



Progetto di Internet of Things

Realizzato da

Gaia Cigna

Eleonora Giuffrida

Prof. Federico Fausto Santoro

Relazione Finale

Invio di dati da una smartband lungo un tunnel di ancore verso un server centrale

Corso di Laurea Magistrale in Informatica

March 5, 2024

Contents

1	Consegna del progetto	1
1.1	Consegna	1
1.2	Premessa	3
2	Smartband	4
2.1	Descrizione	4
2.2	Progettazione	4
2.2.1	Rete neurale	6
3	Ancore	9
3.1	Descrizione	9
3.2	Soluzione	9
3.3	Il messaggio	10
3.4	Progettazione	11
3.4.1	Prima progettazione	11
3.4.2	Seconda progettazione	12
3.4.3	Terza progettazione	12
3.4.4	Triangolazione	13
4	Gateway	14
4.1	Descrizione	14
4.1.1	Soluzione	14
4.1.2	Progettazione	15

5	Server	17
5.1	Descrizione	17
5.2	Progettazione e soluzione	18
6	Conclusioni	20

Chapter 1

Consegna del progetto

1.1 Consegna

In una struttura industriale, suddivisa da almeno due blocchi, vi si trova un tunnel sotterraneo che consente il passaggio tra un blocco e l'altro. In questo tunnel vi sono delle tubature sulle quali, periodicamente, si devono fare dei controlli di manutenzione. L'ambiente non è uno dei migliori, buio e soffocante, ciò comporta spesso a dei malesseri da parte dei manutentori che vanno ad ispezionare il tunnel, anche solo per attraversarlo. Inoltre all'interno del tunnel non sussiste copertura radio di alcun tipo, nemmeno quella cellulare. Il tunnel è da intendersi privo di ostacoli per i manutentori e non sempre diritto. Si richiede quindi una soluzione IoT per la quale si possa monitorare, non precisamente, la posizione dei fari operatori all'interno del tunnel, considerando anche i tempi di percorrenza e di stallo all'interno dello stesso tunnel, monitorando, inoltre, lo stato dell'operatore (a scelta dei progettisti i valori da monitorare). L'operatore potrà utilizzare o un proprio dispositivo commerciale o i progettisti potranno progettare il proprio dispositivo indossabile. Questo sarà alimentato a batteria, a differenza degli altri dispositivi. All'interno del tunnel sarà possibile piazzare dei dispositivi alimentati direttamente, utili alla risoluzione del problema. Al di fuori del tunnel sarà disponibile una rete IP verso la Intranet aziendale. I dati raccolti dovranno essere conservati all'interno di un database e dovranno essere utilizzati per l'amministrazione. L'amministrazione

dovrà avere a disposizione un servizio con il quale possano monitorare la posizione e lo stato degli operatori, lo stato dei dispositivi e la loro configurazione, inoltre anche lo schedule degli operatori. Il progetto verrà valutato valutando l'operatività della soluzione, i protocolli progettati per la soluzione, l'estendibilità e la semplicità di installazione della soluzione, la qualità e usabilità del servizio per l'amministrazione e l'uso della AI all'interno della soluzione proposta. La soluzione dovrà essere pubblicabile presso il repository del corso, accompagnato da relativa documentazione e relazione di lavoro. Di seguito si riporta un'immagine rappresentativa del progetto:

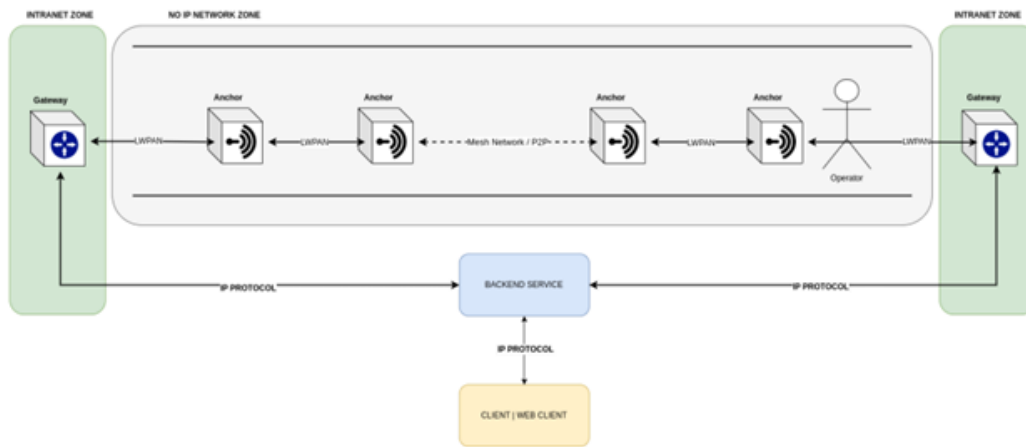


Figure 1.1: Schema esplicativo del progetto

1.2 Premessa

Il seguente progetto è stato implementato su più dispositivi ESP32 usando il framework Platform IO. Si prevede, dunque, che un dispositivo BLE si connetta ad una serie di dispositivi Anchor i quali sono sia collegati ad una rete mesh, che collegati al dispositivo mobile BLE. Gli Anchor, a loro volta, condividono in broadcast il messaggio mandato dal dispositivo BLE all'interno della rete mesh.

Ricordiamo che all'interno del tunnel non vi è connessione alla rete internet, dunque, una volta che i vari Anchor condividono il messaggio, quest'ultimo viene inoltrato ad un gateway che si occuperà di mandare il suo contenuto ad un server. Il server, presi i dati, li manderà alla pagina client in modo che eventuali impiegati esterni al tunnel, possano visualizzare la posizione e lo stato di salute dell'operaio che si trova al suo interno, in tempo reale.[\[Ran\(\)\]](#)

Chapter 2

Smartband

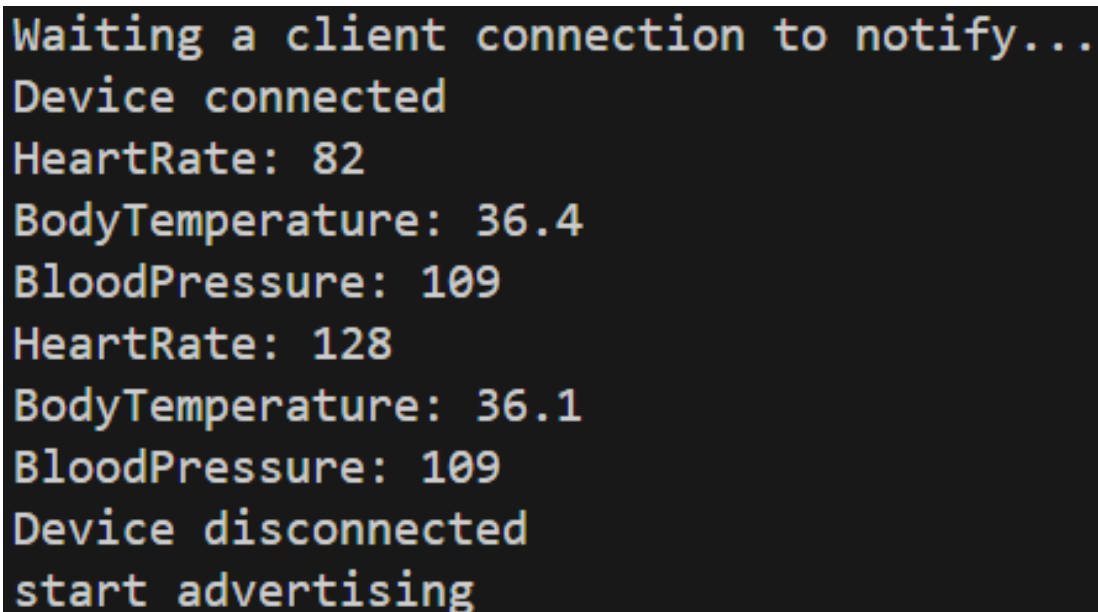
2.1 Descrizione

Con "smartband" si intende un dispositivo atto a raccogliere informazioni sulla salute dell'utente che attraversa il tunnel. In particolare, i dati che si vogliono raccogliere sono la pressione sanguigna, la temperatura corporea e il battito cardiaco. Di conseguenza, tale dispositivo dovrebbe essere dotato di sensori appositi.

