

UNIVERSITÀ TELEMATICA INTERNAZIONALE
UNINETTUNO

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in

Ingegneria Informatica

ELABORATO FINALE

in

Ingegneria del software e programmazione ad oggetti

**Monitoraggio di sistemi di produzione industriale, in
ambito industria 4.0, attraverso dispositivi IoT**

RELATORE

Prof.ssa Patrizia Grifoni

CANDIDATO

Leonardo Barea

CORRELATORE

Prof. Mauro Mazzei

ANNO ACCADEMICO 2020/2021

Con la presente volevo spendere qualche riga per ringraziare tutti coloro che mi sono stati vicini e mi hanno supportato in questo lungo percorso durato tre anni fatto di fatiche e gioie che hanno portato oggi a diventare ciò che sono.

Prima di tutti, vorrei ringraziare la mia compagna Elena che mi ha supportato lungo tutto il percorso, aiutandomi e condividendo insieme a me le gioie degli esiti degli esami e trovando sempre le parole giuste per supportarmi durante i momenti più duri. Grazie a te ho imparato che ogni esame può essere superato, anche quando sembra una montagna insormontabile.

Un ringraziamento va anche alla mia famiglia per avermi sostenuto nella decisione di intraprendere questo percorso di studio e portarlo a termine nel migliore dei modi.

Infine vorrei ringraziare due colleghi di corso, Nicholas e Giampiero, i quali hanno fornito un preziosissimo aiuto alla realizzazione del progetto e con i quali ho avuto il piacere di studiare assieme, portando a casa risultati oltre le aspettative.

Questo percorso mi ha permesso di ritrovare la voglia di studiare, la voglia di imparare, ma soprattutto mi ha permesso di capire che con forza e tanta volontà anche le cose più difficili possono essere superate.

Indice

1	Introduzione.....	1
1.1	Industria 4.0.....	2
1.1.1	Industria 4.0 nel mondo.....	3
1.1.2	Industria 4.0 in Italia	3
1.2	Web application.....	4
1.2.1	Protocollo http	5
1.2.2	Client-Server	6
1.3	Internet of things	6
1.3.1	Industrial Internet of Things	7
2	Panoramica software e framework	7
2.1	Kicad	7
2.2	Arduino IDE	8
2.3	NodeJS	9
2.4	MySQL.....	11
2.5	Angular.....	12
2.6	GitHub.....	13
2.7	Grafana.....	14
3	Caso di studio	16
3.1	Scenario d'uso	16
3.1.1	Caratteristiche del progetto.....	16
3.1.2	Focus del progetto	17
3.2	Panoramica progetto.....	17
3.2.1	Struttura del progetto.....	18
4	Client hardware	20
4.1	Schema elettrico	20
4.1.1	Alimentazione	20
4.1.2	Controllo led.....	24
4.1.3	Gestione pulsanti START/RESET	24
4.1.4	Gestione proximity	25
4.2	ESP32.....	26
4.2.1	Principali caratteristiche	27
4.3	Firmware	29
4.3.1	Librerie	29
4.3.2	Dichiarazione variabili	30
4.3.3	Connessione al wifi	30
4.3.4	Impacchettamento ed invio del messaggio.....	31

5	Server NodeJS	33
5.1	Moduli utilizzati	33
5.1.1	Express	33
5.1.2	Sequelize	34
5.2	Codice.....	34
5.2.1	Importazione di moduli, middleware e file custom	34
5.2.2	Definizione delle CORS policy	36
5.2.3	Definizione porta del server	36
5.2.4	Definizione route	37
5.2.5	Ricezione dati da client hardware.....	38
5.2.6	Configurazione DB.....	39
5.2.7	Modello dati	41
6	Database MySQL	43
6.1	Creazione nuovo utente	43
6.2	Creazione database	47
6.3	Opzionale – visualizzazione su Workbench.....	47
7	Front-end Angular	49
7.1	Componenti applicazione	49
7.1.1	AppComponent	50
7.1.2	Menu.....	50
7.1.3	Home	51
7.1.4	Monitoring.....	53
7.1.5	Card	54
7.1.6	Historical	55
7.2	Grafana	55
7.2.1	Grafico PASS/FAIL di ogni postazione	57
7.2.2	Grafico temporale.....	57
7.2.3	Grafico errori	58
7.3	Servizio.....	59
7.3.1	Richiesta dati	59
8	Conclusioni e sviluppi futuri	62
9	Bibliografia.....	64

Sommario

L'industria nel corso degli anni è stata oggetto di diverse innovazioni, ed insieme ad altri settori, l'industria riveste un ruolo fondamentale nella vita di ogni persona, direttamente o indirettamente.

Allo stesso modo l'informatica, settore molto più “giovane” rispetto all'industria, è uno tra quelli più importanti al giorno d'oggi. È il settore che ha visto un incremento esponenziale dell'innovazione, con una pendenza più ripida rispetto a tutti gli altri settori, passando dalla gestione di pochi bit negli anni '50 a milioni di miliardi di bit di oggi.

Questo sviluppo così repentino ha portato l'informatica ad evolversi non solo in verticale (dove ad un certo punto lo sviluppo è, per forza di cose, rallentato), ma anche in maniera, così trasversale, da arrivare in tutti i settori, uno tra tutti l'industria.

L'informatica applicata agli oggetti prende il nome di **IoT** (Internet of Things), ovvero internet delle cose, mentre in particolare l'informatica legata all'industria prende il nome di **IIoT** (Industrial Internet of Things), dall'altro lato invece l'industria che integra sistemi cibernetici, informatici e connessi, si racchiude all'interno dell' **Industria 4.0**.

L'industria 4.0 si basa sul fatto che, se è possibile “connettere” un oggetto di uso comune, come un orologio, una sveglia, allo stesso modo è possibile connettere anche sistemi più complessi come, macchinari, robot e utensili.

Il progetto prevede la creazione di un web server completo, fornito di una parte back-end ed una front-end, per il monitoraggio di impianti di produzione, attraverso un sito dedicato.

Il seguente elaborato ha lo scopo di analizzare nel dettaglio l'implementazione pratica e gli aspetti fondamentali delle tecnologie utilizzate, nonché fornire un valido supporto al lettore per la comprensione e la riproduzione dello stesso.

Nel secondo capitolo vengono trattate le tecnologie software utilizzate per la realizzazione del progetto. In particolare vengono trattati i software utilizzati per ogni livello applicativo, partendo dall'hardware è stato utilizzato **Kicad**, software open source, per la schematizzazione e realizzazione di schede elettroniche (PCB) con strumenti professionali. Permette di generare dei file *Gerber* (file contenenti le posizioni dei fori, del layout, dei componenti, il numero di layer...), con i quali il produttore di schede è in grado di poter poi produrre i PCB.

Arduino IDE, ambiente di sviluppo per l'omonima scheda di sviluppo (o similari), anch'esso open source, permette la programmazione di microcontrollori della famiglia Atmel o Espressif

(come nel caso del progetto in essere), attraverso un'interfaccia semplice con il supporto di una community online molto attiva, in linguaggio Wiring.

NodeJS è la piattaforma utilizzata per lo sviluppo del livello back-end, il server. Anch'esso open-source, sviluppato in tempi recenti, permette, attraverso il motore V8 di Chrome, l'esecuzione di codice Javascript al di fuori del browser.

MySQL è il DBMS (Data Base Management System) utilizzato per la memorizzazione dei dati di produzione. Si avvale di tabelle, attraverso le quali è possibile instaurare delle relazioni semantiche tra diversi attributi della tabella stessa o di altre tabelle.

Angular, framework anch'esso opensource, per la realizzazione di applicativi web, basati su architettura a pagina singola (*single page application*), attraverso la quale tutto risiede in un'unica pagina ed è il contenuto all'interno di essa a cambiare e non la pagina stessa. In questo modo è possibile creare applicazioni più veloci dove soltanto alcune parti della pagina cambiano mentre le altre rimangono invariate.

Github è un servizio di repositoring, gratuito, all'interno del quale è possibile caricare del software (o anche solo parti di codice), per lavorare in team, rendere pubblico il codice scritto, o più semplicemente avere un "contenitore" (*repository*) remoto del proprio codice, accessibile dovunque. Con Github è possibile anche tenere traccia delle modifiche.

Grafana, applicazione web gratuita, permette di accedere a diversi database o sistemi di backend per fornire una rappresentazione grafica dei dati in essi contenuti.

Grafana permette di accedere direttamente al database attraverso delle query, permette di scegliere tra diversi grafici molto accattivanti e ben fatti e lavorare sul layout di questi.

Nel terzo capitolo è stato discusso lo scenario d'uso e le caratteristiche del progetto, in particolare il progetto è stato articolato in modo da essere quanto più scalabile possibile e permette di agire su ogni livello implementativo senza dover necessariamente stravolgere gli altri livelli. Una delle caratteristiche principali, è stata proprio la possibilità di rendere il progetto open-source e accessibile a chiunque lo volesse consultare e migliorare, il tutto accessibile sul repository pubblico su Github.

Gli scopi principali del progetto sono quelli di garantire manutenzione, efficienza e risparmio. Il tutto è reso possibile dalle scelte progettuali effettuate, dal know-how acquisito in ambito industriale e dalle problematiche riscontrate in macchinari di produzione industriale.

Il fine è appunto quello di garantire un maggior controllo sulla produzione per poter intervenire prontamente sui problemi che si presentano ed avere quindi costi di produzione più bassi, minor numero di fermi macchina, maggiore autonomia e programmazione delle manutenzioni.

Nei capitoli successivi vengono descritti nel dettaglio le parti di codice più salienti, necessarie per la comprensione del funzionamento del progetto stesso, il quale prevede più livelli implementativi:

Il primo è quello hardware, dove un microcontrollore si occupa della raccolta dei dati di produzione ed invia il tutto in formato JSON al server tramite il protocollo HTTP, il server recupera i dati, li spacchetta e salva ogni dato ricevuto in un database MySQL. Il sito costruito in Angular effettuerà delle richieste cicliche (*polling*) al server il quale risponderà fornendo i dati di produzione che verranno successivamente mostrati attraverso un browser. Questi dati saranno visualizzabili in tempo reale, con questi che vengono aggiornati costantemente, oppure sarà possibile visionarli in uno storico con grafici e diagrammi temporali.

Le possibilità di sviluppo di questo progetto sono pressoché illimitate, per come è stato costruito, il sito può essere integrato con una sezione di login, è possibile implementare un sistema di notifica Telegram o mail, è possibile standardizzare il firmware in modo che questo possa comunicare attraverso un protocollo standard, il MODBUS RTU, con i PLC installati sui macchinari, in questo modo è possibile monitorare la produzione senza nemmeno dover modificare l'impianto od i macchinari già presenti.

L'informatica, rende oggi possibile molte delle cose che un tempo erano solo utopie, basta soltanto un po' di fantasia, di spirito critico e di tanto studio!

1 Introduzione

Negli ultimi anni lo sviluppo legato ad internet ed alle aziende ICT ha visto una crescita non più soltanto verticale, ovvero legato a determinati campi, ma anche orizzontale, vedendo settori come l'agricoltura e l'industria sempre più a stretto contatto con quello che è il mondo di internet. Un po' per moda, un po' per scopi effettivamente pratici, ad oggi le aziende stanno cercando di mettersi al pari con quello che è il progresso tecnologico.

A favore di questo processo di transizione, il Governo, ha messo e sta mettendo, in atto delle misure volte ad agevolare le aziende stesse ed in particolare le PMI, le quali costituiscono la gran parte delle aziende italiane, che senza agevolazioni o incentivi difficilmente sarebbero propense (o meglio in grado) di affrontare i costi di un tale cambiamento. Di fatti si è di fronte a quella che ormai viene definita 4° rivoluzione industriale.

Negli ultimi 15 anni, si è visto un'importante balzo in avanti anche per quanto riguarda le tecnologie di internet. Sono infatti nati numerosi framework e linguaggi che prima non esistevano e con loro si sono quindi incrementate le possibilità di nuovi sviluppi. Basti pensare che senza javascript le pagine sarebbero ancora statiche o con limitata dinamicità.

L'IoT (Internet of Things) o meglio definito "internet delle cose" è alla base di questo sviluppo e questo progresso tecnologico che vede interessati sempre più settori e campi d'applicazione. Ormai risulta veramente economico implementare, a livello hardware, un modulo Bluetooth/WiFi, per cui ogni dispositivo, elettrodomestico, capo d'abbigliamento ecc ad oggi può essere connesso ad internet.

L'IoT, quindi, è approdato anche nell'industria e nella robotica al fine di migliorare ed incrementare la produzione, nonché per garantire una gestione più accurata della qualità.

Lo scopo del progetto è quello di mostrare un esempio di applicazione dell'IoT in ambito industriale.

Il progetto prevede tutti i livelli d'implementazione di un caso reale di una linea di produzione industriale con più postazioni, dove l'operatore inserisce il pezzo grezzo da lavorare nella prima postazione e preme il pulsante di start. Da quel momento vengono azionati i nastri automatici che porteranno il pezzo da lavorare nelle varie postazioni. Ad ogni postazione sono presenti una serie di sensori, attuatori e attrezzi per la lavorazione del pezzo e per il controllo dei vari parametri di lavorazione (quote, quantità di olio, temperatura...).

Quindi passando all'hardware, si è progettato e implementato il circuito elettronico al quale è collegato il microcontrollore ESP32 (client) il quale, attraverso un firmware apposito, elabora i diversi dati di produzione/posizione del pezzo e parametri di lavorazione e li invia al server NodeJS attraverso una chiamata HTTP POST.

Successivamente il server che è in ascolto, recepisce il dato inviato da ESP32 (nel caso di più ESP32 ognuno sarà un client e tutti invieranno i dati allo stesso indirizzo del server) ed andrà a salvare questi dati in un database MySQL.

Per la parte di front-end è stato utilizzato il framework Angular, il quale attraverso un'operazione di polling effettua delle chiamate HTTP GET cicliche al server per tenere i dati costantemente aggiornati. Infine, nella parte di front-end è stata utilizzata Grafana, un'applicazione web per la creazione di grafici, la quale effettua direttamente delle query al database per i dati da graficare.

1.1 Industria 4.0

Questo rappresenta il tema principale dell'elaborato e rappresenta anche uno dei temi principali dell'innovazione tecnologica che l'Italia sta portando avanti con la transizione 4.0.

Con industria 4.0 si va ad identificare quella che viene chiamata quarta rivoluzione industriale, ovvero quella che integra internet ai processi produttivi al fine di migliorare le condizioni di lavoro, creare nuovi modelli di business, aumentare la produttività degli impianti e migliorare la qualità dei prodotti.

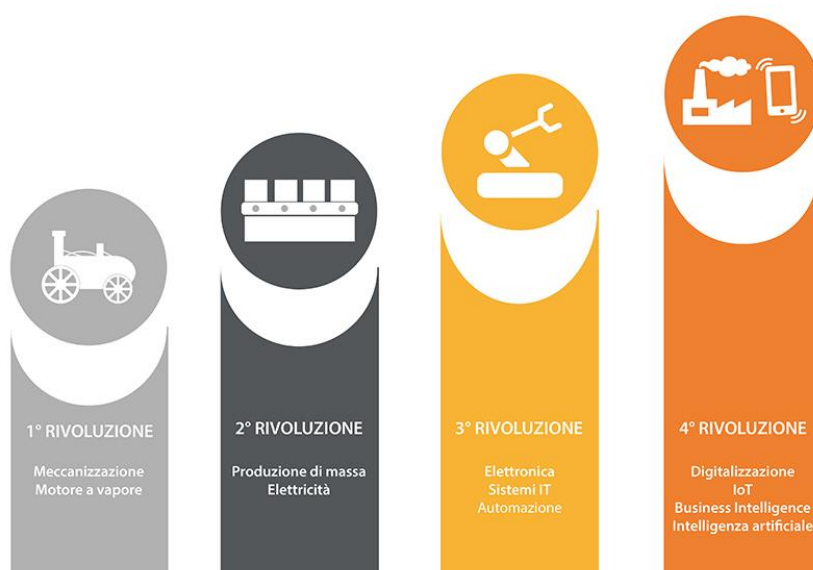


Figura 1.1: Le quattro rivoluzioni industriali nella storia

1.1.1 Industria 4.0 nel mondo

A livello mondiale, il MISE, cita alcune iniziative messe in atto da diversi paesi.

La Francia, ad esempio, ha messo in atto una serie di finanziamenti, prestiti agevolati ed incentivi per consentire alle aziende di effettuare la transizione.

In Germania, oltre ai finanziamenti alle imprese, è in fase di approvazione dal Governo una serie di agevolazioni per investire nelle start-up, spesso iniziatrici di innovazioni tecnologiche importanti.

In Olanda si sta lavorando ad un approccio “*network centric*” in cui il sistema industriale tradizionale si combina con le organizzazioni scientifiche e di ricerca nonché con le istituzioni Governative.

1.1.2 Industria 4.0 in Italia

A livello internazionale invece, il Governo, ha dettato una serie di linee guida per consentire alle imprese di effettuare la transizione tecnologica nel migliore dei modi:

- Neutralità tecnologica
- Operare con azioni orizzontali e non di settore (verticali)
- Orientare strumenti esistenti per favorire il salto tecnologico

Tutte linee guida, basate sulle caratteristiche del settore industriale italiano.

In particolare come nel resto del mondo il piano nazionale prevede degli investimenti alle imprese private ed alle start-up, ma non solo, in quanto lo sviluppo in sé parte dalla scuola. È così che il Governo Italiano, investe nelle competenze, quindi in progetti di Scuola Digitale, percorsi Universitari e ITS, dottorati e ricerca..., tutto questo rientra nel Piano Nazionale Industria 4.0.



Figura 1.2: Prima pagina della presentazione "Piano Nazionale Industria 4.0"

1.2 Web application

L'altro tema principale di questo elaborato sono le **web application**. Molto brevemente una web application, come dice il nome, è un'applicazione (o un insieme di applicazioni) accessibile via web, quindi attraverso internet.

La maggior parte di queste applicazioni utilizzano come terminali i browser, i quali permettono lo scambio di dati attraverso protocolli di rete standard, con server remoti i quali contengono i dati e le pagine dell'applicazione stessa.

Le web application stanno, via a via, diventando sempre più presenti nell'immaginario collettivo e questo è dovuto al fatto che queste non devono avere un supporto per una specifica piattaforma, bensì basta un accesso ad internet, un browser ed il gioco è fatto.

In genere una web app si compone di più livelli, per renderla quanto più scalabile possibile, generalmente si trova un front-end, che è il livello più alto, in cui i dati vengono mostrati all'utente con l'utilizzo di grafica, immagini, video e suoni secondo lo stile definito dal designer. A livello più basso troviamo un back-end, che è il nodo centrale, quella parte che consente la gestione e la manipolazione dei dati e si occupa di gestire le richieste del front-end. Facendo riferimento a questo elaborato, si trova anche un livello di archiviazione, in cui un database memorizza tutti i dati che verranno inviati e gestiti dal back-end.

