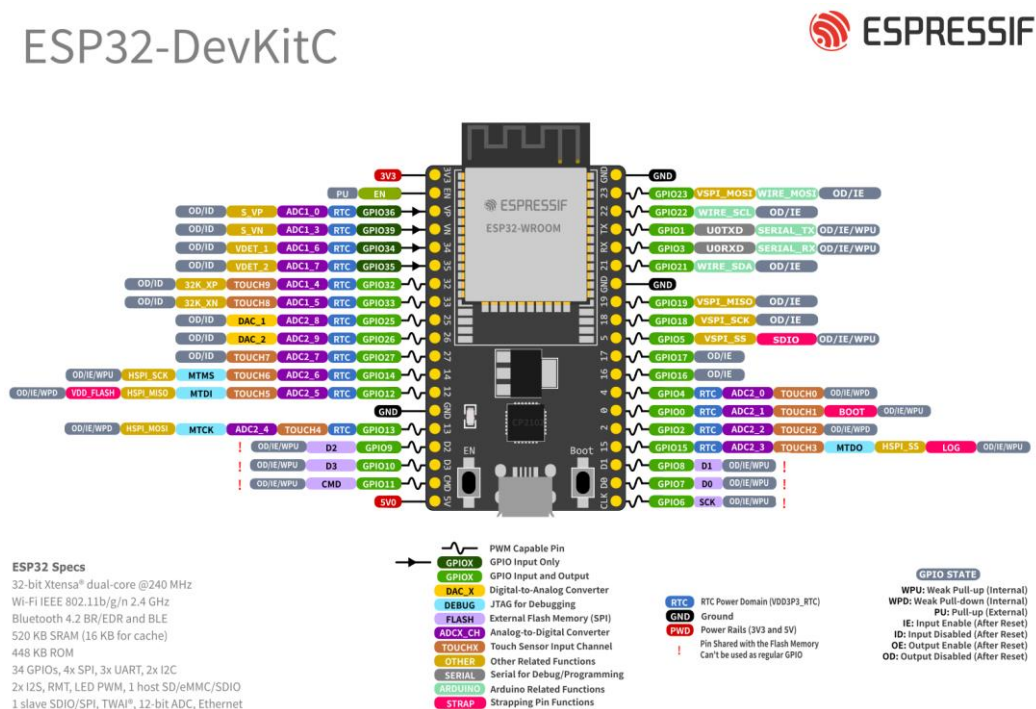


Figura 4.5: ESP32 pinout

4.2.1 Principali caratteristiche

CPU:

- ESP32-D0WDQ6: contiene due processori low-power Xtensa® 32-bit LX6

Memoria Interna:

- 448 kB di ROM per il booting e core functions;
- 520 kB (8 kB RTC FAST Memory inclusa) di on-chip SRAM per dati e istruzioni;
- 8 kB di SRAM in RTC, che è chiamata RTC SLOW Memory e può essere acceduta dal co-processore durante la Deep-sleep mode;
- 1 kbit di eFuse, di cui 320 bits sono usati per il sistema (MAC address e chip di configurazione) e i restanti 704 bits sono riservati per applicazioni dell'utente, inclusi Flash-Encryption e Chip-ID.

WiFi:

- Protocolli: 802.11 b/g/n (802.11n up to 150 Mbps);
- Range di frequenza: 2412 MHz ~ 2484 MHz;
- Modalità: Station/SoftAP/SoftAP+Station/P2P.

Bluetooth:

- Protocolli: Bluetooth v4.2 BR/EDR e BLE specification;
- Ricevitore: NZIF con sensibilità -97 dBm;
- Trasmettitore: Class-1, class-2 and class-3.

Dati più approfonditi possono essere trovati nel documento di specifiche reperibile sul sito ufficiale [21].

4.3 Firmware

Il firmware utilizzato è il ESP32_GENERIC-20241129-v1.24.1.bin reperibile sul sito ufficiale [22].

Dopo aver installato i driver CH340 (necessario solo su dispositivi Windows e MacOS), la procedura di flash del firmware avviene tramite Thonny andando alla voce Run>Configure Interpreter.

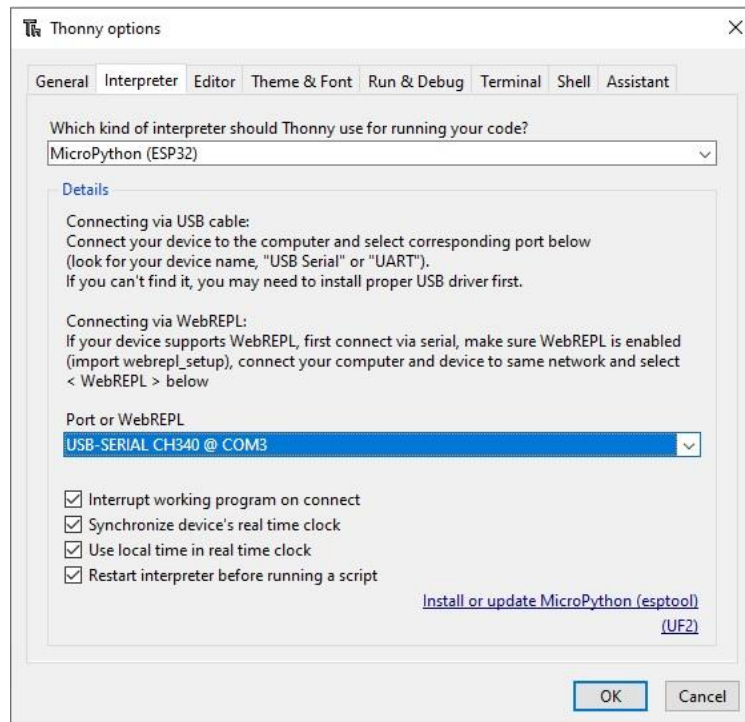


Figura 4.6: Menu configurazione interprete

Selezionando la voce *Install or Update MicroPython (esptool)*, come mostrato in figura 4.6, è possibile andare a selezionare la porta a cui il microcontrollore è collegato e l'immagine del firmware da flashare. La procedura si conclude con il click sul tasto Install (Figura 4.7). Il processo di installazione richiede pochi minuti.

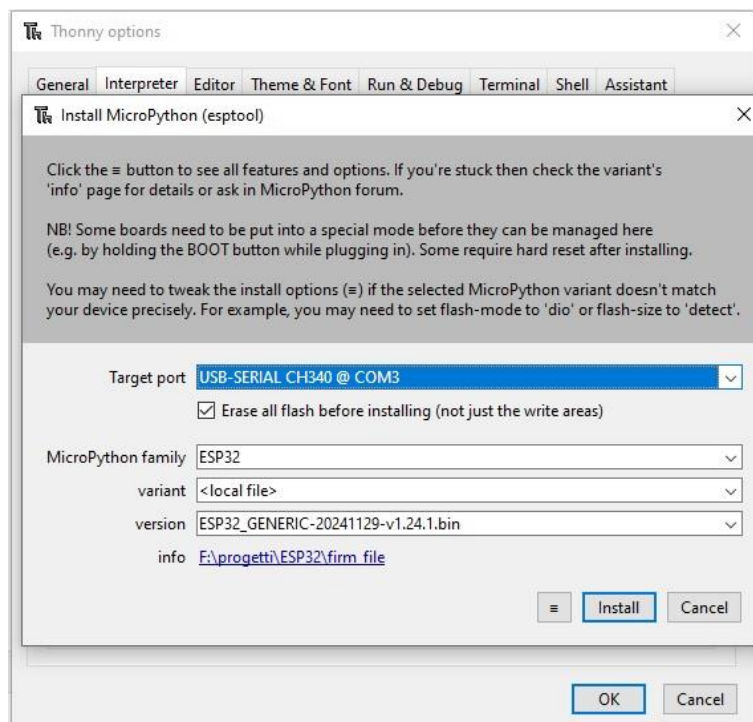


Figura 4.7: Menu installazione firmware

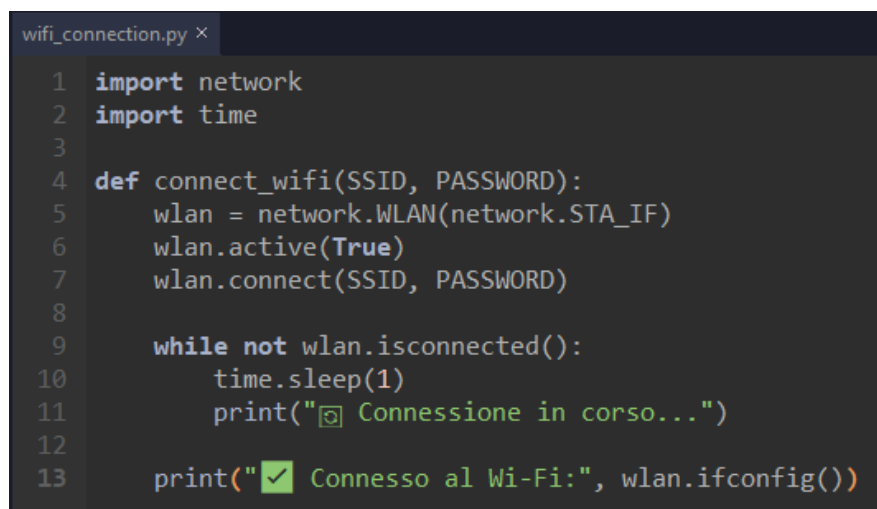
4.3.1 Funzionalità

È possibile distinguere 5 blocchi funzionali principali, ovvero:

1. Connessione al Wifi;
2. Connessione al WebSocket;
3. Lettura dei dati dal sensore;
4. Ricezione dei messaggi;
5. Invio dei messaggi

4.3.2 Connessione al WiFi

La connessione al wifi viene gestita tramite uno script dedicato chiamato *wifi_connection.py*.



```
wifi_connection.py x
1 import network
2 import time
3
4 def connect_wifi(SSID, PASSWORD):
5     wlan = network.WLAN(network.STA_IF)
6     wlan.active(True)
7     wlan.connect(SSID, PASSWORD)
8
9     while not wlan.isconnected():
10         time.sleep(1)
11         print("[🔄] Connessione in corso...")
12
13     print("[✅] Connesso al Wi-Fi:", wlan.ifconfig())
```

Figura 4.8: Script *wifi_connection.py*

Dalla Figura 4.8 è possibile vedere:

Linea 5: viene creato l'oggetto che si occuperà di gestire la connessione impostando la modalità di connessione a STA_IF. In questo modo il microcontrollore si comporterà da “station” e quindi come un normale client che si vuole collegare ad un access-point/router.

Linea 6: viene attivata la station mode.

Linea 7: richiede la connessione al router passando come parametri SSID e Password del router.

Linea da 9 a 11: un ciclo while con un time.sleep di 1 secondo resta in attesa di ottenere la connessione, stampando volta per volta un messaggio per far capire all'utente che la connessione al router è in atto.

Linea 13: tramite un messaggio si informa l'utente dell'avvenuta connessione mostrando anche la configurazione di rete ottenuta.

Questa funzione viene poi importata nel file *main.py* alla linea 4 (Figura 4.9) e utilizzata per realizzare la connessione alla linea 57 (Figura 4.11) dopo aver definito SSID e PASSWORD (in Figura 4.10 e che vanno aggiornati di volta in volta in base al router a cui ci si vuole connettere).

```
1 import uasyncio as asyncio
2 import ujson
3 import array as arr
4 from wifi_connection import connect_wifi
5 from websocket_client import WebSocketClient
6 from sensor_controller import SensorController
```

Figura 4.9: import dei moduli

```
50 #add the credencials
51 SSID = ""
52 PASSWORD = ""
```

Figura 4.10: blocco definizione credenziali WiFi

```
57 connect_wifi(SSID, PASSWORD)
```

Figura 4.11: Funzione connessione al WiFi

4.3.3 Connessione al WebSocket

Una volta connesso al router e dopo aver verificato la presenza del sensore, il microcontrollore si collega al server tramite un websocket e resta in attesa del comando che avvia il recupero e la trasmissione dei dati dal sensore.

