Generatore di frequenze con vari accessori gestito via web

[1 Introduzione 3](#_Toc26695331)

[1.1 Informazioni sul progetto 3](#_Toc26695332)

[1.2 Abstract 3](#_Toc26695333)

[1.3 Scopo 3](#_Toc26695334)

[2 Analisi 4](#_Toc26695335)

[2.1 Analisi del dominio 4](#_Toc26695336)

[2.2 Analisi e specifica dei requisiti 4](#_Toc26695337)

[2.3 Use case 6](#_Toc26695338)

[2.4 Pianificazione 6](#_Toc26695339)

[2.5 Analisi dei mezzi 8](#_Toc26695340)

[2.5.1 Software 8](#_Toc26695341)

[2.5.2 Hardware 8](#_Toc26695342)

[3 Progettazione 9](#_Toc26695343)

[3.1 Design dell’architettura del sistema 9](#_Toc26695344)

[3.1.1 Restful API 9](#_Toc26695345)

[3.2 Design dei dati e database 11](#_Toc26695346)

[3.3 Design delle interfacce 13](#_Toc26695347)

[3.4 Design procedurale 15](#_Toc26695348)

[3.4.1 Applicativo (back-end) 15](#_Toc26695349)

[3.4.1.1 connection to database (jdbc) 15](#_Toc26695350)

[3.4.1.2 data access object (dao) 16](#_Toc26695351)

[3.4.1.3 dati (model) 17](#_Toc26695352)

[3.4.1.4 json (modeljson) 18](#_Toc26695353)

[3.4.1.5 data servlets (servlets.data) 19](#_Toc26695354)

[3.4.1.6 actions servlets (servlets.action) 19](#_Toc26695355)

[3.4.1.7 authentication (auth) 20](#_Toc26695356)

[3.4.1.8 arduino connection controll (acc) 20](#_Toc26695357)

[3.4.2 Applicativo (front-end) 20](#_Toc26695358)

[3.4.3 Acc Client (Arduino) 21](#_Toc26695359)

[4 Implementazione 22](#_Toc26695360)

[4.1 Database 22](#_Toc26695361)

[1.1.1 Tables 22](#_Toc26695362)

[1.1.2 Triggers 22](#_Toc26695363)

[4.2 Backend 22](#_Toc26695364)

[1.1.3 Gradle 22](#_Toc26695365)

[1.1.4 JDBC 22](#_Toc26695366)

[1.1.5 DAO 22](#_Toc26695367)

[1.1.6 Models 22](#_Toc26695368)

[1.1.7 Servlets 22](#_Toc26695369)

[4.3 Frontend 22](#_Toc26695370)

[4.4 Installazione Raspberry 25](#_Toc26695371)

[5 Test 27](#_Toc26695372)

[5.1 Protocollo di test 27](#_Toc26695373)

[5.2 Risultati test 28](#_Toc26695374)

[5.3 Mancanze/limitazioni conosciute 28](#_Toc26695375)

[6 Consuntivo 28](#_Toc26695376)

[7 Conclusioni 28](#_Toc26695377)

[7.1 Sviluppi futuri 28](#_Toc26695378)

[7.2 Considerazioni personali 28](#_Toc26695379)

[8 Bibliografia 28](#_Toc26695380)

[8.1 Bibliografia per articoli di riviste: 28](#_Toc26695381)

[8.2 Bibliografia per libri 28](#_Toc26695382)

[8.3 Sitografia 28](#_Toc26695383)

[9 Allegati 29](#_Toc26695384)

# Introduzione

## Informazioni sul progetto

Docente Responsabile: Fabrizio Valsangiacomo

Apprendista: Giulio Bosco

Data inizio progetto: 05.09.2019

Data consegna progetto: 20.12.2019

## 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.*

## 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.

# Analisi

## 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?
* …

## Analisi e specifica dei requisiti

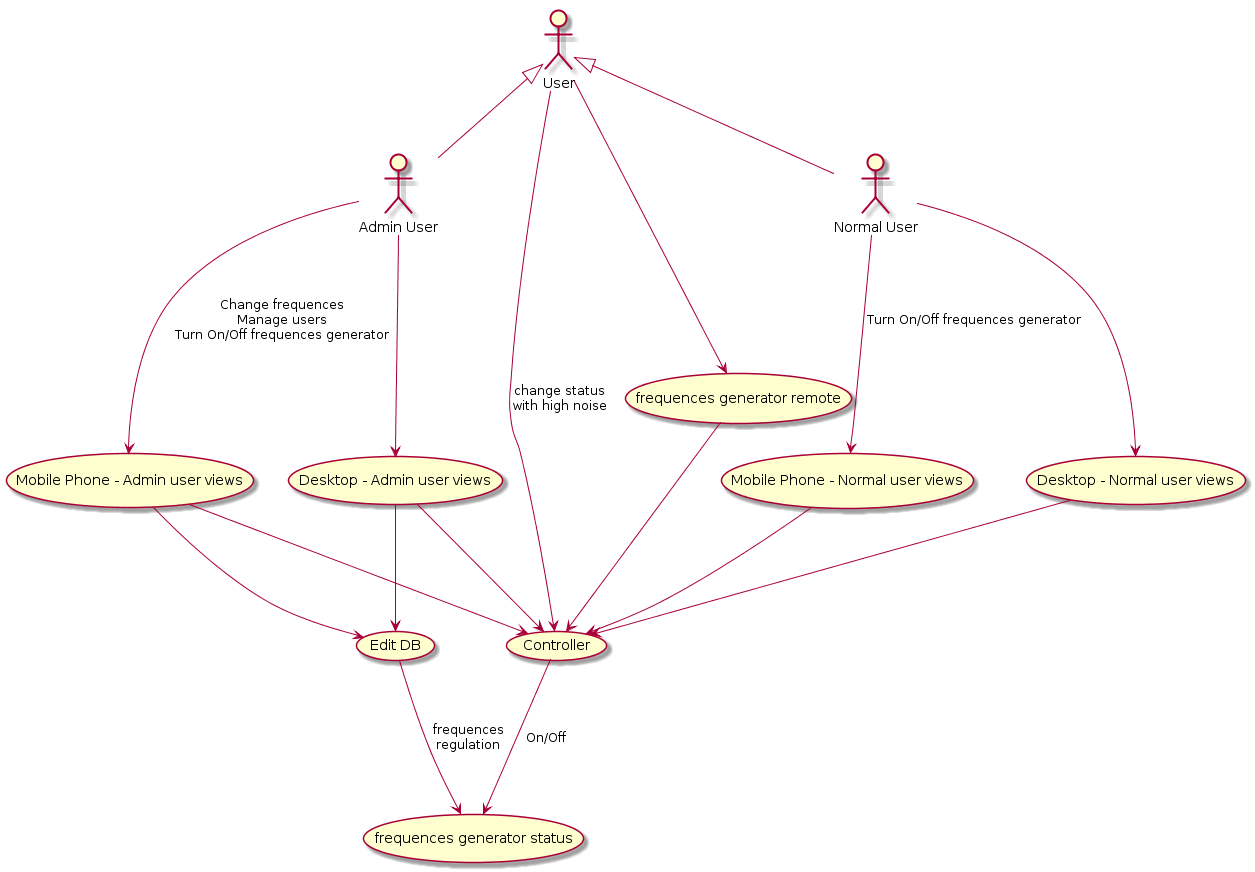
|  |  |
| --- | --- |
| **ID: REQ-01** | |
| **Nome** | Gestione generatore tramite WEB |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
|  | **Sotto requisiti** |
| **Sub REQ 1** | 0-25'000 Hz |
| **Sub REQ 2** | Gestione Wireless del generatore |
| **Sub REQ 3** | Regolazione della frequenza |
| **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** | - |
|  | **Sotto requisiti** |
| **Sub REQ 1** | Tasto Start |
| **Sub REQ 2** | Tasto Stop |