1.2.1 Protocollo HTTP

HTTP, ovvero Hyper Text Transfer Protocol, è il protocollo principale per lo scambio di dati sul web. È un linguaggio di testo (ipertesto) che lavora con un'architettura di tipo client-server, in cui un client effettua una richiesta (*request*) al server e questo gli risponde con la risorsa cercata (*response*).

Una richiesta HTTP generalmente è composta in questo modo:

```
[method] [URI] [version]
[headers]
[body]
```

Figura 1.3: Struttura di una richiesta HTTP

Dove l'*URI* (Uniform Resource Identifier) è una stringa che identifica in modo univoco una risorsa.

Il metodo (*method*), è la parte più importante. Consente di specificare il tipo di richiesta da effettuare al server da parte del front-end (ad esempio). Qui vengono citati soltanto alcuni dei metodi presenti e più nel dettaglio quelli utilizzati nel progetto:

- GET: è il metodo più utilizzato dal front-end e serve per recuperare una risorsa dal server. Nel progetto è utilizzato per la richiesta dei dati da visualizzare sulla pagina web.
- POST: è un metodo che serve per inviare delle risorse al server. Potrebbe essere utilizzato per inviare i dati di un form dal front-end al server, ma nel progetto è stato utilizzato per inviare i dati dalla scheda hardware al server.
- DELETE: è un metodo utilizzato per eliminare una risorsa dal server (nel progetto non viene utilizzato).

Version rappresenta la versione del protocollo HTTP utilizzato. Ad oggi la versione più recente è la 3.0, risalente al 2018/2019.

L'*header* contiene una serie di meta informazioni utili per il coordinamento tra client e server (come ad esempio lo status code, la lunghezza del pacchetto...).

Infine *body* è il corpo della richiesta HTTP, ovvero quella che conterrà le informazioni.

1.2.2 Client-Server

La maggior parte delle applicazioni web, si basano sul modello architetturale client-server, in cui uno o più client si collegano ad internet e tramite un determinato protocollo (nel progetto è stato utilizzato il protocollo HTTP visto precedentemente) effettuano delle richieste al server e questo vi risponde.

Esistono due tipologie di implementazione di un modello client-server in funzione del numero di livelli su cui l'applicazione è costruita:

- 2-tier: basata su due livelli, il client si connette direttamente con il server;
- 3-tier: nel client sono presenti due livelli, uno che si occupa di gestire la comunicazione con il server, mentre la visualizzazione dei dati è in un livello a parte (livello *thin client*).



Figura 1.4: Struttura modello client - server

1.3 Internet of things

Internet of things tradotto sta per “internet delle cose”, è quella categoria di oggetti di un qualsiasi ambito, che in qualche modo sono “connessi”, ovvero presentano una connessione ad internet.

Questi dispositivi ormai si trovano in ogni cosa, dall’agricoltura, alla domotica, nell’automotive e nell’industria. Con pochi Euro o addirittura meno è possibile acquistare delle schede di sviluppo (*development board*) con a bordo un microcontrollore e dei componenti in grado di collegarsi al wifi e quindi di inviare e ricevere dati attraverso internet.

La particolarità di questi dispositivi è che sono davvero piccolissimi, alcuni di questi sono grandi come una moneta e questo permette di integrarli in qualsiasi dispositivo, come orologi, scarpe, auto e tutto ciò che si vuole.

Il basso costo e le dimensioni ridotte sono i fattori che hanno contribuito alla diffusione di questi dispositivi e quindi l'integrazione in diversi oggetti.

1.3.1 Industrial Internet of Things

Come si può ben capire, con Industrial Internet of Things (I-IoT) si identifica quella branca dell'IoT legata al mondo industriale, cercando di rendere più efficienti i vari processi.

Le applicazioni, in campo industriale, di internet, sono molteplici, come riportato in [14]:

- Smart Factory: controllo avanzamento produzione (che è lo scopo principale del progetto), manutenzione, movimentazione materiali, controllo qualità;
- Smart Logistics: tracciabilità / monitoraggio della filiera tramite tag RFID (Radio-Frequency Identification) e sensoristica, gestione delle flotte (es. tramite GPS);
- Smart Lifecycle: miglioramento del processo di sviluppo nuovi prodotti (es. tramite dati provenienti da versioni precedenti dei prodotti connessi).

Quindi integrandosi ai tradizionali modelli di produzione i dispositivi IoT riescono ad apportare diversi reali benefici. In Italia, purtroppo, nonostante il Governo abbia finanziato fondi a sostegno della transizione tecnologica, poche sono le PMI che ad oggi hanno effettivamente avviato progetti di questo tipo, a differenza invece delle grandi imprese che da questo punto di vista sono più istruite.

2 Panoramica software e framework

Qui di seguito verranno affrontati a livello teorico i principali software e framework utilizzati per portare a termine il progetto. Per la spiegazione dettagliata dell'uso che se ne è fatto di questi si rimanda al capitolo successivo.

2.1 Kicad

Kicad è un software open source gratuito per la progettazione di circuiti elettronici. Integra al suo interno un editor per schematici (*Eeschema*), dove è possibile disegnare il proprio schema elettronico con migliaia di componenti già presenti in decine di librerie, ma se il componente che si vuole aggiungere non è presente in queste è possibile creare il componente o la libreria di

componenti custom in pochi semplici passi come è riportato nei manuali ufficiali [16] e [17] scaricabili nella sezione “Documentation” del sito [15].

È presente poi un editor PCB (*Pcbnew*), il quale grazie alle impronte dei componenti, sempre presenti nelle diverse librerie, è possibile definire il layout del PCB per come dovrà poi essere prodotto.

Per aiutare durante la fase di prototipazione, sul software è anche presente un tool per la visualizzazione in 3D dello stampato con i componenti posizionati come da specifiche di chi lo progetta. Questo è utile per vedere se l'impronta del componente scelto corrisponde con il componente in nostro possesso o meglio per analizzare gli ingombri della scheda finita.

Un'altra funzionalità molto interessante è quella messa a disposizione da Eeschema, dove è possibile simulare il circuito disegnato ed effettuare un test delle regole elettriche, grazie al simulatore SPICE integrato.

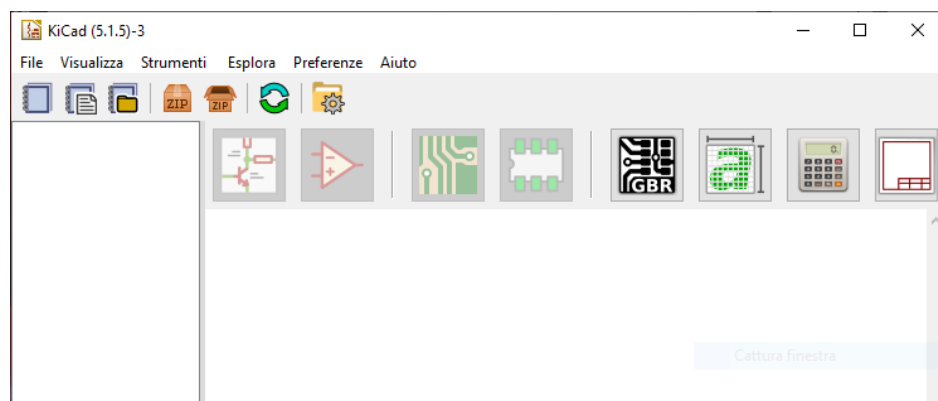


Figura 2.1: Schermata Kicad

2.2 Arduino IDE

L'IDE (Integrated Development Environment) è un ambiente di sviluppo integrato in cui è possibile scrivere del codice ed usufruire di diversi tool per lo sviluppo.

L'IDE di Arduino è un software scritto in Java, che si utilizza per scrivere codice e programmare microcontrollori compatibili con i gestori delle schede installate.

All'interno dell'IDE troviamo come prima cosa una parte del codice già precompilato e questo è sicuramente di supporto ai neofiti ma anche a programmatori più esperti che vogliono evitare di scrivere pezzi di codice ridondante ad ogni nuovo progetto.

Un altro aspetto interessante accanto al gestore di schede è la possibilità di installare librerie direttamente dall'IDE, cercandole per nome e leggendo la descrizione delle funzioni e le schede con cui è compatibile, avendo inoltre la possibilità di selezionare la versione che più si desidera.

Il linguaggio utilizzato per sviluppare su piattaforma Arduino è un derivato del C/C++, il Wiring, semplificato dalle librerie native e questo permette uno sviluppo veloce con una curva di apprendimento relativamente breve, questo è dovuto anche al fatto che le istruzioni da utilizzare prevedono un approccio di alto livello senza dover necessariamente conoscere la struttura fisica del microcontrollore (registri, banchi di memoria...), in quanto tutto ciò è gestito in automatico dall'IDE stesso e dalle librerie.

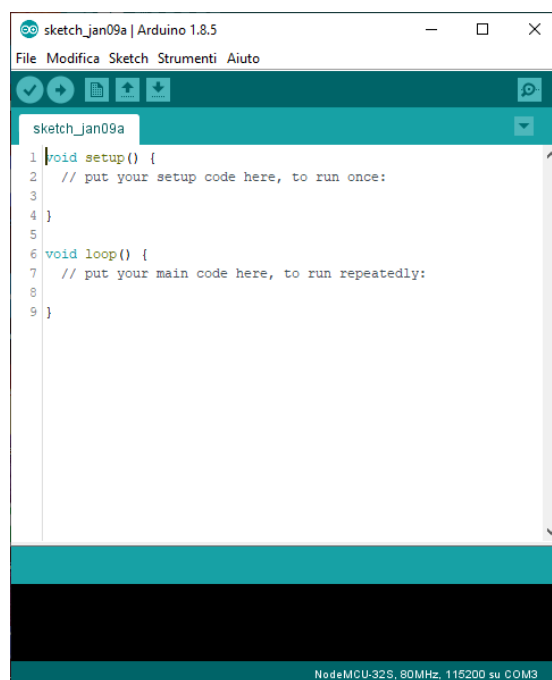


Figura 2.2: Schermata Arduino IDE

2.3 NodeJS

NodeJS è una piattaforma open-source e cross-platform di sviluppo in cui è possibile eseguire del codice Javascript utilizzando il paradigma della programmazione ad oggetti.

Node è nato nel 2009 ed il suo successo è dovuto al fatto che per la prima volta, Javascript è stato utilizzato al di fuori del web. Questo è dovuto al fatto che Node implementa il motore V8 di Chrome, un motore Javascript open-source ad alte performance, scritto in C++, in grado di compilare ed eseguire codice Javascript anche al di fuori del browser.

Un altro motivo di successo di Node è dovuto al fatto che è stato concepito come un gestore asincrono di eventi, infatti ad ogni evento viene richiamata una funzione di callback e questo

permette di ottimizzare l'esecuzione di Node stesso, il quale rimarrà in attesa fino a quando l'evento a cui si era fatta sottoscrizione non emette una notifica, solo in questo caso Node torna operativo ed esegue le funzioni di callback assegnate, tutto ciò che accade prima è delegato al sistema operativo, in modo tale da lasciar libera l'esecuzione di Node per poter gestire eventuali altri eventi. Questa gestione degli eventi è definita non bloccante. Grazie a queste caratteristiche Node è particolarmente adatto a costruire sistemi scalabili e orientati agli eventi.

Nonostante Node sia single-thread, orientato agli eventi e non bloccante, è concesso l'utilizzo di processi figlio utilizzando le API `fork` e allo stesso modo è possibile definire del codice bloccante utilizzando metodi sincroni (ad esempio per la lettura di file di piccole dimensioni o per ragioni di sequenzialità di esecuzione), però essendo un ambiente fortemente sviluppato per applicazioni real-time (sistemi di notifiche push, chat, pubblicazione di dati) l'utilizzo di metodi sincroni viene poco utilizzato o meglio viene utilizzato, in alternativa, con sistemi multi-thread.

Node si compone di innumerevoli moduli, che non sono altro che del codice riusabile, come delle librerie che espongono dei metodi che si possono richiamare dopo averli importati, questi si dividono in tre categorie:

- Interni: scritti direttamente dallo sviluppatore
- Built-in: costruiti internamente a Node
- Esterni: sviluppati da terzi (richiedono di essere installati)

I moduli esterni si possono trovare tutti racchiusi all'interno di **npm** (Node Package Manager), il quale è il più grande contenitore di pacchetti al mondo ed è già installato in Node e grazie alla CLI (Command Line Interface) è possibile, appunto, importare pacchetti presenti in npm scritti da sviluppatori di tutto il mondo. Oltre ai moduli con npm è possibile anche importare dei framework, come *react* ad esempio.

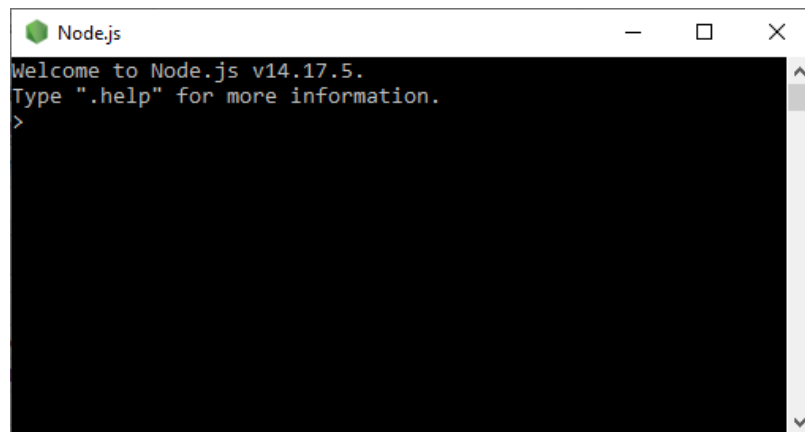


Figura 2.3: Schermata Node.js

2.4 MySQL

MySQL è un software che supporta un veloce, multithreaded, multi utente e robusto server database SQL (Structured Query Language). Il server di MySQL è adatto all'utilizzo con sistemi con un alto carico di produzione.

MySQL è uno dei RDBMS (Relational DataBase Management System) open-source più diffuso ed utilizzato al mondo, era inizialmente di proprietà dell'omonima azienda, ma poi è stato acquisito da Oracle, anch'essa produttrice di DBMS, la quale incorpora MySQL in alcuni dei suoi prodotti.

MySQL è un database relazionale, il che vuol dire che è basato sul concetto matematico/logico di relazione, secondo il quale tutti i dati sono rappresentati da relazioni, infatti vige il modello ER (Entità-Relazione) per la costruzione delle tabelle. Questo permette di associare più dati tra di loro e di trovare un determinato dato in funzione della relazione che questo ha con altri dati.

Con la nuova versione 8.0 di MySQL, ad oggi disponibile, si integra non solo l'SQL ma anche il NoSQL (Not Only SQL), che prima era delegata soltanto ad altri DBMS non relazionali, con l'eccezione che MySQL aggiunge al NoSQL le proprietà ACID (Atomicity, Consistency, Isolation e Durability) alle quali soltanto i DB relazionali facevano riferimento, questo permette di unire la robustezza e l'affidabilità di un DB relazionale con la semplicità, la funzionalità e la scalabilità di un DB NoSQL per creare applicazioni sempre più performanti.

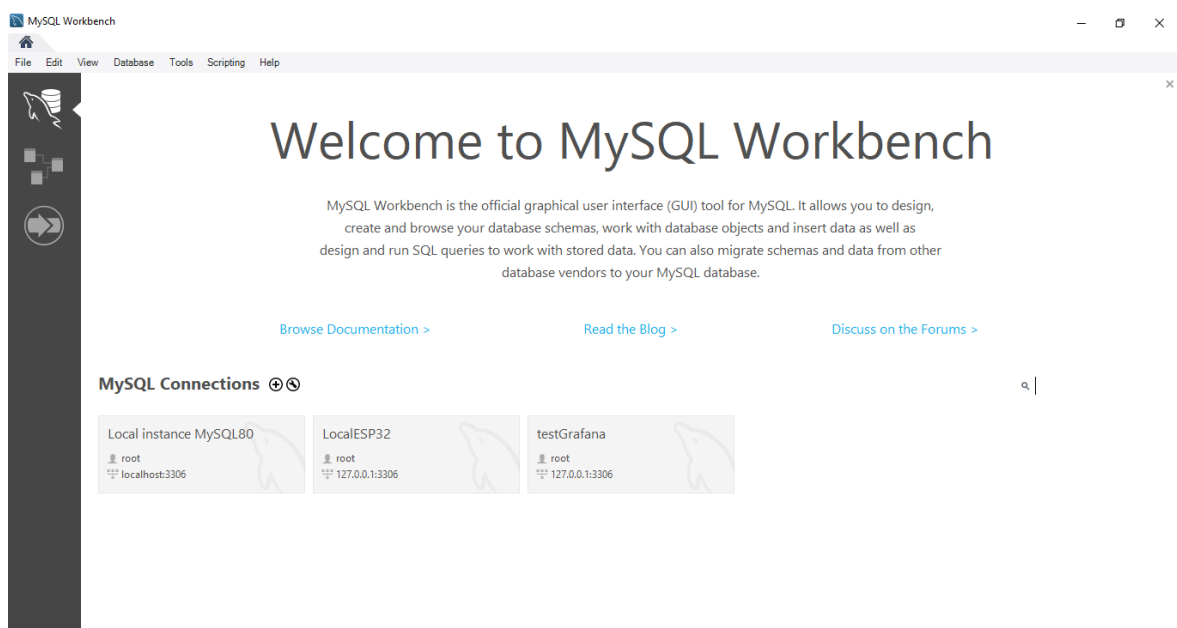


Figura 2.4: Schermata MySQL Workbench

2.5 Angular

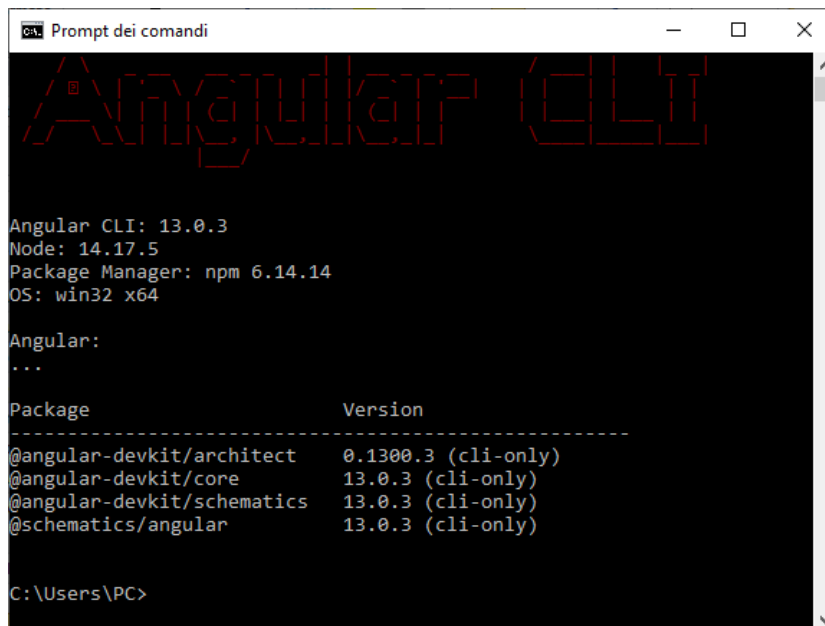
Angular è un framework che permette di costruire applicazioni mono-pagina efficienti e sofisticate. È scritto in TypeScript ed include:

- Framework basato sui componenti, questo permette di creare applicazioni modulari e scalabili, scrivendo poco codice per ogni componente per poterlo riutilizzare in più applicazioni
- Una vasta varietà di librerie che coprono molte funzionalità disponibili (routing, forms, http...)

