UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Progetto di Programmazione ad Oggetti:

Libreria Musicale

Alessandro Rago - 1187504

A.A. 2019/2020

1. Presentazione progetto

Il progetto “Libreria Musicale” consiste in un'applicazione desktop in cui chiunque può decidere di registrarsi, con due principali scopi:

1. Se ci si iscrive come “Utente standard” basterà inserire un proprio username e una propria password con il fine di poter consultare le canzoni già presenti nell’applicazione (visualizzarne il cantante, il produttore, lo scrittore, il testo).
2. Se ci si iscrive come “Artista” (Cantante, Scrittore, Produttore o Cantautore) per la registrazione sarà necessario inserire anche la propria etichetta discografica e si avrà la possibilità di caricare le canzoni con tutte le informazioni a riguardo.

E` inoltre presente un account “Admin” che ha i permessi di agire sulle canzoni caricate da tutti gli utenti, sul creare nuovi utenti o di eliminarli, cambiarne la password ecc.

2. Ambiente di sviluppo

L’intero progetto è stato sviluppato sulla macchina virtuale fornitaci, con quindi sistema operativo Linux Ubuntu, compilatore gcc 7.3.0 e framework Qt 5.9.5.

3. Compilazione, esecuzione, testing

Istruzioni per la compilazione ed esecuzione

Per la compilazione viene già fornito per comodità il file Libreria_Musicale.pro.
Volendo rigenerarlo bisognerà lanciare il comando da terminale

```
qmake -project ,
```

per poi inserire nel file .pro il flag `QT += gui widgets`.
A questo punto basterà lanciare il comando

```
qmake Libreria_Musicale.pro
```

per la generazione del Makefile, e quindi

```
make
```

per la compilazione. Per l’esecuzione dell’applicazione:

```
./Libreria Musicale
```

Istruzioni per il testing

Risulta essere fondamentale che il build venga fatto sulla stessa cartella dove sono presenti le varie cartelle GUI, MODEL, InputXML, Music per poter utilizzare correttamente i file userdata.xml e songdata.xml già popolati, altrimenti si potrà entrare attraverso solo l’utente di tipo “Admin” automaticamente generato con username “admin” e password “admin”, o registrandone uno nuovo.
Ecco una serie di utenti attraverso i quali è possibile effettuare l’accesso:

ADMIN

username: admin **password: admin**

UTENTE STANDARD

username: Alessandro **password: Alessandro**

SCRITTORE

username: Lazza **password: lazza**

CANTAUTORE

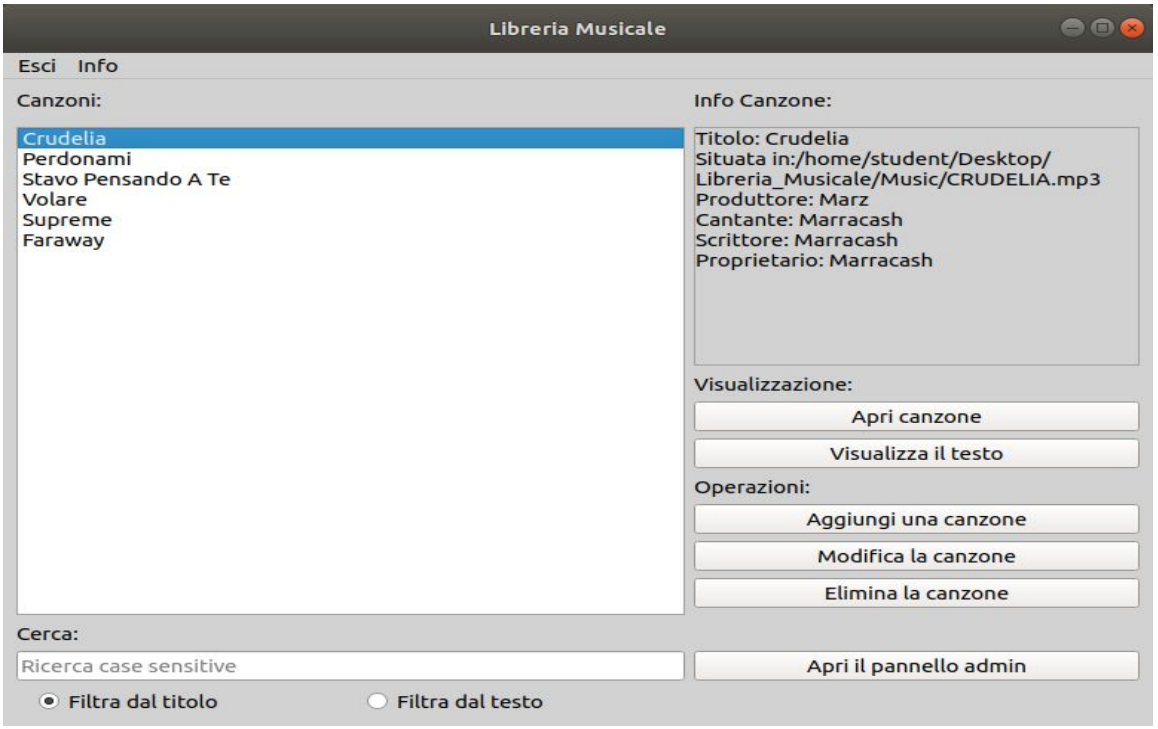
username: Salmo **password: salmo**

Gli ulteriori account già presenti sono visualizzabili nel file userdata.xml in InputXML.

Si consiglia l'accesso come ADMIN per poter testare più funzionalità da un unico account. Anche il file songda.xml è già popolato con 6 canzoni le quali sono già state scaricate e inserite nella cartella Music. Per poter ascoltarle una volta selezionate dall'applicazione sarà necessario avere già installato un qualsiasi player musicale nel proprio PC.
Nel caso possa servire questi sono i comandi per installare Rhythmbox Music Player:

```
sudo add-apt-repository ppa:vascofalves/gnome-backports
sudo apt-get update
sudo apt-get install rhythmbox
```

4. Manuale utente della GUI

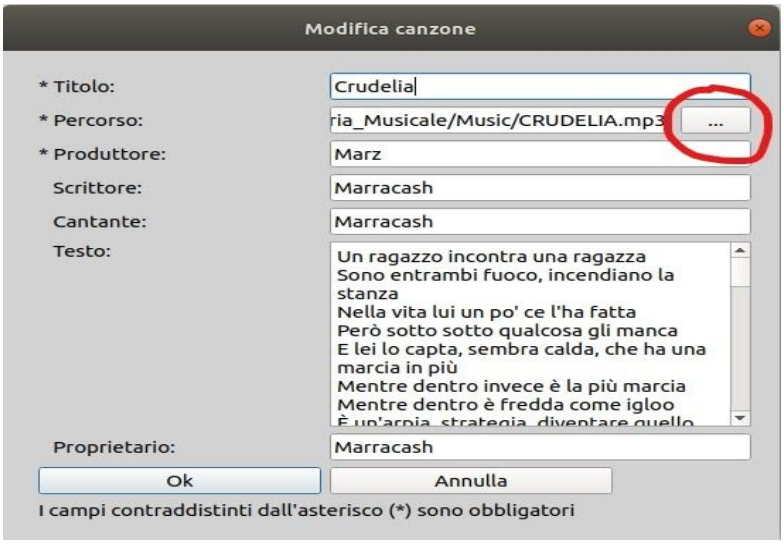


Dopo aver effettuato l'accesso come ADMIN si aprirà la finestra principale attraverso la quale è possibile eseguire tutte le funzioni disponibili, ossia visualizzare il testo della canzone selezionata nella lista a sinistra, aggiungerne una nuova, modificarne o eliminarne una già esistente. Inoltre si può aprire un pannello admin per l'amministrazione degli utenti:

Qui sarà possibile visualizzare, eliminare, modificare, tutti gli utenti registrati o aggiungerne di nuovi. Come nella schermata principale per il testo o per il titolo delle canzoni, anche qui è presente una casella di ricerca per il nome degli utenti, a favore di una ricerca più rapida.

La differenza con gli altri tipi di account è che un artista (di qualsiasi tipo) può modificare ed eliminare solo le canzoni che ha personalmente caricato, selezionandole nella lista, mentre un utente standard non può utilizzare i bottoni dedicati alle “Operazioni” sulle canzoni (i bottoni risultano disabilitati). Inoltre un artista, al posto del bottone per aprire il pannello dell’admin, ha un bottone per visualizzare il suo stipendio che dipenderà da quante canzoni ha già caricato nell’applicazione è che varia dal tipo di artista (un cantautore guadagna di più rispetto a un produttore per esempio).