|  |  |
| --- | --- |
| **ID: REQ-03** | |
| **Nome** | Gestione generatore tramite Rumore |
| **Priorità** | 2 |
| **Versione** | 2.0 |
| **Note** | - |
|  | **Sotto requisiti** |
| **Sub REQ 1** | Gestione tramite decibel |
| **Sub REQ 2** | Timer per spegnimento |

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

## 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

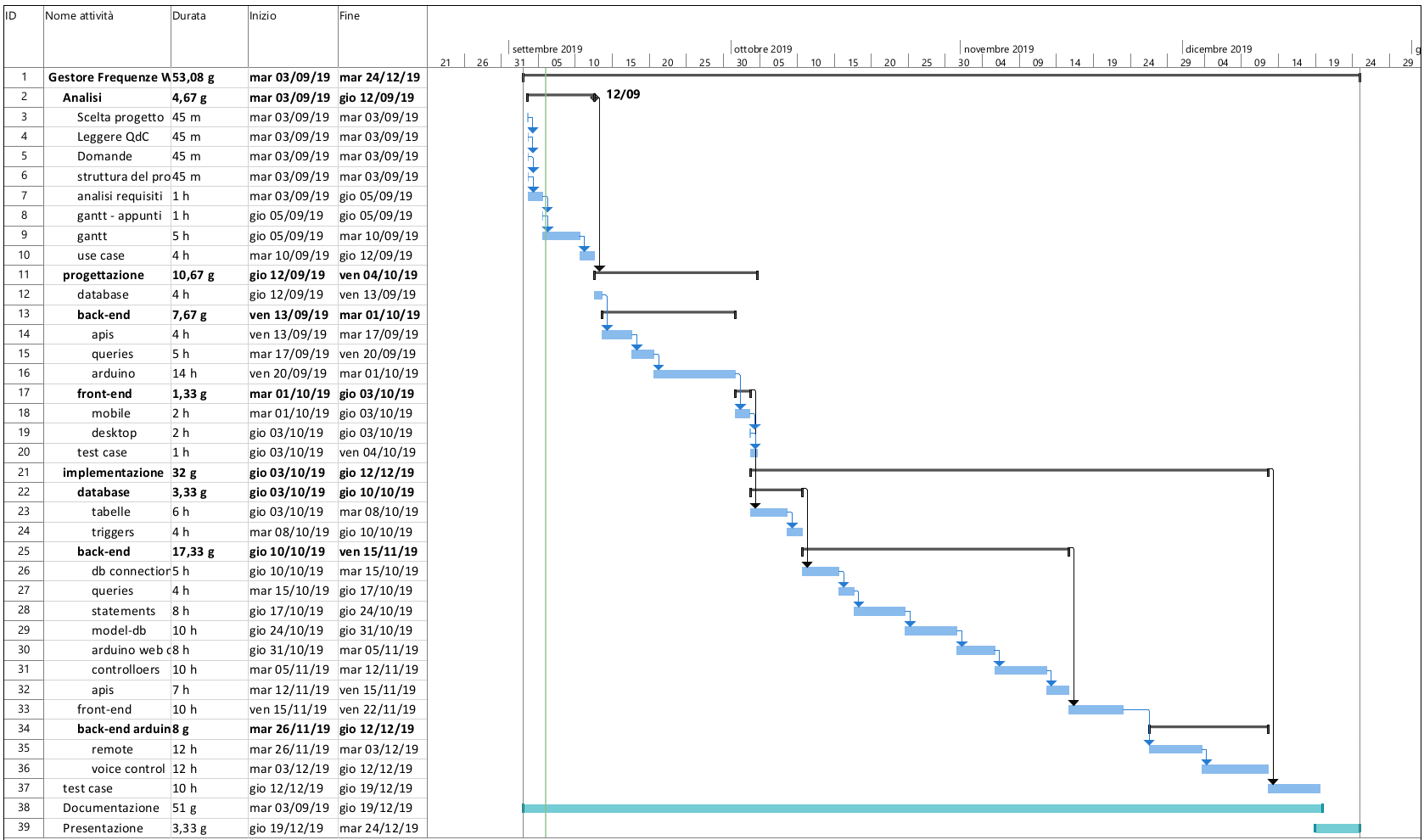


Figura 1: Diagramma di GANTT

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, cosi 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 e` 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.

## Analisi dei mezzi

### Software

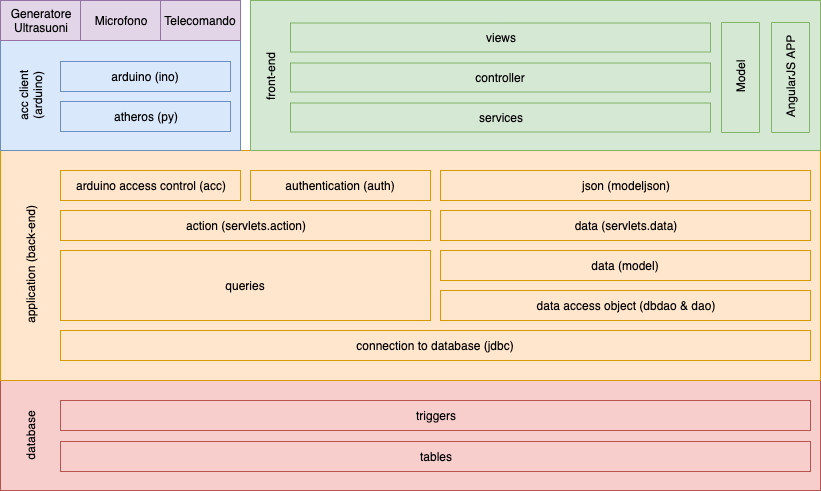
* Arduino IDE (v1.8.9)
* Atom IDE (v1.38.2)
* plantuml (Version 1.2019.9)
* Libre Office (Version: 6.2.5.2)
* Java (v1.8)
  + Libreria org.gretty (v2.2.0)
  + Libreria Junit (v4.12)
  + Libreria JDBC (MySQL Connector Java – v8.0.11)
  + Libreria JSON (org.json:json:20171018)

### 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

# Progettazione

## Design dell’architettura del sistema

Il progetto e 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)*.