Con Angular si possono scrivere applicazioni front-end, mono-pagina, il che vuol dire che tutto il sito creato risiederà su un'unica pagina. Questo permette all'applicazione di essere molto veloce e reattiva, in quanto tutte le risorse (tranne i dati) verranno caricati tutti in una sola volta all'apertura del sito, quindi le uniche richieste che verranno fatte al server successivamente, saranno solo per i dati.

Un componente Angular si compone essenzialmente di 4 file:

- `Component.html`: è il template del componente e qui si va a scrivere il codice HTML relativo al componente che si sta creando. In aggiunta ai tag classici, con Angular, è possibile inserire dei costrutti come `if`, `for` e `switch` direttamente nell'HTML;
- `Component.css`: è il foglio di stile del componente, qui è possibile andare a definire le regole di stile. Questo non deve essere importato nell'HTML in quanto già collegato tramite il file `typescript`;
- `Component.ts`: è il file `typescript` del componente, quel file dove è possibile definire il codice operativo (costrutti, metodi, variabili...) per il componente in linguaggio `typescript` e `javascript`, oltre che importare i moduli necessari per il funzionamento del codice e dei metodi usati;
- `Component.spec.ts`: è un file di test per il componente;



```
Prompt dei comandi

Angular CLI

Angular CLI: 13.0.3
Node: 14.17.5
Package Manager: npm 6.14.14
OS: win32 x64

Angular:
...

Package                                Version
-----
@angular-devkit/architect              0.1300.3 (cli-only)
@angular-devkit/core                   13.0.3 (cli-only)
@angular-devkit/schematics             13.0.3 (cli-only)
@schematics/angular                   13.0.3 (cli-only)

C:\Users\PC>
```

Figura 2.5: Schermata Angular CLI

2.6 GitHub

Per garantire una maggiore esportabilità ed un maggiore controllo degli aggiornamenti si è deciso di includere il progetto in un repository locale con Git ed esportarlo anche su un repository remoto con GitHub.

GitHub è servizio di repositoring per progetti software. È possibile pubblicare progetti, software o anche soltanto parti di codice all'interno di repository privati o pubblici, tenere traccia degli aggiornamenti (aggiungendo ad esempio dei messaggi), inserire un file README contenente informazioni di supporto per l'esecuzione del codice.

All'interno del repository è possibile anche vedere i linguaggi utilizzati per la scrittura del codice e la loro percentuale di utilizzo, nonché tutti gli utenti che hanno apportato dei contributi o hanno segnalato dei bug.

Il link al repository pubblico GitHub del progetto si trova al seguente link:

https://github.com/leo96it/Tesi_webApp

Per collegare il progetto con il proprio repository è necessario prima di tutto aver creato un account su GitHub, successivamente è necessario scaricare ed installare Git dalla relativa pagina di download presente al link in appendice [b].

Una volta fatto questo è necessario creare una cartella, nella directory, dove si andrà ad importare il progetto. Fatto questo con Git CMD (Git Command line, ovvero terminale a riga di comando di Git) ci si posiziona all'interno della cartella appena creata e si eseguono i seguenti comandi:

```
⇒ git init    //inizializzazione repository locale
⇒ git clone "https://github.com/leo96it/Tesi_webApp"
    //clonazione repository remoto
⇒ git pull Tesi_webApp main    // importazione progetto
```

Con il progetto ora nel repository locale è possibile eseguirlo seguendo le indicazioni contenute nel file README.md.

2.7 Grafana

Grafana è un'applicazione web open-source che permette di richiedere dei dati direttamente ad un server od ad un database e successivamente di mostrarli nella tipologia di grafico scelta. Sono supportate le connessioni con pressoché tutti i database più conosciuti e la procedura di connessione con il database scelto è molto semplice ed intuitiva.

Esistono due versioni per l'accesso ai grafici di Grafana, la versione cloud ed una versione installabile ed accessibile collegandosi all'indirizzo di default <http://localhost:3000/> (è possibile modificare la porta 3000 con quella che si vuole, andando a modificare il parametro `http_port` nel file `defaults.ini` all'interno della cartella `grafana/conf`).

Con Grafana è possibile creare delle dashboard, all'interno delle quali è possibile scegliere tra una spiccata varietà di grafici (grafici a torta, istogrammi, grafici a barre, tabelle, grafici temporali...).

Per prelevare i dati da un database, è necessario selezionare il database desiderato nella sezione "datasource", fornire i dati di accesso al database e connettersi ad esso. Successivamente sarà possibile creare la dashboard andando a scegliere la tipologia di grafico ed importare i dati dal DB effettuando delle query SQL (nel caso di MySQL).

Per i grafici basati su una serie temporale, sarà necessario che sia presente sul DB anche una colonna con dati di tipo DATETIME (o similari). Per la visualizzazione dell'arco temporale si può scegliere tra quelli disponibili, oppure crearne di personalizzati. In alternativa è possibile

anche agire direttamente con il mouse per ingrandire o diminuire un'area e quindi anche il relativo specchio temporale.

Grafana è in grado di creare alert, che possono essere di solo tipo visivo evidenziando le misurazioni che non rispettano i criteri dettati dall'utente, o attraverso l'emissione di eventi personalizzati, come per esempio un messaggio e-mail o attraverso Telegram.



Figura 2.6: Schermata esempio Grafana

3 Caso di studio

3.1 Scenario d'uso

Il progetto è stato pensato per fornire un valido strumento di feedback per macchinari industriali e per il loro monitoraggio da remoto.

3.1.1 Caratteristiche del progetto

Le caratteristiche del progetto sono:

- Open-source e multiplatforma: il progetto completo è disponibile su GitHub gratuitamente, sia per la parte hardware che per la parte software. L'applicativo front-end può girare su qualsiasi piattaforma sia desktop che mobile in quanto scritta con Angular e come già spiegato nel capitolo 2.5 necessita soltanto di un browser ed un accesso ad internet
- Low cost: per la realizzazione dell'intero progetto, le spese sostenute sono state:
 - Prototipazione PCB: 4,87€/5pz prodotta in Cina da JLCPCB.
 - Acquisto componenti: in totale 10€ acquistati in parte su Aliexpress, Farnell, RS ed Ebay. La BOM (Bill Of Materials) è presente nel repository GitHub.
 - Minuteria e ferramenta varia: nel progetto il tutto è stato composto artigianalmente, in uno scenario reale si dovranno prevedere collegamenti elettrici, involucri e tutto ciò serve per portare i dati dal centro di controllo del macchinario alla scheda utilizzata nel progetto. Inoltre nel caso di un utilizzo reale si dovrà procedere con l'iter burocratico per l'utilizzo di apparecchiature elettroniche autoprodotte.
 - Eventuale hosting del progetto su un dominio acquistato (vedi Aruba, Altravista...)

Tutti i software ed i framework utilizzati sono open-source e gratuiti, pertanto, per il loro utilizzo, non è previsto il pagamento di nessun canone.

- Scalabile: il progetto si basa interamente sul paradigma Client-Server, per cui dato un server, questo può gestire le richieste di diversi Client. Quindi definito il server, questo

sarà in grado di supportare le richieste di più client front-end (per la visualizzazione dei dati) e più client fisici, ovvero più schede collegate a diversi macchinari

- Modulare: utilizzando, sia per il front-end che per il back-end, i paradigmi della programmazione ad oggetti ed un approccio a moduli e componenti, è possibile in qualsiasi istante aggiungere nuove funzionalità all'applicazione senza dover stravolgere il progetto (ad esempio, inserimento di login e password, form per il caricamento di nuovi macchinari e loro caratteristiche...)

3.1.2 Focus del progetto

Molto spesso gli enti di certificazione per la qualità richiedono un archivio degli scarti e delle non conformità, uno degli scopi principali quindi di questo progetto è quello di fornire questo supporto attraverso il database (in questo caso MySQL). Oltre a questo, gli altri benefici che un'azienda potrebbe avere dall'utilizzo di questo progetto sono:

- Manutenzione: tramite l'applicativo è possibile tenere traccia dell'usura dei componenti meccanici di alcuni macchinari (i torni necessitano di un cambio placchette ogni tot pezzi prodotti)
- Efficienza: è possibile verificare il numero di pezzi prodotti in un'ora, una giornata, o una settimana ed andare ad agire prontamente quando questa statistica peggiora, o semplicemente per migliorare la produzione
- Risparmio: nel caso di un corpo macchine molto vasto, è possibile ridurre il numero di persone che vigilano su di esse, fornendo il responsabile di un tablet dove poter collegarsi e controllare in tempo reale lo stato delle macchine ed eventualmente indirizzare le persone giuste per risolvere eventuali problemi meccanici/elettrici tutto da un'unica postazione mobile.

Questi sono soltanto alcuni dei benefici derivanti da questa tecnologia, altri possibili sviluppi sono descritti nel capitolo “Conclusioni e sviluppi futuri”.

3.2 Panoramica progetto

Nel seguente capitolo si andrà ad analizzare nel dettaglio le scelte progettuali e di sviluppo intraprese. Nei prossimi capitoli invece, si analizzerà il codice nelle sue parti più salienti, andando a fare una piccola divagazione sui moduli e sui pacchetti utilizzati.

3.2.1 Struttura del progetto

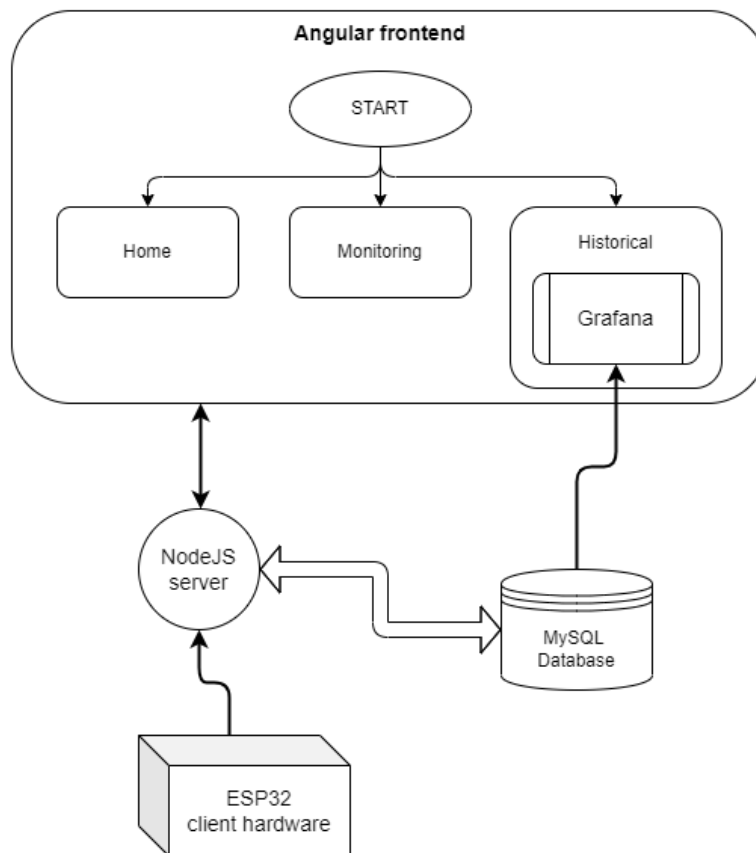


Figura 3.1: Struttura del progetto

Il progetto si compone di quattro macro sezioni principali:

- A. **ESP32:** questo rappresenta la parte hardware del progetto, è il cuore per la gestione del macchinario, si compone di una scheda elettronica accessoria per la gestione di motori, sensori e attrezzature. Nella scheda è installato una dev-board ESP32 con un microcontrollore che ha lo scopo di gestire l'elettronica ad esso collegata e per effettuare le chiamate HTTP verso il server e quindi di connettersi ad internet.
- B. **Server:** scritto con NodeJS, ha lo scopo di recepire i dati di produzione dai diversi client hardware ed instradarli all'interno del database. Il server si occuperà inoltre di "impacchettare" i dati ricevuti dal client hardware in array pronti da essere spediti al front-end, quando ne farà richiesta.
- C. **Database:** scelto MySQL come RDBMS, questo si occuperà del salvataggio di tutti i dati in una tabella, contenente come chiave primaria un ID numerico progressivo e due

colonne riportanti la data di creazione della riga e la data di modifica della riga stessa (in caso la riga non sia stata modificata, la data di modifica sarà uguale a quella di creazione).

- D. **Front-end:** scritto con Angular, ha lo scopo di garantire un accesso remoto ai dati contenuti in server e database. Effettua un polling di chiamate GET al server, il quale in base alla route fornirà i dati di produzione raccolti dai vari client hardware sotto forma di un array di oggetti JSON. Questi verranno mostrati sotto forma di card sulla pagina “Monitoring”.

Il front-end include una sotto sezione aggiuntiva, **Grafana**. Tramite Grafana è possibile visualizzare i dati sotto forma di grafici dal layout accattivante. L’accesso ai dati avviene esternamente all’app scritta in Angular, sarà il server stesso di Grafana, ad effettuare delle query cicliche, direttamente al database MySQL. Quindi Grafana deve essere sempre in esecuzione, così come server e database.

Nei prossimi capitoli verranno analizzate in dettaglio tutte queste sezioni.

4 Client hardware

Come già detto il client hardware è rappresentato da una scheda elettronica per la gestione di sensori, attuatori, motori... il cui centro di controllo e calcolo si trova sulla dev-board ESP32 che verrà trattata nei sotto capitoli a seguire.

4.1 Schema elettrico

Lo schema elettrico ed il circuito utilizzati per questo progetto sono soltanto a titolo esemplificativo, difatti è possibile adattare lo schema ed i componenti in funzione del tipo di macchinario che si vuole decidere di monitorare.

Siccome la maggior parte dei macchinari industriali sono dotati di PLC, con circuito molto più complessi di quello qui sotto descritto, si può optare anche per una soluzione universale con un approccio quanto più plug and play possibile. Questa soluzione verrà descritta nell'apposito capitolo "Conclusioni e sviluppi futuri".

4.1.1 Alimentazione

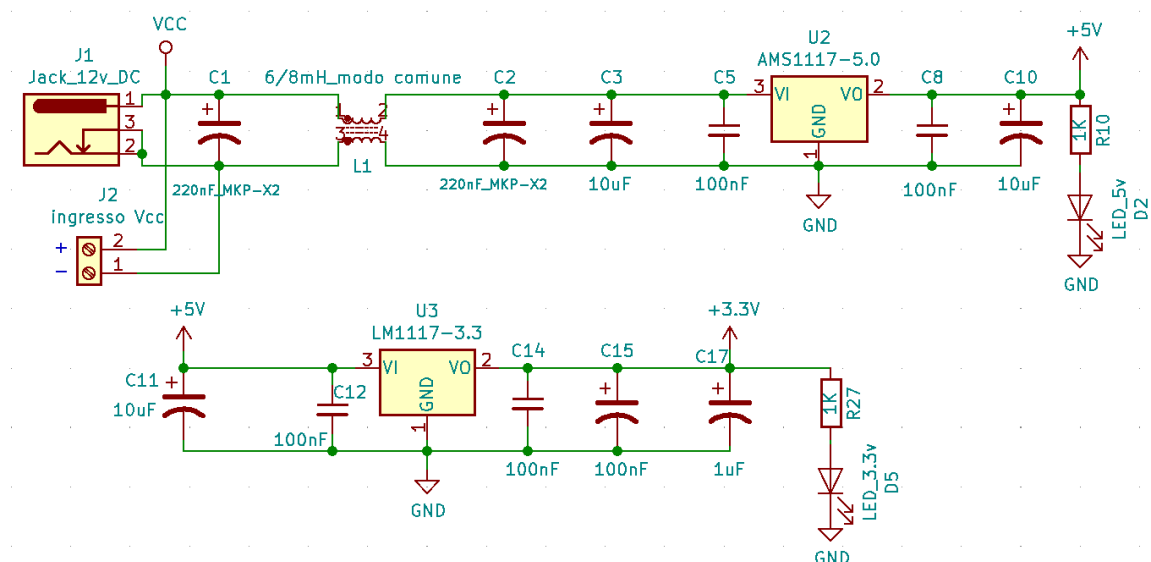


Figura 4.1: Schematico alimentazione

Il circuito di alimentazione si compone essenzialmente di tre parti, una prima parte di filtro in ingresso, una seconda per stabilizzare la tensione intorno ai 5v ed una terza per stabilizzare ulteriormente la tensione intorno ai 3.3v.

La prima parte, come detto, è composta da un filtro in ingresso, in particolare il filtro utilizzato è un filtro pi-greco. Il nome deriva dalla forma dei componenti utilizzati, come si può vedere di seguito:

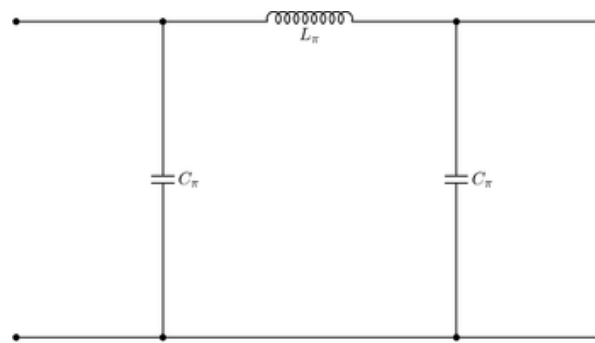


Figura 4.2: Schema filtro pi-greco

In generale, sono presenti due condensatori di filtro ed un'induttanza (nel circuito è stato utilizzata un'induttanza di modo comune, quindi con elemento induttivo sia sul "lato caldo" dei 12v che sul "lato freddo" dei 0v).