Al fine di poter gestire le connessioni di questo progetto e per rientrare nei tempi di sviluppo entro la data di scadenza data, la smartband è stata semplificata quanto più possibile. Nello specifico, non avendo i sensori di cui sopra a disposizione, si è scelto di realizzare un dispositivo che si occupi soltanto di inoltrare dei messaggi su valori numerici randomici con il protocollo BLE. Si è scelto di utilizzare tale protocollo per la sua fondamentale proprietà nell'essere un protocollo a bassissimo consumo energetico, in tal caso, l'ideale per realizzare la smartband poichè si tratta di un dispositivo alimentato a batteria.

2.2 Progettazione

Il codice prodotto per la smartband ha presentato poche, se non nulle, difficoltà nella produzione. Esso si occupa soltanto di condividere con le ancore le quattro caratteris-



```
Waiting a client connection to notify...
Device connected
HeartRate: 82
BodyTemperature: 36.4
BloodPressure: 109
HeartRate: 128
BodyTemperature: 36.1
BloodPressure: 109
Device disconnected
start advertising
```

Figure 2.1: Fig 1: Un esempio di output del braccialetto

tiche, una per ciascun valore di salute, tramite connessione BLE. La quarta caratteristica è stata aggiunta in seguito per condividere l'esito, in booleano, della rete neurale. Tale condivisione avviene ogni 10 minuti con l'ancora più vicina a cui il braccialetto è connesso. L'idea di base che si vuole realizzare è di consentire alla smartband di connettersi in maniera consecutiva tra un'ancora e l'altra ogni qualvolta l'operaio si sposti lungo il tunnel.

Il codice è sviluppato dimodochè, anzitutto, la smartband si connetta alla prima ancora vicina che in ogni istante effettua una scansione dei dispositivi BLE nelle vicinanze. All'allontanarsi da un'ancora, il dispositivo del braccialetto perde la connessione con la prima per riconnettersi alla seconda successiva e ricominciare a inoltrare messaggi.

I dati che le ancore ricevono grazie alla notify di BLE, sono in `std::string` e vengono inviati uno per ogni caratteristica. Sarà poi l'ancora a occuparsi di mettere insieme ciascun valore per combinarli in un unico messaggio.

2.2.1 Rete neurale

Nella progettazione della smartband, è stato implementato il codice che prevede l'uso di una rete neurale, il cui scopo è quello di saper distinguere se un operatore sta bene o sta male, in base ai suoi valori del battito cardiaco, temperatura e pressione sanguigna.

Creazione del file di training

Tramite codice python, è stato creato un file training.csv al cui interno vi sono 10000 righe con valori di heartRate, temperature e bloodPressure nella norma. A questi valori, viene poi assegnato un ulteriore valore finale che indica lo stato di buona salute dell'operatore.

Nello stesso file figurano altre 10000 righe con valori che invece non rispettano i normali valori sopracitati.

I valori finali utilizzati sono:

- "0": l'operatore sta bene;
- "1": l'operatore sta male.

Creazione e addestramento della rete neurale

Per la creazione e l'addestramento, è stata utilizzata una rete neurale sequenziale tramite l'ausilio della libreria Keras.

Con

```
"model.add(Dense(32, input_dim = len(predictors),  
activation = 'relu'))"
```

si aggiunge uno strato "denso" alla rete neurale. Ogni nodo è connesso a tutti i nodi dello strato precedente. Questo strato ha 32 neuroni e come attivazione abbiamo usato "relu" (Rectified Linear Activation), comunemente usata per attivare le reti neurali.

Successivamente è stato aggiunto un altro strato "denso" alla rete neurale con un solo neurone e con funzione di attivazione "sigmoid". Un'attivazione sigmoide è comunemente usata nell'ultimo strato di una rete neurale binaria, poiché fornisce un'output

compreso tra 0 e 1, che può essere interpretato come la probabilità di appartenenza a una classe.

Osserviamo che, nel nostro caso, sarebbe stato necessario usare proprio l'attivazione "sigmoide" perché la rete neurale deve distinguere appunto due casi: "L'operatore sta bene (0)", "L'operatore sta male (1)".