## 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.

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

{

"users": [

{

"username": "giulio.bosco",

"firstname": "Giulio",

"lastname": "Bosco"

},

{

"username": "fabrizio.valsangiacomo",

"firstname": "Fabrizio",

"lastname": "Valsangiacomo"

}

]

}

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=?).

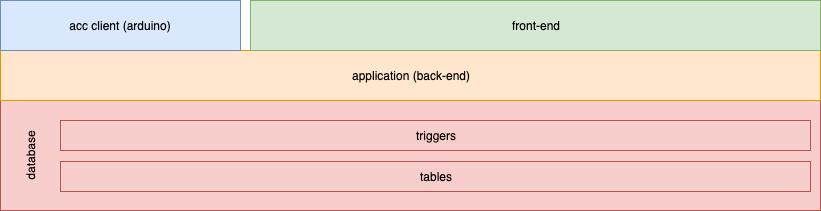
Ognuna di queste richieste può avere una serie di risposte, le quali sono rappresentate da un numero e da una stringa di descrizione, il numero è sempre composto di 3 cifre, la prima indica il tipo di risposta, che puo essere mentre le seconde 2 cifre identificano la risposta:

* 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 |

## 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 perche e dal lato della piattaforma WEB, il progetto e 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 | * Semplice implementazione e integrazione con la banca dati * Facile ricostruzione del database | * Difficile creare dei report sui log |
| 2 | * Facilizza generazione di report sulle azioni eseguite sul database * mantiene separati i dati | * Implementazione del database più complessa |
| 3 | * Facile implementazione del database | * Complica la struttura del software * Nel caso in cui vi fossero in futuro più elementi che interagiscono con lo stesso database, i log non sarebbero più autentici |
| 4 | * Semplificherebbe il codice | * Dovrei prendere del tempo per imparare ad utilizzare un altro software * Richiederebbe più risorse |

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 e una tabella `generator` che da 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 permette 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.

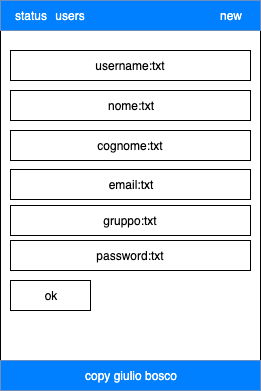
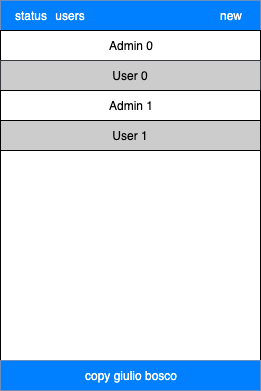
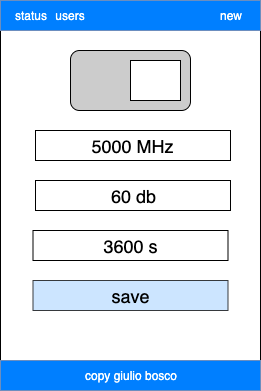
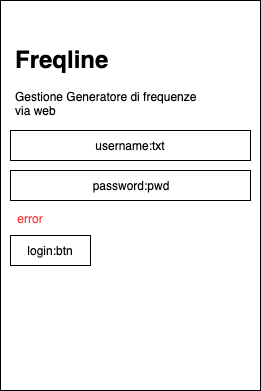


In questo schema si possono notare molte più tabelle e più attributi, per ogni tabella che vi era nello schema antecedente, 6 attributi, che servono per salvare la data e l'autore delle azioni principali che si possono fare sul database (create, update, delete). Inoltre, vi e un altra tabella, nella 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.

## 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 hai 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.



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 vuoi, 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 groppo.

## Design procedurale

## 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.

Il lato back-end dell’applicativo è suddiviso in due parti, i dati (e la loro gestione) e le azioni, cioè le operazioni specifiche che possono venir eseguite. Questa suddivisione è stata fatta per rendere le azioni più performanti e per mantenere l’integrità dei dati. Entrambe le due parti, sono suddivise in moduli.

Lato dei dati, vi sono 4 livelli, uno per l’astrazione dei dati:

* data access object (dbdao & dao): interfaccia dei dati con il database
* modelli (model): rappresentazione dei dati a livello database all’interno dell’applicativo
* data (servlets.data): esposizione http dei dati
* json (modeljson): traduzione dei dati da modelli a rappresentazione JSON, per le richieste delle servlet.

Il lato delle azioni vi sono 3 livelli di cui uno diviso in 2 moduli.

* queries sul database: sono le query, per aggiornare i dati sul database
* action (servlets.action): esposizione http delle azioni
* autenticazione (auth): autenticazione nell’applicativo
* controllo arduino (acc): controllo remoto dell’arduino

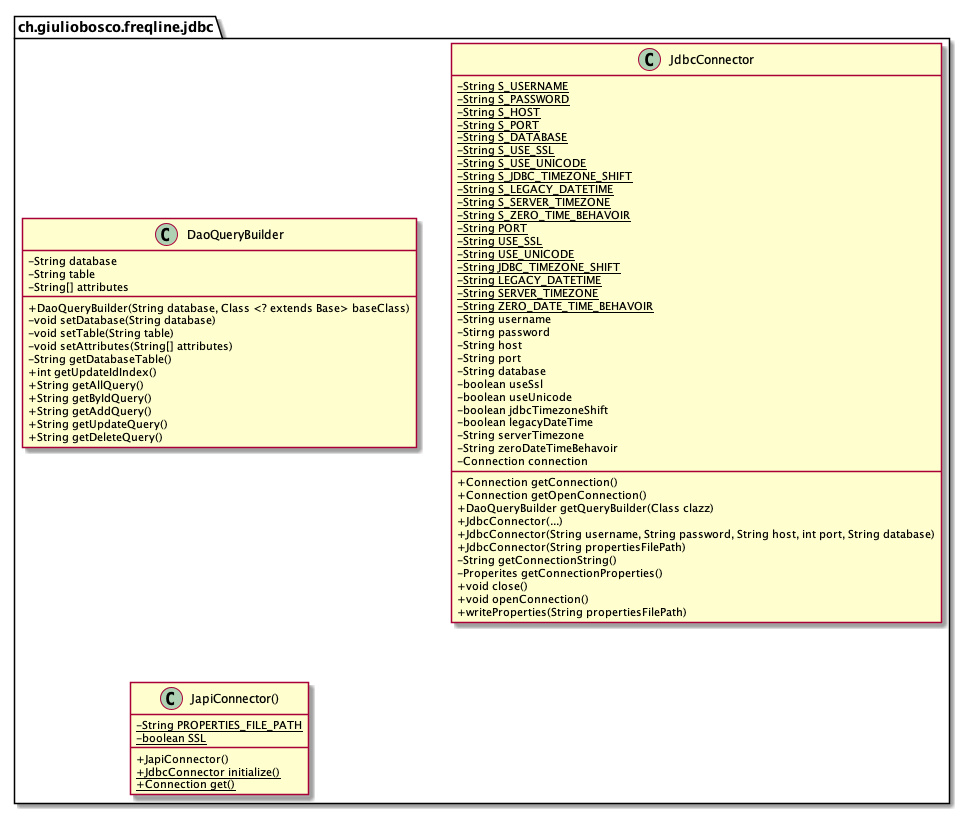
I vari moduli, non verranno sviluppati nello stesso ordine in cui sono posti ora, siccome dipendono l’uno dall’altro, verranno sviluppati in un ordine che mi permetta di sviluppare al meglio il software.

