

# **Generatore di frequenze con vari accessori gestito via web**

**Titolo del progetto:** Generatore di frequenze con vari accessori gestito via web  
**Alunno/a:** Giulio Bosco  
**Classe:** SAMT I4AA  
**Anno scolastico:** 2019/2020  
**Docente responsabile:** Fabrizio Valsangiacomo

1	Introduzione .....	3
1.1	Informazioni sul progetto.....	3
1.2	Abstract.....	3
1.3	Scopo.....	3
	Analisi .....	4
1.4	Analisi del dominio .....	4
1.5	Analisi e specifica dei requisiti .....	4
1.6	Use case .....	6
1.7	Pianificazione.....	6
1.8	Analisi dei mezzi .....	6
1.8.1	Software.....	6
1.8.2	Hardware .....	6
2	Progettazione .....	7
2.1	Design dell'architettura del sistema .....	7
2.2	Design dei dati e database .....	7
2.3	Design delle interfacce.....	7
2.4	Design procedurale .....	7
3	Implementazione.....	8
4	Test .....	8
4.1	Protocollo di test .....	8
4.2	Risultati test .....	9
4.3	Mancanze/limitazioni conosciute .....	9
5	Consuntivo .....	9
6	Conclusioni.....	9
6.1	Sviluppi futuri .....	9
6.2	Considerazioni personali.....	9
7	Bibliografia .....	9
7.1	Bibliografia per articoli di riviste: .....	9
7.2	Bibliografia per libri .....	9
7.3	Sitografia.....	9
8	Allegati .....	10

## 1 Introduzione

### 1.1 Informazioni sul progetto

Docente Responsabile: Fabrizio Valsangiacomo

Apprendista: Giulio Bosco

Data inizio progetto: 05.09.2019

Data consegna progetto: 20.12.2019

### 1.2 Abstract

E' una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all'utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

- **Background/Situazione iniziale**
- **Descrizione del problema e motivazione:** Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
- **Approccio/Metodi:** Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche...)? Quale è stata l'entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
- **Risultati:** Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

Esempio di abstract:

*As the size and complexity of today's most modern computer chips increase, new techniques must be developed to effectively design and create Very Large Scale Integration chips quickly. For this project, a new type of hardware compiler is created. This hardware compiler will read a C++ program, and physically design a suitable microprocessor intended for running that specific program. With this new and powerful compiler, it is possible to design anything from a small adder, to a microprocessor with millions of transistors. Designing new computer chips, such as the Pentium 4, can require dozens of engineers and months of time. With the help of this compiler, a single person could design such a large-scale microprocessor in just weeks.*

### 1.3 Scopo

Creare un interfaccia di gestione per un generatore di frequenze ultrasoniche, con la possibilità di essere gestito da più utenti, quindi si necessita anche una piattaforma per gestire gli utenti.

Il generatore deve poter essere acceso e spento tramite un telecomando, tramite un suono (un suono qualunque più alto di una determinata soglia, in decibel), tramite interfaccia WEB.

## 2 Analisi

### 2.1 Analisi del dominio

Questo capitolo dovrebbe descrivere il contesto in cui il prodotto verrà utilizzato, da questa analisi dovrebbero scaturire le risposte a quesiti quali ad esempio:

- Background/Situazione iniziale
- Quale è e come è organizzato il contesto in cui il prodotto dovrà funzionare?
- Come viene risolto attualmente il problema? Esiste già un prodotto simile?
- Chi sono gli utenti? Che bisogni hanno? Come e dove lavorano?
- Che competenze/conoscenze/cultura posseggono gli utenti in relazione con il problema?
- Esistono convenzioni/standard applicati nel dominio?
- Che conoscenze teoriche bisogna avere/acquisire per poter operare efficacemente nel dominio?
- ...

### 2.2 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Gestione generatore tramite WEB
Priorità	1
Versione	1.0
Note	-
	<b>Sotto requisiti</b>
Sub REQ 1	0-25'000 Hz
Sub REQ 2	Gestione Wireless del generatore
Sub REQ 3	Regolazione della frequenza
Sub REQ 4	Accensione/Spegnimento del generatore
Sub REQ 5	Mantenimento memoria
Sub REQ 6	Scatola finale cablata e protetta (Priorità 3)
Sub REQ 7	Minima attenuazione segnale (Priorità 3)

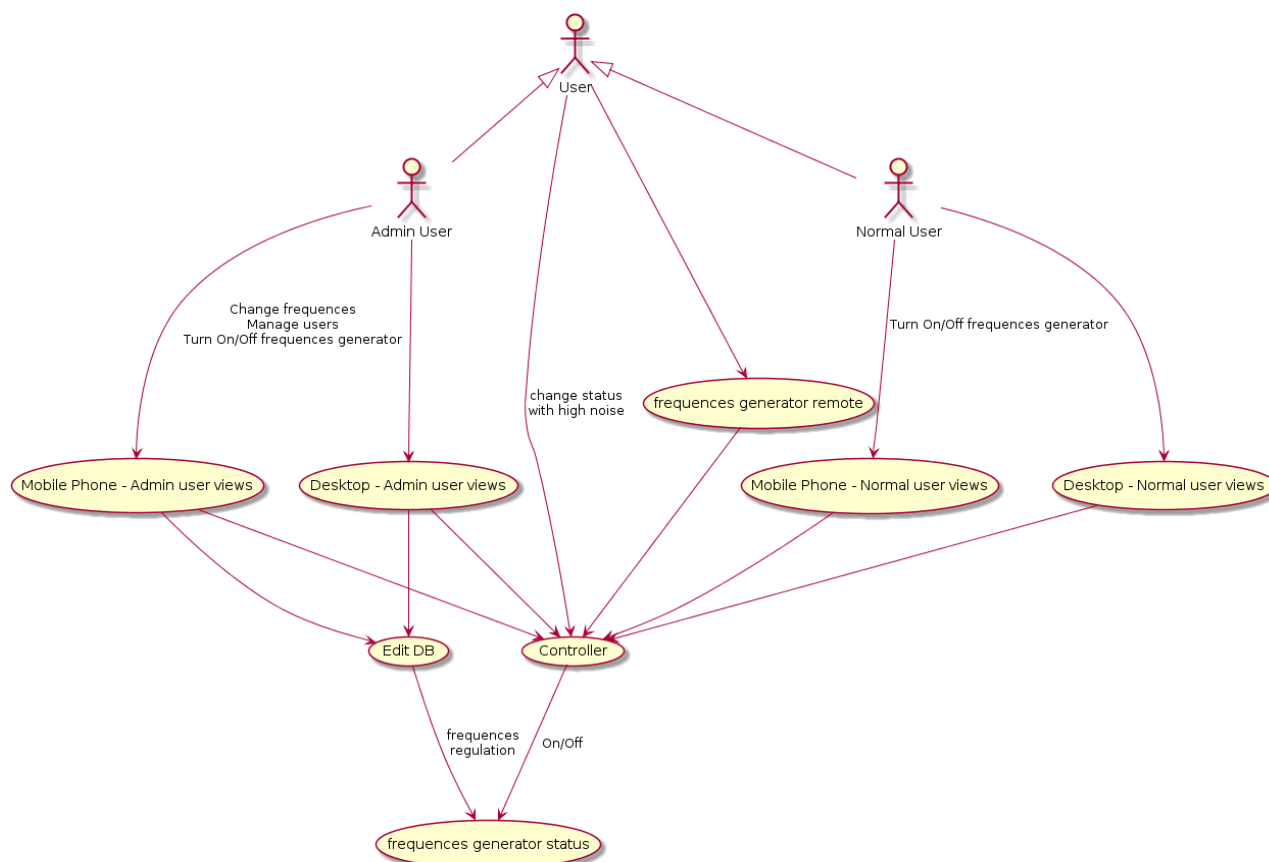
ID: REQ-02	
Nome	Gestione generatore tramite Telecomando
Priorità	2
Versione	2.0
Note	-
	<b>Sotto requisiti</b>
Sub REQ 1	Tasto Start
Sub REQ 2	Tasto Stop

ID: REQ-03	
Nome	Gestione generatore tramite Rumore
Priorità	2
Versione	2.0

<b>Note</b>	-
	<b>Sotto requisiti</b>
<b>Sub REQ 1</b>	Gestione tramite decibel
<b>Sub REQ 2</b>	Timer per spegnimento

<b>ID: REQ-04</b>	
<b>Nome</b>	Pagina WEB - Amministrazione utenti
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	-
	<b>Sotto requisiti</b>
<b>Sub REQ 1</b>	Funzionamento con maggiori browser
<b>Sub REQ 2</b>	Funzione di amministrazione
<b>Sub REQ 3</b>	Pagina gestione utenti
<b>Sub REQ 4</b>	Pagina gestione permessi
<b>Sub REQ 5</b>	Funzione di utente base
<b>Sub REQ 6</b>	Password provvisoria (Priorità 3)

## 2.3 Use case



Questo progetto ha come scopo di sviluppare la gestione di un generatore di frequenze tra 0 e 25'000 Hz, il quale deve poter essere gestito in maniere diverse. La frequenza deve poter essere regolata tramite una pagina web, solamente dall'utente amministratore. Mentre per quanto riguarda l'accensione e lo spegnimento del generatore, deve poter essere fatto tramite la pagina web (da un utente di base), tramite un telecomando e tramite un suono regolato in decibel (che di deve poi spegnere allo scadere di un timer).