Con

```
model.compile(loss='binary_crossentropy',
              optimizer="adam", metrics=['accuracy'])
```

viene compilato il modello. L'algoritmo di ottimizzazione usato è "adam", la funzione di perdita (loss) è "binary_crossentropy", usata per i problemi di classificazione binaria, ed infine le metriche da monitorare durante l'addestramento.

Infine, per verificare come fosse strutturata la rete neurale e monitorare l'accuratezza della rete nel riconoscere i valori nella norma e quelli fuori norma, viene richiamato il metodo "model.summary()".

Implementazione nel codice di "Smarband"

Una volta addestrato adeguatamente il modello, quest'ultimo è stato poi incluso nel codice della Smartband. Quindi, è stata creata un'istanza della classe Tflite denominata "ml" (che sta per machine learning):

```
"Eloquent::TinyML::Tflite<3, 1, TENSOR_SIZE> ml;"
```

Tflite è una classe definita nella libreria "EloquentTinyML.h" per lavorare con modelli di machine learning basati su TensorFlow Lite. Si usa TensorFlow Lite (al posto di TensorFlow) per i dispositivi mobile ed embedded. I valori passati nella classe Tflite sono rispettivamente: InputSize (3), OutputSize (1) ed un valore TensorArenaSize che si riferisca a quanta memoria deve essere allocata ai tensor

```
(TENSOR_SIZE = model_tflite_len).
```

Successivamente viene effettuato l'init all'interno della setup che stampa un messaggio di errore se esso non è andato a buon fine.

Infine all'interno della loop, una volta che i valori di heartRate, bodyTemperature e bloodPressure sono stati correttamente creati, viene richiamata la funzione "ml.predict(inputs, outputs)" che prende i dati di input, li passa attraverso la rete neurale creata per ottenere le previsioni e restituisce i risultati delle previsioni, che possono poi essere ulteriormente elaborati o utilizzati per prendere decisioni. Se la percentuale di predizione sullo stato di salute dell'operatore è maggiore dello 0.8, l'operatore sta male e il valore di monitoraggio dello stato di salute "stamale" viene aggiornato ad 1. Viceversa se l'operatore sta bene avremo "stamale = 0".

Vengono quindi mandati al dispositivo Anchor i valori di heartRate, bodyTemperature e bloodPressure insieme al valore di salute relativo.

Chapter 3

Ancore

3.1 Descrizione

In questo progetto, l'ancora costituisce il fulcro, la sfida più difficile. Di fatto, sono le ancore a dover gestire la più grande mole di lavoro per la comunicazione tra di esse, con il singolo braccialetto e con i due gateway posti a inizio e fine tunnel.

Le ancore sono tra di loro connesse tramite una rete mesh con il protocollo LoRa sino ai due bridge, ossia due dispositivi di destinazione dei dati, in questo caso costituiti dai gateway stessi, posti all'inizio e alla fine del tunnel. Inoltre, mantengono la loro connessione BLE per trovare i braccialetti nel tunnel, pronti per connettersi e inviare i valori di pressione sanguigna, battito cardiaco e temperatura corporea.

3.2 Soluzione

Il codice ottenuto dal duro lavoro di questa fase di progettazione è senza dubbio quello più complesso e lungo.

La prima cosa di cui esso si occupa è certamente la scansione dei dispositivi BLE vicini. In particolare, per semplicità, le ancore conoscono a priori il nome del dispositivo a cui vogliono connettersi. Tale scelta è derivata da una questione di tempistiche, di fatto, in questo modo la scansione è immediata e non perde neppure un secondo a trovare e

connettersi al braccialetto. Una volta connesso, l'ancora si occupa di combinare insieme i valori di salute dell'utente per costruire il messaggio che da qui verrà inoltrato a tutte le altre ancore in broadcast.

Dopo la costruzione del messaggio, avviene la setup della rete mesh creata con LoRa e il suo conseguente inoltro verso il bridge, cioè il gateway, e verso tutte le altre ancore.

[[DavideFa\(\)](#)]

3.3 Il messaggio

La struttura del messaggio è quanto più semplice possibile, pensata per facilitare il suo continuo inoltro lungo tutto il percorso dal braccialetto stesso sino al server.