## 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.



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, cosi 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, cosi 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 instanziazione della connessione tramite file di properties che non viene pubblicato con il codice.

## 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:

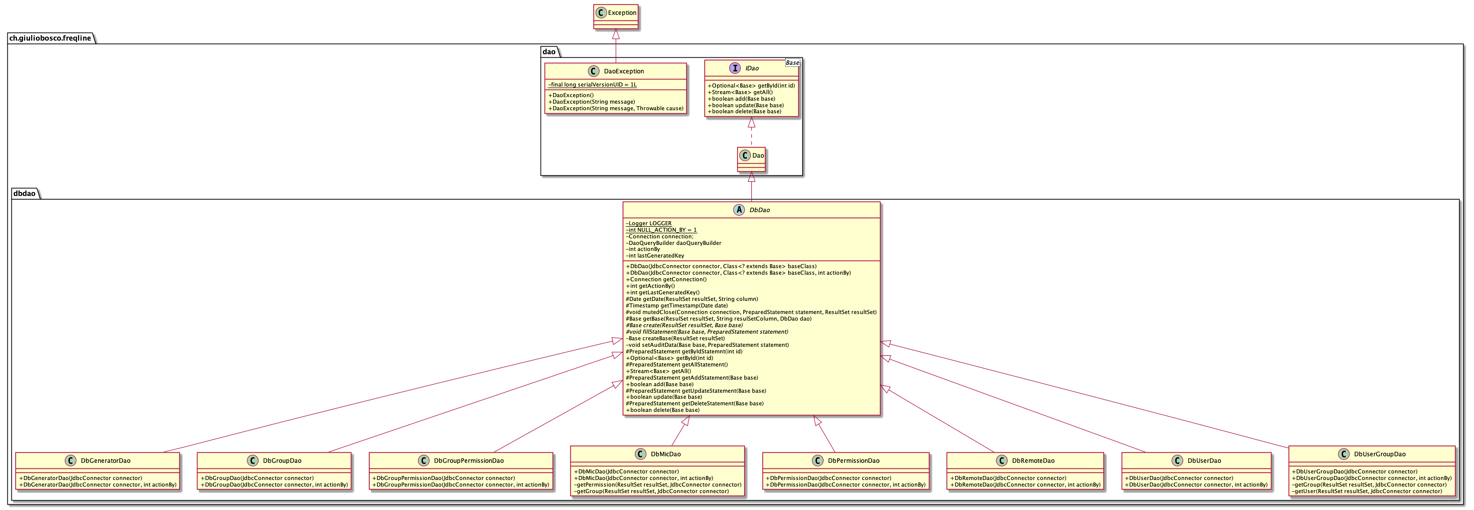
* *getById*: seleziona un elemento dalla tabella, tramite il suo id
* *getAll*: seleziona tutti gli elementi dalla tabella
* *add*: aggiunge un elemento alla tabella
* *update*: aggiorna un elemento nella tabella, sostituisce tutti i parametri
* *delete*: Elimina un elemento dalla tabella, tramite l’id

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 le query, vengono composte automaticamente tramite il modello dei dati.

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 nell’oggetto, che vengano inseriti nello stesso ordine nel prepared statement.



Il modello DAO ha un’interfaccia che le operazioni possibili, che viene implementata per tutti i modelli che rappresentano le tabelle del database. Le azioni sono:

* insert: che inserisce nel database un dato
* update: che aggiorna un dato nel database, il quale utilizza l’ID per aggiornare tutto il resto dei parametri
* delete: elimina un elemento tramite il suo id
* getAll: seleziona tutti gli elementi dal database, del modello in questione
* getById: seleziona un elemento tramite il suo id

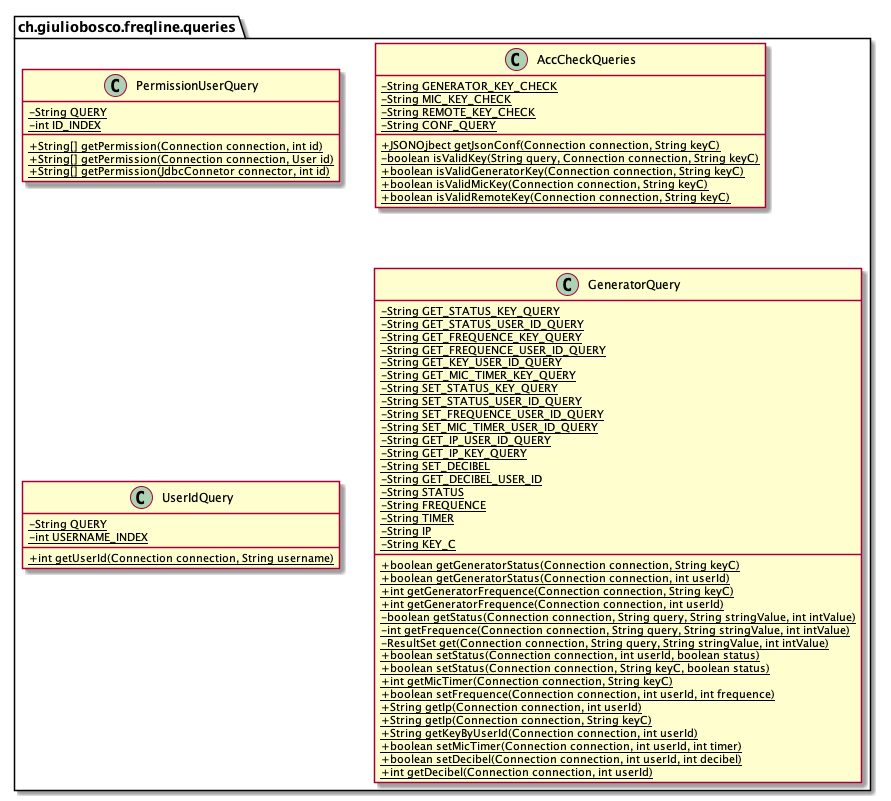
Questa struttura viene utilizzato per facilitare il cambio di database, quindi per ognuno di essi viene implementato un modello di dati. Nel nostro caso è stato implementato solo la classe DbDao, per MySQL.

La classe DbDao è molto complessa siccome è stata sviluppata in maniera astratta, cosi da poter riutilizzare praticamente tutto il codice, per poi adattarla per ogni tabella del database (e modello), la si estende, implementandone i metodi astratti, si può facilmente creare l’accesso hai dati, per ogni tabella. Come si può vedere nel diagramma UML delle classi. Per la classe DbDao contiene i metodi principali, più diversi metodi di aiuto, che servono per la costruzione degli statement SQL da eseguire sul server, per la trasformazione da tipi di dati di MySQL a tipi di dati Java.