Gli alimentatori *switching* (alimentatori a commutazione), ovvero i classici alimentatori per cellulari o altri dispositivi elettronici, come quello utilizzato nel progetto, hanno il vantaggio di essere molto più compatti e di avere un rendimento migliore di quelli tradizionali a trasformatore, però possiedono il grande svantaggio di essere caratterizzati da un elevato *ripple* e dalla generazione di componenti spurie ad alta frequenza, con l'inconveniente di andare ad interferire con il funzionamento degli altri componenti elettronici presenti nel circuito.

I condensatori hanno lo scopo di ridurre il ripple, ovvero quel segnale, che a seguito di un raddrizzamento della corrente (da AC a DC), si sovrappone alla tensione continua di lavoro.

Di seguito un esempio di ripple presente su di un'onda raddrizzata:

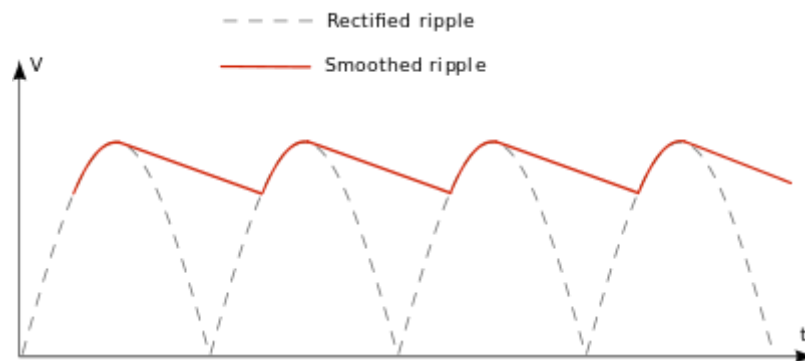


Figura 4.3: Ripple su onda sinusoidale raddrizzata

Di norma si cerca di attenuare quanto più possibile l'effetto del ripple aggiungendo dei filtri (condensatori ed induttori in genere), ma l'effetto del ripple è impossibile e costoso da eliminare del tutto per cui si cerca sempre di adottare la soluzione migliore, al minor costo in funzione dello scopo del circuito.

In una situazione di questo tipo:

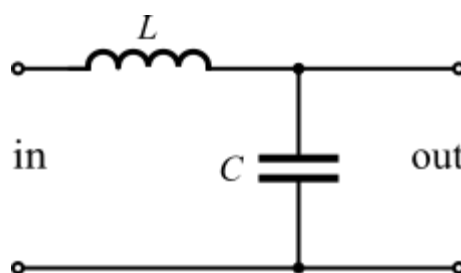


Figura 4.4: Filtro LC passa basso

Condensatore ed induttanza formano quello che è un comune filtro passa basso LC (L = induttanza, C = condensatore), in grado appunto di eliminare, o meglio tagliare, tutte le frequenze definite “alte”, ovvero al di sopra della frequenza di taglio del filtro. Questo filtro se posto in serie ad una fonte di alimentazione in corrente continua (quindi a frequenza ideale 0), andrebbe ad attenuare le frequenze variabili/alternate sovrapposte alla corrente continua, generate appunto dall'alimentatore switching.

Nel caso del circuito utilizzato nel progetto, il filtro a pi-greco è ancora più efficace di quello LC, dato che usa per $L1$ (vedi figura 4.1) una bobina compensata (detta anche bobina di modo

comune), che è formata da 2 avvolgimenti in controfase, avvolti sullo stesso nucleo. In una bobina come questa, i disturbi presenti in uno degli avvolgimenti generano un flusso magnetico che crea una tensione identica sull'altro avvolgimento. Il risultato è che se sul “lato caldo” un disturbo fa aumentare per un istante la tensione, ad esempio, di 1V, anche il “lato freddo” avrà lo stesso aumento, quindi all'uscita della bobina la tensione presente sarà identica su entrambi i poli.

In un circuito pi-greco, inoltre, il carico (parte di circuito a valle del filtro) è disaccoppiato dalla rete elettrica, dall'induttanza, questo aiuta ulteriormente la funzione di filtro.

La seconda parte, si compone di un regolatore di tensione lineare, in grado di abbassare e stabilizzare la tensione ad un valore di circa 5v:

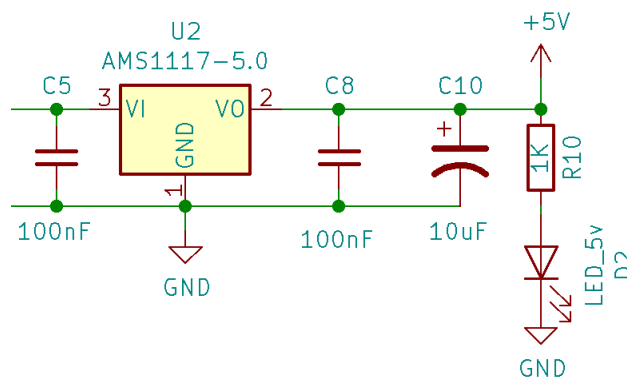


Figura 4.5: Schematico regolatore 5v

Lo schema prevede quanto riportato nel datasheet [b] e si compone di due condensatori di filtro per tenere stabile la tensione, in ingresso ed in uscita. Nello schema ne sono stati previsti altri, in caso dovessero servire. Alla fine è stato aggiunto un diodo led il quale segnala la presenza di tensione sul ramo dei 5v.

Allo stesso modo è stata progettata la terza parte, per abbassare la tensione dai 5v ai 3.3v, sempre con condensatori di filtro e led, con circuito basato sul relativo datasheet [c]:

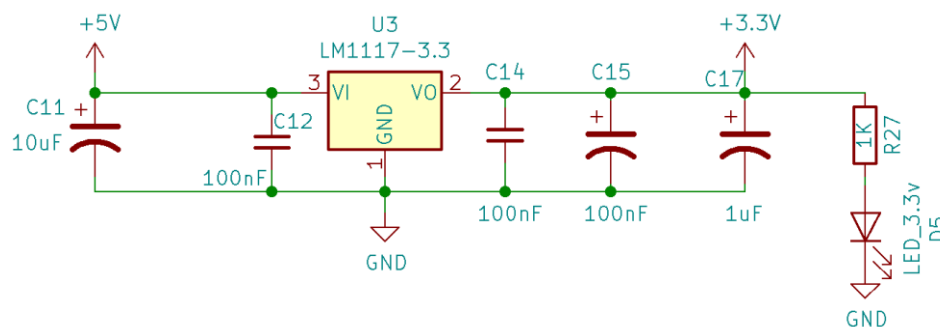


Figura 4.6: Schematico regolatore 3.3v

4.1.2 Controllo led

Nel circuito è stata aggiunta anche una parte di controllo led, sono i led presenti in ogni postazione i quali segnalano se la macchina è in lavorazione (led verde) o in errore (led rosso). Questi sono presenti in quasi tutti i macchinari industriali, per il progetto si è pensato di introdurli in questo modo:

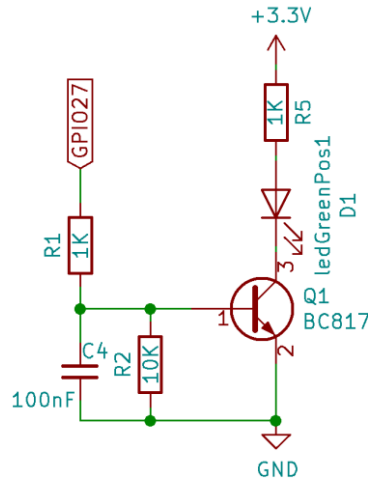


Figura 4.7: Schematico controllo led

Il microcontrollore genera un segnale sulla pista GPIO27 (è presente un GPIO diverso per ogni led), tramite la resistenza R1 si limita la corrente uscente dal microcontrollore, tramite il condensatore C4 si cerca di livellare la tensione in modo da evitare il problema del “rimbalzo” di tensione, la resistenza R2 è la resistenza di pull-down, la quale in assenza di segnali sulla linea GPIO27 fornisce un livello di tensione a Q1 basso (0v). Infine il transistor NPN BC817 (datasheet [d]) disaccoppia l’uscita del microcontrollore dal led, evitando così di danneggiarlo, a seguito di assorbimenti di corrente troppo elevati.

R5 invece limita la corrente assorbita dal led secondo la legge di ohm $R=V/I$.

4.1.3 Gestione pulsanti START/RESET

Nella postazione dell’operatore sono presenti due pulsanti, rispettivamente:

- START: per far partire il ciclo di produzione, dopo che l’operatore ha inserito il pezzo grezzo
- RESET: a seguito di un allarme, l’operatore interviene manualmente e sistema il problema. Per far ripartire il ciclo deve premere il pulsante di reset.

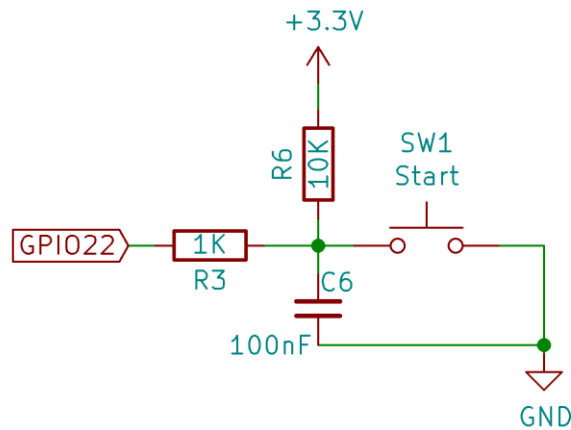


Figura 4.8: Schematico gestione pulsanti

GPIO22 rappresenta un ingresso del microcontrollore. Il sistema di rilevazione del pulsante premuto si basa sul portare a GND l'ingresso del microcontrollore. In condizioni di pulsante non premuto, il microcontrollore leggerà uno stato alto, determinato dalla linea 3.3v collegata alla linea GPIO tramite le resistenze R6 (resistenza di pull-up) e R3 (resistenza per limitare la corrente). Nel caso il pulsante venga premuto, la tensione 3.3v non andrà più nella linea GPIO ma verrà scaricata a massa. Il condensatore C6 funge da anti-rimbalzo.

4.1.4 Gestione proximity

In quasi tutti i macchinari industriali sono presenti dei proximity, dei sensori capacitivi in grado di rilevare la presenza di un corpo metallico a qualche millimetro di distanza, quindi senza contatto. Si è deciso quindi di prevedere un circuito per l'interconnessione di proximity tra i più comuni in commercio, quelli a tre fili.

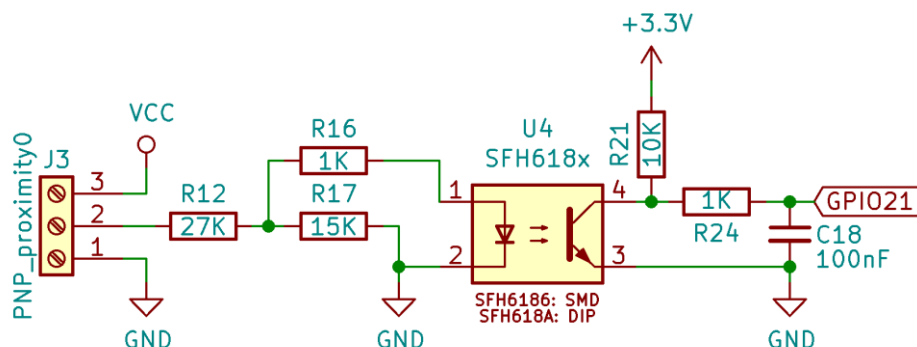


Figura 4.9: Schematico lettura proximity

Il circuito si compone di un partitore iniziale, formato dalle resistenze R12 e R17, per abbassare la tensione dai 12v forniti dal proximity ai circa 5v, più accettabili per il foto-accoppiatore. Il foto-accoppiatore SFH6186 (datasheet [e]) è un componente che serve per disaccoppiare fisicamente ed elettricamente due parti di circuito con tensioni diverse. Se il proximity rileva il pezzo, la tensione passerà attraverso il partitore, il quale, la abbasserà a 5v ed alimenterà i piedini 1 e 2 di U4, a cui è collegato internamente un diodo led. Il led illuminandosi attiva il foto-transistor di U4 il quale entra in conduzione e porta la pista GPIO21, del microcontrollore, a GND. In condizioni di “pezzo non rilevato” la pista GPIO21 resterà alta (a 3.3v). Con il foto-accoppiatore si protegge la parte di circuito collegata al microcontrollore evitando che vi possano entrare tensioni che lo brucerebbero.

4.2 ESP32

La scheda utilizzata per il controllo dell'hardware è una dev-board ESP32, la quale monta un microcontrollore ESP32, prodotto dalla Espressif, dotato di un modulo per la connettività Bluetooth e WiFi. Con una serie di piedini (il numero varia di versione in versione ma la funzionalità non cambia), tra i quali vi sono i GPIO che permettono di interagire col microcontrollore stesso. Fornisce supporto per diverse interfacce tra cui SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, pulse counter, capacitive touch sensor, ADC, DAC, tutti elencati qui sotto:

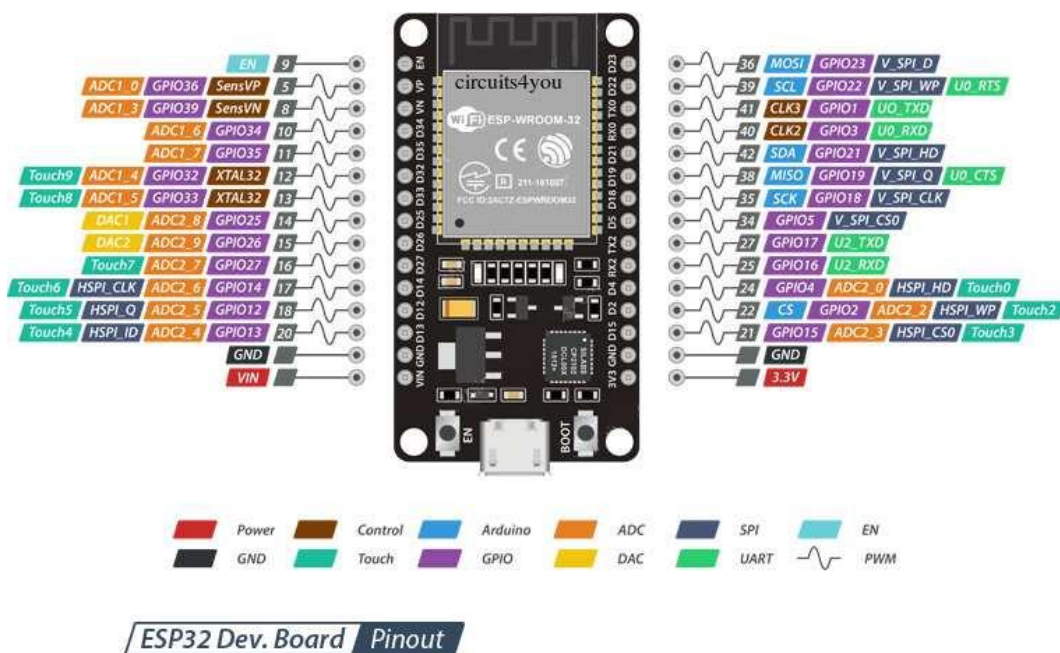


Figura 4.10: Dev. Board pinout ESP32

4.2.1 Principali caratteristiche

CPU e memoria

- Xtensa® single-/dual-core 32-bit LX6 microprocessor(s), fino a 600 DMIPS
- 448 kB ROM
- 520 kB SRAM
- 16 kB SRAM in RTC
- memoria Flash 4 MB

Wi-Fi

- Range di frequenza: 2,4 GHz - 2,5 GHz
- 802.11 b/g/n HT40 Wi-Fi transceiver, baseband, stack e LwIP
- Modalità Sniffer / Station / SoftAP e Wi-Fi direct mode
- Velocità massima dati 150 Mbps@11n HT40, 72 Mbps@11n HT20, 54 Mbps@11g e 11 Mbps@11b
- Massima potenza di trasmissione 19,5 dBm@11b, 16,5 dBm@11g, 15,5 dBm@11n
- Sensibilità minima del ricevitore -97 dBm
- Supporta la modalità di protezione: WEP, WPA/WPA2 PSK/Enterprise
- Crittografia con accelerazione hardware: AES / SHA2 / ECC / RSA-4096

Bluetooth

- Conforme alle specifiche Bluetooth v4.2 BR/EDR e BLE
- Ricevitore NZIF con una sensibilità di -97dBm
- Trasmettitore classe 1, classe 2 e classe 3
- Potenza di trasmissione +12dBm
- Ricevitore NZIF con una sensibilità di -97dBm
- Multi-conessioni Bluetooth e BLE

E molte altre caratteristiche possono essere trovate nel sito o nella documentazione tecnica presente al link [f].

Il linguaggio utilizzato per la programmazione, è un linguaggio simile al C/C++, per l'esattezza chiamato Wiring e viene utilizzato nell'IDE Arduino per i microcontrollori compatibili.

4.3 Firmware

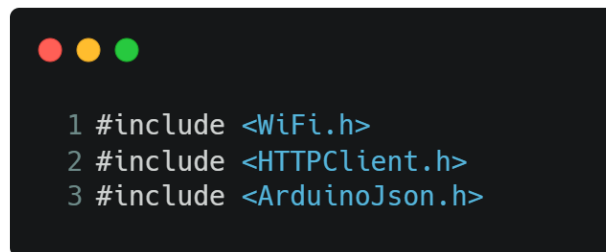
Il firmware lo possiamo suddividere in due macro-blocchi, una prima parte più importante, legata all'accesso ed alla comunicazione attraverso internet ed una seconda parte meno importante legata alla gestione del macchinario (quindi sensori, attuatori, motori ecc...).

Nel seguente capitolo verrà trattata soltanto la prima parte legata all'accesso alla rete, in quanto più significativa per quel che concerne il progetto stesso.

La seconda parte non ha una rilevanza particolare ai fini del progetto ed è stata usata solamente per la generazione dei dati da visualizzare.

4.3.1 Librerie

Per il progetto sono state utilizzate essenzialmente quattro librerie:



```
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3 #include <ArduinoJson.h>
```

Figura 4.11: Inclusione librerie

1. `WiFi.h`: è la libreria che permette al modulo di collegarsi ad internet. Fornisce dei metodi per il funzionamento sia da server (e quindi accettare connessioni in entrata) che da client (e quindi fornire connessione in uscita). Effettua la connessione secondo gli standard WEP e WPA2, utilizzando quindi SSID e password per la connessione alla rete.
2. `HTTPClient.h`: è la libreria che permette di utilizzare il modulo come un web client ed espone una serie di funzioni per effettuare delle chiamate GET, POST e PUT ad un web server, secondo il protocollo HTTP.
3. `ArduinoJson.h`: è la libreria che permette di raggruppare i dati prodotti, dal modulo, in un oggetto di tipo JSON. Fornisce diversi metodi interessanti come l'allocazione della memoria, cast esplicito o implicito, filtri dati... e molti altri che possono essere approfonditi sul sito [35].