Come detto poc'anzi, la connessione BLE aggiorna i dati che vengono di volta in volta ricevuti come `std::string`. La connessione LoRa, però, pretende l'inoltro di tale messaggio in byte. Il numero di byte che LoRa può inoltrare è però, in questo caso, molto piccolo: non più di 50 byte di stringa per volta. La soluzione a tale problema è stata quella di combinare ogni valore numerico in un ordine fisso all'interno di una stringa `std::string`. Ciò significa che ogni informazione utile al server è stata combinata in un messaggio che avesse una struttura che il server stesso conosce e sa come leggere. L'ordine con cui è stato formato il messaggio è come segue:

1. Anchor ID,
2. Position,
3. Heart Rate,
4. Body Temperature,
5. Blood Pressure.

Ciascuna di queste informazioni è separata l'una dall'altra dai due punti. Ovviamente, si tratta di soli valori numerici, talvolta interi, talvolta decimali, e nessun altro tipo di carattere. Questa scelta ha anche reso possibile la conversione del messaggio in byte per l'inoltro con il protocollo LoRa.

L'ultimo valore inserito per formare l'intera stringa del messaggio è un valore binario (0 o 1) che viene prodotto dalla rete neurale. Se è 0, l'utente non ha valori di salute preoccupanti, 1 altrimenti. In totale, quindi, nel messaggio vengono concatenati un totale di 6 valori numerici da conservare nel database.

3.4 Progettazione

Prima di riuscire a giungere alla soluzione appena descritta, il numero di versioni del codice e di protocolli pensati e applicati per il funzionamento, non è affatto piccolo. Le difficoltà riscontrate durante la lunga fase di progettazione del codice dell'ancora hanno prodotto versioni tra di esse molto differenti. Di seguito si riportano le progettazioni precedenti.

3.4.1 Prima progettazione

La prima versione del dispositivo Anchor, prevedeva la semplice implementazione di PainlessMesh e BLE. Per prima cosa gli Anchor cercavano il dispositivo BLE al fine di prenderne i dati. Nel setup si inizializzavano sia il protocollo BLE che il protocollo mesh. Una volta preso il messaggio, i dispositivi IOT, si connettevano alla rete mesh usando i rispettivi MAC address, per poi usare la funzione `mesh.sendBroadcast(message)` al fine di condividere il messaggio tra i vari nodi. Infine, gli anchor posti all'estremità del tunnel, avrebbero inoltrato il messaggio al dispositivo Gateway che a sua volta avrebbe implementato la rete PainlessMesh per poi inviare il messaggio al server tramite Wi-Fi.

Output risultante

Nel Monitor Seriale, appariva che il dispositivo Anchor si connetteva solo al protocollo BLE, non creando mai una rete mesh. Questo è accaduto perché il singolo nodo Anchor non riusciva a ricevere e trasmettere dati implementando contemporaneamente BLE e PainlessMesh, poiché gli ESP32 hanno un unico modulo di connessione di 2.4 GHz che è condiviso sia Bluetooth (dunque sia BT che BLE) che da Wi-Fi.

3.4.2 Seconda progettazione

Si seguiva la stessa soluzione esposta in “Progettazione 1” con la differenza che questa volta si implementava una funzione che, tramite l’uso di un timer, gestiva la condivisione del modulo di connessione del dispositivo ESP32 tra BLE e PainlessMesh.

Output risultante

Nel Monitor Seriale, appariva che il dispositivo Anchor riusciva a connettersi tramite BLE e successivamente alla rete mesh, tuttavia sono immediatamente insorti problemi relativi alla sincronizzazione tra i nodi Anchor.

3.4.3 Terza progettazione

Dopo le diverse problematiche riscontrate dalle progettazioni precedenti, si è deciso di introdurre le task e, di conseguenza, di uno scheduler che le avrebbe eseguite nel corretto ordine.