Nel diagramma si può notare, un utente che tramite un forte rumore oppure con un telecomando può accendere o spegnere il generatore.

Oppure l'utente tramite l'applicativo (con un utente di base) può accendere o spegnere il generatore, mentre se esegue il login con un utente amministratore deve essere in grado di accendere o spegnere il generatore, cambiare la regolazione del generatore e gestire gli utenti.

## 2.4 Pianificazione

Prima di stabilire una pianificazione bisogna avere almeno una vaga idea del modello di sviluppo che si intende adottare. In questa sezione bisognerà inserire il modello concettuale di sviluppo che si seguirà durante il progetto. Gli elementi di riferimento per una buona pianificazione derivano da una scomposizione top-down della problematica del progetto.

La pianificazione può essere rappresentata mediante un diagramma di Gantt

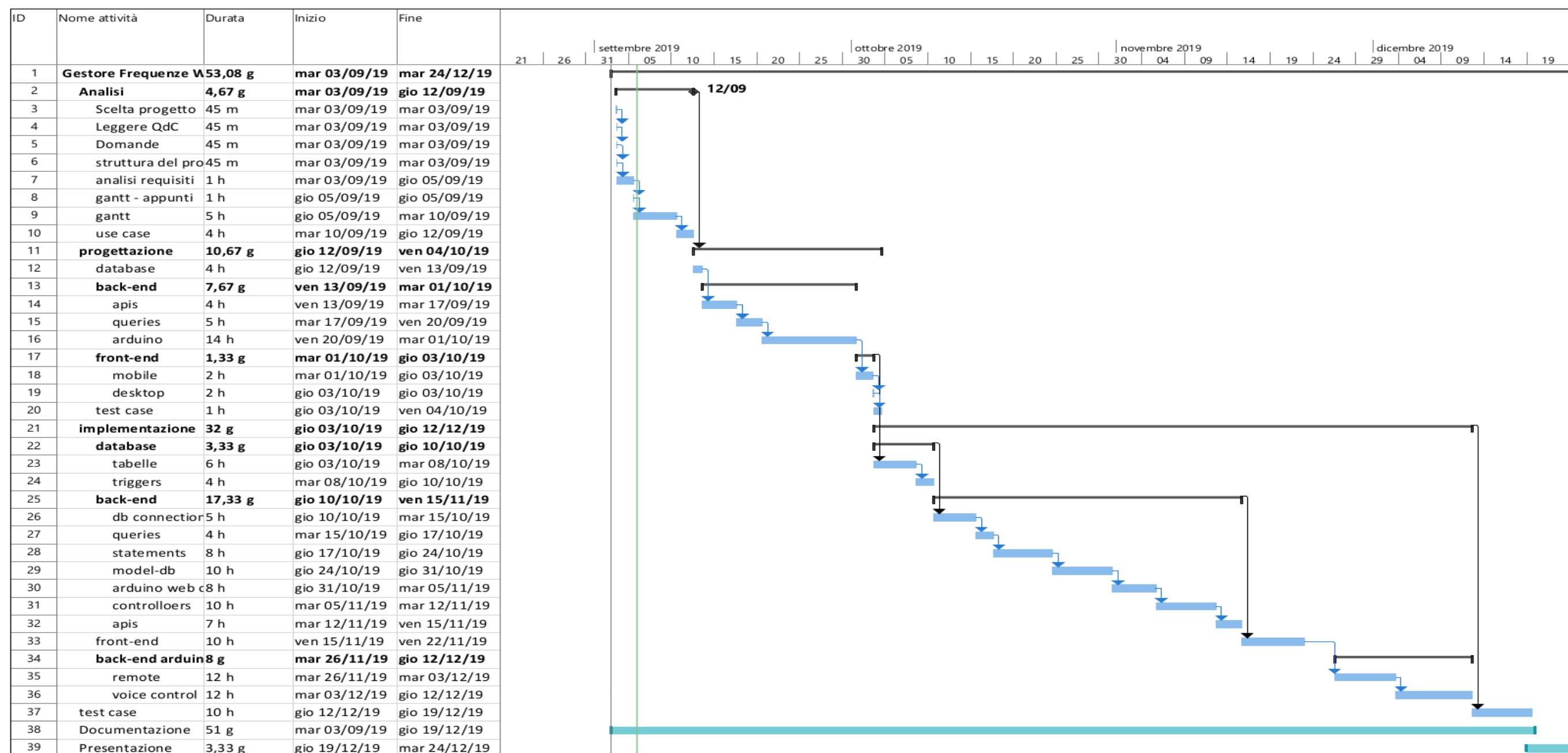


Figura 1: Diagramma di GANTT

Per lo sviluppo di questo progetto, ho pianificato lo sviluppo con 5 giorni di analisi, 10 di progettazione e 32 di implementazione, per giorni si intendono i giorni di progetto, che sono il martedì, giovedì e venerdì, in tutti e tre i giorni saranno a disposizione 4 ore di lezione (45 minuti per ora), ovvero un totale di 3 ore. La prima pianificazione che avevo pensato, sarebbe finita a metà novembre, nella quale sarebbe stato compreso l'utilizzo di un framework nello sviluppo del progetto nel lato back-end. Siccome avrei finito il progetto molto prima ho deciso che per una volta avrei scritto tutto il codice da solo. Quindi il codice di astrazione del database lo ho scritto a mano, così come la costruzione delle interfacce di comunicazione che vi sono fra l'applicativo back-end e front-end. Ho fatto questa scelta siccome ho un progetto relativamente semplice, quindi sviluppare tutto il codice che solitamente è in un framework mi avrebbe permesso di capire come funzionano la maggior parte dei framework a basso livello, questo mi porterebbe molta esperienza e conoscenza.

## **2.5 Analisi dei mezzi**

### **2.5.1 Software**

- Arduino IDE (v1.8.9)
- Atom IDE (v1.38.2)
- plantuml (Version 1.2019.9)
- Libre Office (Version: 6.2.5.2)
- Java (v1.8)
  - Libreria org.gretty (v2.2.0)
  - Libreria Junit (v4.12)
  - Libreria JDBC (MySQL Connector Java – v8.0.11)
  - Libreria JSON (org.json:json:20171018)

### **2.5.2 Hardware**

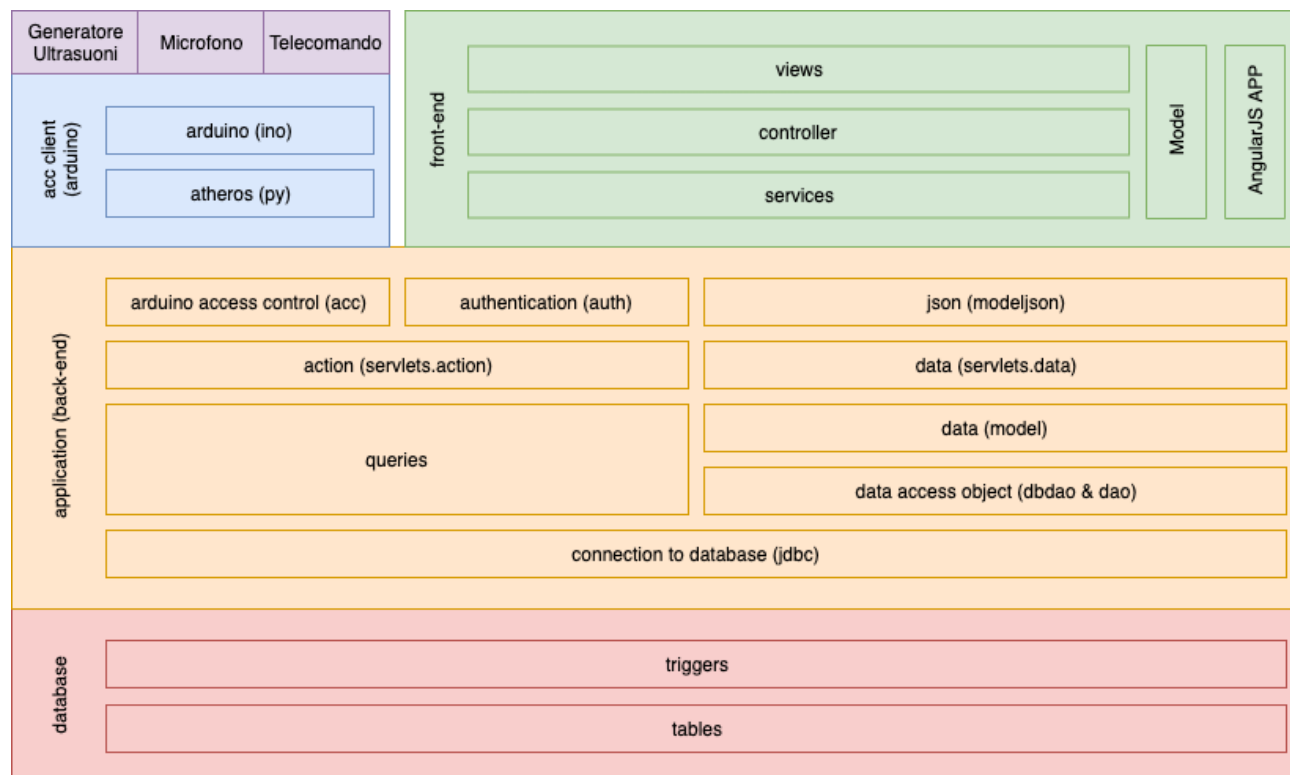
- Mac Book Pro 2018 Intel Core i7 CPU 3.1GHz RAM 16GB
- Arduino UNO (Rev 3)
- Arduino YUN (Rev 2)
- Raspberry PI 3 Model B
- Circuito amplificatore