La classe DbDao viene estesa per ogni tabella la quale contiene solamente i costruttori ed i metodi per la trasformazione da ResultSet (Ritorno dei dati da MySQL) a oggetti Java e la trasformazione da oggetti Java a statement (inserimento dati nei prepared statement).

## Queries

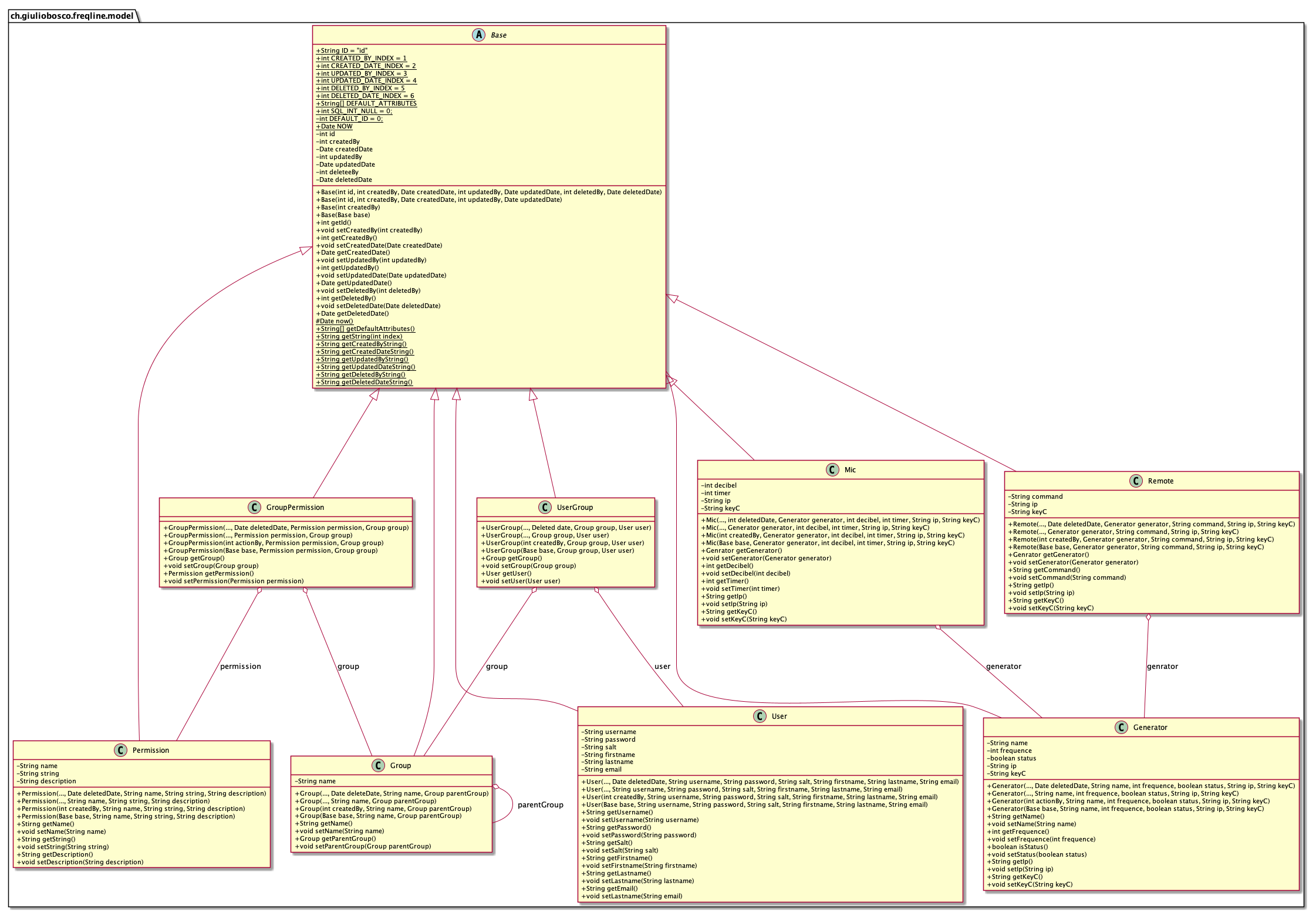
Per ogni tipo di query, vi è una classe. Tutte le query per i prepared statement sono i delle costanti di tipo stringa, che vengono inserite negli statement e poi vengono assegnati i valori, per poi essere eseguiti. Per facilitare l’utilizzo dei metodi sono tutti statici, che possono ritornare degli oggetti. Questo per facilitarne e velocizzarne l’utilizzo.



## 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 ordine, per mantenere la logica), con i costruttori ed i setter ed i getter, siccome in Java è best-practice mantenere tutti gli elementi private, creandone dei metodi per settare e dei metodi per richiedere il valore, questo si fa per poter regolare i permessi. Per esempio, mettere il setter di un attributo ad un livello di accesso (come private) ed il getter ad un altro livello (come public).

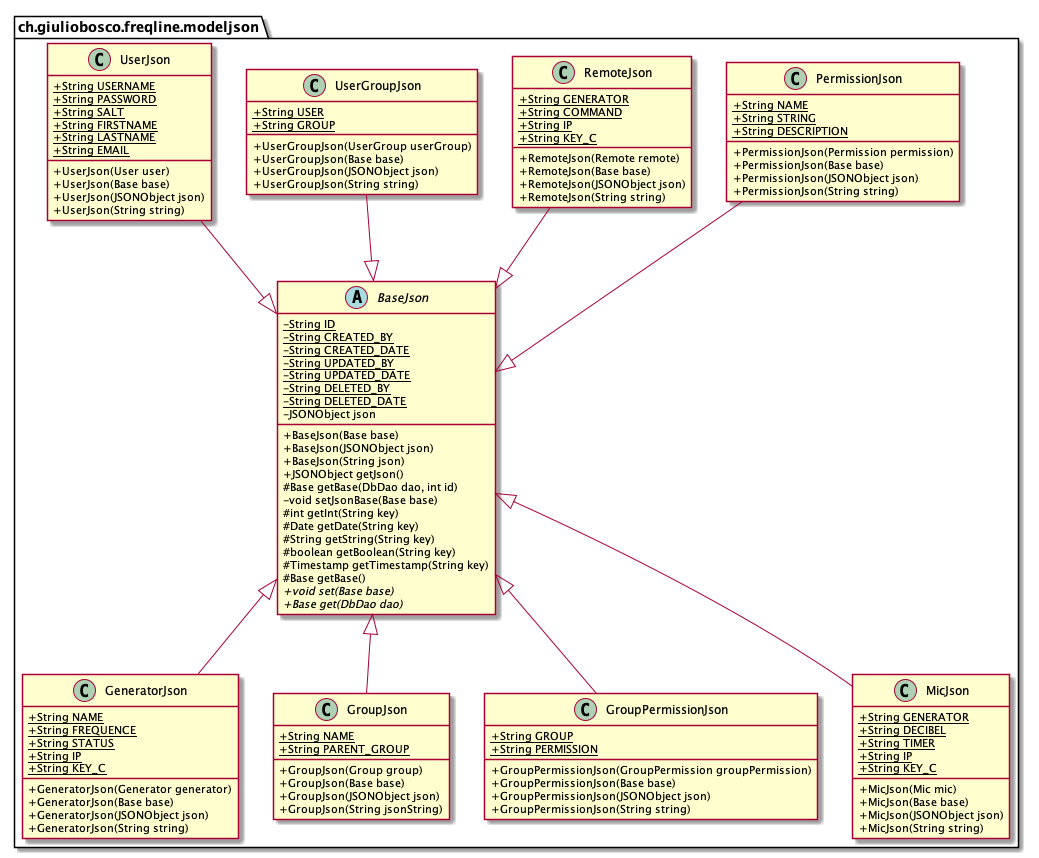
Anche in questo caso il codice ripetitivo verrà riciclato, utilizzando la tecnica di programmazione ad ereditarietà`. In questa parte del codice vi saranno tutti gli elementi che sono utilizzati per l’audit delle tabelle.