4.3.2 Dichiarazione variabili

```
1 const char* ssid      = "yourSSID";
2 const char* password = "yourPassword";
3
4 const String serverUrl = "http://192.168.1.2:3000/produzione/inserisci";
```

Figura 4.12: Dichiarazione variabili

Per eseguire il codice è necessario cambiare il valore delle linee 1 e 2 con l'SSID e la password del proprio wifi. Mentre per il server si dovrà inserire l'indirizzo IP del computer sul quale è in esecuzione (per l'esecuzione locale del server), o in alternativa l'indirizzo su cui viene hostato (nel caso si decida di usufruire di un servizio di hosting).

4.3.3 Connessione al wifi

```
1 void connectESP()
2 {
3   Serial.println("WiFi setup");
4   WiFi.mode(WIFI_STA); //imposto il wifi
5
6   Serial.println("");
7   delay(10);
8   WiFi.begin(ssid, password);
9   while (WiFi.status() != WL_CONNECTED)
10  {
11    delay(500);
12    Serial.print(".");
13  }
14   Serial.println("WiFi connected");
15   Serial.println("IP address: ");
16   Serial.println(WiFi.localIP());
17 }
```

Figura 4.13: Funzione connessione al wifi

La configurazione della connessione avviene attraverso funzione in Figura 4.13:

Linea 4: viene impostato il tipo di connessione del modulo come stazione (STA). In questo caso il router fornirà la connessione al modulo e si dovrà collegare ad esso tramite SSID e password

(se richiesta). È possibile però impostare il modulo anche come Access Point (AP), in quel caso sarà il modulo stesso a fornire connettività ad altri device presenti sulla sua rete.

Linea 8: vengono impostati SSID (Service Set Identifier) e password per il collegamento alla rete.

Linee 9-13: tramite la funzione di libreria `WiFi.status()` si ritorna lo stato della connessione, connesso o non connesso. Per cui finché il modulo non risulta connesso rimane dentro al ciclo e controlla lo stato della connessione ogni 500ms.

Linea 16: se tutto è andato a buon fine, il modulo si è connesso e tramite la funzione di libreria `WiFi.localIP()`, viene ritornato l'IP assegnato dal router al modulo.

4.3.4 Impacchettamento ed invio del messaggio

```
1 void sendMessage()
2 {
3     String json;
4     if(WiFi.status() == WL_CONNECTED)
5     {
6         //Apro una connessione di tipo HTTP tramite l'url del server
7         http.begin(serverURL);
8         Serial.println("Apro connessione HTTP");
9
10        //Specifico il tipo di file da inviare nell'header
11        http.addHeader("Content-Type", "application/json");
12
13        //Trasformo i dati in formato JSON
14        StaticJsonDocument<256> doc;
15        doc["idMachinery"] = idMachinery;
16        doc["idProduct"] = idProduct;
17        doc["position"] = positionP;
18        doc["status"] = pass;
19        doc["errorCode"] = errorCode;
20
21        serializeJson(doc, json);
22
23        //Effettuo la chiamata POST al server e stampo la risposta
24        int httpResponseCode = http.POST(json);
25        Serial.print("HTTP Response code: ");
26        Serial.println(httpResponseCode);
27        Serial.print("HTTP payload: ");
28        Serial.println(http.getString());
29
30        Serial.println("Chiudo connessione HTTP");
31        Serial.println("-----");
32        http.end(); //chiudo la connessione
33    }
```

Figura 4.14: Funzione per gestione ed invio messaggi HTTP

Attraverso questa funzione, vengono passati i dati di produzione, impacchettati in un oggetto di tipo JSON e inviati al server tramite una POST.

`http` è un dato di tipo `HTTPClient` per l'accesso ai metodi della libreria.

Linea 4: tutta l'operazione deve essere svolta verificando ad ogni invio che il modulo ESP sia connesso.

Linea 7: viene aperta una connessione di tipo HTTP con il server, raggiungibile tramite il relativo indirizzo presente nelle variabili globali.

Linea 11: alla connessione HTTP vengono aggiunti degli *headers* (un'intestazione) per specificare il tipo del contenuto del messaggio (*content-type*), il quale, in questo caso, è in formato JSON.

Linea 14: viene creato un `JsonDocument` per salvare la rappresentazione in memoria dell'oggetto. In questo caso viene utilizzata l'allocazione statica della memoria, in quanto l'oggetto è di piccole dimensioni (per le dimensioni dell'oggetto è consigliabile utilizzare il tool online fornito dal produttore della libreria, ArduinoJson Assistant, accessibile tramite il link [g]).

Linee 14-19: Un `JsonDocument` vuoto, diventa un oggetto non appena vengono aggiunti dei "membri" ad esso. Con queste istruzioni vengono aggiunti questi membri, che altro non sono che i dati da inviare.

Linea 21: tramite il metodo `serializeJson(doc, output)` viene prodotto un oggetto JSON (parametro `output`) a partire da un `JsonDocument` (parametro `doc`).

Linea 24: con `http.POST(json)` viene inviato al server l'oggetto JSON tramite la chiamata POST. Questa fornirà una risposta che verrà immagazzinata all'interno di `httpResponseCode`.

Linea 28: tramite il metodo `http.getString()` della libreria `HTTPClient` verrà stampata su monitor seriale la risposta del server in formato stringa.

5 Server NodeJS

Il server rappresenta lo strato back-end del progetto, si è deciso di implementarlo utilizzando NodeJS.

Il server riveste un ruolo chiave all'interno del progetto, è ciò che sta alla base di tutto, senza il server il progetto non potrebbe minimamente funzionare a differenza degli altri strati, a cui invece, il progetto potrebbe anche fare a meno.

Lo scopo del server è quello di gestire le richieste di client e front-end e favorire lo scambio dati tra questi ed il database. Se si volesse fare un'analogia con una strada, il server sarebbe il vigile che gestisce tutto il traffico derivante dalle diverse strade (*route*) e confluenti tutte in un unico punto (il server per l'appunto).

Si può suddividere, a livello logico, il codice del server in tre parti: una prima parte relativa alla ricezione dei messaggi da parte dei diversi client hardware, una seconda parte relativa al salvataggio dei dati nel database ed una terza parte relativa alla richiesta dei dati da parte del front-end.

5.1 Moduli utilizzati

Come già accennato nei capitoli precedenti, Node si compone di diversi moduli. Questi sono come delle classi in Java, con l'unica eccezione che attraverso il gestore dei moduli NPM, è possibile accedere ad una miriade di moduli già scritti e testati da milioni di persone (per cui si presuppone che siano funzionanti e senza bug), per cui sarà sufficiente installarli da riga di comando con: `npm i nome_modulo`, il modulo verrà così installato a livello locale nel progetto e sarà possibile usarlo dopo averlo importato nel progetto, come verrà mostrato in 5.2.1.

5.1.1 Express

Express rappresenta il modulo più importante tra tutti quelli utilizzati nel server. È quel modulo che permette di creare web server con NodeJS, basti pensare che ogni settimana viene scaricato ed installato circa ventidue milioni di volte nel mondo, questo solo per capire quanto viene utilizzato a livello globale, tant'è che ormai è stato definito come il framework server “di fatto” di Node.

Express fornisce una serie di API che consentono di gestire le chiamate HTTP provenienti da diverse route.

5.1.2 Sequelize

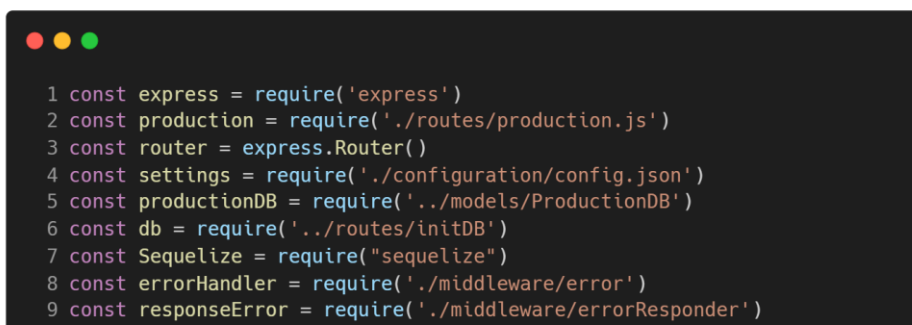
Sequelize è un modulo meno utilizzato di Express, registra un trend di poco più di un milione di download settimanali, ma non per questo vuol dire che sia meno valido.

Sequelize è un modulo basato su *promise*, ovvero oggetti rappresentanti l'eventuale compimento o fallimento di un'operazione asincrona e del suo risultato. In particolare viene definito ORM (Object Relational Mapper), ovvero uno strumento che consente l'interazione con il database senza conoscere di preciso il linguaggio utilizzato dal database (SQL). In parole povere, Sequelize fornisce un livello di astrazione tra server e database, per cui non si rende necessario conoscere la sintassi specifica del DBMS utilizzato. Sequelize fornisce supporto a diversi DBMS, tra cui Postgres, MySQL, MariaDB, SQLite, DB2 e Microsoft SQL. Per accedere ad uno di questi sarà necessario fornire le credenziali di accesso al DB ed il nome del DBMS utilizzato, tra quelli citati. Successivamente l'inserimento di righe da parte del server avviene tramite funzioni javascript, come si vedrà nei prossimi paragrafi.

5.2 Codice

Di seguito verranno riportati le parti di codice utili alla comprensione del progetto.

5.2.1 Importazione di moduli, middleware e file custom



```
1 const express = require('express')
2 const production = require('./routes/production.js')
3 const router = express.Router()
4 const settings = require('./configuration/config.json')
5 const productionDB = require('./models/ProductionDB')
6 const db = require('./routes/initDB')
7 const Sequelize = require("sequelize")
8 const errorHandler = require('./middleware/error')
9 const responseError = require('./middleware/errorResponder')
```

Figura 5.1: Importazione moduli

Nei diversi file, sono stati importati alcuni moduli, il primo è **Express** descritto nel capitolo precedente ed il secondo è un modulo interno, scritto appositamente per l'applicazione, serve per gestire le chiamate ad una path particolare, in questo modo tutte le chiamate aventi URL che inizia con `localhost:3000/produzione/` saranno gestite all'interno di questo file. Questo permette di avere un codice più pulito e modulare.

Il terzo è **Router**, un modulo built-in di express che permette di creare delle parti di percorso univoche per più route, come fatto per *production.js*.

Settings invece non è propriamente un modulo, ma è associato ad un file JSON (*config.json*) contenente tutti i parametri di configurazione utilizzati nel progetto. In questo modo se si dovesse cambiare la porta del server od il nome del database, lo si potrebbe fare semplicemente da quel file, senza dover cercare la costante all'interno di tutto il progetto.

productionDB è un file sempre custom, un modello ed ha lo scopo di definire, per l'appunto, un modello per la scrittura dei dati sul DB, in sostanza va a definire quali saranno i dati ed il loro tipo.

Il prossimo si tratta sempre di un modulo custom, che consente di eseguire delle funzioni base sul DB, come l'autenticazione, test di connessione e sincronizzazione.

Sequelize, come già descritto, permette di accedere al database in modo più semplice attraverso delle API mascherate da funzioni universali per diversi DB.

Gli ultimi due, sono dei *middleware* custom, i quali espongono delle funzioni, in particolare questi middleware servono per la cattura e la gestione degli errori.

La sintassi prevede di associare ad una costante, con `require`, l'importazione del modulo. Nel caso si tratti di modulo esterno basta scrivere il nome del modulo stesso, nel caso di modulo interno si deve indicare il percorso relativo (ovvero partendo dal file in cui il modulo viene importato).

5.2.2 Definizione delle CORS policy

CORS (Cross-Origin Resource Sharing) è un meccanismo basato sugli headers HTTP, che permettono al server di indicare qualsiasi origine oltre la sua, dal quale il browser dovrebbe permettere il caricamento di risorse.

```
1 // CORS policy
2 app.use((req, res, next) =>
3 {
4   res.header("Access-Control-Allow-Origin", "*");
5   res.header("Access-Control-Allow-Methods", "GET,PUT,POST,DELETE,PATCH,OPTIONS");
6   res.header("Access-Control-Allow-Headers", "Content-Type, Authorization, Content-Length, X-Requested-With");
7   next();
8 })
```

Figura 5.2: Definizione ed utilizzo delle CORS policy

Molto brevemente si può vedere che con questa piccola parte di codice si va a definire l'origine ("*" sta a significare qualsiasi path), i metodi (quindi i classici GET, PUT, POST...) e gli headers (quindi il tipo di contenuto della richiesta, la lunghezza...).

Se non venissero implementati, Google potrebbe segnalare un errore relativo all'assenza di questi headers, ad esempio:

```
✖ ▶ Refused to set unsafe header "origin"
✖ Access to XMLHttpRequest at 'http://localhost:3000/hp' from origin 'http://localhost:3000' header is present on the requested resource.
✖ ▶ Uncaught (in promise) Error: Network Error
    at e.exports (spread.js:25)
    at XMLHttpRequest.d.onerror (spread.js:25)
```

Figura 5.3: Errore Google per CORS non implementate

5.2.3 Definizione porta del server

Il server per essere messo in esecuzione necessita che gli venga assegnata una porta:

```
1 const port = settings.server.port;
2 app.listen(port, () => console.log(`Server app listening on port ${port}!`));
3
4 process.on('uncaughtException', function (err)
5 {
6   console.log('Caught exception: ', err);
7 });
```

Figura 5.4: Definizione porta server ed inizializzazione

Linea 2: Il server viene messo in ascolto nella porta locale definita nel file *config.json*.

Linea 4: Nel caso di eccezioni non gestite il server terminerebbe la sua esecuzione, per cui con questo comando si vanno a catturare tutte le eccezioni non gestite.

5.2.4 Definizione route

Per permettere al server di identificare l'origine della risorsa, è possibile creare delle route di navigazione, per cui, in funzione dell'URL a cui viene fatta la richiesta, il server gestirà quella richiesta secondo delle determinate funzioni.



```
1 app.use(express.json());
2 app.use(express.text());
3
4 app.use('/produzione', production);
5
6 app.get('/', (req, res) => { res.send("Accesso eseguito!").status(200) });
7
8 app.all("*", (req,res) => { res.send("404 non trovato").status(404) });
```

Figura 5.5: Definizione delle route del progetto

Questo è un primo esempio di utilizzo delle route di express.

Linee 1-2: utilizzo di middleware di express per permettere la lettura di testi e JSON in arrivo dalle varie richieste.

Linea 4: nel caso di più percorsi con la stessa parte iniziale di URL è possibile definire un modulo interno che li gestisca tutti (ad esempio “localhost:3000/produzione/inserisci” e “localhost:3000/produzione/rimuovi” e “localhost:3000/produzione/modifica”, tutti hanno lo stesso prefisso).

Linea 6: con questa API si gestisce una GET con path localhost:3000/. Collegandosi al browser con questa URL automaticamente viene effettuata una GET, il server istanzierà una funzione di callback ricevente i parametri `req` (*request*, ovvero la richiesta) e `res` (*response*, ovvero la risposta del server) la quale risponderà al client con la stringa “Accesso eseguito”.

Linea 8: può succedere che venga digitata un'URL non gestita e non esistente nel progetto, tale per cui Google segnalerebbe un errore. Per evitare ciò è bene gestire le richieste a tutte le URL non gestite andando a definire una funzione che le racchiuda tutte. Con “*” si identificano tutte le route, deve essere posto alla fine delle definizioni delle route, in questo modo verrà eseguita per ultima ed andrà a catturare soltanto le route non gestite, restituendo un messaggio di errore con il relativo status code.

5.2.5 Ricezione dati da client hardware

La ricezione dei dati dal client hardware, avviene attraverso una POST, il client effettua una post al server ed il server la intercetta.

```
1 var dati = [];
2
3 // localhost:3000/produzione/inserisci
4 router.post("/inserisci", (req,res) =>
5 {
6     let {idMachinery,idProduct,position,status,errorCode} = req.body;
7     let dato = req.body;
8     let i;
9
10    dati.forEach( (element, index) => {
11        if(element.idMachinery === dato.idMachinery) { i = index }
12    });
13
14    if(i === undefined || (i === 0 && dati.length < 1)) { dati.push(req.body) }
15    else { dati.splice(i, 1, req.body) }
16
17    res.send("Ricevuto").status(200);
18
19    addRow({idMachinery,idProduct,position,status,errorCode});
20 });
21
22
23 function addRow({idMachinery,idProduct,position,status,errorCode}) {
24     productionDB.create({idMachinery,idProduct,position,status,errorCode})
25     .catch(err => console.log(err));
26 }
```

Figura 5.6: Route per ricezione dati da client hardware

Linea 1: viene dichiarata una variabile array “dati” per immagazzinare i dati ricevuti dal client hardware.

Linee 4-20: tramite l’API POST all’URL *localhost:3000/produzione/inserisci* viene inserito il corpo della richiesta nella variabile “dati”.

Linee 6-7: variabili per contenere e scorporre la richiesta ricevuta dal client hardware.

Linee 10-12: tramite un `foreach` viene iterato l'array *dati* per cercare se nei dati ricevuti è presente o meno un oggetto con lo stesso `idMachinery`, in tal caso salva l'indice nella variabile `"i"`.


Linee 14-15: se nella ricerca precedente è stato trovato un elemento con `idMachinery` uguale a quello della request, allora sostituisce l'elemento *i*-esimo dell'array `"dati"` con quello ricevuto.

Linee 23-26: con la funzione, vengono inseriti i dati ricevuti nel database, creando una nuova riga (funzione del modulo `sequelize`). Questa funzione di `sequelize` aggiunge automaticamente due colonne alla tabella: `createdAt` (indica data e ora di creazione della riga) e `updatedAt` (indica data e ora di modifica della riga, se non modificata, data e ora di creazione).

5.2.6 Configurazione DB

La configurazione del database si compone di più parti:

- Creazione di un'istanza del modulo `sequelize`



```
1 const Sequelize = require("sequelize");
2 const settings = require('../configuration/config.json');
3
4 const db = new Sequelize(settings.database.dbname, settings.database.user,
5                           settings.database.password, settings.database.options);
```

Figura 5.7: Inizializzazione DB con Sequelize

In questa parte, viene creata un'istanza del modulo `sequelize` e vengono impostati una serie di parametri che vanno a definire l'host per la connessione al DB, il DBMS utilizzato (`mysql`) ed altri parametri propri di MySQL, tutti riportati sul file di configurazione `config.js`.