### 3 Progettazione

#### 3.1 Design dell'architettura del sistema

Il progetto è sviluppato su quattro diversi elementi, che sono messi in evidenza nello schema sottostante:



Alla base di tutto vi è il database nel quale vengono salvati tutti i dati dell'applicativo, il quale è diviso in 2 due parti, il back-end ed il front-end.

Tutto il sistema, verrà strutturato in maniera modulare, in modo che tutti gli elementi del progetto siano indipendenti, quindi siano più facili da testare, singolarmente, per cercare di avere meno problemi possibili durante il corso del progetto, che siano anche più facili da mantenere e modificare in futuro e che possano essere riutilizzati in progetti futuri. Alcuni moduli del progetto verranno ripresi da vecchi progetti che ho svolto in questi anni alla SAMT e probabilmente riscritti siccome ora ho più conoscenze ed esperienza, viene utilizzato lo stesso concetto, ma riscrivendo il codice solamente in parte oppure totalmente. I vari moduli, possono avere dei sotto moduli, per semplificare ancora lo sviluppo ed il mantenimento dell'applicativo.

Per ogni modulo è stata fatta una progettazione, in alcuni più approfondita mentre in altri meno. Questo perché a dipendenza dei moduli vi sono più o meno complessi, mentre per quelli che si pensa di prendere da dei vecchi progetti non è proprio stata fatta, verranno fatti degli adattamenti direttamente durante lo sviluppo.

Per esempio, la parte relativa al *acc client (Arduino)* verrà preso dal progetto *domotics* (<https://github.com/giuliobosco/domotics>), come il modulo *application (front-end)*.

##### 3.1.1 Restful API

Questo applicativo WEB, è sviluppato suddiviso in 2 elementi, front-end e back-end, nel lato front-end vi sono le grafiche dell'applicativo e l'interpretazione dei dati. Mentre nel lato back-end vi è l'interazione con il

database, quindi la creazione e l'aggiornamento dei dati. Per costruire l'applicativo con questa struttura ho deciso di utilizzare le best practices del trend attuale dello sviluppo web.

Quindi per comunicare fra i due elementi dell'applicativo utilizzerò le Restful API che fondamentalmente sono delle stringhe in formato JSON, le quali possono essere richieste tramite delle richieste http, con i suoi vari metodi. Per progettare le API, mi sono documentato su [restapitutorial.com](http://restapitutorial.com).

Esempio di Restful API (<http://localhost/api/v1/users>):

```
{
  "users": [
    {
      "username": "giulio.bosco",
      "firstname": "Giulio",
      "lastname": "Bosco"
    },
    {
      "username": "fabrizio.valsangiaco",
      "firstname": "Fabrizio",
      "lastname": "Valsangiaco"
    }
  ]
}
```

Ad ogni API viene associato un indirizzo nel web server, come per esempio:

<http://localhost/api/servletAddress>

Il protocollo HTTP prevede la possibilità di implementare diversi metodi per eseguire le richieste e diverse possibili risposte per ogni richiesta. Le richieste più comuni sono GET, POST, PUT e DELETE, le quali sono implementate nei relativi metodi per ogni servlet:

- doGet: serve per eseguire la richiesta GET  
serve per richiedere gli elementi della api, se termina con un numero questo deve essere l'id dell'elemento (riferimento con SQL: `SELECT * FROM table WHERE id=?`). Altrimenti ritorna tutti gli elementi (riferimento con SQL: `SELECT * FROM table`).
- doPost: serve per eseguire la richiesta POST  
serve per creare un nuovo elemento (riferimento SQL: `INSERT INTO table (?)`).
- doPut: serve per eseguire la richiesta PUT  
serve per aggiornare un elemento (riferimento SQL: `UPDATE table SET x=? WHERE id=?`).
- doDelete: serve per eseguire la richiesta DELETE  
serve per eliminare un elemento (riferimento SQL: `DELETE FROM table WHERE id=?`).

Ognuna di queste richieste può avere una serie di risposte, le quali sono rappresentate da un numero e da una stringa di descrizione, il numero è sempre composto di 3 cifre, la prima indica il tipo di risposta, che può essere mentre le seconde 2 cifre identificano la risposta:

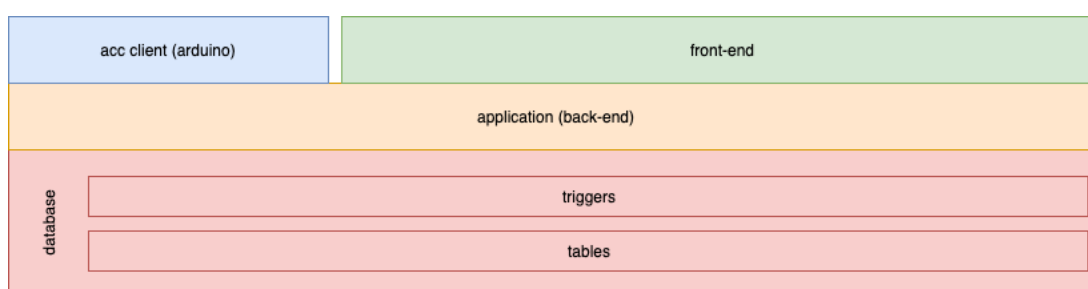
- 1xx Informational: risposta al client di tipo informativo
- 2xx Success: la richiesta ha una risposta con esito positivo
- 3xx Redirection: ridirezionamento su un'altra pagina.
- 4xx Client Error: il client ha fatto una richiesta non valida
- 5xx Server Error: la richiesta ha provocato un errore sul server

Qui sotto sono elencati i metodi più frequenti ed utilizzati (soprattutto per quanto riguarda le Restful API)

ID Risposta	Stringa	Descrizione
200	OK	la richiesta ha una risposta con esito positivo
201	CREATED	la richiesta ha una risposta con esito positivo, creato (p.s.: creato record MySQL)

204	NO CONTENT	risposta con esito positivo, ma non ha contenuto
304	NOT MODIFIED	redirect non modificato
400	BAD REQUEST	richiesta sconosciuta
401	UNAUTHORIZED	non autorizzato per eseguire la richiesta
403	FORBIDDEN	richiesta possibile ma non accettabile dal server, con autenticazione non cambia
404	NOT FOUND	elemento non trovato
405	NOT ACCEPTABLE	richiesta non accettabile
409	CONFLICT	richiesta non processabile, perché contiene dei conflitti (conflitti nella modifica)
500	SERVER ERROR	errore nel server, la richiesta ha provocato un errore nel server
501	NOT IMPLEMENTED	l'elaborazione della richiesta non è ancora stata implementata

### 3.2 Design dei dati e database



Per lo sviluppo del database, ho prima di tutto creato il minimo indispensabile per il progetto, quindi tutte le tabelle di cui necessito, tutti gli attributi, senza il quale il progetto non funziona.

Lista delle tabelle:

- generators
- users
- groups
- permissions

Dopo di che ho pensato potesse essere una buona idea tenere traccia delle operazioni eseguite sul database. Questo perché dal lato della piattaforma WEB, il progetto è piccolo. Quindi potrei investire del tempo nello sviluppare questa parte del progetto, che potrebbe comunque essere riutilizzata in qualunque progetto.

Per eseguire i log delle azioni effettuate sulle banche dati, vi sono diversi modi:

1. Eseguire il log delle azioni di tutte le tabelle in una tabella di log, nella quale si inserisce la query eseguita, l'autore, la data e l'ora.
2. Eseguire il log in una tabella dedicata per ogni tabella, nella quale si inseriscono tutte le azioni che vengono eseguite sulle tabelle.
3. Eseguire il log nell'applicativo, quindi inserire una parte del software che esegua il log del database.
4. Si potrebbe utilizzare un software di monitoraggio della banca dati.