Il websocket viene gestito tramite la classe *websocket_client.py* basata sulla libreria MicroPython usocket. Questa classe mette a disposizione i metodi per l'apertura e chiusura del websocket e per l'invio e ricezione dei dati.

```
5 from websocket_client import WebSocketClient
```

Figura 4.12: Import del modulo

```

59 #add the endpoint
60 SERVER_URL = "ws://address:port"
61
62 try:
63     ws = WebsocketClient(SERVER_URL)
64     print("✅ Connesso al WebSocket server")
65
66     loop = asyncio.get_event_loop()
67     loop.run_until_complete(main())
68     loop.run_forever()
69
70 except Exception as e:
71     print("❌ Errore:", e)
72
73 finally:
74     #insert here a disconnect message to tell
75     ws.close()
76     print("🔌 Connessione WebSocket chiusa")

```

Figura 4.13: Blocco gestione WebSocket e coroutine

Linea 5: viene importata la classe WebsocketClient (Figura 4.12).

Nella figura 4.13 è possibile vedere:

Linea 60: in questa linea bisogna inserire l'indirizzo e la porta che fanno riferimento al server.

Linea 63: viene creata la connessione websocket con il server.

Linee da 73 a 76: ci si assicura che sia in caso di errore che di fine esecuzione la connessione al websocket venga chiusa.

La ricezione e l'invio dei messaggi avviene in modo asincrono utilizzando la libreria uasyncio [23], libreria che consente di definire delle funzioni chiamate "coroutine" che sono in grado di mettere in pausa la loro esecuzione per permettere ad altre courtine di essere eseguite. Questo permette di realizzare quelli che vengono chiamati "cooperative multitasking programs". In questo modo il microcontrollore può restare in ascolto del comando start/stop e allo stesso tempo recuperare e inviare i dati dal sensore quando il flag stream è impostato su "true".

In totale sono state create 4 coroutine.

4.3.4 Lettura dei dati del sensore

Per la lettura dei dati dal sensore sono stati utilizzati gli script *circular_buffer.py* e *_init_.py* presenti nella cartella *max30102* (per maggiori informazioni consultare la repository originale del progetto [24]). Questi script fungendo da driver per il dispositivo permettono di rilevare i valori di IR e RED riflessi e catturati dal fotodiodo. La conversione di questi valori in HR e

SpO2 avviene grazie alla classe *sensor_controller.py* basata sullo script *pulse-oximeter.py* (è possibile consultare la repository originale a [25]).

```
9 dati = arr.array('i',[0,0])
```

Figura 4.14: Dichiarazione array dati

```
37 async def get_data():
38     global dati
39     while True:
40         sensor.get_values(dati)
41         print(f"SpO2={dati[0]}, HR={dati[1]}")
42
43         await asyncio.sleep(0.001)
```

Figura 4.15: Coroutine di recupero dati

Linea 9: viene definito un vettore che conterrà all'indice 0 il valore di SpO2 e all'indice 1 il valore di HR (Figura 4.14).

Nella Figura 4.15 è possibile vedere:

Linea 40: la scrittura dei valori sul vettore dati avviene per riferimento, viene passato al metodo *get_values* dell'oggetto *sensor* di classe *SensorController* il vettore che viene aggiornato ad ogni nuova lettura.

Linee 43: definisce il ripetersi della coroutine ogni millesimo di secondo.

4.3.5 Ricezione dei messaggi

```
11 async def look_for_received_msg():
12     global stream
13     while True:
14         rep = await ws.recv_async()
15         if rep is not None:
16             msg = ujson.loads('{' + rep)
17             stream = msg.get("stream") == "true"
18             print("📬 Ricevuto:", msg)
19             await asyncio.sleep(0.1)
20
```

Figura 4.16: Coroutine di ricezione messaggio

Nella coroutine chiamata "look_for_received_msg" (Figura 4.16) è possibile vedere come il microcontrollore resta in ascolto di messaggi da parte del server. Quando arriva un messaggio, viene valutato il valore dell'attributo "stream" e il valore booleano risultante viene scritto all'interno della variabile globale stream. Il messaggio viene infine stampato.

4.3.6 Invio dei messaggi

```
21  async def send_data():
22      global stream
23      global dati
24
25      while True:
26          if stream:
27              # Crea il JSON con i dati del sensore
28              data = {
29                  "hr": dati[1],
30                  "spo2": dati[0],
31                  "sender": "sensor"
32              }
33              ws.send(ujson.dumps(data))
34              print("📡 Dati inviati:", data)
35              await asyncio.sleep(1)
```

Figura 4.17: Coroutine di invio messaggio

La coroutine `send_data` (Figura 4.17) è formata da un loop dove al suo interno si verifica lo stato della variabile globale `stream` tramite un `if`. In caso di riscontro positivo, il microcontrollore procede con il recupero dei dati dal vettore seguito dall'inserimento dei valori ottenuti in una stringa che sarà convertita in JSON per poi essere inviata al server (linea 33). Il campo `sender` serve a specificare al server che il messaggio in arrivo viene dal microcontrollore.

4.3.7 Creazione e avvio delle coroutine

```
45  async def main():
46      asyncio.create_task(look_for_received_msg())
47      asyncio.create_task(send_data())
48      asyncio.create_task(get_data())
```

Figura 4.18: Coroutine di creazione task

Nella coroutine `main` (Figura 4.18) vengono creati i task.

```
66      loop = asyncio.get_event_loop()
67      loop.run_until_complete(main())
68      loop.run_forever()
```

Figura 4.19: Esecuzione delle coroutine

Infine viene richiamato l'event loop (Figura 4.19) che permette di mettere in esecuzione le coroutine.

5 Server Node JS

Il server è la parte centrale del progetto in quanto si occupa di realizzare quelle che sono le funzioni principali che verranno adesso analizzate.

Per questo progetto sono stati realizzati due servizi indipendenti che hanno scopi diversi.

Un primo servizio principale si occupa di ricevere e inoltrare i messaggi tra microcontrollore e front-end, permettendo quindi al front-end di comunicare al microcontrollore quando avviare o stoppare il flusso di dati e anche di ricevere questi valori. Il secondo ruolo è quello di, alla richiesta del front-end, avviare e stoppare la registrazione dei valori che vengono salvati su un database. La fase di registrazione avviene in due passaggi. Quando la registrazione viene attivata, il server crea un file csv e inizia, in tempo reale, a scrivere i dati sul file. Alla fine della registrazione, i dati vengono caricati a blocchi di n linee (con n parametro definibile tramite un file di configurazione) sul database.

Il secondo servizio si occupa di esporre delle API che permettono di estrarre i dati dal database e quindi permettono al front-end di rappresentare graficamente i dati salvati.

5.1 Moduli principali utilizzati

La scelta di utilizzare Node JS è dovuta anche al fatto che dispone di un'ampia quantità di moduli che permettono in poche righe di realizzare intere applicazioni web. Il gestore dei pacchetti npm permette una semplice installazione e gestione delle dipendenze dei moduli necessari.

Segue una breve analisi dei moduli più importanti utilizzati.

5.1.1 WS

È un implementazione client/server di un WebSocket robusta, ampiamente testata e di semplice utilizzo. Con decine di migliaia di download ogni settimana costituisce la scelta principale per la realizzazione di WebSocket client o server in Node. Nella sezione dedicata verrà mostrato come con qualche riga di codice è possibile creare un WebSocket server [26].

5.1.2 Express

Express è un framework web minimale e flessibile per la realizzazione di applicazioni web. È tra i framework Node più popolari e costituisce la base per molti altri framework oltre che il cuore di molte applicazioni. Permette la creazione e la gestione di middleware che rispondono a chiamate HTTP, permette la definizione di routing table che consentono l'esecuzione di azioni

differenti sulla base del metodo HTTP e dell'URL, consente il render automatico di pagine HTML sulla base degli argomenti e dei template passati [27].

In questo progetto è stato utilizzato per la creazione di API REST.

5.1.3 MySQL2

È un client MySQL per Node JS orientato alle performance. Supporta prepared statements, codifiche non utf8, compressione, ssl e molto altro. Rappresenta una soluzione ampiamente testata e robusta per connettere le proprie webapp a database MySQL. Nel progetto viene utilizzato da entrambi i servizi al fine di creare, popolare e interrogare il database [28].

5.2 Codice

In questa sezione saranno trattate le porzioni di codice più rilevanti per il progetto. Verranno divise in due parti in base al servizio.

5.2.1 Servizio 1

Il servizio 1 (che nel repository è rappresentato dal contenuto della cartella backend) è composto dai file:

- server.js: costituisce il file principale, qui viene gestita la comunicazione tra microcontrollore e terminale front-end. Gestisce anche la registrazione del flusso di dati sul database;
- fileHandler.js: classe custom realizzata per semplificare la creazione e la scrittura di file csv da parte del server;
- csvImporter.js: classe realizzata per semplificare il caricamento dei file csv sul database da parte del server.

5.2.2 Gestione della comunicazione

Per poter creare un WebSocket bisogna per prima cosa richiamare il modulo ws (vedi Figura 5.1).

```
8  const WebSocket = require("ws");
9  const fileHandler = require("../fileHandler.js");
10 const csvImporter = require("../csvImporter");
```

Figura 5.1: Import dei moduli

Nel server è presente un oggetto template e delle variabili utilizzate come flag, tutti con visibilità globale. Il primo fa da contenitore per il server a quelle che sono le informazioni che

faranno parte dei messaggi scambiati. Mentre il microcontrollore e il terminale front-end inviano solo parte delle informazioni, il server aggrega i dati e inoltra i messaggi completi.

```
14 // template base dei messaggi che vengono scambiati tra microcontrollore e frontend
15 let template = {
16   spo2: "",
17   hr: "",
18   stream: "",
19   rec: "",
20   sender: "",
21 };
22 let rec = false;
23 let recSwitchedOn = true;
24 let fh;
25 let tmStmp;
26 let loadToDb = false;
```

Figura 5.2: Dichiarazione delle variabili globali

Linee da 15 a 21 (Figura 5.2): l'oggetto template composto dai campi che in ordine rappresentano:

- spo2: il valore di saturazione rilevati dal sensore;
- hr: il valore di frequenza cardiaca rilevato dal sensore;
- stream: un flag che se posto a true/false dice al microcontrollore di avviare/stoppare l'invio dei dati;
- rec: un flag che se posto a true/false dice al server di avviare/stoppare la registrazione del flusso dei dati;
- sender: indica il mittente del messaggio.

Linee 22 e 23: flag che aiutano la gestione dell'avvio e dell'arresto della registrazione.

Linea 24: variabile che conterrà l'oggetto fileHandler per la gestione della creazione e scrittura del file csv.

Linea 25: variabile che conterrà un timestamp usato per la creazione del nome univoco del file csv;

Linea 26: flag utilizzato per la gestione del caricamento dei dati del file csv nel database.