Da qui in poi, l'accesso al database avverrà tramite l'istanza `"db"`.

- Test connessione

```
1 db.authenticate()  
2 .then(() =>  
3 {  
4   console.log("Database connected");  
5 })  
6 .catch((err) => console.log(err));
```

Figura 5.8: Test connessione/autenticazione DB

Dopo aver istanziato `db`, è bene effettuare un test della connessione, per verificare che tutto sia andato a buon fine, per cui tramite il metodo `authenticate()` di `sequelize` viene restituita un'eccezione nel caso di connessione fallita, mentre se tutto è andato a buon fine, viene stampata la stringa "database connected".

- Sincronizzazione al database

```
1 db.sync(  
2 {  
3   force: false,  
4   alter: true  
5 })  
6 .then(() =>  
7 {  
8   console.log("Database synchronized");  
9 })  
10 .catch((err) => console.log(err));
```

Figura 5.9: Sincronizzazione DB

Se il test di connessione è andato a buon fine, ora sarà possibile accedere al DB con i metodi propri del modulo `sequelize`.

Nel caso si debba accedere alle tabelle del database in sola lettura, il problema di sincronizzazione non si pone. Ma nel caso si debba accedere anche in scrittura (come nel progetto), si dovrà effettuare una sincronizzazione tra quanto gli viene passato e quanto effettivamente è presente sul database.

Il metodo `sync()` di `sequelize` permette di fare ciò.

Linea 3: con il parametro `force` impostato su `false`, si va a indicare a `sequelize` che nel caso la tabella, su cui andiamo a scrivere, esiste già, non ne ricrea una nuova.

Linea 4: con `alter` impostato a `true`, si dà a `sequelize` la facoltà di modificare la tabella esistente sul database per farla corrispondere ai dati che gli vengono inviati. Ad esempio, se nella tabella del DB sono presenti due colonne, nome e cognome, e da Node si cerca di inserire nome, cognome ed indirizzo, automaticamente `sequelize` aggiungerà una colonna per l'indirizzo. Tutte le righe prima di quella appena inserita avranno parametro indirizzo impostato a `null`.

Anche in questo caso viene restituita un'eccezione nel caso di sincronizzazione fallita, mentre se tutto è andato a buon fine, viene stampata la stringa "Database synchronized".

5.2.7 Modello dati

Il modello dei dati, rappresenta la struttura e tipologia dei dati che il server dovrà ricevere.

Normalmente può anche essere evitata questa parte, il server funzionerebbe lo stesso, basterebbe soltanto definire la tipologia di dato in ricezione come `any`, ovvero qualsiasi dato. Il problema più grosso di questo approccio è che qualsiasi dato potrebbe essere inviato al server, anche dati infetti, che potrebbero creare una breccia nel server e permettere l'accesso a malintenzionati.


Per questo motivo, è sempre buona norma, definire un modello dati in ingresso, così che, se un tipo di dato non dovesse corrispondere, l'accesso al server verrebbe bloccato.

Questa soluzione però oltre a comportare una buona prassi in fatto di sicurezza, limita la scalabilità dell'applicazione, in quanto se un giorno si dovesse decidere di inviare un parametro aggiuntivo, oltre ad aggiornare il client hardware, si dovrebbe aggiornare anche il server (il database non necessita di modifiche in quanto si occupa di tutto il modulo `sequelize`).

Di seguito viene rappresentato il modello dati utilizzato nel progetto, dove è possibile notare la presenza di sei parametri, ovvero `id`, `idMachinery`, `idProduct`, `position`, `status` ed `errorCode`, ognuno dei quali rappresenta un dato fondamentale per conoscere la posizione del pezzo in lavorazione ed il suo stato.

Siccome alcuni di questi dati permettono assieme di identificare univocamente un pezzo all'interno del ciclo produttivo, sono stati impostati come obbligatori (`allowNull: false`).

Nella figura 5.10 la loro rappresentazione a livello di codice:



```
1  const Sequelize = require("sequelize");
2  const db = require("../routes/initDB");
3  const settings = require('../configuration/config.json');
4
5  const table = db.define(settings.database.tableName,
6  {
7    id: { type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true },
8    idMachinery: { type: Sequelize.STRING, allowNull: false },
9    idProduct: { type: Sequelize.STRING, allowNull: false },
10   position: { type: Sequelize.STRING, allowNull: false },
11   status: { type: Sequelize.BOOLEAN, allowNull: false, default: false },
12   errorCode: { type: Sequelize.INTEGER, allowNull: true }
13  },
14  {
15    tableName: settings.database.tableName
16  });
```

Figura 5.10 Modello dati del progetto

6 Database MySQL

Per il salvataggio dei dati si è deciso di utilizzare come database MySQL, uno dei database relazionali più conosciuti ed utilizzati.

MySQL è accessibile sia da riga di comando che tramite interfaccia grafica (chiamata *Workbench*), possono essere utilizzati entrambi, ma per i successivi sotto-capitoli verrà utilizzato il Workbench per mostrare la tabella creata e riempita durante i test mentre la riga di comando per la procedura di creazione del database.

6.1 Creazione nuovo utente

La procedura per la creazione dell'utente è una fase che non rientra nel codice del progetto ma è una parte fondamentale per l'esecuzione dello stesso. Questa fase la si può inquadrare in una fase preliminare.

Innanzitutto è necessario installare MySQL sul proprio computer, in Windows ci si può avvalere del mysql installer per Windows, scaricabile dal sito visitabile al link [h].

Scaricato questo si segue tutta la procedura di installazione, al termine verrà mandato in esecuzione il workbench, il quale apparirà come segue:

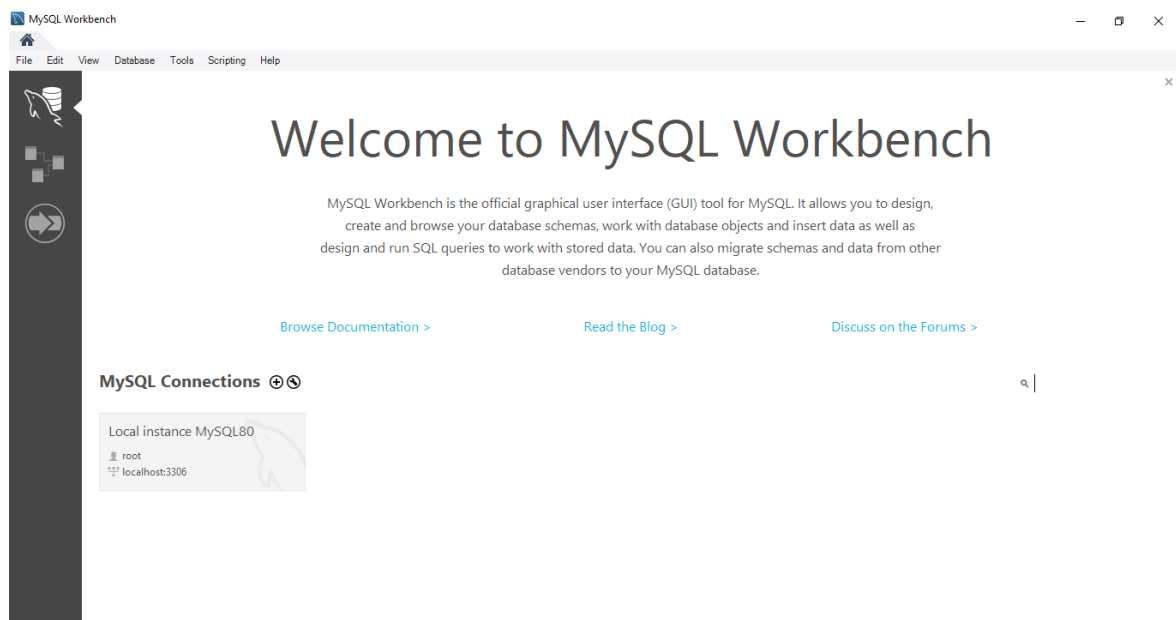


Figura 6.1: Schermata MySQL Workbench, con utente root

Se non espressamente voluto in fase di installazione, l'unico utente presente sarà l'utente root, ovvero l'utente "amministratore", con tutti i privilegi. Solitamente è bene creare un utente nuovo per l'accesso al database, al quale verranno forniti soltanto i privilegi necessari per il compito che deve svolgere.

La procedura migliore e più veloce per la creazione di un nuovo utente è tramite linea di comando.

Aprire il CMD (Command line) e digitare il comando `mysql` per entrare nella modalità di scrittura di codice SQL.

Possono presentarsi degli errori, i più comuni sono:

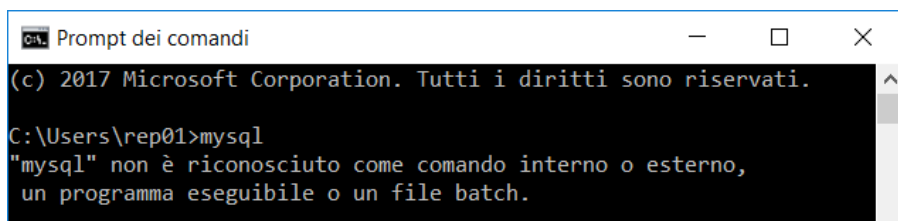


Figura 6.2: Errore variabile d'ambiente non impostata

Questo avviene nel caso non siano state impostate le variabili d'ambiente Windows, per cui è necessario impostarle.

Oppure un altro errore comune potrebbe essere questo:

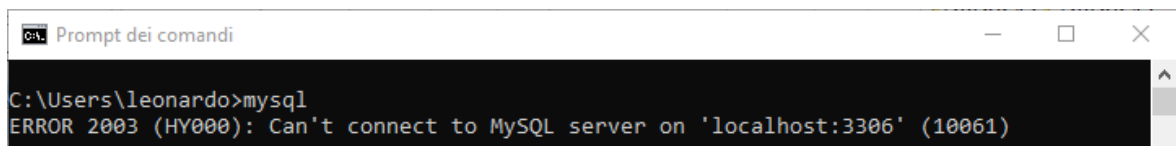


Figura 6.3: Errore servizio non attivo

Questo è dovuto al fatto che il servizio MySQL non è in esecuzione, per cui bisogna accedere ai servizi Windows, trovare il servizio, che di norma viene chiamato MySQL80, ed avviarlo:

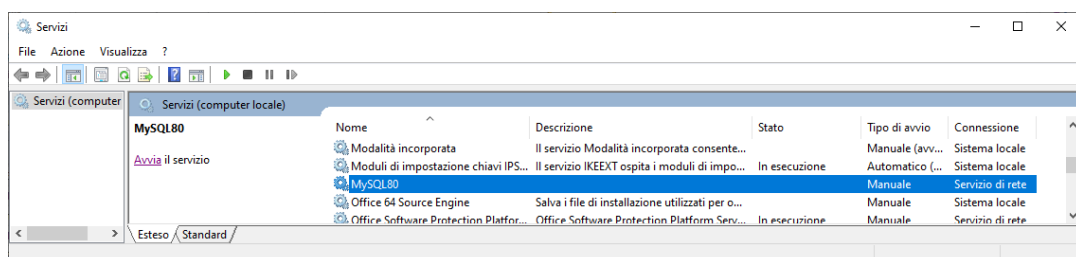
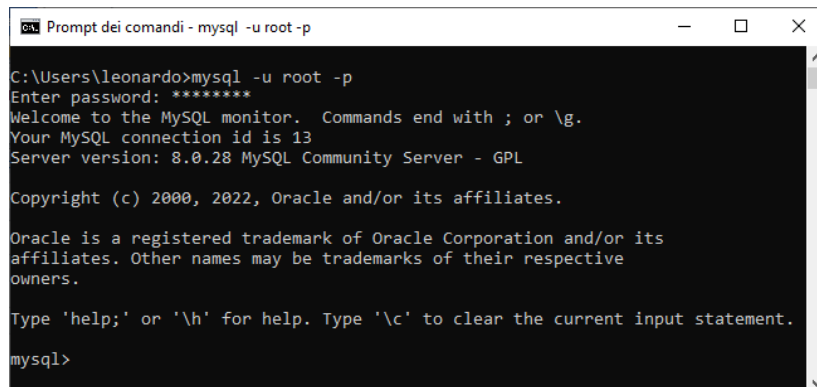


Figura 6.4: Attivazione servizio

Una volta fatto questo digitando `mysql -u root -p` ed inserendo la password impostata durante la fase di installazione sarà possibile accedere in modalità SQL all'utente `root`:



```
Prompt dei comandi - mysql -u root -p
C:\Users\leonardo>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

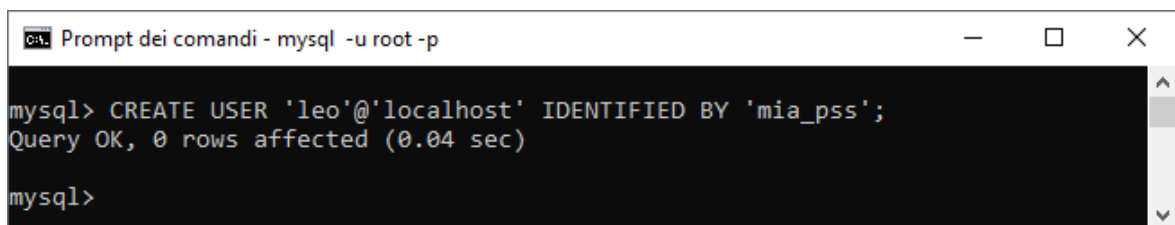
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

Figura 6.5: Accesso a MySQL da CMD

Ora sarà possibile digitare il codice SQL che si vuole, in questo caso si dovrà creare un nuovo utente nel seguente modo:



```
Prompt dei comandi - mysql -u root -p

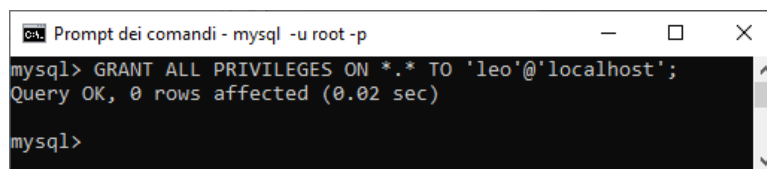
mysql> CREATE USER 'leo'@'localhost' IDENTIFIED BY 'mia_pss';
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Figura 6.6: Creazione nuovo utente

Dove “leo” è l'username del nuovo utente, localhost è l'indirizzo della connessione del database, in questo caso in locale sul pc. Infine “mia_pss” è la password per accedere all'utente appena creato.

Successivamente è necessario impostare i privilegi necessari per l'utente creato, quindi:



```
Prompt dei comandi - mysql -u root -p

mysql> GRANT ALL PRIVILEGES ON *.* TO 'leo'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql>
```

Figura 6.7: Assegnazione privilegi al nuovo utente

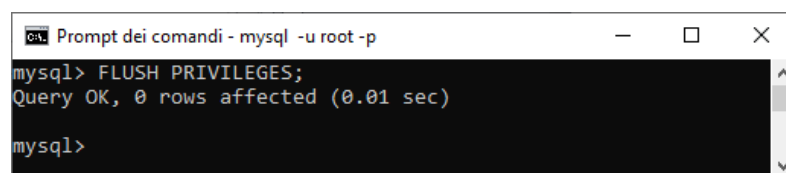
In questo caso sono stati garantiti tutti i privilegi necessari per questo utente (ALL PRIVILEGES) a tutte le tabelle presenti in tutti i database (*.*).

È possibile anche garantire l'accesso soltanto ad alcune determinate azioni, digitando:

- `ALL PRIVILEGES`: consente l'accesso completo al database (o meglio, all'intero MySQL Server);
- `CREATE`: consente all'utente di creare nuovi database e tabelle;
- `DROP`: consente all'utente di cancellare database e tabelle;
- `DELETE`: consente all'utente di cancellare record dalle tabelle;
- `INSERT`: consente all'utente di inserire record nelle tabelle;
- `SELECT`: consente all'utente di interrogare i database per leggerne il contenuto;
- `UPDATE`: consente all'utente di modificare i record presenti nelle tabelle;
- `GRANT OPTION`: consente all'utente di aggiungere o rimuovere privilegi ad altri utenti.

Ed impostare il database e la tabella specifica a cui concedere tali privilegi digitando `nomeDatabase.nomeTabella` al posto di `*.*`.

Terminata questa operazione, bisogna effettuare un aggiornamento interno dei privilegi. Lo si può effettuare mediante il seguente comando:



```

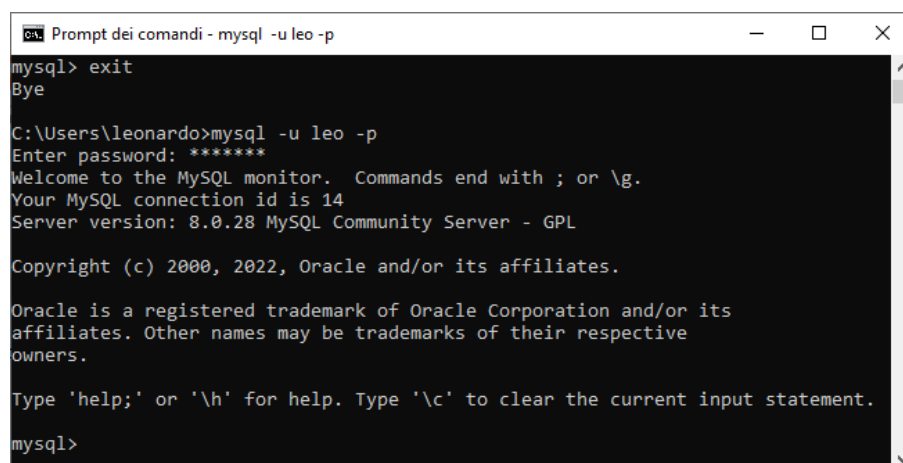
C:\> Prompt dei comandi - mysql -u root -p

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)

mysql>
  
```

Figura 6.8: Aggiornamento privilegi

Fatto questo l'utente sarà creato e potrà essere effettuato l'accesso digitando come prima i comandi di accesso, dopo essere usciti dalla modalità SQL:



```

C:\> Prompt dei comandi - mysql -u leo -p

mysql> exit
Bye

C:\Users\leonardo>mysql -u leo -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, 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.

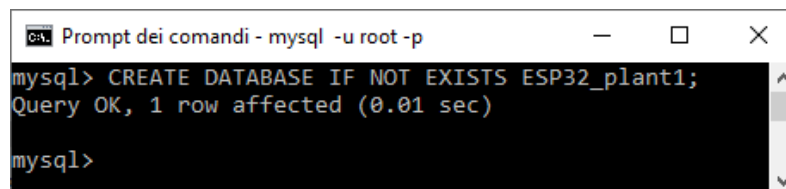
mysql>
  
```

Figura 6.9: Autenticazione con nuovo utente

6.2 Creazione database

Terminata la procedura per la creazione del nuovo utente, è necessario creare il database all'interno del quale il modulo *sequelize* di NodeJS andrà a creare la tabella per i dati ricevuti ed ad inserirveli.