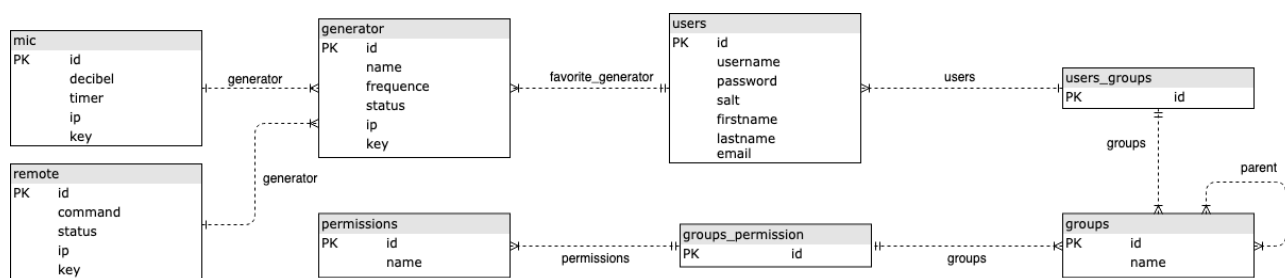
Analizzando queste opzioni ho trovato questi pro e questi contro, di ogni metodologia.

Metodo	Punti a favore	Punti a sfavore
1	<ul style="list-style-type: none"> <li>• Semplice implementazione e integrazione con la banca dati</li> <li>• Facile ricostruzione del database</li> </ul>	<ul style="list-style-type: none"> <li>• Difficile creare dei report sui log</li> </ul>
2	<ul style="list-style-type: none"> <li>• Facilita generazione di report sulle azioni eseguite sul database</li> </ul>	<ul style="list-style-type: none"> <li>• Implementazione del database più complessa</li> </ul>

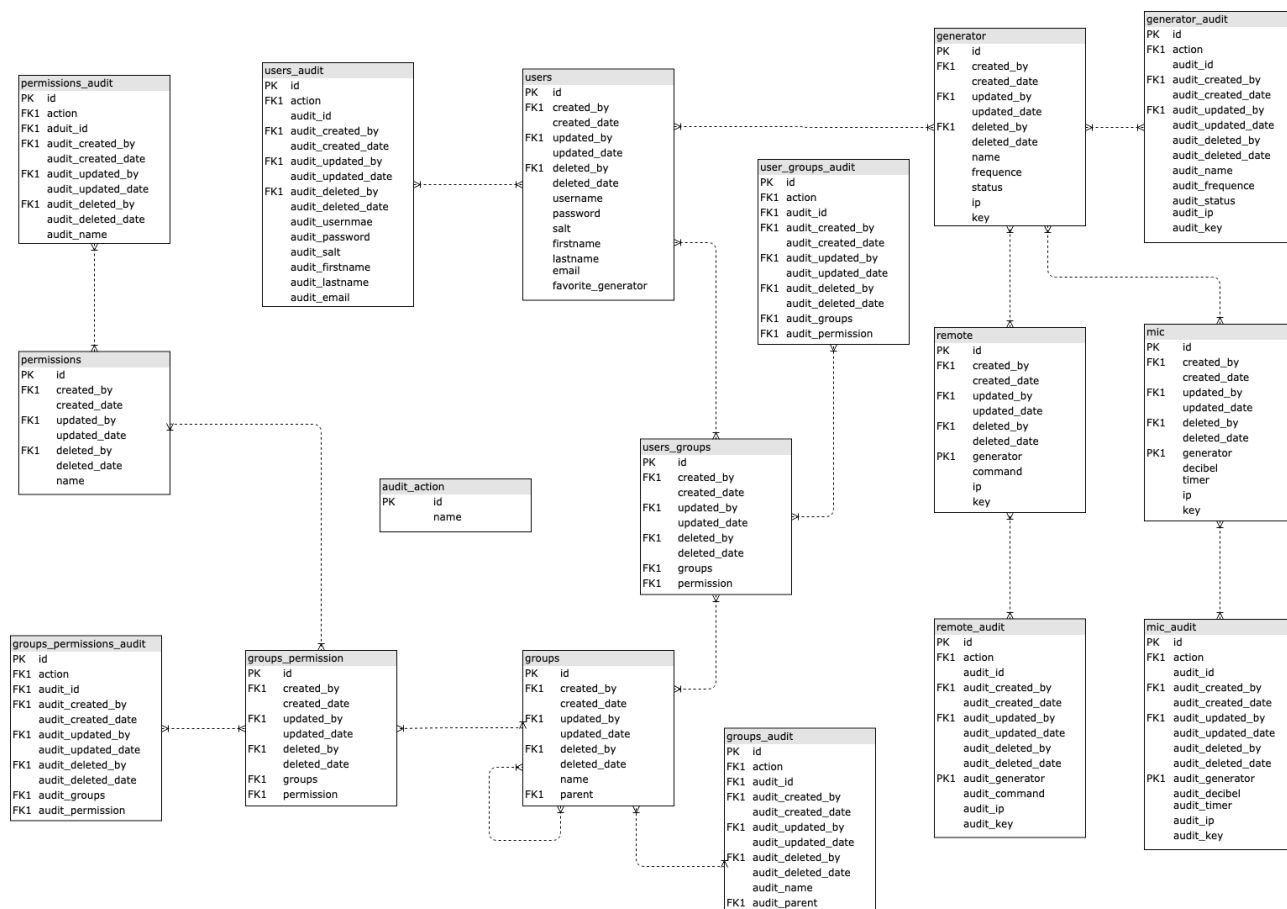
	<ul style="list-style-type: none"> <li>• mantiene separati i dati</li> </ul>	
3	<ul style="list-style-type: none"> <li>• Facile implementazione del database</li> </ul>	<ul style="list-style-type: none"> <li>• Complica la struttura del software</li> <li>• Nel caso in cui vi fossero in futuro più elementi che interagiscono con lo stesso database, i log non sarebbero più autentici</li> </ul>
4	<ul style="list-style-type: none"> <li>• Semplificherebbe il codice</li> </ul>	<ul style="list-style-type: none"> <li>• Dovrei prendere del tempo per imparare ad utilizzare un altro software</li> <li>• Richiederebbe più risorse</li> </ul>

Analizzando questi punti, ho deciso di utilizzare il secondo metodo.

Del database, vi sono 2 schemi. Uno che contiene lo schema di base, quindi solamente le informazioni rilevanti per il progetto. Ed uno completo che comprenderà anche gli attributi e le tabelle legati al log.



In questo schema vi sono le tabelle con gli attributi minimi, per far funzionare l'applicativo. Vi è una tabella `generator` che dà la possibilità di inserire diversi generatori, i quali hanno un nome, una frequenza alla quale devono lavorare, uno stato, un IP ed una key (pensata per creare una comunicazione sicura fra il server ed i controller degli altoparlanti). Questo per permettere al progetto di essere espandibile. Dopo di che vi sono le tabelle **users**, **groups**, e **permissions**, che servono per gestire i permessi. fra le varie tabelle vi sono anche le tabelle ponte, per permettere le relazioni molti a molti.




In questo schema si possono notare molte più tabelle e più attributi, per ogni tabella che vi era nello schema antecedente, 6 attributi, che servono per salvare la data e l'autore delle azioni principali che si possono fare sul database (create, update, delete). Inoltre, vi è un'altra tabella, nella quale si inseriscono gli audit, cioè gli stessi parametri, con in più l'azione eseguita ed un id per ogni audit. Le relazioni fra le tabelle di audit e la tabella `audit\_aciton` non sono rappresentate, come le relazioni fra tutti i campi discussi prima e la tabella `users`, questo per permettere una miglior leggibilità dello schema.

### 3.3 Design delle interfacce

Le interfacce grafiche sono state pensate per dispositivi mobile, e poi adattate a desktop, questo perché la maggior parte degli utenti utilizza dispositivi mobili, con schermi lunghi e stretti che richiedono l'utilizzo delle dita come puntatore, quindi si necessitano icone grandi. Poi le interfacce sono state adattate anche ai dispositivi mobili.

Le interfacce principali sono quella di login (per loggarsi nell'applicativo), quella degli utenti (lista di utenti), quella di modifica dei dati degli utenti e quella della gestione del generatore di frequenze.