Il server si comporta come un intermediario che smista i messaggi. Per gestire questa operazione, nel momento in cui si crea una connessione con un client, al client viene assegnata un'etichetta che lo classifica in base al tipo, dove "sensore" individua il microcontrollore e "client" è il terminale che accede da front-end. Ogni messaggio ricevuto dal server viene smistato sulla base del mittente, le comunicazioni del sensore vanno al client e viceversa.

```

35 // creazione del websocket
36 const wss = new WebSocket.Server({ port: 3000 });
37
38 wss.on("connection", (ws, req) => {
39   // distingo il microcontrollore dal client
40   if (req.headers["sec-websocket-protocol"]) {
41     console.log(`benvenuto: ${req.headers["sec-websocket-protocol"]}`);
42     clients.add({ client: ws, type: "client" });
43   } else {
44     console.log("nuovo sensore connesso!");
45     clients.add({ client: ws, type: "sensor" });
46   }
47
48   // gestione delle operazioni alla ricezione del messaggio
49   ws.on("message", async (message) => {
50     try {
51       const data = JSON.parse(message);
52       template.sender = data.sender;
53
54       // differenzia il comportamento in base al mittente del messaggio
55       if (data.sender == "sensor") {
56         template.spo2 = data.spo2;
57         template.hr = data.hr;
58         // invia i dati del sensore al client frontend
59         for (let client of clients) {
60           if (client.type == "client") {
61             client.client.send(JSON.stringify(template));
62           }
63         }
64       }
65     } catch (error) {
66       console.log(error);
67     }
68   });
69 }

```

Figura 5.3: Creazione e gestione del WebSocket Server

Nella Figura 5.3 è possibile vedere:

Linea 36: crea un server WebSocket in ascolto sulla porta 3000.

Linea 38: gestisce la connessione di nuovi client. Il server può ricevere i messaggi di tutti i client, ma quando invia un messaggio questo non avviene in broadcast, quindi è bene gestire l'invio dei messaggi in caso di più client (dove in questo caso si hanno il microcontrollore e il client frontend).

Linee da 40 a 46: tramite il controllo degli header, il server distingue il microcontrollore dal frontend in quanto il client frontend specifica un header. Tramite un controllo quindi ai dispositivi connessi viene applicata un'etichetta e vengono aggiunti ad un Set.

Linea 49: intercetta la ricezione di un messaggio da parte del server.

Linee 51 e 52: Il messaggio viene convertito in JSON e viene estratta l'informazione sul mittente che viene aggiunta al template.

Linee da 55 a 63: verifica che il mittente sia il sensore, e se lo è, al template vengono aggiunti i valori della spo2 e hr che verranno mandati al client in tempo reale. All'interno di questo controllo ci sono dei blocchi relativi alla registrazione che verranno trattati nelle sezioni successive.

```
138     ws.on("close", () => {
139         for (let client of clients) {
140             if (client.client == ws) {
141                 console.log(`${client.type} disconnesso`);
142             }
143         }
144     });
145 });
146
147 console.log("✅ Server WebSocket in ascolto su ws://0.0.0.0:3000");
```

Figura 5.4: Gestione chiusura della connessione

Nella Figura 5.4 si ha:

Linee da 138 a 145: viene gestita la chiusura del WebSocket avvertendo il server tramite un messaggio testuale il tipo del client che si è disconnesso.

5.2.3 Gestione della registrazione

Come menzionato in precedenza, la registrazione viene suddivisa in scrittura del file csv e caricamento del file sul database. Questa scelta è stata pensata per via della natura dinamica dei dati. Trattandosi di un flusso continuo, per ridurre il numero di inserimenti nel database (e quindi di accessi alla base dati), viene creato il file csv che viene salvato in memoria e funge da buffer. Alla chiusura della registrazione, le righe del file vengono caricate in blocchi (con la dimensione del blocco definibile nel file di configurazione) nel database.

```

65 // gestione della registrazione
66 if (rec) {
67     // recSwitchedOn serve a determinare il cambiamento di stato del f
68     if (recSwitchedOn) {
69         recSwitchedOn = false;
70         tmStmp = timestamp();
71         // creo il file usando un timestamp come identificatore univoco
72         fh = new fileHandler(`./files/rec_${tmStmp}.csv`);
73         loadToDb = true; //flag che aiuta a stabilire quando bisogna car
74         try {
75             await fh.init(); // inizializzazione e scrittura dei campi
76             await fh.write(`spo2,hr,time,file_name`);
77         } catch (err) {
78             console.error(`Errore nella gestione file: ${err.message}`);
79         }
80     }
81     const content = `${template.spo2},${
82         template.hr
83     },${timestamp()},rec_${tmStmp.replace(/\s/g, "")}`;
84     try {
85         await fh.write(content); //scrittura nel file dei valori ricevut
86     } catch (err) {
87         console.error(`Errore nella gestione file: ${err.message}`);
88     }

```

Figura 5.5: Gestione della registrazione

Nella Figura 5.5 è possibile vedere:

Linee 66 e 68: dato che per tutto il periodo di registrazione il flag rec è attivo. Il doppio controllo con recSwitchedOn ha lo scopo di determinare il momento di commutazione di rec da false a true e viceversa.

Linea 69: una volta determinata la commutazione, il flag torna a false in attesa della prossima registrazione.

Linea 70: viene generato un timestamp di tipo “yyyy-mm-dd hh-mm-ss” che verrà utilizzato per dare un nome univoco al file.

Linea: 72: Dato che questo blocco condizionale sarà eseguito solo nel momento di commutazione da false a true, questa linea di codice ha lo scopo di creare il file che sarà usato per la specifica registrazione in corso.

Linee da 74 a 79: linee che vengono eseguite una sola volta per registrazione, hanno lo scopo di inizializzare il file creando l’intestazione (con quelli che poi saranno i campi principali del database).

Linee da 81 a 83: ad ogni ricezione mette insieme i dati ricevuti dal microcontrollore, un timestamp per indicare il momento di inserimento del dato e infine associa il nome del file (utile a raggruppare i valori per file all’interno della base dati).

Linea 85: scrive all'interno del file il contenuto precedentemente preparato.

```
89     } else {  
90         recSwitchedOn = true;  
91         if (typeof fh !== "undefined") {  
92             // terminata la fase di registrazione  
93             try {  
94                 await fh.close();
```

Figura 5.6: Gestione di fine registrazione

Nella Figura 5.6 viene mostrato:

Linee da 89 a 94: nel caso in cui il controllo su rec alla riga 66 dia esito negativo, si accede a questo blocco else. Qui viene resettato il flag recSwitchedOn e se il canale di scrittura sul file è ancora aperto, lo stesso viene chiuso.

```
112     } else {  
113         if (data.rec == "true") {  
114             //gestione del flag per  
115             rec = true;  
116         } else {  
117             rec = false;  
118         }
```

Figura 5.7: Gestione flag di registrazione

Nella Figura 5.7 si ha:

Linee da 112 a 118: questo blocco else si collega al controllo alla riga 55 in cui viene verificato che il messaggio sia inviato dal microcontrollore. Se non lo è, questo significa che è stato inviato dal terminale front-end quindi qui viene gestita la commutazione del flag rec.

5.2.4 Inserimento dei dati nel database

L'inserimento dei dati presi dal file csv nel database avviene tramite la classe custom csvImporter. Questa classe, tramite i suoi metodi, gestisce l'interazione con il database e il caricamento massivo delle righe del file.

Questa classe necessita di importare dei moduli appositi per funzionare (vedi Figura 5.8).

```
6     const fs = require("fs");  
7     const csv = require("csv-parser");  
8     const mysql = require("mysql2/promise");  
9     const config = require("config");
```

Figura 5.8: Import dei moduli

Linea 6: il modulo fs permette all'applicazione di interagire con il file system della macchina e quindi in questo progetto è stato usato per la creazione e l'accesso in lettura e scrittura dei file.

Linea 7: csv-parser è un modulo rapido ed efficiente per la conversione di file csv in oggetti JSON.

Linea 8: mysql2 è stato discusso precedentemente, e come suggerisce il nome, è il modulo che mette a disposizione gli oggetti e i metodi per interagire con il database MySQL.

Linea 9: il modulo config semplifica la gestione dei file di configurazione. In questo progetto è stato utilizzato per leggere dei parametri di configurazione relativi al database da un file.

Il file di configurazione è un semplice file chiamato *default.json* (vedi Figura 5.9) presente nella sottocartella *config* del backend del progetto ed è strutturato come segue.

```
1  {
2      "table": "sensor_data",
3      "batchSize": 500,
4      "dbConfig": {
5          "host": "",
6          "user": "",
7          "password": "",
8          "database": "smart_oximeter",
9          "port": 3306
10     }
11 }
```

Figura 5.9: File di configurazione default.json

I campi vuoti vanno riempiti in base alla configurazione del database. I parametri già impostati di default possono essere eventualmente variati. Per una questione di consistenza è meglio non variarli dopo la creazione del database.

```
11  class csvImporter {
12      constructor(tmStmp) {
13          this.tmStmp = tmStmp;
14          this.csvPath = `./files/rec_${tmStmp}.csv`;
15          this.table = config.get("table");
16          this.batchSize = config.get("batchSize");
17          this.dbConfig = {
18              host: config.get("dbConfig.host"),
19              user: config.get("dbConfig.user"),
20              password: config.get("dbConfig.password"),
21              database: config.get("dbConfig.database"),
22              port: config.get("dbConfig.port"),
23          };
24          this.buffer = [];
25      }
26  }
```

Figura 5.10: Caricamento dei dati di configurazione

In fase di costruzione l'oggetto di classe csvImporter carica i dati di configurazione (mostrato in Figura 5.10).

Il metodo init ha il compito di creare la connessione con il database.

```
30  async init() {
31      try {
32          this.connection = await mysql.createConnection(this.dbConfig);
33          console.log("✅ Connessione al DB riuscita.");
34      } catch (err) {
35          if (err.errno == 1049) {
36              this.connection = await mysql.createConnection({
37                  host: config.get("dbConfig.host"),
38                  user: config.get("dbConfig.user"),
39                  password: config.get("dbConfig.password"),
40                  port: config.get("dbConfig.port"),
41              });
42              const dbQuery = `CREATE DATABASE IF NOT EXISTS ${this.dbConfig.database}`;
43              try {
44                  await this.connection.execute(dbQuery);
45                  console.log(`🏗 database creato`);
46                  await this.connection.end();
47              } catch (err) {
48                  console.error("❌ con la verifica esistenza del database", err);
49              } finally {
50                  this.connection = await mysql.createConnection(this.dbConfig);
51              }
52          }
53      }
54
55      const tableQuery = `CREATE TABLE IF NOT EXISTS ${this.table} (
56          id int NOT NULL AUTO_INCREMENT,
57          spo2 int NOT NULL,
58          hr int NOT NULL,
59          date datetime NOT NULL,
60          file_name varchar(50) NOT NULL,
61          PRIMARY KEY (id)
62      );`;
63
64      try {
65          await this.connection.execute(tableQuery);
66          console.log(`🏗 verifica esistenza tabella.`);
67      } catch (err) {
68          console.error("❌ errore nella verifica esistenza della tabella:", err);
69      }
70  }
```

Figura 5.11: Connessione con il database

Nella Figura 5.11 è possibile vedere:

Linea 32: viene tentata la connessione al database con i parametri caricati dal file di configurazione. In caso di errore interviene un blocco catch.

Linee da 35 a 41: se l'errore restituito è 1049, ovvero database sconosciuto, viene effettuato un nuovo tentativo di connessione, ma questa volta non ad un database specifico ma al server MySQL.

Linee da 42 a 46: effettuata la connessione al server MySQL viene a questo punto creato il database che precedentemente non è stato trovato. Al termine della creazione la connessione viene terminata.

Linea 50: a seguito della creazione del database viene fatto un nuovo tentativo di connessione.

Linee da 55 a 70: dopo che l'esistenza del database è stata accertata, nel caso in cui la tabella di riferimento del progetto non dovesse esistere, viene creata.