Questa nuova implementazione prevedeva 3 funzioni di gestione principali, una per ciascuna task specifica:

- Una funzione `handleSync()` il cui compito era occuparsi della sincronizzazione tra i dispositivi Anchor. Finché il numero di nodi non raggiungeva il massimo numero stabilito, si stampava una lista di nodi connessi. Al raggiungimento di tale numero, si passava a disattivare la rete mesh e attivare contemporaneamente la connessione BLE con `BLETask.enable()` in modo da trovare in qualsiasi parte del tunnel un operatore.
- • Una funzione `handleBLE()` che si occupava di implementare la stessa logica pensata prima per la connessione BLE, ma con alcune modifiche. Innanzitutto non si cercava più per un dispositivo Bluetooth generico, ma per lo specifico dispositivo rinominato “SmartbandEsp32”. Se un Anchor non trovava alcun dispositivo “SmarbandEsp32”, dopo un certo lasso di tempo si raggiungeva un timeout per la connessione BLE, dove si disabilitava la connessione Bluetooth e si attivava

la connessione alla rete mesh. Infine se l'Anchor trovava la "SmarbandEsp32", semplicemente i dati di quest'ultima venivano salvati all'interno di un messaggio definito globalmente dall'Anchor, in modo da aggiornarne il valore. Una volta finita la comunicazione tra i due dispositivi, l'Anchor si occupava di chiudere la connessione BLE e lasciare lo spazio per la connessione mesh.

- Una funzione `handleMesh()`, la quale si occupava di gestire la connessione alla rete mesh. Al suo interno si faceva un controllo in cui si stampava la lista dei nodi connessi. Dopo di che, se un Anchor aveva ricevuto il messaggio, questo lo si mandava in broadcast agli altri nodi connessi in attesa di riceverlo.

Output risultante

Nel Monitor Seriale, apparivano i messaggi di controllo della sincronizzazione. Essa avveniva, ma accadevano due problemi fondamentali:

- I nodi si connettevano tra loro solo per pochi secondi e poi, nonostante la rete mesh fosse ancora attiva, essi non riuscivano più a mantenere la connessione tra loro.
- Per i dispositivi che si connettevano al protocollo BLE, a volte succedeva che il loro modulo di connessione rimaneva occupato nel protocollo BLE, nonostante questa problematica fosse stata gestita, impedendo alla rete mesh di inicializzarsi.

3.4.4 Triangolazione

Un'altra idea, durante la fase di progettazione, per rilevare la posizione dell'utente nel tunnel è stata quella della triangolazione.

L'algoritmo di triangolazione prevedeva l'utilizzo del valore rssi di ciascun anchor. Questa implementazione richiedeva che fossero connessi almeno 3 anchor, in modo che la posizione esatta dell'operatore potesse essere trovata con un algoritmo di triangolazione basato sul calcolo dell'inverso del quadrato.

Chapter 4

Gateway

4.1 Descrizione

Il gateway costituisce l'anello di congiunzione tra server e ancore. Con le ultime stabilisce la connessione mesh LoRa, con il server, invece, si occupa di inoltrare il messaggio con una semplice connessione HTTP, come payload di una richiesta POST.

4.1.1 Soluzione

Il codice del gateway risulta sicuramente meno complesso di quello dell'ancora, ma non è stato comunque sottovalutato.

Dopo la ricezione come bridge del messaggio da parte delle ancore, il compito del gateway è di riconvertire nuovamente il messaggio da byte a String. In seguito sarà spiegata la ragione di tale scelta.

Il messaggio passa ora alla connessione HTTP con il server, il quale attende già richieste da parte di eventuali client. Una volta stabilita la connessione, viene effettuata una richiesta POST che include il messaggio ricevuto dalle ancore e stampa il codice di risposta 200 se è tutto avvenuto correttamente. Non appena il messaggio è inoltrato, viene chiusa la connessione col server e si rimette in attesa di altri messaggi dalle ancore riconnettendosi alla mesh e ricominciando da capo il ciclo.

L'inoltro del messaggio come payload di una richiesta POST avviene grazie all'omonima

funzione della libreria HTTPClient.h, la quale pretende come tipo di messaggio, lo String. Di conseguenza, la scelta della conversione del messaggio da byte a String è stata obbligata.

4.1.2 Progettazione

Conseguentemente alle difficoltà riscontrate nella progettazione delle ancore, la produzione della soluzione appena proposta è stata partorita dopo diverse altre versioni. Verranno di seguito descritte.