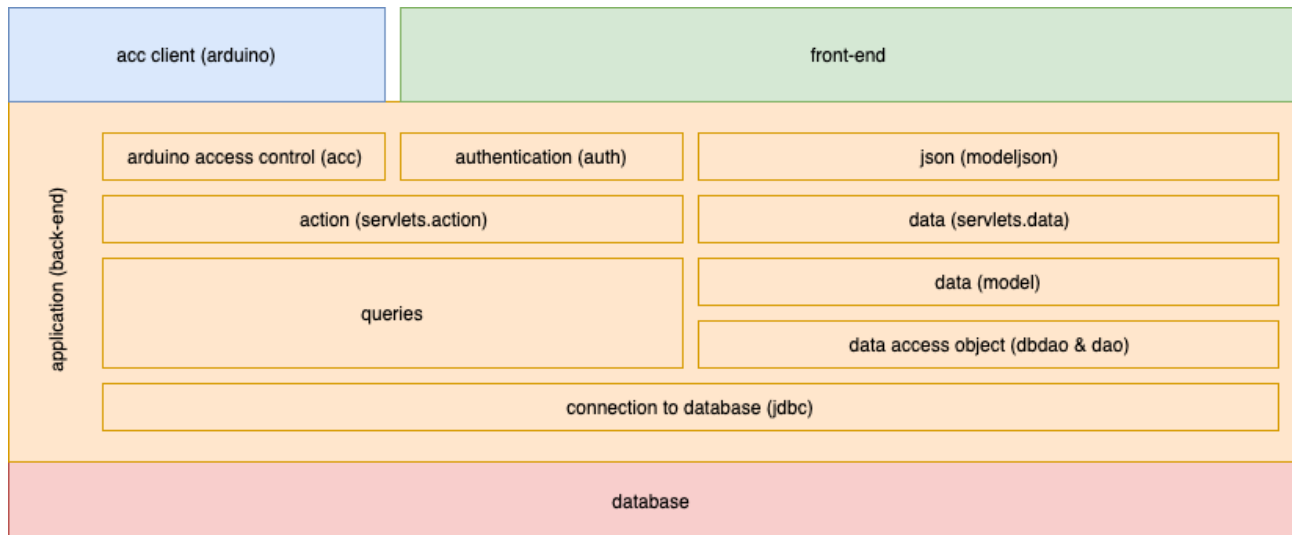
<b>Freqline</b> Gestione Generatore di frequenze via web <input type="text" value="username:txt"/> <input type="password" value="password:pwd"/> <div style="color: red; font-size: small;">error</div> <input type="button" value="login:btn"/>	<div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">status users new</div> <div style="text-align: center;">   <input type="text" value="5000 MHz"/>  <input type="text" value="60 db"/>  <input type="text" value="3600 s"/>  <input type="button" value="save"/> </div> <div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">copy giulio bosco</div>	<div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">status users new</div> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">Admin 0</td></tr> <tr><td style="text-align: center;">User 0</td></tr> <tr><td style="text-align: center;">Admin 1</td></tr> <tr><td style="text-align: center;">User 1</td></tr> </table> <div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">copy giulio bosco</div>	Admin 0	User 0	Admin 1	User 1	<div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">status users new</div> <div style="padding: 5px;"> <input type="text" value="username:txt"/>  <input type="text" value="nome:txt"/>  <input type="text" value="cognome:txt"/>  <input type="text" value="email:txt"/>  <input type="text" value="gruppo:txt"/>  <input type="password" value="password:txt"/>  <input type="button" value="ok"/> </div> <div style="background-color: #007bff; color: white; padding: 2px; text-align: center;">copy giulio bosco</div>
	Admin 0						
	User 0						
	Admin 1						
User 1							

Le interfacce grafiche principali sono 4, una per il login nell'applicativo, una per gestire il generator, una con l'elenco delle pagine ed una per la modifica dei dati.

- Mockup 0: Si vede il nome dell'applicativo, con sotto 2 campi di input di tipo text box, una per lo username, ed una per la password. Sotto vi è un campo di testo di colore rosso solitamente nascosto, che mostra gli errori, come la password sbagliata ed infine vi è un input di tipo submit per eseguire il login.
- Mockup 1: È la pagina di gestione del generatore di frequenze, nel quel vi è un interruttore che cambia lo stato del generatore, poi vi è sino 3 input per numeri, i quali servono rispettivamente per la frequenza del generatore (in megahertz). Poi vi è un input per la regolazione dei decibel del microfono ed uno per i secondi dopo i quali viene spento il generatore (se acceso dal microfono). Infine vi è un input di tipo submit per salvare le modifiche fatte.
- Mockup 2: La lista degli utenti, accessibile solamente se l'utente è amministratore, contiene una lista con tutti gli utenti dell'applicativo. Viene mostrato nome e cognome. Quando viene cliccato new (in alto a destra, per creare un nuovo utente), oppure il nome di un utente, viene aperto il Mockup 3, con tutti i campi vuoti, nel caso di un utente nuovo, mentre con i dati dell'utente se è già esistente, serve per modificarlo.
- Mockup 3: Vi sono i dettagli dell'utente, quindi 5 input di tipo text ed una select. Gli input di tipo text servono per lo username, il nome, il cognome, l'indirizzo email e la password. La select serve per la selezione del gruppo.

### 3.4 Design procedurale

#### 3.4.1 Applicativo (back-end)



Per quanto riguarda la progettazione del back-end, che sarà abbastanza complesso, siccome ho deciso di non utilizzare un framework per interfacciarmi con il database e con l'interfaccia grafica. Quindi l'intero codice deve essere scritto a mano. L'applicativo deve contenere un sistema di gestione degli utenti, con differenti livelli di permessi basata su database e vi deve essere un sistema gestire il generatore di frequenze.

Siccome il lato back-end dell'applicativo è molto complesso, ho deciso di suddividerlo in dei sotto moduli:

- connection to database (jdbc): layer nel quale avviene la connessione al database
- data access object (dbdao & dao): interfaccia dei dati con il database
- data (model): rappresentazione dei dati
- data (servlets.data): una sottocartella nella quale vengono esposti i dati
- actions (servlets.action): una sotto cartella di *servlets* nella quale vengono esposte le azioni possibili sull'applicativo
- json (jsonmodel): in questo modulo, vengono trasformati gli elementi modello in elementi in formato JSON, usati come interfaccia per il lato front-end dell'applicativo
- authentication (auth): modulo di autenticazione e gestione dei permessi
- arduino access controll (arduino): modulo di gestione del controller arduino

I vari moduli, non verranno sviluppati nello stesso ordine in cui sono posti ora, siccome dipendono l'uno dall'altro, verranno sviluppati in un ordine che mi permetta di sviluppare al meglio il software.

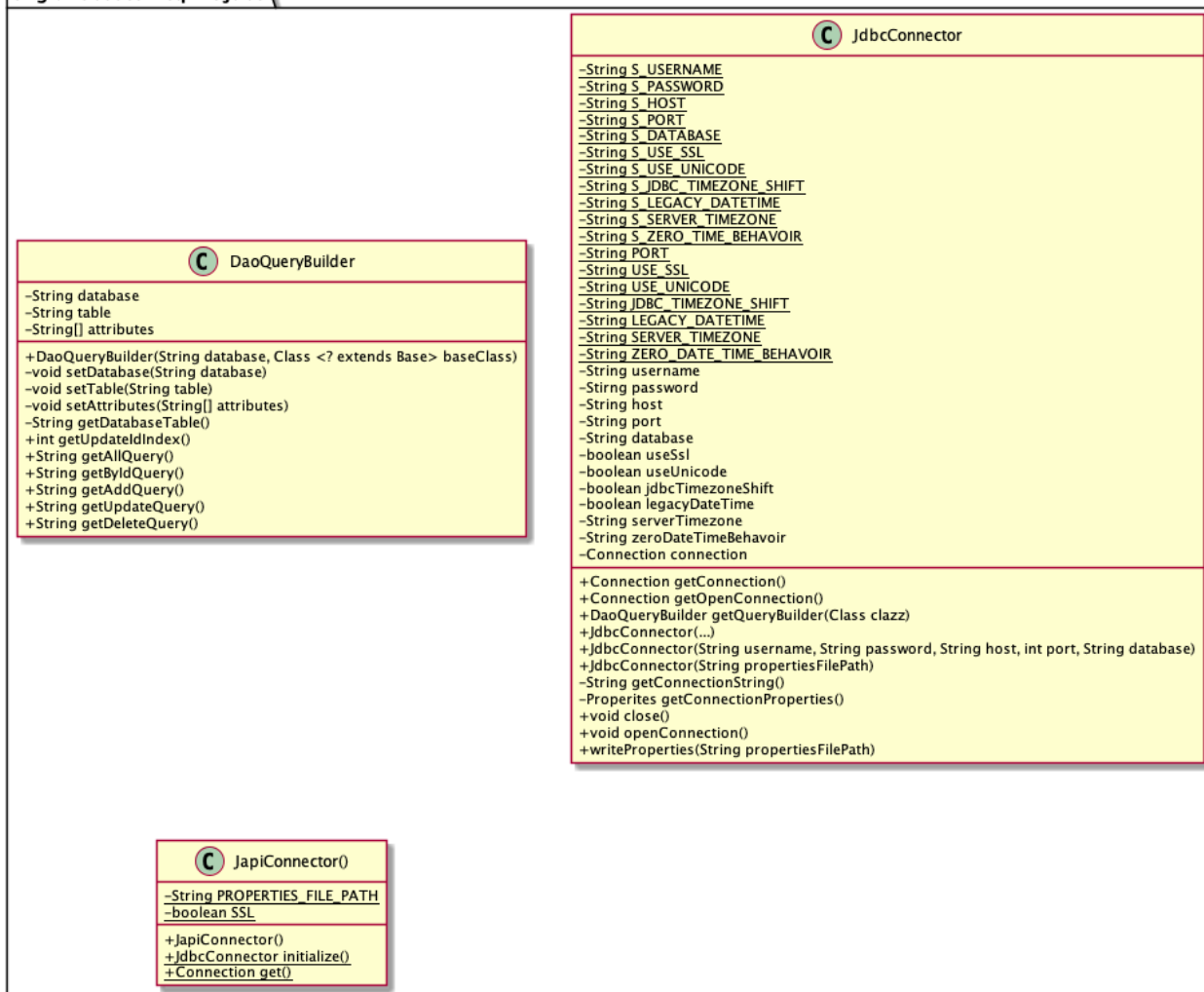
##### 3.4.1.1 connection to database (jdbc)