Nel file *server.js* al termine della registrazione, viene eseguita la porzione di codice per l'import massivo dei dati dal file csv al database.

```
95 //insert data into db
96 if (typeof tmStmp !== "undefined" && loadToDb == true) {
97   (async () => {
98     const importer = new csvImporter(tmStmp);
99     try {
100       await importer.import();
101       loadToDb = false;
102     } catch (err) {
103       console.error("Errore durante l'importazione:", err);
104     }
105   })();
106 }
```

Figura 5.12: Import del contenuto del file nel database

Nella Figura 5.12 è possibile vedere:

Linea 96: viene effettuato un controllo per determinare se il file relativo alla registrazione appena terminata deve essere caricato nel database.

Linea 98: l'oggetto *csvImporter* è già dotato dei dati necessari alla connessione con il database, quindi viene solamente indicato il timestamp che identifica il file da caricare.

Linee 100 e 101: viene eseguito il metodo che avvia l'import del file e si resetta il flag che permette di determinare se il file csv deve essere importato nel database.

5.2.5 Servizio 2

Il servizio 2 (che nel repository è rappresentato dal contenuto della cartella *db_api*) espone delle API REST che consentono l'estrapolazione di alcuni dati dal database. È composto dai file:

- *main.js*: è il file in cui, grazie al modulo *express*, viene creato il server che gestisce il servizio;
- *routes.js*: è il file in cui vengono definite le route (gli endpoint) che saranno raggiungibili dal client. Ad ogni route è associato un URL, un metodo HTTP e una funzione che viene eseguita quando quel percorso viene richiesto;

- `dbManager.js`: è il file in cui vengono definite le funzioni richiamate dalle route.

5.2.6 Server

```
1  const express = require("express");
2  const routes = require("../routes/routes");
3  const cors = require("cors");
4
5  const app = express();
6  app.use(express.json());
7  app.use(cors());
8
9  app.use("/charts", routes);
10
11 app.listen(3001);
12
13 console.log("✅ Server API in ascolto su http://localhost:3001");
```

Figura 5.13: API server

Nella Figura 5.13 è possibile vedere:

Linee da 1 a 3: definiscono i moduli utilizzati all'interno del file. Tra cui il modulo `cors` che permette la gestione delle CORS (Cross-Origin Resource Sharing).

Linea 5: crea un'istanza dell'applicazione Express che sarà usata per definire il server e le route.

Linea 6: configura Express al parsing dei payload JSON in ingresso, semplificando la gestione dei dati inviati nei request body.

Linea 7: consente la gestione delle richieste Cross-Origin, permettendo l'accesso alle API da domini differenti.

Linea 9: definisce il modulo interno (in questo caso il modulo `routes` che verrà trattato in seguito) che si occuperà di gestire le richieste che arrivano a tutti gli end point con root comune `"/charts"`. Nel caso di un server locale tutti gli end point con `"localhost:3001/charts/"` in comune.

Linea 11: avvia il server in ascolto sulla porta 3001.

5.2.7 Routers

Il file `routes.js` si occupa di definire le funzioni associate ad ogni endpoint. La gestione delle route del progetto segue uno schema comune, quindi a titolo esemplificativo ne sarà discussa solamente una.

```

4   const router = express.Router();
5
6
7   router.get("/filename", async (req, res) => {
8     const db = new dbManager();
9     try {
10      const data = await db.getFileNames();
11      res.status(200).json(data);
12    } catch (err) {
13      console.error("Errore durante l'importazione:", err);
14    }
15  });

```

Figura 5.14: Filename API

Nella Figura 5.14 è possibile vedere:

Linea 4: crea l'oggetto Router che consente di definire le varie route.

Linea 7: definisce il metodo HTTP con cui interrogare l'endpoint. L'endpoint, seguendo l'esempio di un server locale mostrato in precedenza, sarà raggiungibile a "localhost:3001/charts/filename". Nello specifico questa API permette di ottenere dal database la lista dei nomi dei file csv, nomi univoci che di fatto identificano le singole registrazioni. Questo permette di ottenere la lista delle registrazioni memorizzate sul database.

Linea 8: crea l'oggetto di tipo dbManger, oggetto preposto alla connessione e l'interrogazione del database.

Linea 10: esegue la funzione che interroga la base dati per ottenere i dati richiesti. Questi dati vengono appunto memorizzati nella costante "data".

Linea 11: invia la risposta alla richiesta, specificando lo status della risposta e trasmettendo il payload in formato json.

5.2.8 Interrogazione del database

Nella gestione dell'interrogazione del database da parte della classe *dbManager.py* si ha una componente in comune con quanto visto nella sezione precedente riguardante l'interazione con il database.


```

1  const mysql = require("mysql2/promise");
2  const config = require("config");
3
4  class dbManager {
5    constructor() {
6      this.table = config.get("table");
7      this.dbConfig = {
8        host: config.get("dbConfig.host"),
9        user: config.get("dbConfig.user"),
10       password: config.get("dbConfig.password"),
11       database: config.get("dbConfig.database"),
12       port: config.get("dbConfig.port"),
13     };
14   }
15
16   async init() {
17     this.connection = await mysql.createConnection(this.dbConfig);
18     console.log("✅ Connessione al DB riuscita.");
19   }

```

Figura 5.15: Costruttore della classe dbManager

Nella Figura 5.15 è possibile vedere:

Linee da 4 a 14: in fase di costruzione dell'oggetto, viene caricato il file *default.json* associato a questo servizio.

Linee da 16 a 19: il metodo `init` è più snello rispetto al precedente perché qui non viene gestita la casistica in cui il database non esiste.

Anche qui i metodi per interrogare il database seguono tutti una struttura comune, quindi ne sarà trattato solo uno a titolo esemplificativo.

```

34  async getRecordingData(name) {
35    await this.init();
36    const query = `SELECT spo2, hr, date FROM ${this.table} WHERE file_name = "${name}"`;
37
38    try {
39      const [results, fields] = await this.connection.execute(query);
40      return { success: true, results };
41    } catch (err) {
42      console.error("❌ Errore durante getFileNames:", err);
43      return { status: "err", err };
44    }
45  }

```

Figura 5.16: Recupero dati da database

Infine nella Figura 5.16 si ha:

Linea 34: in questo caso il metodo, per recuperare i dati, richiede un parametro, ovvero il nome di una registrazione presente sul database.

Linea 35: prima di proseguire con la richiesta viene effettuata la connessione al database.

Linea 36: nella costante “query” viene memorizzata appunto la query con cui verrà interrogato il database. Questo metodo in particolare, dato il nome di una registrazione, restituisce l’insieme di tutti i valori di SpO2 e HR associati a quella registrazione.

Linee 39 e 40: in questo blocco viene eseguita la query e il risultato viene restituito alla funzione chiamante (e quindi alla route che richiama questo metodo).

6 Database MySQL

Per il salvataggio e l'accesso dei dati all'interno di questo progetto è stato scelto MySQL, ovvero uno dei DBMS relazionali più popolari. La scelta di MySQL offre una soluzione potente e robusta per l'integrazione di database all'interno dei progetti.

MySQL si compone di una parte server e una parte client ed esistono svariati modi per integrarlo all'interno di un progetto e metodologie di accesso e di gestione sia da riga di comando che da interfaccia grafica.

6.1 Tool utilizzato nel progetto

Il funzionamento del progetto è indipendente dalle specifiche architetturali scelte per realizzare l'istanza server di MySQL, a patto però che il database server sia in funzione, raggiungibile dai servizi che fanno parte del progetto e che si sia in possesso delle credenziali di accesso al database da inserire nel file di configurazione *default.json*.

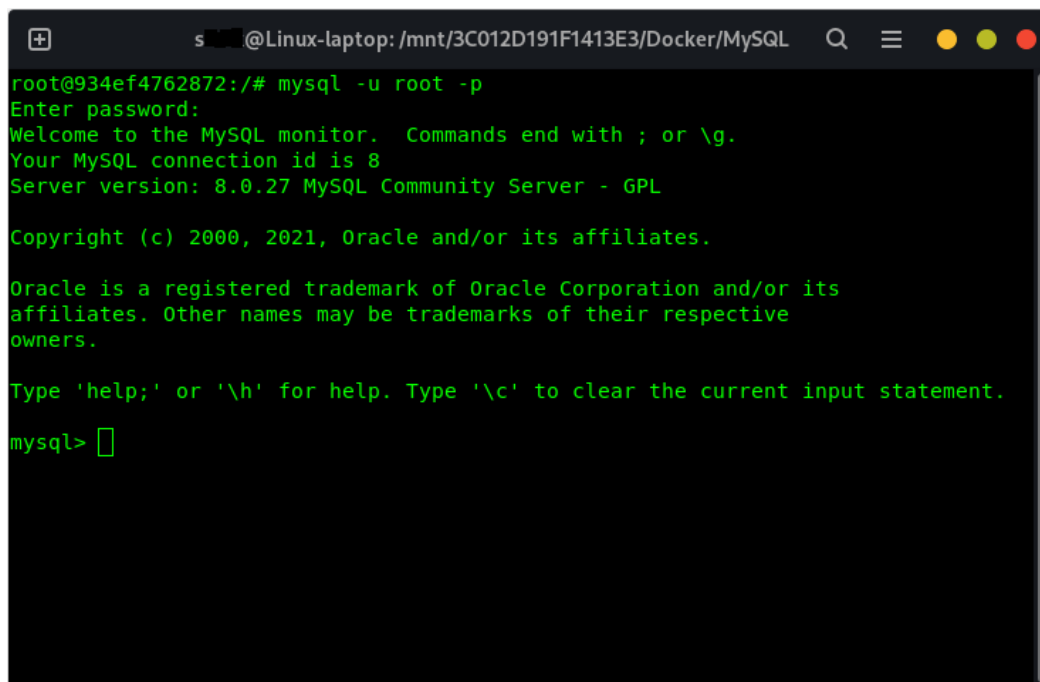
In fase di creazione del progetto, I tool utilizzati per eseguire l'istanza server di MySQL sono stati la suite XAMPP in ambiente Windows e Docker in ambiente Linux. In entrambi i casi è possibile l'accesso al database sia da riga di comando che da interfaccia grafica tramite phpMyAdmin.

L'accesso da riga di comando può essere fatto utilizzando la shell integrata in XAMPP o accedendo da terminale al container Docker su cui è in esecuzione l'istanza di MySQL.

6.2 Creazione utente

Al primo avvio si ha a disposizione l'utente root, per motivi di sicurezza è buona pratica creare un altro utente che possiede i soli privilegi sufficienti ad eseguire i comandi necessari per il corretto funzionamento dell'applicazione. A titolo esemplificativo verrà mostrato come creare un utente con i permessi necessari da terminale.

Avviata la shell è possibile avviare il client MySQL con il comando *mysql* seguito dal parametro *-u* che permette di specificare quale utente sta tentando l'accesso e il parametro *-p* che specifica che verrà inserita una password. Premuto il tasto invio il sistema chiederà di inserire la password dell'utente (vedi Figura 6.1).