Prime progettazioni

La prima versione del gateway era un codice che si occupava di due semplici compiti: la connessione alla rete mesh e la connessione con il server. La prima era gestita da una task apposita che aveva come unico compito quello di connettersi a tutte le altre ancore tramite la libreria painlessmesh.h per ricevere da essere il messaggio inoltrato dal braccialetto. Una volta stabilita la connessione con la mesh e ricevuto il messaggio, partiva la seconda connessione, cioè quella con il server, anch'essa grazie a una task. Quest'ultima avveniva con il protocollo http, includendo le librerie WiFi.h e HTTPClient.h. Il codice aveva una struttura lineare. Nella setup veniva inizializzata la mesh e le task venivano inserite nello scheduler, la prima delle quali, la MeshTask, veniva abilitata esattamente in questo punto del codice. La loop si occupava soltanto di eseguire lo scheduler.

Nella parte più alta del codice, vengono settati il ssid, la password e la porta per l'inizializzazione della rete mesh, successivamente troviamo il ssid, la password, l'indirizzo del server e la relativa porta che stabilisce la connessione con la rete WiFi verso il server.

La prima task basa la sua esecuzione sulla funzione handleMesh, la quale aggiorna la rete mesh e invia in broadcast il messaggio che ha ricevuto dalle altre ancore. Se il messaggio non è vuoto, viene disabilitata la MeshTask e abilitata la WiFiTask. Quest'ultima fa eseguire la funzione handleWiFi, nella quale viene connesso il gate-

way alla rete WiFi.

Con l'introduzione della task di sincronizzazione delle ancore, è stata inserita successivamente anche nel gateway la stessa task. In questo modo, le ancore e il gateway avrebbero iniziato la loro esecuzione contemporaneamente e in modo sincronizzato. La `SyncMeshTask` è stata introdotta nel gateway come task iniziale, prima ancora della `MeshTask`. Essa è stata inizializzata in modo da essere eseguita solo una volta, come nelle ancore, per l'inizio dell'esecuzione di tutti i codici in sincro tra ancore, braccialeto e gateway.

Chapter 5

Server

5.1 Descrizione

Tutti i messaggi ottenuti dal braccialetto confluiscono come destinazione finale nel server. Il server gestisce un database atto come contenitore di tutte le informazioni del braccialetto. Esse corrispondono ai campi della tabella “bracelet data”, ossia: id, anchor id, position, heart rate, body temperature, blood pressure, high values.

In ordine, riportiamo il significato di ciascun campo. L’id è una semplice variabile auto increment per elencare e ordinare tutti i dati, fissata come chiave primaria. L’anchor id specifica l’identificativo di ogni ancora. Grazie a questo campo, l’utente dal server può apprendere da quale parte è arrivato il messaggio, e di conseguenza capire in che punto del tunnel si trova l’utente col braccialetto. Con la position, si ottiene la stessa informazione di prima, ma ad ampio raggio, poiché tutte le ancore sono tra di loro raggruppate in aree di tot ancore. Ogni area corrisponde alla position, rappresentata da un numero intero positivo. L’heart rate, la body temperature e la blood pressure corrispondono esattamente agli omonimi dati di salute dell’utente nel tunnel raccolti dal braccialetto. High Values è il campo binario che raccoglie il responso della rete neurale.

5.2 Progettazione e soluzione

Per costruire il server, è stato necessario l'uso di XAMPP, in modo da avere a disposizione sia del database su phpmyadmin sia dell'ultima versione di php. Quest'ultimo linguaggio ha ricoperto un ruolo fondamentale per la costruzione del codice del server.

La prima progettazione del server costituiva un semplice backend molto scarno e privo di CSS. Esso stampava su browser soltanto una table in HTML all'interno della quale venivano caricati tutti i dati accumulati nel database. La table era l'esatta rappresentazione su browser del database installato su *phpmyadmin* nel localhost.