La connessione al database avviene tramite il driver di default di java, che deve poter essere istanziata in due modi:

- utilizzando un costruttore e passando i vari parametri per la connessione
- utilizzando un file di properties

Una volta istanziata la connessione, bisogna poterla aprire, chiudere e bisogna poter scrivere in un file di properties le proprietà utilizzate al momento.

ch.giulibosco.frequeline.jdbc



Nel seguente diagramma si vede, il modulo di connessione al database, che è composto di una classe che serve per creare una connessione al database, che può essere utilizzata in diversi progetti, nel quale si possono configurare le opzioni di connessione più comuni, così che non vi sia il rischio di fare errori di battitura creando la stringa di connessione. Per ogni opzione vi è un valore di default, così che non sia necessario impostare tutti i valori. Mentre hostname, username, password e database sono sempre obbligatori. Siccome il codice del progetto è posto su un repository pubblico di GitHub, è buon abitudine non salvare nel codice le credenziali in generale, per questo motivo ho deciso di implementare anche una modalità di istanziazione della connessione tramite file di properties che non viene pubblicato con il codice.

### 3.4.1.2 data access object (dao)

L'interfaccia dei dati fra le rappresentazioni dei modelli e il database (*dbdao*), che sfrutterà il modulo di connessione al database, per collegarsi ad esso. Verrà implementato sul modello DAO (Data Access Object), che dovrà essere sviluppato per ogni tabella. Il modello DAO comprende 5 possibili interazioni con il database:

- *getById*: seleziona un elemento dalla tabella, tramite il suo id
- *getAll*: seleziona tutti gli elementi dalla tabella
- *add*: aggiunge un elemento alla tabella
- *update*: aggiorna un elemento nella tabella, sostituisce tutti i parametri
- *delete*: Elimina un elemento dalla tabella, tramite l'id



Ognuno di questi metodi, comprende l'inizializzazione della *query*, con il *prepared statement*, l'inserimento dei dati nello *statement*, la sua esecuzione e l'analisi del suo risultato nel caso sia una *query* di selezione. Per questo motivo, ho deciso di progettare il software in maniera che sia il più astratto possibile, cioè che sia meno ripetitivo possibile.

Questo vuol dire, che ogni metodo è scritto una volta sola. Per poter sviluppare in questa maniera, alcune parti devono essere diverse per ogni tabella, quindi riscritte ogni volta per ogni tabella. Mentre per quanto riguarda le *query*, verrà automatizzato il tutto. Utilizzando un metodo laborioso, ma che permette di non dover scrivere tutte le *query*.

Le *query* verranno automaticamente create prendendo l'oggetto *model* che rappresenta la tabella, dal quale verranno presi tutti gli attributi e dal quale verrà generata la *query*, per preparare lo *statement*.

Il *prepared statement* non dovrà essere riempito ogni volta manualmente per ogni tabella, per questo è importante che vengano scritti gli attributi nello stesso ordine sia nel oggetto, che vengano inseriti nello stesso ordine nel *prepared statement*.

### 3.4.1.3 dati (model)

Per ogni tabella del database, vi è una relativa classe che la rappresenta, che deve contenere principalmente gli stessi attributi della tabella (nello stesso, per mantenere la logica), con i costruttori ed i setter.

Anche in questo caso il codice ripetitivo verrà riciclato, utilizzando la tecnica di programmazione ad ereditarietà. In questa parte del codice vi saranno tutti gli elementi che sono utilizzati per l'audit delle tabelle.

### 3.4.1.4 json (modeljson)

Questo sotto modulo del back-end si occupa di trasformare i modelli in elementi JSON, così che siano pronti per essere inviati come risposte delle Restful API.

### 3.4.1.5 data servlets (servlets.data)

Le data servlets, sono dei programmi scritti in java per elaborazione di dati e/o operazioni lato server, che rispondono alle richieste HTTP di un certo indirizzo. Nel nostro caso sono state suddivise in due blocchi (questo e *servlets.action*), per mantenere la divisione fra le varie componenti.

Per quanto riguarda le data servlets, servono per la costruzione delle Restful API, che verranno descritte e spiegate nel capitolo **XXXXXX**, anche per il loro sviluppo ho scelto di utilizzare un approccio ereditario, in maniera da astrarre il codice. Nel senso di scrivere il codice, generico, che possa essere usato per tutte le servlet, modificandolo marginalmente.

### 3.4.1.6 actions servlets (servlets.action)

Mentre la per quanto riguarda le Restful API che non riguardano i dati, ma le azioni come il login, oppure delle azioni specifiche cambiare la frequenza, oppure accendere o spegnere il generatore. Altre invece dovranno essere sviluppate per la comunicazione con l'Arduino.

Queste API, vengono separate siccome ogni una

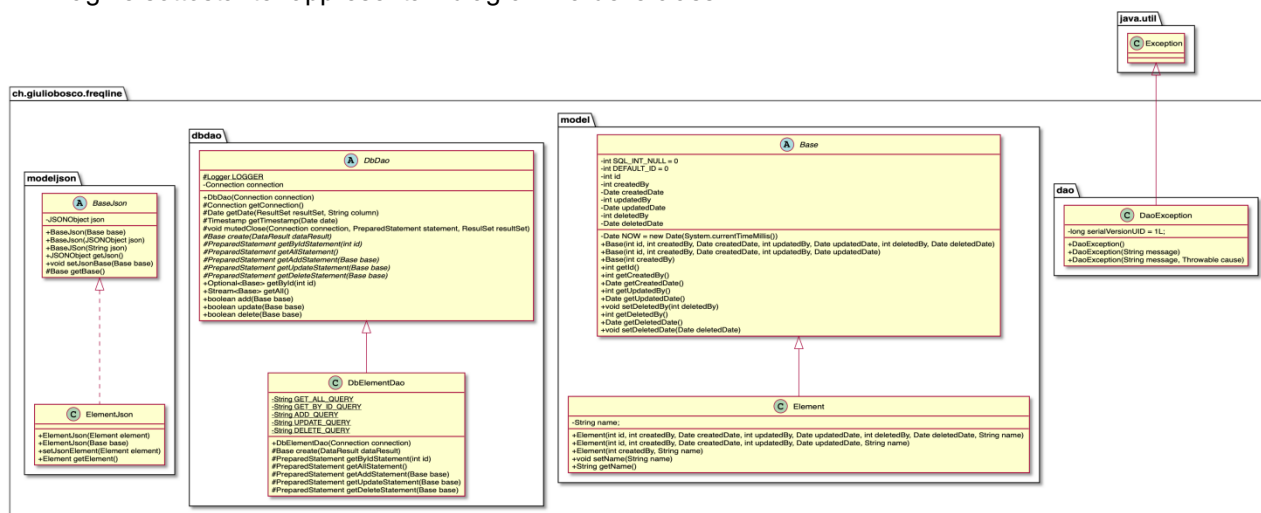
### 3.4.1.7 authentication (auth)

Per utilizzare questo applicativo bisogna essere autenticati, quindi vi è un modulo nel quale vengono controllati gli accessi, tramite username e password (con controllo di permessi), mentre viene utilizzata una key, per gli arduino.

### 3.4.1.8 arduino connection controll (acc)

Quindi anche per quanto riguarda i modelli di dati (*model*) necessito una struttura che sia in parte astratta. I modelli verranno trasformati in delle stringhe in formati JSON, anche per questo processo ho deciso di utilizzare una struttura astratta per riciclare il più possibile il codice.

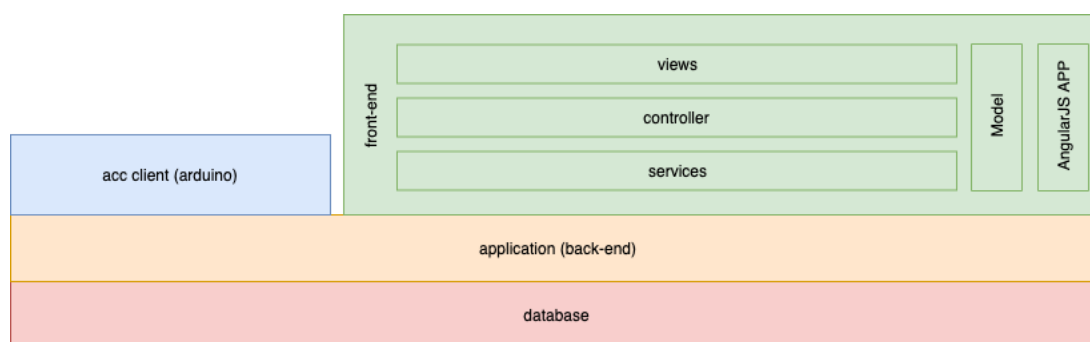
L'immagine sottostante rappresenta il diagramma delle classi.