Nel diagramma delle classi si nota un modello Base che contiene l’id e gli attributi di audit. La classe viene estesa da una classe per ogni tabella del database, che contiene tutti gli attributi delle varie tabelle (eccetto id e attributi di audit, perché sono già contenuti nel modello Base). Sono anche rappresentati i legami fra i vari elementi, tramite delle composizioni.

## json (modeljson)

Questo sotto modulo del back-end si occupa di trasformare i modelli in elementi JSON, cosi che siano pronti per essere inviati come risposte delle Restful API. Anche in questo modulo è stato utilizzato un approccio astratto ed ereditario.

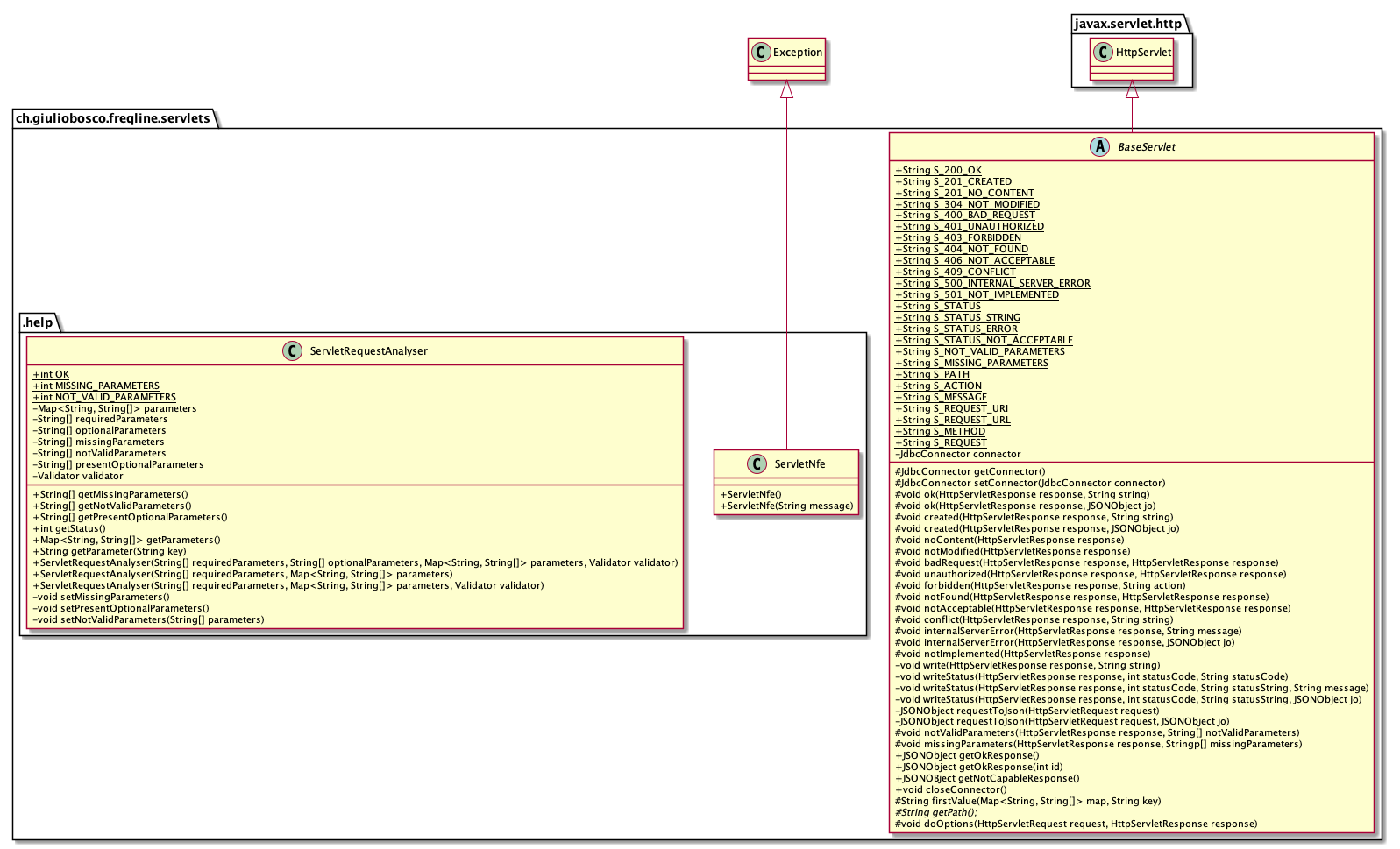


Nel diagramma soprastante, si nota che la rappresentazione è molto simile a quella dei modelli, l’id e gli elementi di audit sono gestiti dal BaseJson dal quale discendono le classi corrispondenti ad ogni modello dei dati. Nei quali vengono gestiti gli attributi specifici.

## servlets (servlets & servlets. Help)

Le servlets, sono dei programmi scritti in java per elaborazione di dati e/o operazioni lato server, questi programmi sono pensati per interpretare e rispondere le richieste http. Siccome in questo caso viene utilizzato il protocollo http, ma per rispondere alle richieste con delle stringhe JSON, nelle quali deve essere sempre inserito lo status code http della richiesta ed il messaggio dello status code. Quindi è stata creata una BaseServlet, la quale contiene tutti i metodi per inviare le varie riposte, un metodo per ogni codice di errore. Per alcuni errori bisogna passare come perimetro un messaggio, mentre in altre come il metodo notFound necessita la richiesta http, cosi da automatizzare il messaggio di riposta, per esempio ritorna il messaggio, dell’elemento non trovato, anche la path dell’elemento.

Per facilitare l’analisi delle richieste http, vi è una classe che analizza i parametri inviati nella richiesta. Utilizza un array di stringhe (che sarebbero le key gli elementi) per gli elementi richiesti ed uno per gli elementi opzionali, la classe controlla che i parametri richiesti esistano e ne fa la validazione, mentre per quanto riguarda quelli opzionali se esistono li inserisce in un array contenetene tutti gli elementi presenti e li valida.