La procedura è molto semplice e come prima è possibile utilizzare la linea di comando:



```
mysql> CREATE DATABASE IF NOT EXISTS ESP32_plant1;
Query OK, 1 row affected (0.01 sec)

mysql>
```

Figura 6.10: Creazione database

Fatto questo il database è pronto per ricevere i dati dal server. È bene ricordare che, affinché tutto funzioni, il servizio di MySQL deve essere attivo sempre, come d'altronde il server NodeJS, cosicché i dati possano sempre essere memorizzati quando un client hardware ci si connetterà.

Tutte le query, sono comunque disponibili per la consultazione nel repository gitHub.

6.3 Opzionale – visualizzazione su Workbench

Nel caso non si volesse utilizzare la linea di comando per visualizzare le tabelle ma si volesse utilizzare il Workbench di MySQL allora bisogna aggiungere una connessione a quelle esistenti.

Questo è possibile farlo aprendo il Workbench e cliccando la “+” nella sezione “MySQL Connections”:

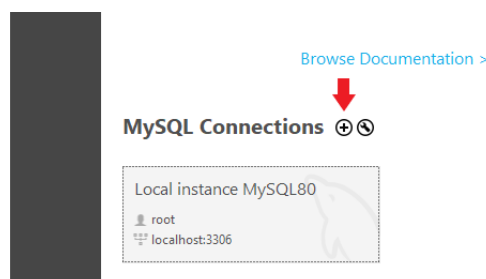


Figura 6.11: Creazione di una nuova connessione

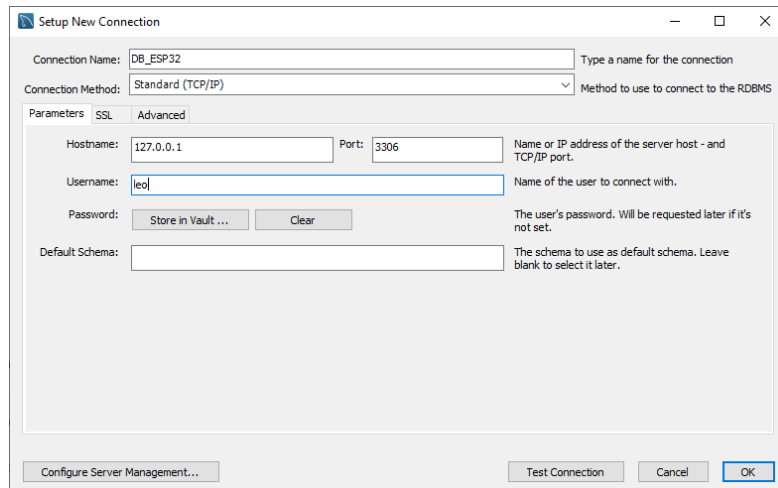


Figura 6.12: Parametri nuova connessione

Si dovrà inserire il nome della connessione, l'*hostname* (che di default è 127.0.0.1, ovvero il localhost) e l'*username* dell'utente creato in precedenza. Cliccando su "Test Connection" verrà effettuato un test sulla connessione, verrà chiesta la password e se tutto andrà a buon fine verrà visualizzato il seguente messaggio:

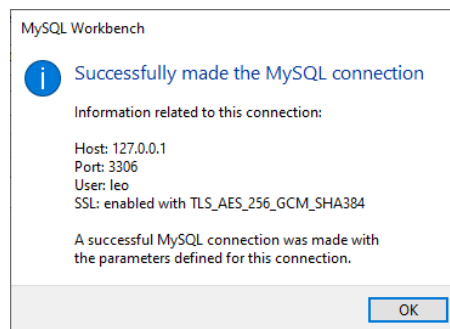


Figura 6.13: Finestra connessione correttamente creata

Dopodiché la connessione sarà disponibile nel Workbench e per accedere sarà sufficiente cliccarci sopra ed inserire la password. All'interno si avrà la possibilità di scrivere del codice SQL ed eseguirlo per visualizzare/modificare/creare dati in linea con i privilegi concessi all'utente in uso.

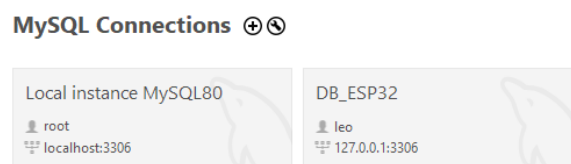


Figura 6.14: Elenco connessioni disponibili

7 Front-end Angular

Il front-end generalmente è l'ultimo livello di un'applicazione, ovvero quel livello che permette di mostrare i dati all'utente sotto forma di grafici, tabelle od in generale utilizzando funzioni grafiche.

Per questo progetto si è deciso di utilizzare il framework Angular sviluppato da Google, in quanto nello scenario applicativo è molto richiesto dal mondo del lavoro.

Esso, come già detto, permette di creare applicazioni mono-pagina, il cui contenuto viene caricato sul browser tutto insieme, durante l'accesso al sito, e modificato dinamicamente durante l'esecuzione.

Angular implementa una strutturazione interna, a componenti. Questa particolarità lo rende modulare al massimo e si avvicina molto ai concetti di programmazione ad oggetti studiati nel corso di Ingegneria del software.

7.1 Componenti applicazione

L'applicazione del progetto si compone di diversi moduli, in Angular chiamati *componenti* i quali svolgono ognuno una determinata funzione:

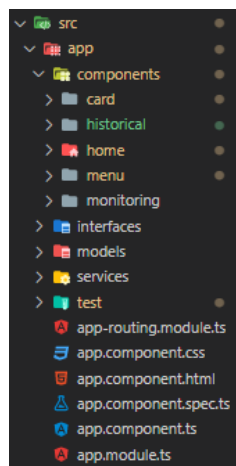


Figura 7.1: Struttura dell'app

Ogni componente si compone di quattro file, come spiegato nel capitolo 2.5.

Nei seguenti sotto-capitoli verranno trattati tutti i componenti presenti, dando maggiore enfasi al file *Component.ts* e *Components.html*, ovvero quelli che contengono le principali funzioni javascript/typescript.

Il resto dei file invece, viene lasciata al lettore la possibilità di visionarli sul repository GitHub.

Struttura dei componenti

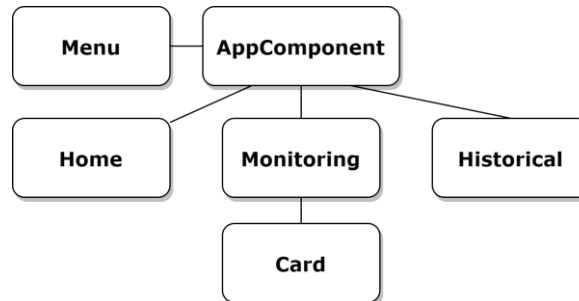


Figura 7.2 Struttura componenti Angular

7.1.1 AppComponent

È il componente principale, questo viene generato in automatico da Angular durante la creazione di un nuovo progetto.

In questo modulo è stata definita la toolbar ed il menù del sito, i quali contengono rispettivamente, logo e nome dell'app ed i bottoni per navigare nelle altre sezioni del sito.

7.1.2 Menu

Questo componente rappresenta un componente inserito direttamente nell'AppComponent e quindi presente in ogni schermata.

Si compone essenzialmente di tre bottoni e tramite questi è possibile selezionare quale tra gli altri tre componenti principali (Home, Monitoring e Historical) visualizzare.

La cosa interessante presente in questo componente è la presenza del ***routerLink***, ovvero una direttiva propria di Angular, che permette di navigare tra i componenti del progetto:

```
1 <nav>
2   <ul class="navbar">
3     <li> <button [routerLink]="['/home']" routerLinkActive="router-link-active">Home</button> </li>
4     <li> <button [routerLink]="['/monitoring']">Monitoring</button> </li>
5     <li> <button [routerLink]="['/hystorical']">Historical</button> </li>
6   </ul>
7 </nav>
```

Figura 7.3: Esempio utilizzo routerLink

Basterà indicare la route presente nel file *app-routing.module.ts* ed Angular mostrerà il componente ad essa associato.

7.1.3 Home

È il componente che consente la visualizzazione della home, ovvero la pagina che appare all'apertura del sito.

La funzione principale di questo componente è la visualizzazione delle immagini dei macchinari. Tramite un selettore è possibile scegliere un macchinario tra quelli che stanno trasmettendo dati al server.

Passando il mouse sopra i pallini, sarà possibile visualizzare una breve descrizione della postazione di lavorazione in cui il pallino è posto.

Questa parte è stata fatta implementando l'immagine del macchinario in formato SVG (*ScalableVectorGraphics*) ed andando a rendere reattiva (*responsive*) l'immagine direttamente inserendo del codice all'interno dell'immagine SVG. Questa operazione è stata svolta con **Inkscape**, un software di disegno vettoriale che permette di aggiungere semplice codice XML alle immagini (come un *mouseover*, *mouseleave*...), direttamente da un pannello di configurazione e quindi definire il colore, piuttosto che l'opacità dell'immagine o il testo da mostrare.

Di seguito viene rappresentato il pannello “proprietà oggetto” all'interno della quale è stato possibile inserire le funzionalità sopra descritte.

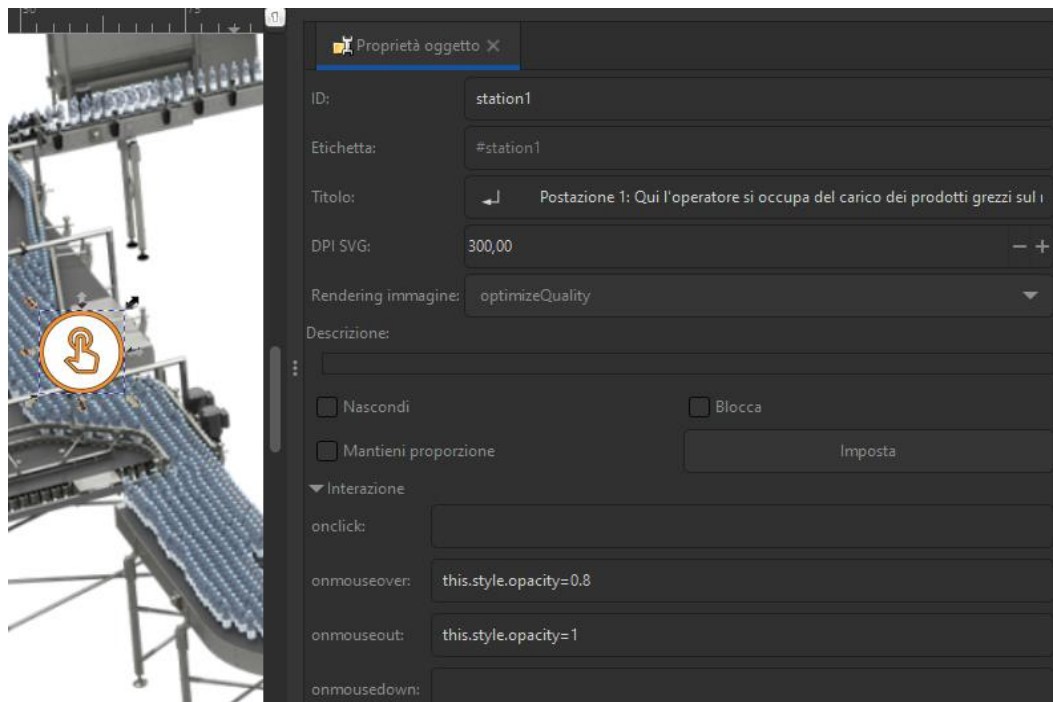


Figura 7.4: Schermata proprietà oggetto Inkscape

Risultano particolarmente interessanti, due funzioni utilizzate nel codice HTML, attraverso le quali è possibile, grazie ad Angular, definire dei blocchi condizionali. Di cui si riporta un esempio non tratto dal codice utilizzato per il progetto:

```

1 <div *ngFor="let i of test" [value]="i">{{i}}</div>
2
3 <div [ngSwitch]="idMachinery">
4   <div *ngSwitchCase="'caseN'"> code... </div>
5 </div>

```

Figura 7.5: Esempio di utilizzo di blocchi condizionali in HTML

Grazie ad Angular è infatti possibile poter inserire dei blocchi condizionali, come If/Else, For, Switch. Rendendo di fatto il linguaggio HTML un linguaggio dinamico ed ampliando così le possibilità di rendere il codice più modulare

7.1.4 Monitoring

È il componente principale, quello che permette di monitorare lo stato di produzione di ogni macchinario in tempo quasi reale.

Nel template di questo componente è possibile visualizzare tutti i dati di produzione inviati da ogni client hardware, sotto forma di schede. Le schede rappresentano un altro componente integrato in Monitoring.

È possibile selezionare solo uno dei macchinari presenti, attraverso un apposito form posto in alto a sinistra e visualizzare i dati in cambiamento solo di quel macchinario.

Il form presenta il meccanismo del ***databinding***, tale per cui è possibile passare dati dal template HTML al file di gestione *monitoring.component.ts*.

```
1 <div class="form-group mx-sm-3 mb-2">
2   <input type="search" name="machineryID" class="form-control"
3     placeholder="ID machinery..." required="#"
4     [(ngModel)]="selected_idMachinery">
5 </div>
6 <div class="form-group">
7   <button type="button" name="searchButton" class="btn btn-primary mb-2"
8     (click)="selectMachinery($event)">Search</button>
9   <button type="button" name="reloadButton" class="btn btn-secondary mb-2 ml-2"
10     (click)="reloadMachinery($event)">Reload</button>
11 </div>
```

Figura 7.6: Esempio utilizzo del databinding (ngModel)

Il meccanismo del databinding è fornito proprio da ngModel, il quale permette di accedere ad una variabile presente nel file *monitoring.component.ts* e modificarla (in questo caso inserire il valore immesso dall'utente).

L'altra particolarità fornita sempre da Angular è la possibilità di accedere ad una funzione (nell'immagine, *selectMachinery(...)*), al verificarsi di un evento, come ad esempio il click sul pulsante.

La funzione più importante svolta all'interno del componente è quella di richiesta dei dati.

I dati vengono richiesti al server tramite l'apposito servizio, che verrà descritto nei capitoli successivi, nel componente però è stata eseguita una sottoscrizione al servizio, tale per cui ogni volta che i dati, nel servizio, cambieranno, questa modifica sarà notificata a tutti gli iscritti (in gergo *subscriber*). Come un meccanismo di newsletter, dove un elemento centrale, notifica agli iscritti eventuali novità (*news*).

Di seguito viene riportata la funzione di sottoscrizione al servizio, la quale avviene all'avvio del componente (*ngOnInit*) e viene disiscritta soltanto alla distruzione del componente (*ngOnDestroy*).

```
1 constructor(private mpService: MachineryProductService) { }
2
3 ngOnInit(): void
4 {
5   this.subscription = this.mpService.getDati().subscribe(res =>
6   {
7     this.machineryData = Object.assign([], res);
8   });
9 }
```

Figura 7.7: Esempio sottoscrizione ad un servizio

Linea 3: tramite il costruttore si importa il servizio.

Linea 5: si accede al servizio e si effettua una sottoscrizione al metodo *getDati*, per cui ogni volta che questo metodo effettuerà una notifica, questa verrà catturata e posta all'interno della variabile *subscription* di tipo *Subscription* (questo serve per effettuare, successivamente, una disiscrizione).

Linea 7: il contenuto di *res* viene copiato in *machineryData*, tramite il metodo *Object.assign()*, in quanto l'operatore "=" in Javascript ha una funzione particolare ed assegnerebbe ad entrambe le variabili il contenuto di *machineryData*

7.1.5 Card

Questo rappresenta un sotto-componente di *Monitoring* e presenta una serie di parametri che consentono al componente padre di mostrare i dati all'utente.

Si è scelto questo approccio perché più modulare e scalabile, non è possibile sapere in anticipo quanti macchinari staranno trasmettendo in un certo istante di tempo. Per cui, in linea con la programmazione orientata agli oggetti, si è deciso di separare questo componente e renderlo a sé stante, in questo modo tramite un semplice ciclo *for* sarà possibile stampare quante *card* saranno necessarie in modo del tutto automatico.

Anche in questo caso si è utilizzato il concetto del databinding passando l'array di macchinari dal componente padre (*Monitoring*) al componente figlio (*Card*).

Con il costrutto *ngFor* è possibile iterare anche su di un componente, passandogli ad ogni ciclo dei dati diversi:

```
1 <!-- Products cards data -->
2 <div *ngFor="let machine of machineryData">
3   <app-card [machineData]="machine"></app-card>
4 </div>
```

Figura 7.8: Esempio utilizzo di *ngFor* per iterare su un componente

I dati vengono immessi nel componente *Card* attraverso il databinding, quindi nel componente che riceve i dati è necessario specificare la variabile che si occuperà di ricevere quei dati. Per fare questo in Angular si deve utilizzare il decoratore *@Input()*:

```
1 @Input()
2 machineData!: ProductionDataModel;
```

Figura 7.9: Esempio utilizzo decoratore *Input* per databinding

7.1.6 Historical

Questo è l'ultimo dei componenti e permette di visualizzare i grafici relativi allo storico di produzione. Non presenta nessuna funzione o nessun costrutto in quanto viene utilizzato come contenitore per i grafici importati da Grafana.

7.2 Grafana

Per la sezione relativa allo storico, si è deciso di utilizzare Grafana in quanto permette di accedere direttamente al database, quindi non c'è la necessità lato front-end di gestire i dati da visualizzare.

L'accesso al database avviene effettuando delle query in linguaggio SQL (per quanto riguarda l'utilizzo di Grafana con MySQL), andando a selezionare le *tuple* che servono per il grafico.

Esistono due tipologie di grafico che si possono creare, il primo è su base temporale (*time based*), ovvero, è possibile selezionare i dati di interesse per l'asse y, mentre sull'asse delle x sarà necessario selezionare una colonna che presenta un dato in formato DATE_TIME, DATE o TIME (o mascherarlo come tale).

La seconda tipologia di grafico (*table*), non prevede la presenza obbligatoria di dati "temporali" in quanto i dati stessi vengono rappresentati sotto forma di tabella, come se fosse il risultato di una query classica al database, in questo però non abbiamo nessun riferimento al tempo. Il formato table permette inoltre la visualizzazione con istogrammi, grafici a torta e molti altri ancora.