Per lo sviluppo del software, e le connessioni al database, ho deciso di basarmi sul modello DAO (Data Access Object).

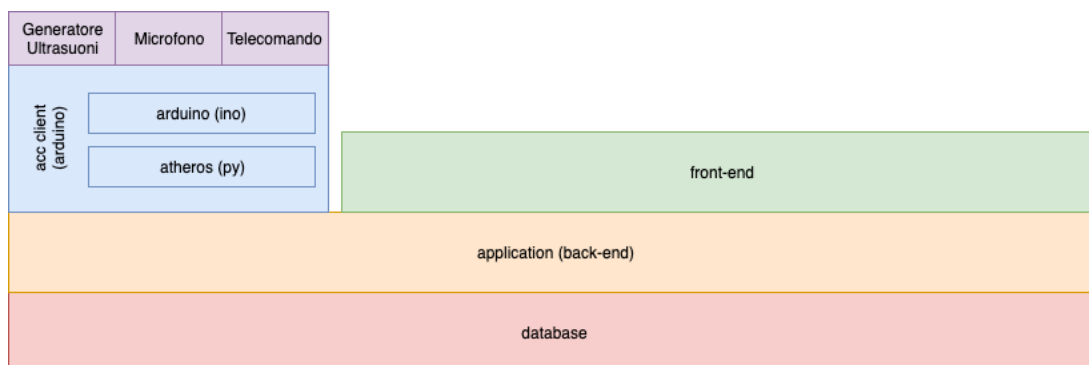
Prima cosa che viene fatta durante lo sviluppo, è la scrittura delle classi che rappresentano le varie tabelle del database. Dopo di che si implementano le classi che si collegano al database. Poi delle classi che trasformano i modelli delle tabelle in JSON, nascondendo le informazioni che non devono essere accessibili tramite le API.

### 3.4.2 Applicativo (front-end)



Il lato front-end, dell'applicativo, verrà sviluppato basandosi su

### 3.4.3 Acc Client (Arduino)



## 4 Implementazione

### 4.1 Database

#### 1.1.1 Tables

#### 1.1.2 Triggers

### 4.2 Backend

#### 1.1.3 Gradle

#### 1.1.4 JDBC

#### 1.1.5 DAO

#### 1.1.6 Models

#### 1.1.7 Servlets

##### 1.1.7.1 Actions

##### 1.1.7.2 Data Servlets

### 4.3 Frontend

Il front-end è sviluppato con il *web framework* javascript AngularJS, cioè un applicativo che serve semplificare lo sviluppo delle applicazioni web, alleggerire il lavoro degli sviluppatori e mantenere un codice pulito e ben leggibile. AngularJS è un framework sviluppato e mantenuto da Google, il quale può essere utilizzato con due architetture di pattern per lo sviluppo web, cioè MVC (Model View Controller) e MVVM (Model View ViewModel). MVC a 3 elementi fondamentali che sono:

- Model: i modelli, quindi i dati
- View: Le view, le interfacce grafiche che vengono mostrate agli utenti
- Controller: Le azioni che vengono fatte, come caricare le view e modificare i dati

Mentre la struttura MVVM, è una architettura sviluppata sugli eventi, quindi tutte le azioni devono essere scatenate da un evento. Anche in questa struttura vi sono 4 elementi:

- Model: i modelli dei dati
- View: Le view, le interfacce grafiche che vengono mostrate agli utenti
- ViewModel: Un'astrazione dei dati da inserire nelle view
- Binder: Si occupa di legare i comandi agli eventi che avvengono nella view

Per questo progetto verrà utilizzato il pattern MVC. Siccome AngularJS è un framework javascript, viene scritto in dei file JS, utilizzando le strutture tipiche del linguaggio.

La WebAPP in AngularJS, deve essere inizializzata, la quale può essere utilizzata in modalità pagina singola ed in modalità pagine multiple, il punto di forza di utilizzare un framework per utilizzare diverse pagine, è che non è necessario ricaricare più volte le pagine per navigare fra esse, ma la base dell'applicativo viene caricata all'inizio, poi mano a mano che si necessitano pagine diverse vengono caricate solamente quelle, per esempio il footer, la barra di ricerca, vengono caricate solamente una volta, così come gli script (JS) e i fogli di stile (CSS). Questo permette una maggiore velocità di caricamento ed una minore latenza durante la navigazione.

Il seguente codice serve per inizializzare l'applicazione ed il *router*, cioè il sistema che per ogni indirizzo scritto nella barra di ricerca, carica la view ed il controller giusto.

Nel codice sono configurate tutte le possibili view, per ognuna di esse, bisogna specificare a quale indirizzo corrisponde, quale file HTML caricare ed eventualmente si può inserire direttamente nella configurazione, altrimenti si può inserire nella view, nel caso la view dovesse caricare altri componenti, i quali hanno dei controller specifici.

In questo caso la webapp verrà chiamata **FreqlineAPP**, e dovrà caricare i seguenti 2 moduli ngRoute, che serve per analizzare la richiesta e reindirizzare sulla giusta view ed il modulo ngSanitize che serve per inserire tramite javascript del codice html, che venga interpretato e non scritto come testo.

Nella configurazione vengono configurate 7 views, la pagina index, quella degli utenti, la pagina di dettaglio degli utenti, che necessita un id, una pagina con lo stato del generatore di frequenze e due di errore, che corrispondono rispettivamente all'errore HTTP 401 e 404 che sarebbero UNAUTHORIZED (operazione non autorizzata) e NOT FOUND (elemento non trovato). In oltre viene configurato un redircet, che viene eseguito per ogni richiesta sconosciuta, alla pagina dell'errore 404.

```
var app = angular.module('FreqlineAPP', ['ngRoute','ngSanitize']);

app.config(function ($routeProvider) {
    $routeProvider.when('/', {
        templateUrl: 'views/index.html'
    });

    $routeProvider.when('/users', {
        templateUrl: 'views/users.html'
    });

    $routeProvider.when('/user/:id', {
        templateUrl: 'views/user.html',
    });

    $routeProvider.when('/login', {
        templateUrl: 'views/login.html'
    });

    $routeProvider.when('/status', {
        templateUrl: 'views/status.html',
        controller: 'StatusController'
    });

    $routeProvider.when('/401', {
        templateUrl: 'views/errors/401.html',
        controller: 'Error401Controller'
    });

    $routeProvider.when('/404', {
        templateUrl: 'views/errors/404.html',
        controller: 'Error404Controller'
    });

    $routeProvider.otherwise({
        redirectTo: '/404'
    });
}).run(function($rootScope, $route) {
    $rootScope.$route = $route;
});
```

Per richiedere i dati all'back-end vengono utilizzati i services, cioè` degli elementi che si occupano di fare le richieste AJAX, che con i browser moderni ha un problema, cioè` mantiene in cache i dati, quindi le pagine non vengono aggiornate. Per ovviare a questo problema, si invia come parametro HTTP, lo UNIX time, che sarebbe il numero di secondi passati dal primo gennaio 1970.

Per ogni service vi possono essere molteplici azioni, come per esempio nel service di login, vi è un azione per eseguire il login ed una per controllare se il login è stato fatto.

Ogni service è` pero` legato ad un indirizzo del back-end. Poi per ogniuna delle azioni si associa un metodo specifico, come GET, POST, PUT e DELETE. Tutte queste richieste sono fatte in maniera asincrona, questo vuol dire che il caricamento della pagina continua indifferentemente dalla risposta del server, durante le richieste dei dati. Quando poi il server risponde i dati vengono automaticamente inseriti nelle view, questo grazie al framework AngularJS. Questa tecnica viene utilizzata per migliorare l'esperienza utente, quindi iniziare a caricare le componenti grafiche della pagina e secondariamente inserirne i dati. All'utente non sembra che vi sia un tempo di caricamento così lungo, siccome vede che la pagina inizia a prendere forma.

```
app.factory('LoginService', ['$http', function($http) {
    var service = {};
```

```

var address = "localhost";
var port = 8080;
var baseApi = "/freqline-be"
var urlBase = "http://" + address + ":" + port + baseApi;
var baseApi = baseApi;
urlBase += "/action/login";
$http.defaults.withCredentials = true;

service.login = function (username, password) {
    let url = urlBase + "?t=" + new Date().getTime() + "&username=" + username + "&password=" +
password;
    let data = "username=" + username + "&password=" + password;
    return $http.post(url, data, {}).then(function (response){
        return response.data;
    },function (error){
        return error;
    });
};

service.isLoggedIn = function() {
    let url = urlBase + "?t=" + new Date().getTime();
    let data = "";

    return $http.get(url, data, config).then(function (response) {
        return {response: response.data.isLoggedIn};
    }, function(error) {
        return {response: false};
    });
};

return service;
}]);