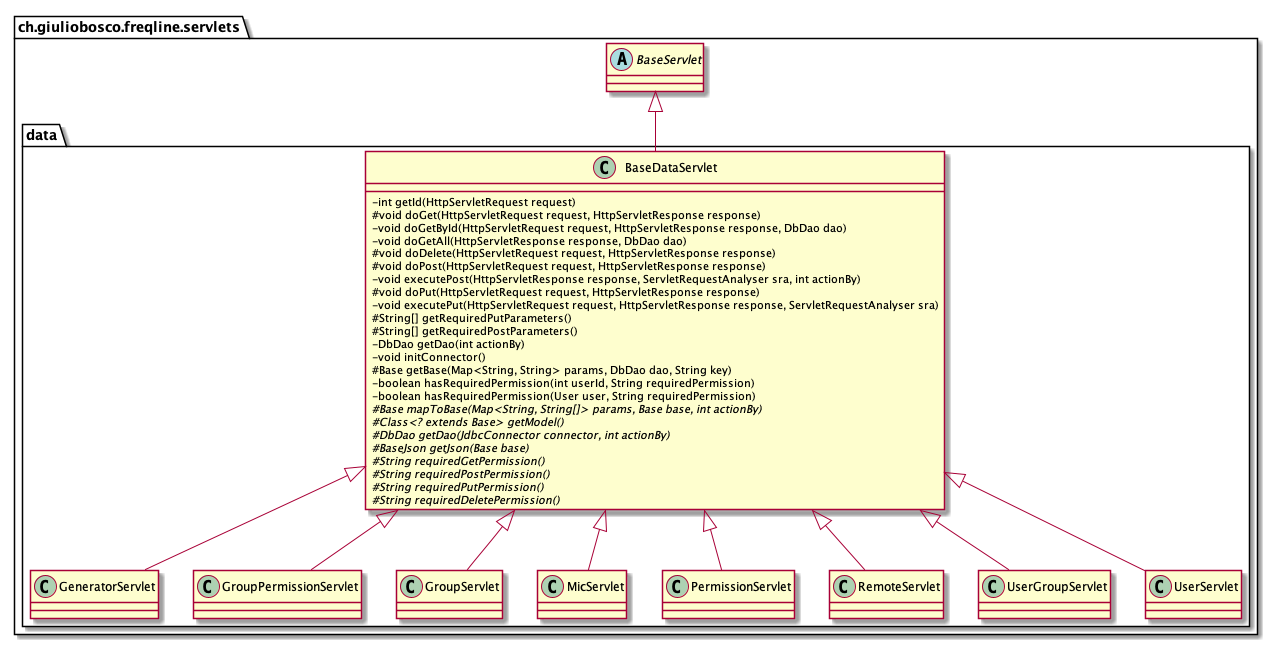


## data servlets (servlets.data)

Le data servlets, sono le servlets specifiche per i dati, quindi tutte molto simili che devono eseguire le richieste dei client per quanto riguarda la gestione dei dati. Esse hanno a disposizione le operazioni CRUD (Create Read Update Delete), per le quali vi sono dei permessi da applicare per ogni operazione.

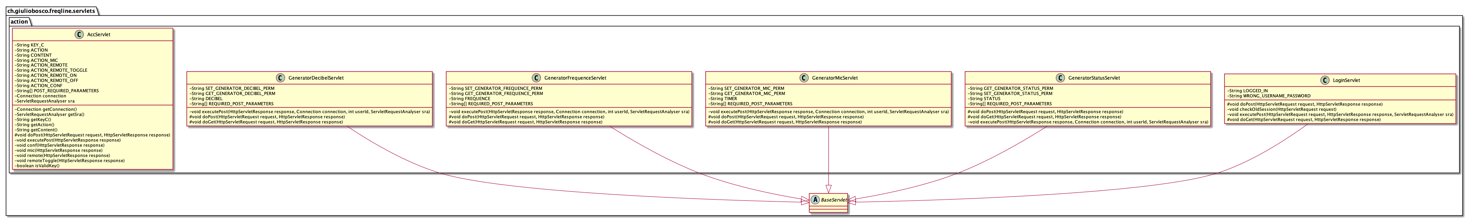
Per alcune operazioni il codice è stato suddiviso in più metodi, cosi da facilitarne la lettura. Le risposte delle richieste vengono inviate in formato JSON, come viene detto nelle best-practice delle Restful API (spiegate nel capitolo …………) per la comunicazione fra l’applicativo back-end e front-end.

Anche per questo modulo vi è una classe con tutta la logica e la struttura del codice; la quale viene estesa da una classe per ogni modello, che implementa i metodi astratti che servono per la regolazione dei permessi, per la creazione della classe di astrazione dei dati (model) ed convertitore da model a json.



## actions servlets (servlets.action)

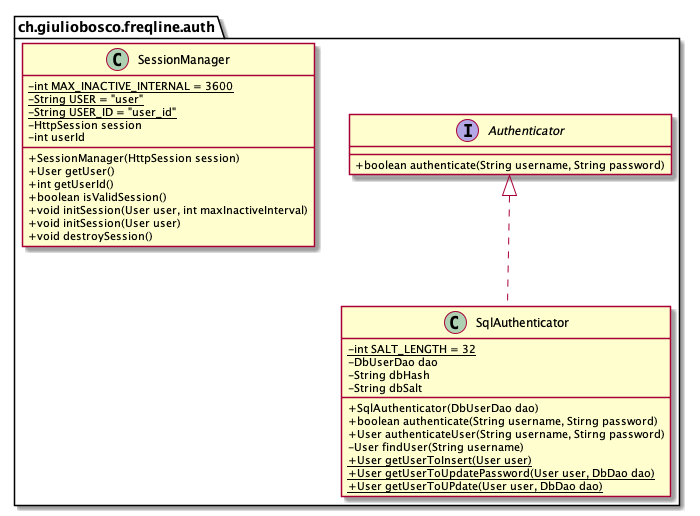
Mentre la per quanto riguarda le Restful API che non riguardano i dati, ma le azioni come il login, oppure delle azioni specifiche cambiare la frequenza, accendere o spegnere il generatore, quelle di controllo dell’Arduino. Per ognuna di questa API vi è una servlet indipendente.



Le quali sono molto semplici, servono semplicemente per eseguire delle operazioni di base, come il login, oppure controllare se l’utente e` collegato. Per eseguire queste operazioni vengono utilizzate le queries nel caso in qui le operazioni sono di base, mentre per quanto riguarda le operazioni più complicate vi possono essere dei piccoli moduli, cosi da separare ulteriormente il codice e renderlo più facile da leggere e mantenere.

## authentication (auth)

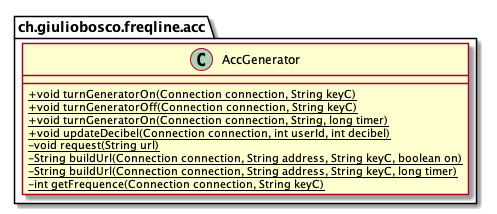
Per utilizzare questo applicativo bisogna essere autenticati, quindi vi è un modulo nel quale vengono controllati gli accessi, tramite username e password (con controllo di permessi), mentre viene utilizzata una key, per gli arduino.