A terminal window titled 's@Linux-laptop: /mnt/3C012D191F1413E3/Docker/MySQL' showing the MySQL command-line interface. The user 'root' has logged in successfully. The output includes the MySQL version (8.0.27), copyright information, and a welcome message. The prompt 'mysql>' is visible at the bottom.

```
root@934ef4762872:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.27 MySQL Community Server - GPL

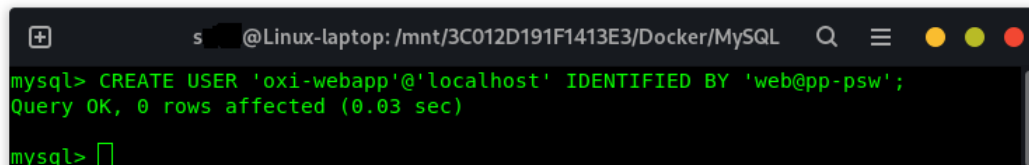
Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

Figura 6.1: Accesso a MySQL

In MySQL è possibile creare degli utenti specificando se questi debbano potersi collegare dalla stessa macchina su cui risiede il server (localhost), da un certo indirizzo ip oppure da qualsiasi macchina.

A terminal window titled 's@Linux-laptop: /mnt/3C012D191F1413E3/Docker/MySQL' showing the execution of the 'CREATE USER' command. The command creates a user named 'oxi-webapp' with a password 'web@pp-psw' that can only connect from 'localhost'. The output confirms the successful creation of the user.

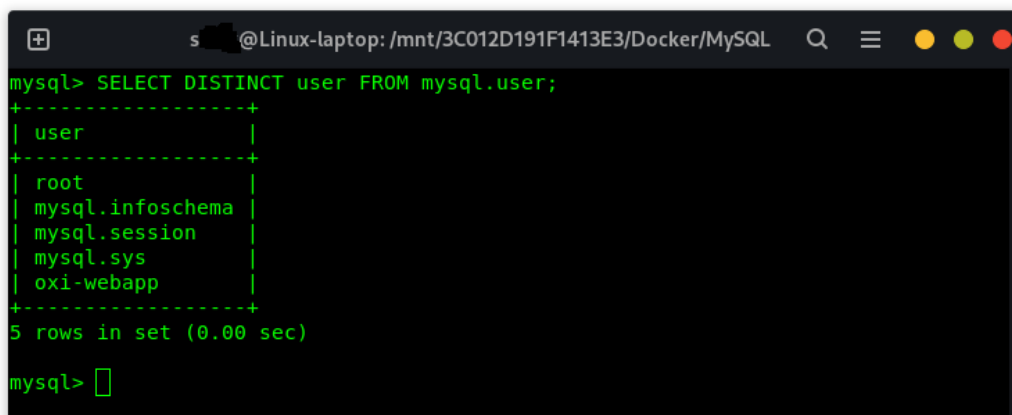
```
mysql> CREATE USER 'oxi-webapp'@'localhost' IDENTIFIED BY 'web@pp-psw';
Query OK, 0 rows affected (0.03 sec)

mysql>
```

Figura 6.2: Creazione utente

Tramite il comando `CREATE USER` è possibile creare l'utente, la parte precedente a `@` definisce lo username. Nell'esempio in Figura 6.2 viene mostrato che la connessione deve avvenire da localhost. Infine quello che segue `IDENTIFIED BY` è la password associata all'utente.

È possibile andare a verificare la corretta creazione dell'utente tramite una `SELECT` sulla tabella `user` di sistema (come mostrato in Figura 6.3).



```
mysql> SELECT DISTINCT user FROM mysql.user;
+-----+
| user                |
+-----+
| root                |
| mysql.infoschema    |
| mysql.session       |
| mysql.sys           |
| oxi-webapp          |
+-----+
5 rows in set (0.00 sec)

mysql>
```

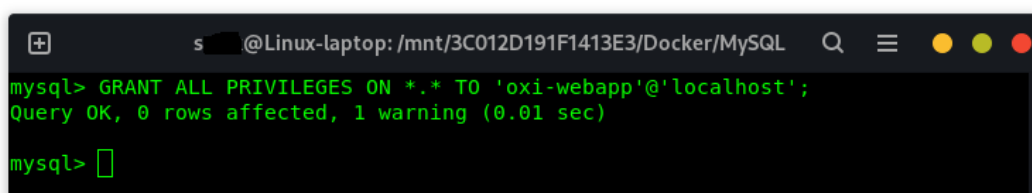
Figura 6.3: Verifica creazione utente

Da notare come oltre agli utenti presenti di default, l'utente appena creato è presente.

Dopo la creazione bisogna assegnare all'utente i permessi. Questi permessi includono:

- Create: consente all'utente di creare database;
- Select: consente all'utente di cercare informazioni all'interno del database;
- Update: consente all'utente di modificare il contenuto delle righe delle tabelle;
- Insert: consente all'utente di inserire dati all'interno delle tabelle;
- Delete: consente all'utente di eliminare righe e colonne dalle tabelle;
- Drop: consente all'utente di cancellare interi database;
- All privileges: concede accesso completo al database.

Il comando per assegnare i permessi all'utente è il comando GRANT seguito dalla lista dei permessi. È possibile assegnare permessi a tutti i database o ad un singolo database.



```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'oxi-webapp'@'localhost';
Query OK, 0 rows affected, 1 warning (0.01 sec)

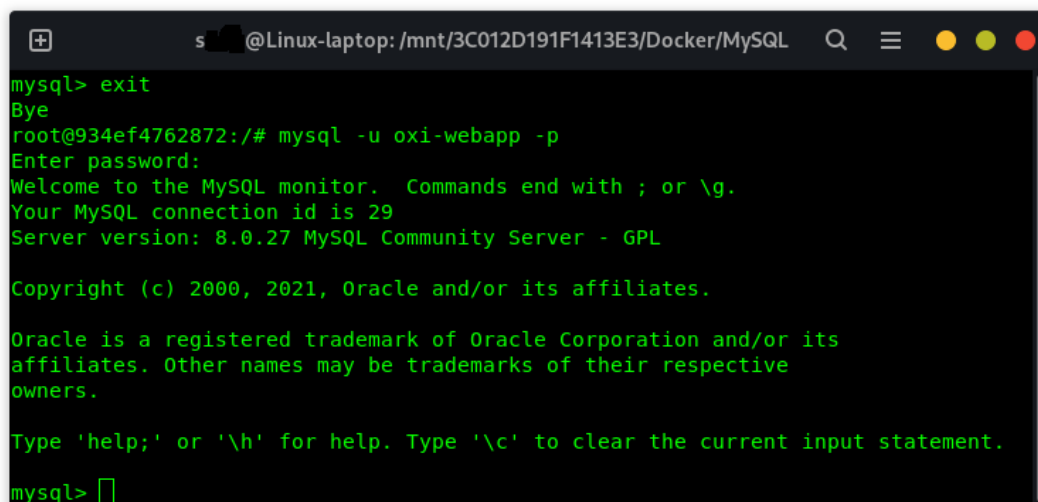
mysql>
```

Figura 6.4: Assegnazione dei permessi

Nell'esempio in Figura 6.4 viene mostrato come concedere tutti i privilegi a tutti i database.

È possibile verificare quali privilegi sono assegnati agli utenti tramite il comando SHOW GRANTS FOR seguito dal nome dell'utente.

A questo punto è possibile lanciare una nuova sessione digitando il comando exit e accedendo con l'utente appena creato (Figura 6.5).

A terminal window titled 's@Linux-laptop: /mnt/3C012D191F1413E3/Docker/MySQL' showing the MySQL command-line interface. The user enters 'exit', then 'mysql -u oxi-webapp -p'. The prompt 'Enter password:' is shown. The user enters a password, and the terminal displays: 'Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 29 Server version: 8.0.27 MySQL Community Server - GPL Copyright (c) 2000, 2021, Oracle and/or its affiliates. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners. Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. mysql>'.

```
mysql> exit
Bye
root@934ef4762872:/# mysql -u oxi-webapp -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 8.0.27 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

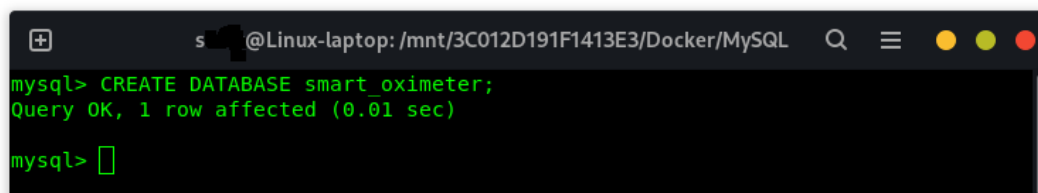
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

Figura 6.5: Accesso col nuovo utente

6.3 Creazione del database

La creazione del database, nel caso non esista già, viene gestita dal backend della webapp. Ma a scopo esemplificativo verrà mostrato come creare da riga comando un nuovo database con il comando `CREATE DATABASE` seguito dal nome del database (come mostrato in Figura 6.6).

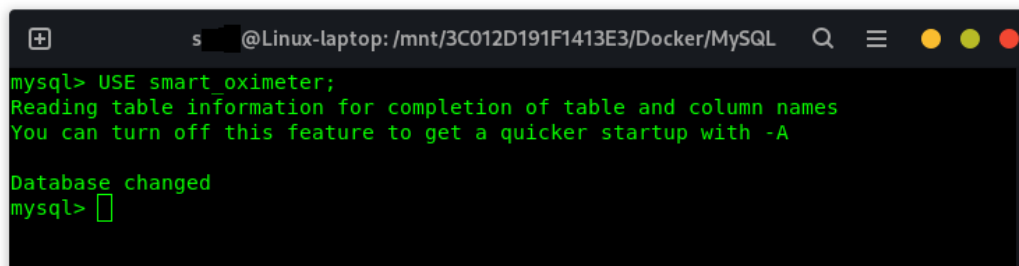
A terminal window titled 's@Linux-laptop: /mnt/3C012D191F1413E3/Docker/MySQL' showing the MySQL command-line interface. The user enters 'CREATE DATABASE smart_oximeter;'. The terminal displays: 'Query OK, 1 row affected (0.01 sec)'. The prompt 'mysql>' is shown.

```
mysql> CREATE DATABASE smart_oximeter;
Query OK, 1 row affected (0.01 sec)

mysql>
```

Figura 6.6: Creazione del database

A seguito della creazione, per iniziare ad operare con il database bisogna selezionarlo con il comando `USE` seguito dal nome del database (come mostrato in Figura 6.7).

A terminal window titled 's@Linux-laptop: /mnt/3C012D191F1413E3/Docker/MySQL' showing the MySQL command-line interface. The user enters 'USE smart_oximeter;'. The terminal displays: 'Reading table information for completion of table and column names You can turn off this feature to get a quicker startup with -A Database changed'. The prompt 'mysql>' is shown.

```
mysql> USE smart_oximeter;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

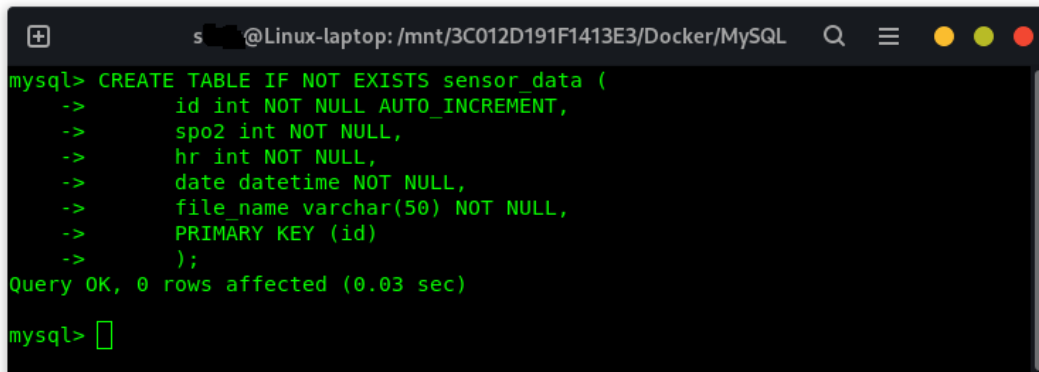
Database changed
mysql>
```

Figura 6.7: Selezione del database

6.4 Creazione della tabella

Il database che fa parte del progetto è costituito da una sola tabella visto che il suo scopo principale è memorizzare i dati provenienti dal sensore.

La creazione della tabella, nel caso non fosse già presente, è gestita dal backend della webapp, ma come per la creazione del database, anche la tabella può essere creata in anticipo da riga di comando grazie al comando `CREATE TABLE` seguito dal nome della tabella e dei campi che la compongono.