In questa versione iniziale, il codice, seppur corto, quasi da subito ha dato chiari segnali di funzionamento, dopo pochi test. Esso partiva con la connessione al database, inserendo le credenziali di root di XAMPP e poi stampando un eventuale messaggio di errore, in caso si presentasse. Di seguito, è stata implementata in HTML la struttura scarna del frontend, costruendo qui la table. In mezzo alle righe di HTML, in PHP viene eseguita la query che seleziona tutte le righe della tabella bracelet data per stamparle di seguito. In particolare, per ogni riga del database, viene creata una riga nella table in HTML.

Una volta gestita la selezione delle informazioni contenute nel database, si è trattato di gestire la connessione con il gateway. Per fare ciò, è bastato usare la funzione *file_get_contents* di php che riceve il payload di http dal gateway e lo conserva temporaneamente nella variabile *\$data*. Se tale variabile non è vuota, viene effettuata l'explode della stringa ricevuta. Poiché stringa ha una struttura semplice di soli numeri separati da due punti, con la funzione di php explode si separano tutti i dati numerici e si possono inserire nel database con il minimo sforzo. L'inserimento avviene con la tipica query che inserisce ogni valore nei campi rispettivi della tabella del database.

Per permettere che questa soluzione funzioni, è fondamentale anzitutto attivare il database da XAMPP e connettere il server a una rete WiFi e da qui ottenere, con una semplice *ipconfig* da terminale, l'IP privato della rete. Con quest'ultimo, basti scrivere su terminale il comando di php che consente di avviare il server, specificando una porta, tipicamente la 8080. Lo stesso IP è importante che venga scritto anche sul

codice del gateway, separatamente rispetto alla porta, dunque su due variabili globali distinte. Sul browser, scrivendo l'IP privato e la porta, si potrà vedere su schermo la pagina del server in funzione. Per vedere se i dati vengono ricevuti in tempo reale, basta aggiornare la pagina.

La soluzione finale costituisce l'aggiunta di un frontend alla versione iniziale del server e di una pagina iniziale di login in cui l'utente può autenticarsi per accedere alle informazioni.

L'introduzione della pagina di login ha conseguito la creazione di una seconda tabella nel database che si occupi di conservare le credenziali di accesso degli utenti.

Una volta effettuato il login, l'utente ha modo di osservare tutti i dati in tempo reale nella pagina successiva, la dashboard. Tali valori sono tutti raffigurati in una tabella al centro della pagina. La tabella è l'esatta rappresentazione della tabella "bracelet data" del database, con la sola differenza che l'ultimo campo, "high values", non compare. Tuttavia, tale campo è utile poichè, se esso è uguale a 1, il CSS della pagina si occupa di colorare il testo di tutta la riga corrispondente di rosso. Ciò serve per far risaltare alla vista i valori troppo alti rilevati dal braccialetto.

Chapter 6

Conclusioni

Il lavoro in questo progetto è stato ricco di imprevisti, difficoltà e incidenti. Il tempo non è stato dalla nostra parte e per tali ragioni diverse sono le migliorie che si possono ancora applicare.

Per ragioni di tempistiche strette e di insufficienza di materiale, non è stato possibile testare il funzionamento su un numero molto alto di ancore e di braccialetti. In linea teorica, il progetto per come è stato strutturato dovrebbe riuscire a risolvere tale problematica.

Il server può essere ancora migliorato, ad esempio inserendo un'autenticazione più sicura nella fase di login, salvando le password non in chiaro nel database. La pagina che gestisce la tabella di tutti i dati ricevuti può anche essere resa interattiva. Si può aggiungere una cancellazione automatica dei valori del database meno recenti per non saturare lo spazio della pagina nel browser oltre che l'aggiornamento automatico dell'interfaccia.

Un modo per rendere tale progetto più efficiente è di sostituire i dispositivi ESP32 con dei microcontrollori IOT che gestiscano contemporaneamente la rete WiFi e Bluetooth da due moduli differenti: tale soluzione semplificherebbe anche la progettazione.

Bibliography

- [Ran()] 160+ esp32 projects, tutorials and guides with arduino ide. URL <https://randomnerdtutorials.com/projects-esp32/5/>.
- [DavideFa()] DavideFa. Esp32 lora + mesh. URL <https://www.hackster.io/davidefa/esp32-lora-mesh-1-the-basics-3a0920#code>.