Per quanto riguarda la autenticazione tramite utente, vi sono due possibilità, una che controlla solamente username e password dell’utente siano corretti ed una che controlla se sono corretti e carica l’utente. Inoltre vi è un gestore delle sessioni per controllare velocemente che l’utente sia già autenticato ed abbia permessi corretti.

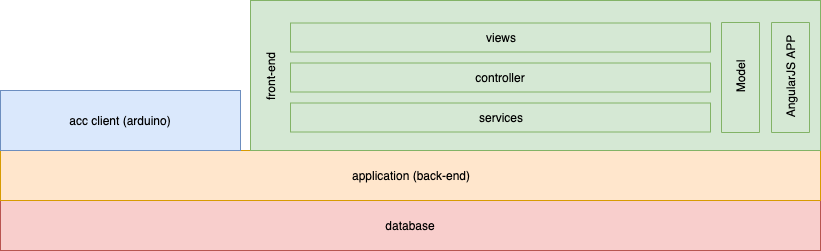
Siccome la password dell’utente nel database viene salvata sotto forma di hash con anche il salt, cioè per migliorare la sicurezza viene presa la stringa della password, ci viene aggiunto dietro il salt (che sarebbe una stringa di caratteri casuali) e poi ne viene fatto un hash. Viene utilizzato il salt per rendere più complesso il reverse-engineering della password. Per gestire l’inserimento e l’aggiornamento di un utente sono state creati dei metodi apposta, così da mantenere l’integrità dei dati.

## arduino connection controll (acc)



Per quanto riguarda la comunicazione con l’Arduino, sono state create delle classi, le azioni da inviare al Arduino, sono state raccolte in una classe, cosi che venga controllato tutto in un punto singolo, l’aggiornamento dei dati nel database e l’esecuzione delle richieste. Per ogni richiesta che viene fatta, si necessita la connessione al database ed i parametri per configurare l’Arduino.

## Applicativo (front-end)

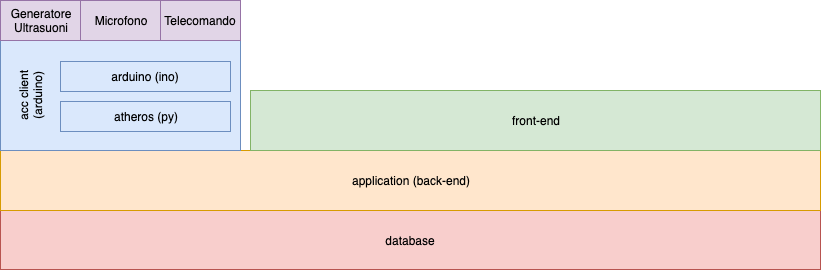


Il lato front-end, dell’applicativo, verrà sviluppato basandosi su è anch’esso basato sui Modelli, per la rappresentazione dei dati. Utilizzando il modello MVC (verrà spiegato nell’implementazione). Buona parte dell’applicativo è eseguito sul client, quindi anche su esso vi deve essere una struttura ben precisa. Sfruttando le Restful API ed AJAX (Asynchronous JavaScript and XML), le varie grafiche vengono create con i vari dati.

## Acc Client (Arduino)

Per quanto riguarda il controller, per il generatore di onde ad ultrasuoni, il telecomando ed il microfono, è un Arduino YÙN, che sarebbe un microcontrollore, composto da due componenti, una scheda ATmega32u4 ed un chip Athreos AR9331. Sulla scheda ATmega32u4 vi è un microsistema di controllo simile a quello di un Arduino UNO, quindi il collegamento hai pin ed alla porta seriale; in più vi è un collegamento con il chip, chiamato Bridge. Il chip ha un sistema operativo UNIX/Linux di base, minimale, OpenWRT, che è fatto apposta per circuiti embedded, quindi molto leggera e con il software di base. Il chip è collegato alla porta USB, posta sulla scheda, alla porta ethernet ed al chip Wi-Fi.

Il lato Arduino del chip Athreos viene programmato in python e può accedere al bridge verso il lato ATmega32u4, il quale viene programmato in una versione semplificata di C, adeguata all’Arduino (sviluppata apposta). Alla scheda ATmega32u4, vi sono collegati il generatore, il microfono ed il telecomando.



# Implementazione

## Database

Il database è stato sviluppato con SQL, per un server MySQL.

Inizialmente volevo utilizzare MySQL, poi spostando il progetto sul Raspberry PI, mi sono reso conto che non è stato implementato MySQL vi è a disposizione mariadb, che è molto simile ma per esempio il driver è diverso. Alcune piccolezze per quanto riguarda il back-up dei database.

Quindi dopo aver sviluppato tutto ho dovuto cambiare il driver da MySQL a mariadb, nella classe JdbcConnector, in particolare, la linea di caricamento del driver alla linea 465:

//Class.forName("com.mysql.cj.jdbc.Driver");

Class.forName(“org.mariadb.jdbc.Driver”);

Poi modificare anche la linea 409 per la stringa di connessione:

//return "jdbc:mysql://" + this.host + ":" + this.port + "/" + this.database;

Return “jdbc:mariadb://” + this.host + “:” + this.port + “/” + this.database;

### Tables

Ho trasformato il diagramma entità relazione in codice SQL, partendo dalle tabelle marginali per arrivare, per arrivare a quelle centrali del corpo. Cosi da poter inserire subito le foreign key.

Le varie tabelle hanno tutte degli attributi standard, che sono:

* id: Numero intero incrementale ed univoco che rappresenta una riga della tabella
* created\_by: Id dell’utente che ha creato il record
* created\_date: Timestamp del momento in cui è stato creato il record
* updated\_by: Id dell’utente che ha eseguito l’ultima modifica sul record (quanto viene creato ha lo stesso valore di created\_by)
* updated\_date: Timestamp del momento in cui è stato modificato l’ultima volta il record (vale la stessa del created\_by)
* deleted\_by: Id dell’utente che ha eliminato il record
* deleted\_date: Timestamp del momento in cui e` stato eliminato il record

Il codice SQL per i seguenti punti e`:

create table <table\_name> (

id INT AUTO\_INCREMENT PRYMARY KEY,

created\_by INT NOT NULL,

created\_date TIMESTAMP NOT NULL,

updated\_by INT NOT NULL,

updated\_date TIMESTAMP NOT NULL,

deleted\_by INT,

deleted\_date TIMESTAMP,

<other\_attributes>

);

### Triggers

## Backend

### Gradle

### JDBC

### DAO

### Models

### Servlets

#### Actions

#### Data Servlets

## 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, cosi come gli script (JS) e i fogli di stile (CSS). Questo permette una maggiore velocita di caricamento ed una minore latenza durante la navigazione.

Il segue 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). Inoltre, viene configurato un redirect, 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 però legato ad un indirizzo del back-end. Poi per ognuna 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 cosi 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 hai 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 invia 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 service, 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") {

$locatsion.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, cosi 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* cioè 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.

## 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` statua 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+rpi1-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.

# Test

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. 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 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. 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; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## 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.

## 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.

# Consuntivo

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

# 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

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# Bibliografia

## 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,

## 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.

## 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.

# 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
* …