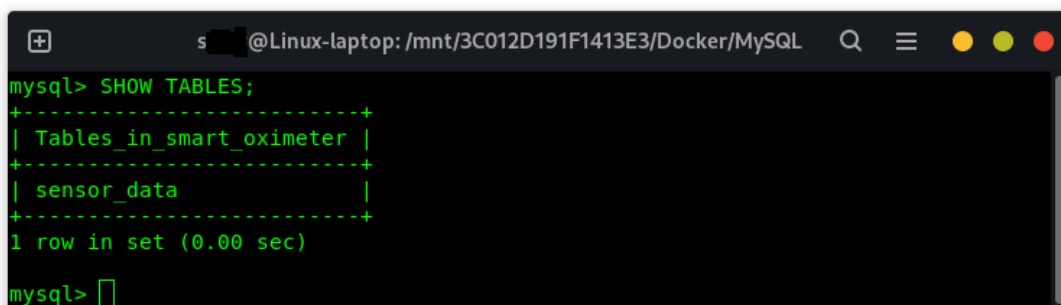
```
mysql> CREATE TABLE IF NOT EXISTS sensor_data (
->   id int NOT NULL AUTO_INCREMENT,
->   spo2 int NOT NULL,
->   hr int NOT NULL,
->   date datetime NOT NULL,
->   file_name varchar(50) NOT NULL,
->   PRIMARY KEY (id)
-> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

Figura 6.8: Creazione della tabella

La tabella è composta da soli cinque campi dove i valori più importanti sono `spo2` e `hr` che sono proprio i campi di riferimento per i valori che arrivano dal sensore. Il campo `id` con incremento automatico funge da chiave primaria. Mentre `date` rappresenta un riferimento temporale al momento in cui i dati sono stati registrati. Il campo `file_name` è utile per raggruppare i dati per registrazione. Questo permette di distinguere in modo netto quali dati vengono da quale registrazione. Un esempio viene mostrato in Figura 6.8.

Una volta creata la tabella è possibile verificare l'effettiva creazione con il comando `SHOW TABLES` (vedi Figura 6.9).



```
mysql> SHOW TABLES;
+-----+
| Tables_in_smart_oximeter |
+-----+
| sensor_data               |
+-----+
1 row in set (0.00 sec)

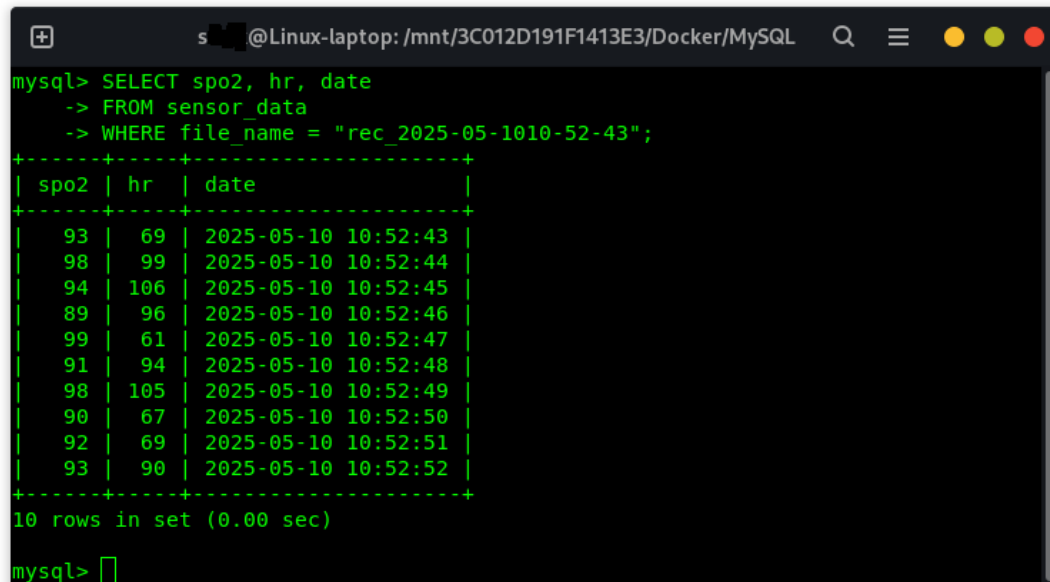
mysql>
```

Figura 6.9: Visualizzazione delle tabelle create

6.5 Visualizzazione dei dati

Dopo aver creato il database e la tabella e dopo che la webapp ha inserito dei dati, è possibile accedere ai dati tramite il comando `SELECT`.

La lista completa delle istruzioni è presente all'interno del progetto, tutte le query di accesso ai dati sono scritte nel file *dbManager.py*. A titolo esemplificativo qui verrà mostrato come consultare i dati tramite riga di comando.



```
mysql> SELECT spo2, hr, date
-> FROM sensor_data
-> WHERE file_name = "rec_2025-05-1010-52-43";
+-----+
| spo2 | hr | date                |
+-----+
| 93    | 69 | 2025-05-10 10:52:43 |
| 98    | 99 | 2025-05-10 10:52:44 |
| 94    | 106 | 2025-05-10 10:52:45 |
| 89    | 96 | 2025-05-10 10:52:46 |
| 99    | 61 | 2025-05-10 10:52:47 |
| 91    | 94 | 2025-05-10 10:52:48 |
| 98    | 105 | 2025-05-10 10:52:49 |
| 90    | 67 | 2025-05-10 10:52:50 |
| 92    | 69 | 2025-05-10 10:52:51 |
| 93    | 90 | 2025-05-10 10:52:52 |
+-----+
10 rows in set (0.00 sec)

mysql>
```

Figura 6.10: Esempio di interrogazione del database

La query in Figura 6.10 è quella utilizzata da uno dei grafici all'interno del progetto per visualizzare i dati. Questa select permette, specificando una precisa registrazione, di recuperare tutti i valori di spo2, hr e riferimenti temporali associati.

7 Front-End

Il front-end è quella parte del software che si occupa dell'interazione con l'utente. Ha lo scopo di interfacciare in modo semplice e tramite interfacce grafiche, ricche di componenti testuali e bottoni, l'utente con il back-end. Questo permette un utilizzo semplice ed immediato del software anche da parte di utenti senza una preparazione tecnica di base nel mondo dell'informatica. Tramite il front-end è possibile fare richieste al back-end e visualizzare le risposte [29].

In questo progetto per la realizzazione del front-end sono stati utilizzati HTML, CSS e JavaScript. Dove con HTML vengono definite le strutture delle pagine web, CSS permette di agire sullo stile delle pagine rendendole esteticamente più accattivanti oltre che responsive (e quindi in grado di adattarsi alle dimensioni di schermi differenti). Infine JavaScript permette di gestire la parte dinamica e quindi la comunicazione con i servizi che compongono il back-end.

La libreria ChartJS ha permesso di semplificare molto la parte di visualizzazione dei dati in quanto permette la renderizzazione dei grafici all'interno di semplici canvas. Anche per questo motivo non è stato necessario utilizzare un framework particolare per realizzare l'interfaccia.

7.1 Struttura del Front-End

Il front-end del progetto è costituito da due pagine html distinte. La prima, la principale, si occupa dei comandi di controllo del microcontrollore e della visualizzazione dello stream di dati in tempo reale. La seconda pagina invece si occupa della visualizzazione dei dati salvati nel database tramite dei grafici.

L'organizzazione dei file all'interno della directory del progetto è mostrata nella Figura 7.1.

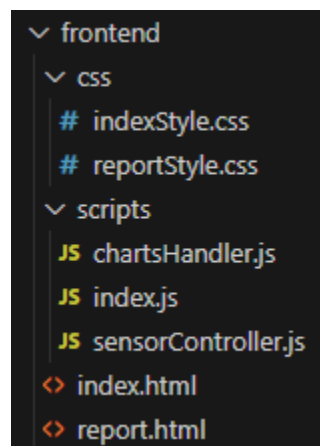


Figura 7.1: Struttura del front-end

Nella root è possibile trovare i file html dove *index.html* rappresenta la pagina principale e *report.html* la pagina che si occupa della visualizzazione dei grafici. Le sottocartelle css e scripts contengono i file css e javascript che sono poi richiamati all'interno dei file html.

In questo elaborato sono state trattate solo le funzionalità introdotte dai file javascript in quanto esse rappresentano la parte dinamica che si occupa dell'interazione con il back-end.

7.2 Interfaccia Principale

L'interfaccia della pagina index è semplice e lineare. Come viene mostrato in Figura 7.2, l'interfaccia è costituita da un menù di navigazione superiore dove Home indica la pagina index stessa e Report la pagina omonima.

Gli indicatori SpO2 e HR vengono utilizzati per la visualizzazione del flusso di dati in tempo reale proveniente dal sensore.

Il tasto Start permette l'avvio del flusso di dati. Al click il tasto commuta al valore Stop indicando il cambio di funzione del tasto stesso.

La checkbox Record viene utilizzata per avviare e terminare la registrazione.

Di default, quando il flusso di dati non è attivo, la checkbox è disattivata. Mentre quando la registrazione è attiva, per interrompere il flusso premendo su Stop bisogna prima terminare la registrazione.

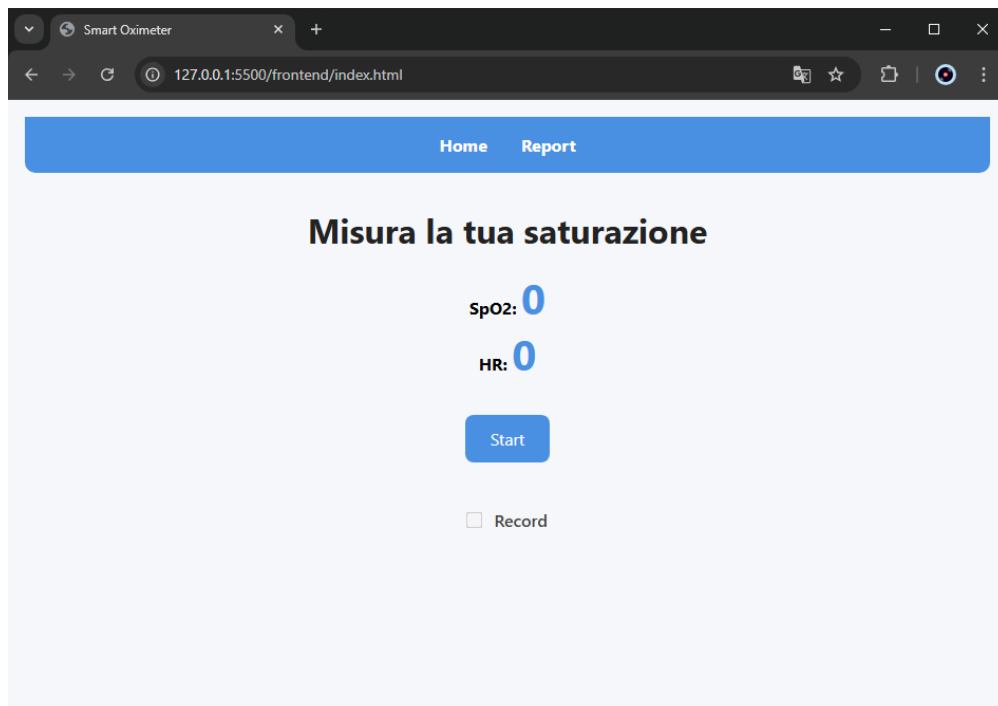


Figura 7.2: Homepage

7.3 Comunicazione con il server

Lo script `sensorController`, come suggerisce il nome stesso, si occupa di gestire le funzionalità che permettono di inoltrare i messaggi per il sensore (inteso come microcontrollore) e la ricezione dei dati.

```
1  const button = document.getElementById("activator");
2  const text = document.getElementById("tex");
3  const check = document.getElementById("record");
4  const spo2 = document.getElementById("spo2");
5  const hr = document.getElementById("hr");
6  const rec = document.getElementById("record");
```

Figura 7.3: Recupero degli elementi HTML

Linee da 1 a 6 (di Figura 7.3): vengono recuperati gli elementi html della pagina rendendoli disponibili all'interno dello script.