Nel progetto sono state utilizzate entrambe le tipologie di grafico. In particolare, in relazione al grafico basato sul tempo, come si può notare nel progetto non esiste nessuna riga di codice relativa alla memorizzazione temporale in quanto, grazie a sequelize, nel database per ogni riga sarà presente una colonna relativa a data e ora di creazione della riga stessa ed una relativa a data e ora di modifica della stessa (coincidente con quella di creazione se non sono avvenute modifiche).

Siccome uno stabilimento produttivo si può comporre di più macchinari connessi, si è deciso di implementare nello storico, la possibilità di visualizzare un solo macchinario o più macchinari o tutti, attraverso l'utilizzo delle "Variables" di Grafana. Ottenendo un risultato come il seguente:

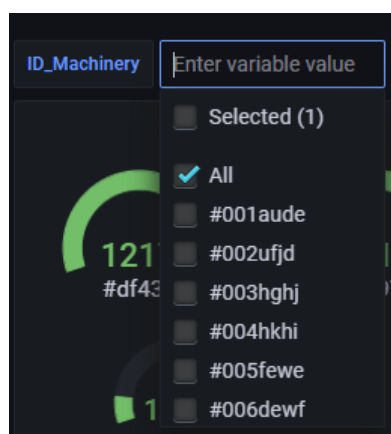


Figura 7.10 Funzione di selezione del macchinario

Nei prossimi sotto-capitoli verranno mostrate le diverse tipologie di grafico utilizzate.

7.2.1 Grafico PASS/FAIL di ogni postazione

Nel grafico in oggetto, sono presenti delle rappresentazioni simili a tachimetri (meglio definiti *gauge*):

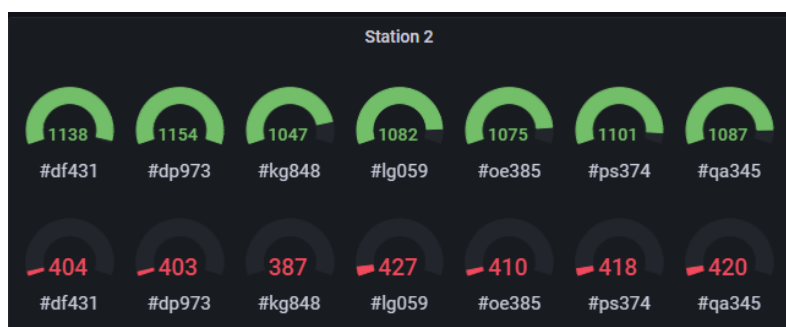


Figura 7.11 Grafici PASS/FAIL per la stazione 2

Per ogni id-prodotto è possibile visualizzare il numero dei “PASS” (in verde nella riga in alto) ed i “FAIL” (in rosso nella riga in basso). Il tutto in relazione al macchinario selezionato, per ogni codice prodotto (#df431, #dp973...).

Questo è stato scelto perché molto spesso, nelle aziende, non si tiene uno storico dei pezzi prodotti buoni e degli scarti ottenuti per ogni postazione, ma soltanto uno storico in funzione del prodotto finito. Questa è una cattiva gestione della produzione in quanto grazie ad un sistema come quello sopra descritto è possibile intervenire preventivamente sulle postazioni che generano più scarti, diminuendo così gli scarti e di conseguenza aumentando la produzione.

7.2.2 Grafico temporale

Un'altra funzionalità importante è quella di tenere traccia giornalmente o settimanalmente, dei pezzi prodotti.

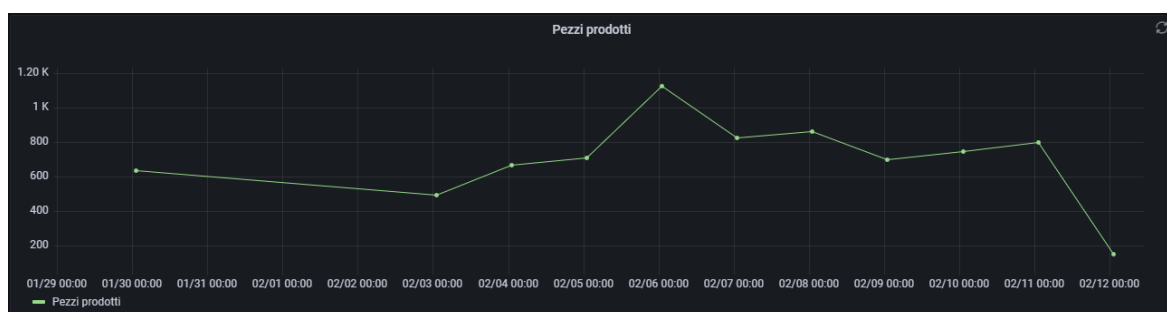


Figura 7.12 Grafico temporale di pezzi prodotti

Non è nulla di nuovo in quanto ogni azienda mette in atto un sistema di controllo della produzione giornaliera. Però la cosa interessante adottata nel grafico sopra presente è la possibilità di visionare, quasi in tempo reale, la produzione odierna. Analizzando soprattutto come varia la produzione nelle diverse fasce orarie.

7.2.3 Grafico errori

Sempre in relazione agli obbiettivi del seguente progetto (manutenzione, efficienza e risparmio), è stato inserito un grafico per il tracciamento degli errori, in funzione del loro codice.

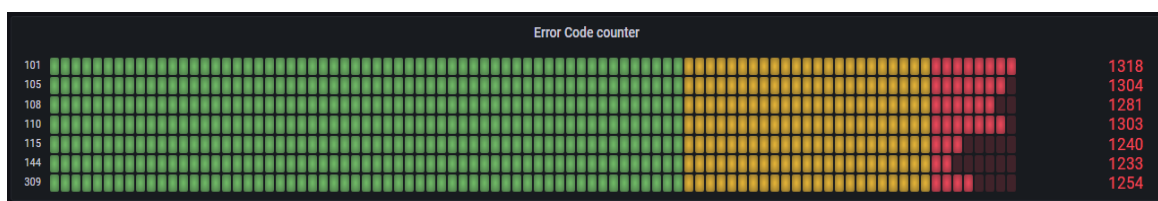


Figura 7.13 Grafico per il conteggio degli errori

Spesso, in un impianto di produzione, si possono verificare svariati errori. Di norma si tende a sistemare il problema senza preoccuparsi di come cercare di evitarlo.

Con un tracciamento del numero degli errori in funzione del loro codice, è possibile controllare quali errori si verificano più spesso in modo da poter agire di conseguenza.

Nel grafico sono stati aggiunti degli indicatori visivi che permettono di verificare quando un certo errore si verifica più o meno spesso degli altri, suddividendoli in tre colori.

Grazie a questo sistema è possibile ridurre i fermi macchina ed incrementare la produzione con un notevole risparmio sui costi di manutenzione/riparazione.

7.3 Servizio

I servizi in Angular sono dei componenti a sé stanti, che svolgono delle operazioni. Essenzialmente sono degli oggetti che includono al loro interno delle funzioni javascript, che possono essere richiamate dai vari componenti, dopo che sono stati importati in questi.

L'importazione avviene tramite il meccanismo del *dependency injection*, un design pattern tipico della programmazione orientata agli oggetti.

In Angular si tratta di un meccanismo per cui Angular stesso, si occupa di creare ed iniettare un'istanza del servizio all'interno del componente.

Per questo motivo la classe deve avere il decoratore `@Injectable()` :

```
1 @Injectable()  
2 export class MachineryProductService implements OnDestroy  
3 { ... }
```

Figura 7.14: Esempio utilizzo Injectable

7.3.1 Richiesta dati

Il servizio, in questo caso, si occupa di effettuare le richieste GET al server e di fornire i dati agli altri componenti del front-end attraverso il metodo `getDati()` :

```
1 private serverUrl: string = 'http://localhost:3000/produzione/monitora';  
2 private stopPolling = new Subject();  
3  
4 constructor(private http: HttpClient) { }  
5  
6 getDati(): Observable<any>  
7 {  
8     return timer(1,1000).pipe(switchMap( () =>  
9         this.http.get(this.serverUrl) ),  
10    retry(1),  
11    share(),  
12    takeUntil(this.stopPolling));  
13 });
```

Figura 7.15: Metodo per richiesta dati al server

Linea 1: viene specificato l'URL del server a cui effettuare le richieste

Linea 2: viene definita una variabile di tipo Subject, la quale permette di bloccare il meccanismo del polling tramite la funzione descritta in figura 7.16

Linea 4: viene importata un'istanza del modulo HttpClient all'interno della variabile http

Linee 6-12: Il metodo ritorna un *Observable* contenente la risposta alla richiesta HTTP effettuata al server nella URL indicata. In particolare viene effettuato un polling di richieste ogni 1000ms attraverso il metodo *timer* che presenta la seguente struttura:

- *timer(x, y):* Funzione che permette di impostare un intervallo di x millisecondi la prima volta e y millisecondi tra una richiesta e l'altra
- *switchMap:* trasforma un input *Observable* in un altro *Observable*, nel nostro caso, trasformeremo ogni numero emesso dal timer in una richiesta HTTP al backend
- *retry(n):* Risottoscrive su un errore n volte, se il campo è vuoto lo farà all'infinito
- *share():* Per default, gli *Observables RxJs* creano un nuovo task per ogni *subscriber* mentre si verifica la *subscription*. Nel nostro caso, tre *subscribers* finirebbero per creare un'istanza del meccanismo di polling 3 volte, quindi 3 richieste HTTP. L'operatore *share()* assicura che ogni *subscriber* utilizzi lo stesso *polling*.
- *takeUntil(...):* Per fermare un *Observable stream* mentre sta emettendo (es. componente distrutto).

Successivamente, ogni volta che si cambia schermata, e si passa quindi ad un altro componente, quel componente viene distrutto e tutti i suoi dati vengono eliminati. Quindi in questo caso non ha senso lasciare il polling delle richieste attivo, per cui è buona norma fermarlo attraverso il metodo precedentemente citato:

```
1 ngOnDestroy(): void {
2   this.stopPolling.next(void 0);
3 }
4
5 stopPollingFunction(): void {
6   this.stopPolling.next(void 0);
7 }
```

Figura 7.16: Metodi per fermare il polling

Linee 1-3: Metodo per effettuare uno stop del polling dopo l'evento di distruzione del componente.

Linee 5-7: Metodo per effettuare uno stop del polling quando necessario, differisce dal precedente, per la possibilità di essere chiamato da altri componenti.

8 Conclusioni e sviluppi futuri

L'elaborato ed il progetto stesso, mostrano in maniera esaustiva lo scopo principale e basilare dell'obiettivo posto inizialmente, il monitoraggio di uno o più sistemi di produzione industriale attraverso dispositivi IoT.

Il progetto ha comportato diverse sfide, ma la sfida più grande è stata sicuramente lo studio e l'apprendimento di diversi linguaggi e framework. Questo ha portato alcune difficoltà in fase di sviluppo, il dover unire diversi livelli applicativi è risultato inizialmente molto complicato, ma grazie ad una community online ben attiva ed a diversi tutorial e corsi, sempre online, è stato possibile portare a termine il progetto e renderlo perfettamente funzionante. Inoltre grazie alla modularità del progetto ed alla presenza di alcuni software (**Postman**) è stato possibile scorporare i tre livelli applicativi e lavorare separatamente su ognuno di essi semplificando il lavoro.

Nonostante ciò, i possibili sviluppi futuri legati a questa tecnologia e applicazione, sono pressoché illimitati. Un'interessante sviluppo potrebbe essere quello legato all'autenticazione, attraverso la quale gestire l'accesso di più aziende. Attualmente il sistema è pensato per lavorare in locale su una rete aziendale privata, per cui non si pone il problema, ma nel caso di utilizzo di servizi di hosting pubblici si rende necessaria l'autenticazione.

Un altro progetto interessante potrebbe essere quello di permettere di interagire con la macchina, impostando alcuni parametri, non critici, direttamente da remoto. Questa possibilità potrebbe ampliare notevolmente la platea di utilizzatori.

A questa però è da aggiungere un livello di protezione robusto ed efficace. Nel caso di sola lettura dei dati, il problema non si pone in quanto i dati per quanto possano essere strutturati non sono per niente sensibili, non contengono informazioni riservate, ma aggiungendo la possibilità di interagire con il macchinario le probabilità di arrecare danno, aumentano esponenzialmente. Non solo è possibile modificare parametri del macchinario, comportando dei danni economici, ma soprattutto si mette a rischio la salute e l'incolumità degli operatori (allagamenti, incendi, schiacciamenti, infortuni...).

Uno sviluppo meno "delicato" ma altrettanto interessante, potrebbe essere quello relativo alla standardizzazione dell'hardware/firmware.

È risaputo, che quando un qualcosa viene reso standard, questo permette risparmi di tempo, fatica e quindi di denaro. La soluzione potrebbe essere quella di implementare, nel modulo per la

comunicazione wifi (il microcontrollore ESP32), un protocollo standard per la comunicazione con i PLC.

La maggior parte dei macchinari industriali, sono oggi comandati da PLC. I PLC per comunicare tra di loro, o con sistemi esterni, utilizzano un protocollo che oramai è diventato uno standard di fatto per l'automazione industriale. Il protocollo MODBUS RTU.

Dotando, per l'appunto, il microcontrollore, con questo protocollo non sarà nemmeno più necessario implementare un hardware dedicato per ogni impianto produttivo (come invece è stato fatto nel progetto) ma sarà necessario soltanto dotare l'hardware di una porta RS232 o RS485 per comunicare in seriale con il PLC ed implementare a livello firmware tutti i comandi necessari per leggere ed eventualmente scrivere sul PLC (questi comandi di norma sono forniti dal costruttore del macchinario).

In conclusione, l'effort apportato per la realizzazione del progetto è sicuramente non trascurabile ma grazie a questo è stato possibile entrare nel dettaglio di ogni livello applicativo, dall'hardware al front-end, potendo così creare un sistema che fosse del tutto flessibile e fatto appositamente per lo scopo per cui è stato ideato. Inoltre ha permesso lo studio e l'apprendimento di tecnologie, oggi, utilizzatissime a livello lavorativo consentendo quindi di ampliare notevolmente il portfolio delle capacità, necessarie per immettersi nel mondo lavorativo.

9 Bibliografia

- [1] Andrew S. Tanenbaum, David J. Wetherall, *Reti di calcolatori*, Pearson, 2018
- [2] Marc Wandschneider, *Learning Node.js. A Hands-On Guide to Building Web Applications in Javascript*, Addison-Wesley, 2013.
- [3] Paolo Atzeni, Stefano Ceri, Piero Fraternali, *Basi di dati*, McGraw-Hill Education, 2018
- [4] Eric Elliot, *Programming Javascript Application*, O'Reilly, 2014
- [5] Giuliano Ortolani, Ezio Venturi, *Manuale di elettrotecnica ed automazione*, Hoepli, 2014
- [6] https://it.wikipedia.org/wiki/Industria_4.0 consultato in data 08/01/2022
- [7] <https://www.mise.gov.it/index.php/it/incentivi/impresa/contratti-di-sviluppo/68-incentivi/2035915-materiale-informativo-industria-4-0> consultato in data 08/01/2022
- [8] <https://it.wikipedia.org/wiki/Uri> consultato in data 09/01/2022
- [9] <https://www.flaviobiscaldi.it/blog/protocollo-http-cosa-e-come-funziona> consultato in data 09/01/2022
- [10] https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol consultato in data 09/01/2022
- [11] <https://www.nextre.it/web-app/> consultato in data 09/01/2022
- [12] https://it.wikipedia.org/wiki/Applicazione_web 09/01/2022
- [13] <https://www.ionos.it/digitalguide/hosting/tecniche-hosting/http-header/> consultato in data 09/01/2022
- [14] https://blog.osservatori.net/it_it/cos-e-internet-of-things consultato in data 09/01/2022
- [15] <https://www.kicad.org/> consultato in data 09/01/2022
- [16] https://docs.kicad.org/6.0/en/getting_started_in_kicad/getting_started_in_kicad.html consultato in data 09/01/2022
- [17] <https://docs.kicad.org/6.0/en/kicad/kicad.html> consultato in data 09/01/2022
- [18] https://it.wikipedia.org/wiki/Arduino_IDE consultato in data 09/01/2022
- [19] <https://www.arduino.cc/en/software> consultato in data 09/01/2022
- [20] <https://nodejs.org/en/about/> consultato in data 09/01/2022
- [21] [https://it.wikipedia.org/wiki/V8_\(motore_JavaScript\)](https://it.wikipedia.org/wiki/V8_(motore_JavaScript)) consultato in data 10/01/2022
- [22] <https://v8.dev/> consultato in data 10/01/2022
- [23] <https://it.wikipedia.org/wiki/Node.js> consultato in data 10/01/2022
- [24] <https://docs.npmjs.com/about-npm> consultato in data 10/01/2022
- [25] <https://dev.mysql.com/doc/refman/8.0/en/> consultato in data 10/01/2022
- [26] <https://www.mysql.com/it/why-mysql/> consultato in data 10/01/2022

- [27] https://it.wikipedia.org/wiki/Modello_relazionale consultato in data 10/01/2022
- [28] <https://www.html.it/pag/32137/introduzione-ai-rdbms/> consultato in data 10/01/2022
- [29] <https://lefred.be/content/top-10-reasons-for-nosql-with-mysql/> consultato in data 10/01/2022
- [30] <https://angular.io/docs> consultato in data 10/01/2022
- [31] <https://www.elprocus.com/pi-filter-circuit-working-and-its-applications/> consultato in data 13/01/2022
- [32] https://it.wikipedia.org/wiki/Alimentatore_elettrico consultato in data 13/01/2022
- [33] <https://www.espressif.com/en/products/socs/esp32> consultato in data 13/01/2022
- [34] <https://www.arduino.cc/reference/en/> consultato in data 14/01/2022
- [35] <https://arduinojson.org/> consultato in data 14/01/2022
- [36] <https://www.npmjs.com/package> consultato in data 15/01/2022
- [37] <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> consultato in data 15/01/2022

Appendice A – LINK

- [a] https://github.com/leo96it/Tesi_webApp
- [b] <http://www.advanced-monolithic.com/pdf/ds1117.pdf>
- [c] <https://www.onsemi.com/pdf/datasheet/lm1117-d.pdf>
- [d] <https://www.diodes.com/assets/Datasheets/ds11107.pdf>
- [e] <https://www.vishay.com/docs/83673/sfh618a.pdf>
- [f] <https://www.espressif.com/en/support/documents/technical-documents>
- [g] <https://arduinojson.org/v6/assistant/>
- [h] <https://dev.mysql.com/downloads/installer/>
- [i] <https://git-scm.com/downloads>
- [j] <https://nodejs.org/it/download/>
- [k] <https://grafana.com/grafana/download>