```

I controller servono per inserire i dati nelle view e gestire le azioni dell'utente sulla view. Quindi essi prima richiedono i dati ai services, sempre in modo asincrono; quando li ricevono li inseriscono nelle view, quando poi viene eseguita un'azione sulla view. (che è controllata dal controller, come un click su un bottone), il controller esegue un'operazione, che può essere inviare i dati nuovi al server oppure li aggiorna. In questo controller si vede solamente il lato delle operazioni eseguite dal controller. Esso tramite il Model di login, che contiene username e password esegue la richiesta ai services, nel caso in cui la risposta del service è positiva cambia la view in quella dello stato del generatore di frequenze.

```

app.controller('LoginController', ['$scope', '$location', 'LoginService', function
($scope, $location, loginService) {
    $scope.login = function(login) {
        loginService.login(login.username, login.password).then(function (data) {
            if (data.message === "ok") {
                $location.path('/status');
            } else {
                $scope.message = data.message;
            }
        })
    }
}]);

```

Le view sono semplicemente i file html, che interpretati dal browser creano le pagine web che l'utente finale vede. Se il controller non è selezionato direttamente nella configurazione, ne carica uno (come nel caso seguente). Il codice della view sottostante rappresenta un form di login, il quale contiene un titolo della pagina, un sotto titolo 2 campi di input uno text per lo username ed uno per la password di tipo password, così che rimanga nascosta. Infine vi è un bottone per eseguire il login di tipo submit. Ed un campo per scrivere gli errori di login. Il tutto in un form.

```

<div ng-controller="LoginController" class="full-size">
    <form novalidate class="css-form">
        <h3>Freqline</h3>

```

```
<p>Gestore di generatore di frequenze</p>
<p>&nbsp;</p>
<div class="user-input-wrp">
  <br/>
  <input type="text" class="inputText" ng-model="login.username" required>
  <span class="floating-label">username</span>
</div>
<div class="user-input-wrp">
  <br/>
  <input type="password" class="inputText" ng-model="login.password" required>
  <span class="floating-label">password</span>
</div>
<label style="color:red; padding: 10px 10px 0 10px">
  {{message}}
</label>
<br>
<input type="submit" ng-click="login(login)" value="Login" />
<p style="padding: 15px 10px;">&copy; 2019 Giulio Bosco</p>
</form>
</div>
```

I model, che sarebbero i dati servono per comunicare fra controller, views, e services. Nel caso di AngularJS e piu` in generale di JavaScript, che e` un linguaggio non tipizzato (che vuol dire che non bisogna dichiarare i tipi di dati), i model possono contenere qualunque cosa. Quindi sono molto variabili.

In questo applicativo lo stile e` fatto con CSS (Cascading Style Sheets). Utilizzando poche regole per mantenere la grafica pulita e semplice, seguendo la filosofia *less is more* cioe` rendere le interfacce più semplici ed intuitive possibile.

In questo capitolo dovrà essere mostrato come è stato realizzato il lavoro. Questa parte può differenziarsi dalla progettazione in quanto il risultato ottenuto non per forza può essere come era stato progettato.

Sulla base di queste informazioni il lavoro svolto dovrà essere riproducibile.

In questa parte è richiesto l'inserimento di codice sorgente/print screen di maschere solamente per quei passaggi particolarmente significativi e/o critici.

Inoltre dovranno essere descritte eventuali varianti di soluzione o scelte di prodotti con motivazione delle scelte.

Non deve apparire nessuna forma di guida d'uso di librerie o di componenti utilizzati. Eventualmente questa va allegata.

Per eventuali dettagli si possono inserire riferimenti ai diari.

#### 4.4 Installazione Raspberry

Il database, il back-end ed il front-end per poter essere messi in un apparecchio di piccole dimensioni e rimanere indipendenti vengono messi su un Raspberry PI, che funge da server.

Questo Raspberry PI e` stato installato, utilizzando la distribuzione di default Raspbian, che e` una distribuzione Linux basata su Debian. La quale e` stata scaricata dal sito ufficiale del progetto Raspberry.

Primo passo scrivere l'immagine sulla micro sd, con il comando:

```
sudo dd bs=1m if=path_of_your_image.img of=/dev/rdiskN conv=sync
```

Questo processo potrebbe durare diverso tempo, alla fine inserire la micro sd nel Raspberry, collegarlo ad uno schermo, ad una tastiera, ad una rete via cavo ed all'alimentazione.

Quando sullo schermo compare la finestra di login (da CLI), inserire come nome utente `pi`, mentre come password `raspberry`, e` molto consigliato abilitare ssh, cosi da potersi collegare in remote, con il comando:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

Poi come primo passo installare `mariadb`, siccome `mysql` non e' disponibile per Raspbian, poi installare java e gradle per poter avviare la nostra applicazione:

```
sudo apt update
sudo apt install mariadb-server-10.5
sudo mysql_secure_installation
sudo apt install openjdk-8-jdk
```

Poi testare che java sia installato correttamente:

```
$ java -version
openjdk version "1.8.0_212"
OpenJDK Runtime Environment (build 1.8.0_212-8u212-b01-1+rpil-b01)
OpenJDK Client VM (build 25.212-b01, mixed mode)
$ javac -version
javac 1.8.0_212
```

Dopo di che scaricare gradle, ricordarsi di inserire il proxy (in questo caso sostituire con il proprio username: `user.name` e la propria password `pwd`), dopo di che estrarre il pacchetto ed aggiungere gradle alla variabile `path`, per poterlo eseguire come comando.

```
curl -proxy http://user.name:pwd@10.20.0.1:8080 -O /tmp
https://services.gradle.org/distributions/gradle-5.2.1-bin.zip
sudo unzip -d /opt/gradle /tmp/gradle-*.zip
echo "export GRADLE_HOME=/opt/gradle/gradle-5.2.1" >>
/etc/profile.d/gradle.sh
echo "export PATH=${GRADLE_HOME}/bin:${PATH}" >> /etc/profile.d/gradle.sh
sudo chmod +x /etc/profile.d/gradle.sh
source /etc/profile.d/gradle.sh
gradle -v
```

Poi creare il database per il progetto utilizzando lo script SQL `sql/db.sql`, con il comando:

```
mysql -u root -p < sql/db.sql
```

Poi creare un nuovo utente con tutti i permessi sul database:

```
mysql -u root -p
```



```
CREATE USER 'freqline'@'localhost' IDENTIFIED BY '1234qwer';
GRANT ALL PRIVILEGES ON freqline.* TO 'freqline'@'localhost';
```

Dopo di che provare ad avviare la web-app, con il comando:

```
./gradlew appRun
```

Per testare il programma collegarsi con un browser all indirizzo:

```
http://ip:8080/freqline/
```

Si dovrebbe vedere la pagina.

## 5 Test

### 5.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

<b>Test Case:</b>	TC-001	<b>Nome:</b>	Import a card with KIC, KID and KIK keys, but not shown with the GUI
<b>Riferimento:</b>	REQ-012		
<b>Descrizione:</b>	Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI		
<b>Prerequisiti:</b>	Store on local PC: Profile_1.2.001.xml (appendix <b>h n</b> ) and Cards_1.2.001.txt (appendix <b>h n</b> ). PIN (OTA_VIEW_PIN_PUK_KEY) and ADM (OTA_VIEW_ADM_KEY) user right not set.		
<b>Procedura:</b>	<ol style="list-style-type: none"> <li>Go to "Cards manager" menu, in main page click "Import Profiles" link, Select the "1.2.001.xml" file, Import the Profile</li> <li>Go to "Cards manager" menu, in main page click "Import Cards" link, Select the "1.2.001.txt" file, Delete the cards, Select the "1.2.001.txt" file, Import the cards</li> <li>Research the "41795924770" Card, Click the imsi card link Check the card details</li> <li>Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap) FROM otacardkey a where imsi='340041795924770' ORDER BY keyset;</li> </ol>		
<b>Risultati attesi:</b>	Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details)		

## 5.2 Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l'errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l'errore con eventuali ipotesi di correzione.

## 5.3 Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

## 6 Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap 1.7) (ad esempio Gantt consuntivo).

## 7 Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un'aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

### 7.1 Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

### 7.2 Considerazioni personali

Cosa ho imparato in questo progetto? ecc

## 8 Bibliografia

### 8.1 Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell'autore o degli autori, o nome dell'organizzazione,
2. Titolo dell'articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell'articolo,

### 8.2 Bibliografia per libri

1. Cognome e nome (o iniziali) dell'autore o degli autori, o nome dell'organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell'editore,
5. Anno di pubblicazione,
6. ISBN.

### 8.3 Sitografia

<https://www.raspberrypi.org/downloads/raspbian/> Download Raspbian 29.11.2019

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

- <http://standards.ieee.org/guides/style/section7.html>, *IEEE Standards Style Manual*, 07-06-2008.

## **9 Allegati**

---

Elenco degli allegati, esempio:

- Diari di lavoro
- Codici sorgente/documentazione macchine virtuali
- Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
- Documentazione di prodotti di terzi
- Eventuali guide utente / Manuali di utilizzo
- Mandato e/o Qdc
- Prodotto
- ...