```
10  let socket = new WebSocket("ws://localhost:3000", "client");
11  let msg = {
12    stream: "false",
13    rec: "false",
14    sender: "client",
15  };
```

Figura 7.4: WebSocket client

La Figura 7.4 mostra:

Linea 10: JavaScript consente nativamente l'apertura di `WebSocket` come client. In questo caso si effettua la connessione con il server passando un parametro aggiuntivo per comunicare al server che questa connessione indicata come "client" è con il front-end.

Linee da 11 a 15: la variabile `msg` rappresenta il template dei messaggi che il front-end invierà al server. Le informazioni che ha bisogno di comunicare sono appunto i parametri elencati, parametri che il server utilizza per attivare e disattivare il flusso di dati e la registrazione.

```
17  socket.onopen = () => {
18    console.log("WebSocket connected.");
```

Figura 7.5: Gestione apertura connessione

Linea 17 di Figura 7.5: questo evento permette l'esecuzione di funzioni all'apertura della connessione. All'interno di questa funzione anonima vengono gestiti gli event listener del bottone e della checkbox. Gli event listener sono posizionati in questo punto in modo che gli

elementi eseguano le funzioni solo se la connessione va a buon fine mentre restano inerti se per qualche motivo non è possibile creare la connessione.

```
20 button.addEventListener("click", () => {
21     console.log("eseguito");
22     if (button.textContent == "Start") {
23         button.textContent = "Pause";
24         msg.stream = "true";
25         socket.send(JSON.stringify(msg));
26         console.log("stream attivato");
27         check.disabled = false;
28     } else {
29         button.textContent = "Start";
30         msg.stream = "false";
31         socket.send(JSON.stringify(msg));
32         console.log("stream disattivato");
33         check.disabled = true;
34     }
35 });
```

Figura 7.6: Gestione click su bottone

Nella Figura 7.6 si ha:

Linea 20: permette la gestione dell'evento click sul bottone e quindi si esegue ogni volta che viene cliccato il pulsante.

Linee da 22 a 34: in questo blocco sulla base dello stato del pulsante viene comunicato al server di avviare o interrompere lo stream di dati. Questo è permesso grazie all'invio dei valori true/false registrati nel msg template. In aggiunta viene abilitata o disabilitata la checkbox che gestisce la registrazione.

```
37 check.addEventListener("change", (e) => {
38     if (e.target.checked) {
39         msg.rec = "true";
40         socket.send(JSON.stringify(msg));
41         console.log("registrazione avviata");
42         button.disabled = true;
43     } else {
44         msg.rec = "false";
45         socket.send(JSON.stringify(msg));
46         console.log("registrazione stoppata");
47         button.disabled = false;
48     }
49 });
```

Figura 7.7: Gestione cambio di stato checkbox

In Figura 7.7 viene mostrato:

Linea 37: permette la gestione dell'evento change della checkbox e quindi interviene ogni volta che registra un cambiamento di stato.

Linee da 38 a 48: sulla base dello stato della checkbox viene comunicato al server di avviare o interrompere la registrazione. In maniera analoga a prima questo è permesso inviando al server un valore true/false all'interno del msg template.

```
51 window.onbeforeunload = function(event) {  
52     msg.stream = "false";  
53     msg.rec = "false"  
54     socket.send(JSON.stringify(msg));  
55 };
```

Figura 7.8: Gestione cambio/refresh pagina

Linee da 51 a 55 di Figura 7.8: in questo blocco viene intercettato un evento sulla finestra. Un attimo prima di un cambio pagina o un refresh, viene inviato un messaggio al server per dire di interrompere il flusso di dati e la registrazione.

```
59 socket.onmessage = function (event) {  
60     let message = JSON.parse(event.data);  
61     console.log(message);  
62     if (message.sender == "sensor") {  
63         spo2.textContent = message.spo2 ? message.spo2 : "NaN";  
64         hr.textContent = message.hr ? message.hr : "NaN";  
65         rec.textContent = message.rec ? message.rec : "NaN";  
66     }  
67 };
```

Figura 7.9: Gestione ricezione messaggio

In Figura 7.9 si vede:

Linea 59: permette di intercettare la ricezione di un messaggio in arrivo e di eseguire un blocco di codice per la gestione del messaggio stesso.

Linea 60: il messaggio in formato stringa viene convertito in formato json in modo da semplificare la lettura delle informazioni.

Linee da 62 a 66: viene controllato che il messaggio smistato dal server abbia come mittente il sensore. In caso positivo i valori di SpO2 e HR vengono scritti all'interno degli elementi html che ne permetteranno la visualizzazione.

7.4 Visualizzazione dei dati

L'interfaccia della pagina report (mostrata in Figura 7.10) mantiene lo stile della pagina index, pur ovviamente variando nelle funzioni. Il menu di navigazione resta il medesimo, ma questa

volta sia il menu di navigazione che il menu a tendina restano ancorati alla finestra e sempre visibili anche quando durante la visualizzazione dei dati c'è bisogno di effettuare uno scroll down.

Il menu alla voce “Seleziona la registrazione” al caricamento della pagina recupera l'elenco di tutte le registrazioni presenti sul database. Selezionando una delle registrazioni ne vengono recuperati i dati che popolano i grafici.

Un menu di sintesi mostra la media della SpO2 e dell'HR relativi alla registrazione selezionata. Il colore varia in base al valore. Per la SpO2:

- valore < 90: critico, colore rosso
- $90 < \text{valore} < 94$: accettabile, colore arancione
- valore > 94: buono, colore verde

Per l'HR:

- valore < 45 e valore > 130: critico, colore rosso
- $45 < \text{valore} < 60$ e $100 < \text{valore} < 130$: accettabile, colore arancione
- $60 < \text{valore} < 100$: buono, colore verde

Una piccola leggenda riassume lo schema dei colori.

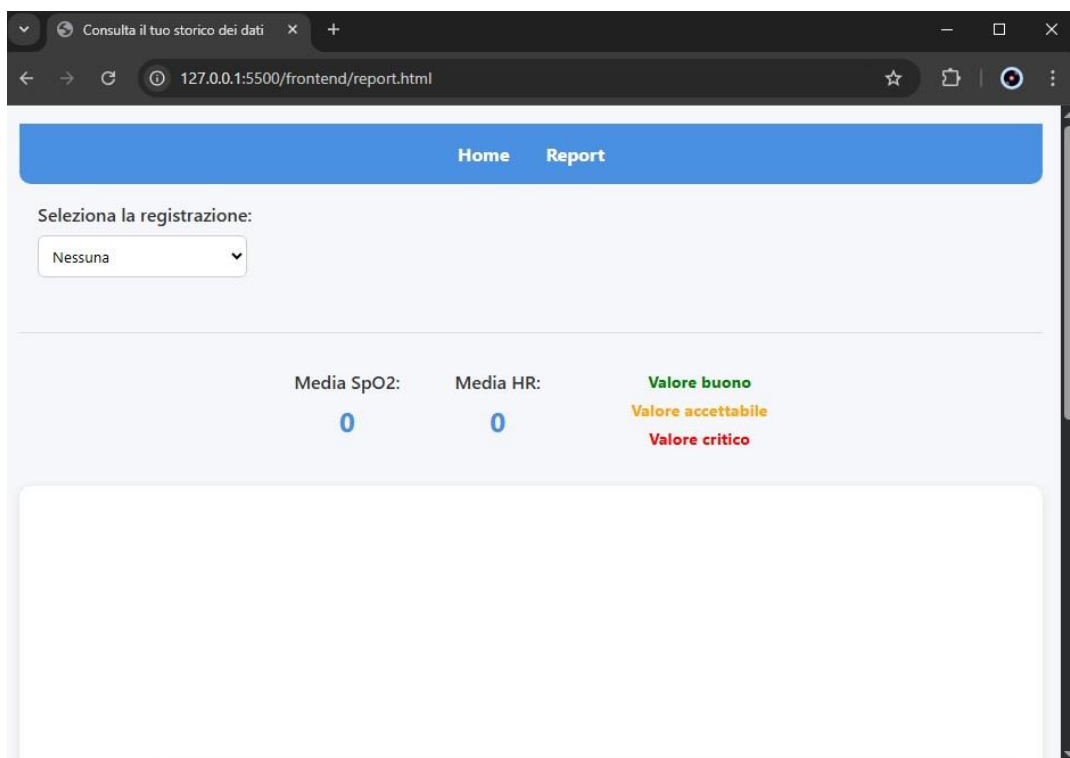


Figura 7.10: Pagina report

A seguire, scorrendo verso il basso nella pagina è possibile trovare 3 differenti grafici.



Figura 7.11: Grafico a linee

Un grafico a linee (Figura 7.11) che mostra l'andamento nel tempo dei valori di SpO2 e HR. Utile per individuare a colpo d'occhio picchi o crolli nella frequenza cardiaca e nella saturazione e risalire alla fascia oraria.

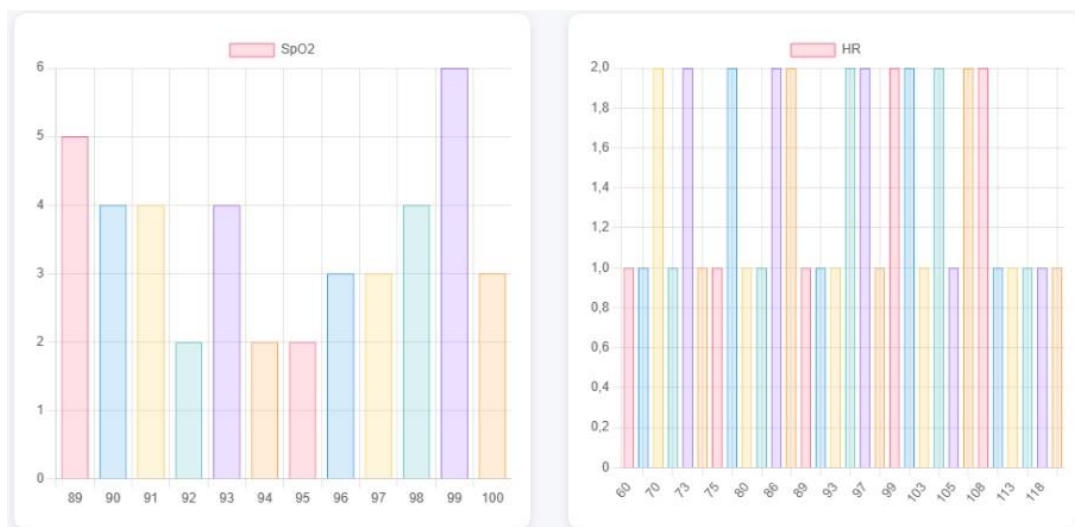


Figura 7.12: Grafici a barre

Due grafici a barre (Figura 7.12) che mostrano il numero di occorrenze di ogni valore, ovvero il numero di volte in cui ogni singolo valore si è ripetuto durante tutto l'arco temporale. Questo può essere utile per determinare a colpo d'occhio se il numero dei valori critici rilevati durante la registrazione è maggiore o minore dei valori buoni.

7.5 Recupero dei dati dal database

Come menzionato nelle sezioni precedenti, il recupero dei dati dal database da parte del front-end passa per un servizio che espone delle REST API. Lo script che si occupa di recuperare i

dati e popolare i grafici è *index.js*. Il compito di *chartsHandler* invece è quello di renderizzare i grafici.

```
1 import { chartHandler } from "../chartsHandler.js";
2
3 const select = document.getElementById("recordings");
4 const lineCtx = document.getElementById("lineChart").getContext("2d");
5 const barSpo2Ctx = document.getElementById("spo2Chart").getContext("2d");
6 const barHrCtx = document.getElementById("hrChart").getContext("2d");
7 const avgSpo2 = document.getElementById("spo2_avg");
8 const avgHr = document.getElementById("hr_avg");
```

Figura 7.13: Recupero degli elementi HTML

Linee da 3 a 8 di Figura 7.13: analogamente a quanto visto prima, anche *index.js* come prima cosa richiama gli elementi html che devono essere utilizzati all'interno dello script.

```
13 try {
14   fetch("http://localhost:3001/charts/filename")
15     .then((response) => response.json())
16     .then((data) => {
17       if (data.success) {
18         data.results.forEach((el) => {
19           const op = document.createElement("option");
20           op.value = el.file_name;
21           op.textContent = el.file_name;
22           select.appendChild(op);
23         });
24       }
25     });
26 } catch (err) {
27   console.log(`errore nel recupero dei nomi: ${err}`);
28 }
```

Figura 7.14: Popolamento del menu a tendina

Nella Figura 7.14 viene mostrato:

Linea 14: *fetch* è un API messa a disposizione da JavaScript per effettuare richieste HTTP e processare le risposte [30]. In questo caso viene effettuata una richiesta GET per ottenere la lista dei nomi delle registrazioni memorizzate nel database.

Linee 15 e 16: effettuata la richiesta, la risposta viene convertita in formato json e ne viene estratto il blocco dei dati.

Linee da 17 a 23: se la richiesta ha avuto esito positivo, tramite un ciclo *forEach*, elemento per elemento, i nomi delle registrazioni vengono aggiunti al menu a tendina.


```

30 select.addEventListener("change", (event) => {
31     console.log(`change event detected, value: ${event.target.value}`);

```

Figura 7.15: Gestione selezione registrazione

Linea 30 di Figura 7.15: aggiunge un event listener al menu a tendina. Ad ogni cambiamento del valore selezionato, quindi ogni volta che viene selezionato il nome di una delle registrazioni caricate in precedenza, il codice all'interno di questo blocco viene richiamato. In questo modo viene gestito il recupero dei dati che vanno a popolare i grafici ad ogni cambio di voce nel menu.

```

33     fetch(`http://localhost:3001/charts/avg/${event.target.value}`)
34     .then((response) => response.json())
35     .then((data) => {
36         let spo2;
37         let hr;
38         // recupera i valori
39         if (data.success) {
40             data.results.forEach((el) => {
41                 spo2 = el.spo2_avg;
42                 hr = el.hr_avg;
43             });
44
45             // varia il colore in base al valore
46             if (spo2 < 90) {
47                 avgSpo2.style.color = "red";
48             } else if (spo2 <= 94) {
49                 avgSpo2.style.color = "orange";
50             } else {
51                 avgSpo2.style.color = "green";
52             }
53
54             // varia il colore in base al valore
55             if (hr < 45 || hr >= 130) {
56                 avgHr.style.color = "red";
57             } else if (hr <= 60 || hr > 100) {
58                 avgHr.style.color = "orange";
59             } else {
60                 avgHr.style.color = "green";
61             }
62
63             // assegna i valori nei relativi span
64             avgSpo2.textContent = spo2;
65             avgHr.textContent = hr;
66         }
67     });

```

Figura 7.16: Recupero e scrittura dei valori medi

In Figura 7.16 si ha:

Linea 33: effettua una GET ad un endpoint che richiede il passaggio di un parametro. Il parametro richiesto è il nome della registrazione. Per recuperarlo si controlla il valore (target) correntemente selezionato nel menu a tendina. Il risultato di questa GET è dato dalla media dei valori di SpO2 e HR di quella specifica registrazione.

Linee da 39 a 61: sulla base dei valori di media ottenuti, si va a modificare quella che è la proprietà style degli elementi html in cui i valori verranno scritti. In questo modo si realizza la logica del cambio di colore che indica quando un valore è buono, accettabile o critico.

Linee 64 e 65: i valori di media recuperati vengono scritti negli elementi html di riferimento.

```
69     fetch(`http://localhost:3001/charts/file/${event.target.value}`)
70     .then((response) => response.json())
71     .then((data) => {
72         const ch = new chartHandler();
73         const chart = ch.drawLineChart(lineCtx, data, formerLineChart);
74         formerLineChart = chart;
75     });
```

Figura 7.17: Creazione grafico a linee

La Figura 7.17 mostra:

Linea 72: viene creato l'oggetto chartHandler che tramite le funzionalità di ChartJS permette il rendering dei dati all'interno di elemento canvas. Per maggiori informazioni sul rendering dei grafici, consultare la documentazione ufficiale della libreria [31].

Linea 73: viene eseguito il metodo definito in chartHandler che va ad effettuare il rendering del grafico a linee. Gli oggetti passati sono una canvas, i dati ottenuti dal database e un oggetto formerLineChart, oggetto che viene utilizzato come riferimento alle vecchie canvas. L'utilizzo di questo oggetto è importante perché prima di poter creare una nuova istanza di Chart bisogna utilizzare il metodo *destroy()* per eliminare le precedenti istanze create. Senza questo accorgimento, dopo aver renderizzato il primo grafico, selezionando una nuova registrazione dal menu, si andrebbe incontro ad un errore.

8 Conclusione e sviluppi futuri

Lo scopo principale del progetto, nonché dell’elaborato, ovvero quello di mostrare uno dei possibili modi per realizzare un sistema che rispecchia l’approccio IoT nel campo della Telemedicina, è stato ampiamente discusso e argomentato.

La realizzazione del progetto nella sua interezza presenta una difficoltà non banale. Mettere in comunicazione dispositivi differenti tra loro richiede diverso tempo e impegno. Da un lato la componente hardware necessita di un approfondimento sulle specifiche e sulle funzionalità del microcontrollore, la configurazione dell’ambiente di sviluppo idoneo per programmarlo e infine la realizzazione dei collegamenti elettrici con il sensore. La parte software richiede delle conoscenze sulle architetture software, sugli stack protocollari e sulle varie tecnologie e framework back-end e front-end. Ma grazie alle conoscenze sviluppate in ambito accademico, ai libri, articoli, manuali e il contributo della community (specialmente in ambito open source) è stato possibile affrontare questa sfida ottenendo i risultati sperati.

Benché il sistema realizzato copra una casistica limitata, lo stesso si presta ad interessanti sviluppi futuri. Al momento il sistema non gestisce connessioni di più utenti o più sensori. Ma con i dovuti accorgimenti e miglioramenti è possibile renderlo compatibile per l’utilizzo contemporaneo di più dispositivi e di più utenti.

Un altro interessante sviluppo futuro è l’integrazione di un sistema di login che distingua il paziente dal personale medico in modo da realizzare una piattaforma strutturata che semplifica l’interazione stessa tra questi due attori. Per lo sviluppo di questo punto occorre mettere in atto misure avanzate di sicurezza e un rigido controllo delle autorizzazioni in quanto si tratta di dati sensibili.

Ma lo sviluppo futuro forse più interessante è quello di estendere la piattaforma a più dispositivi hardware. Ovvero renderla compatibile a collezionare i dati provenienti da più dispositivi medici come apparecchi per il controllo della pressione, della temperatura, della qualità dell’aria e così via. In questo modo è possibile realizzare un vero e proprio ecosistema, che grazie ai dati raccolti e organizzati all’interno della web app, permette di migliorare la qualità di vita delle persone.

Dal punto di vista hardware un interessante sviluppo futuro è sicuramente quello del design di PCB (Printed Circuit Board). Tramite software come KiCad è possibile sviluppare modelli di circuiti stampati che consentono di realizzare dispositivi hardware compatti e ottimizzati e quindi ad esempio realizzare veri e propri dispositivi wearable.

In conclusione, affrontare un progetto che coinvolge diverse tecnologie e diverse metodologie non è stato semplice, ma è stato sicuramente formativo. Approfondire argomenti come le architetture a microservizi, lo sviluppo di API, la configurazione e l'utilizzo di microcontrollori e l'IoT ha permesso oltre al miglioramento delle competenze tecniche un ampliamento del bagaglio culturale in una maniera possibile solo tramite un approccio pratico.

9 Bibliografia

- [1] Andy King, Programming the Internet of Things: An Introduction to Building Integrated, Device-to-Cloud IoT Solutions, O'Reilly Media, 2021
- [2] Michael McCarthy, Barry Burd, Ian Pollock, Concise Guide to the Internet of Things: A Hands-On Introductions to Technologies, Procedures and Architectures, Springer, 2025
- [3] <https://www.salute.gov.it/new/it/tema/telemedicina/i-servizi-di-telemedicina/> consultato in data 03/05/2025
- [4] <https://www.who.int/europe/news/item/10-10-2024-the-rise-of-telehealth-in-the-european-region--insights-from-norway> consultato in data 03/05/2025
- [5] <https://www.jnjmedicalcloud.it/it-it/services/news-center/blt44a7303563636923> consultato in data 03/05/2025
- [6] <https://www.pnrr.salute.gov.it/portale/pnrrsalute/dettaglioContenutiPNRRSalute.jsp?lingua=italiano&id=5833&area=PNRR-Salute&menu=missionesalute> consultato in data 03/05/2025
- [7] <https://www.pnrr.salute.gov.it/portale/pnrrsalute/dettaglioContenutiPNRRSalute.jsp?lingua=italiano&id=5803&area=PNRR-Salute&menu=investimenti> consultato in data 03/05/2025
- [8] https://it.wikipedia.org/wiki/Applicazione_web consultato in data 03/04/2025
- [9] https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol consultato in data 03/04/2025
- [10] <https://it.wikipedia.org/wiki/WebSocket> consultato in data 03/04/2025
- [11] <https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/> consultato in data 5/04/2025
- [12] https://it.wikipedia.org/wiki/Sistema_client/server consultato in data 02/04/2025
- [13] <https://ordr.net/article/iot-healthcare-examples> consultato in data 05/05/2025
- [14] <https://it.wikipedia.org/wiki/Thonny> consultato in data 02/04/2025
- [15] <https://it.wikipedia.org/wiki/MicroPython> consultato in data 02/04/2025
- [16] Eric Elliot, Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries, O'Reilly Media, 2014
- [17] <https://en.wikipedia.org/wiki/Chart.js> consultato in data 02/04/2025

- [18] <https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf> consultato in data 04/04/2025
- [19] J. Allen, "Photoplethysmography and its application in clinical physiological measurement", Regional Medical Physics Department, Freeman Hospital, Newcastle upon Tyne, UK, 20/02/2007
- [20] S. Oak, P. Aroul, "How to Design Peripheral Oxygen Saturation (SpO2) and Optical Heart Rate Monitoring (OHRM) Systems Using the AFE4403", Texas Instruments, 03/2015
- [21] https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html#related-documents consultato in data 04/04/2025
- [22] https://micropython.org/download/ESP32_GENERIC/ consultato in data 04/04/2025
- [23] <https://docs.micropython.org/en/v1.14/library/uasyncio.html> consultato in data 6/04/2025
- [24] <https://github.com/n-elia/MAX30102-MicroPython-driver> consultato in data 10/05/2025
- [25] <https://github.com/dkallen78/PulseOximeter/blob/main/pulse-oximeter.py> consultato in data 10/05/2025
- [26] <https://www.npmjs.com/package/ws> consultato in data 7/04/2025
- [27] <https://www.npmjs.com/package/express> consultato in data 7/04/2025
- [28] <https://www.npmjs.com/package/mysql2> consultato in data 7/04/2025
- [29] https://it.wikipedia.org/wiki/Front-end_e_back-end consultato in data 7/04/2025
- [30] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch consultato in data 7/04/2025
- [31] <https://www.chartjs.org/> consultato in data 02/04/2025