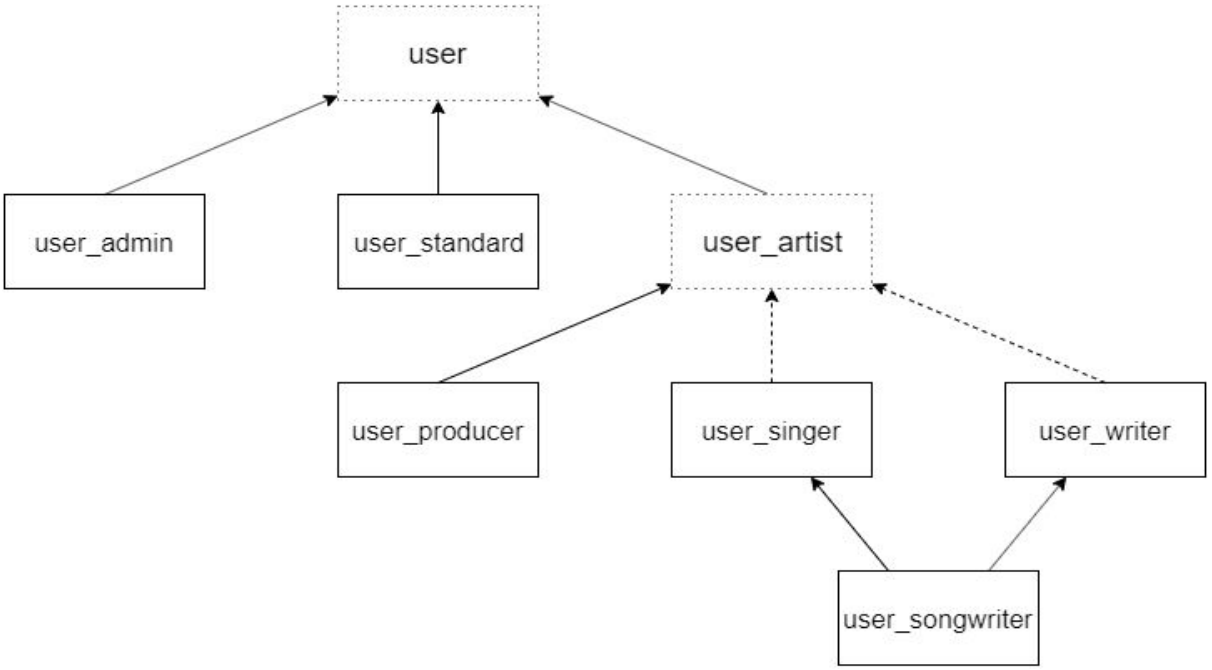
Il bottone “Apri canzone” permette a un qualsiasi utente di ascoltare la canzone selezionata attraverso un proprio player musicale, a patto il percorso in cui è stata salvata (visualizzabile in alto a destra nella sezione “Info Canzone”) sia lo stesso anche per l’utente che vuole aprirla, altrimenti si aprirà un messaggio d’errore.
Per questo motivo si consiglia l’accesso da ADMIN, così facendo si potrà modificare la canzone selezionata attraverso l’apposito bottone e quindi cambiare il percorso per raggiungere la cartella Music in cui sono già stati caricati i vari file mp3 di test:



Utilizzando il pulsante evidenziato si può modificare il percorso fino alla cartella Music.
Una volta salvato le modifiche si potrà aprire correttamente la canzone dalla schermata principale con il bottone “Apri canzone”.

5. Gerarchia principale

La gerarchia principale G è la seguente:

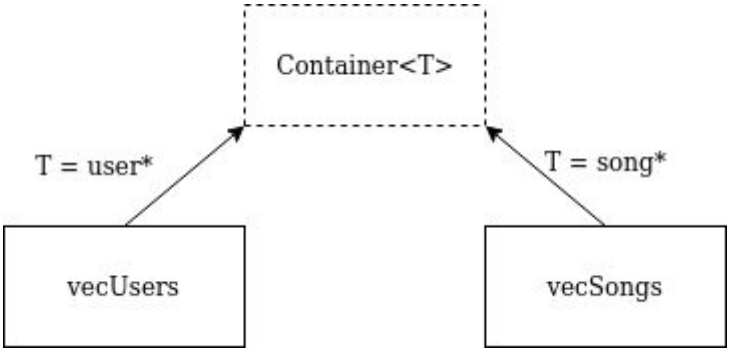


La classe base “user” è astratta, infatti sono presenti i seguenti metodi virtuali puri:
virtual bool isAdmin() const = 0 (per riconoscere se un certo utente è di tipo admin o no)
virtual bool isArtist() const = 0 (per riconoscere se un certo artista è di tipo admin o no)
virtual bool canAdd() const = 0 (per capire se l’utente può aggiungere una canzone)
virtual bool canEdit() const = 0 (per capire se l’utente può editare una canzone)
virtual bool canDelete() const = 0 (per capire se l’utente può eliminare una canzone)
virtual string getType() const = 0 (per ritornare il tipo dell’utente)
virtual user clone() const = 0* (utilizzato come costruttore di copia polimorfo)

User ha come classi figlie “user_admin” per distinguere l’account dell’admin che possiede tutti i permessi, “user_standard” che rappresenta l’utente normale con i permessi base è “user_artist” che è a sua volta una classe astratta ereditando da user i precedenti metodi virtuali puri e aggiungendo:
virtual int getDefaultSalary() const = 0 (per la restituzione del salario base di un certo artista)

Da essa derivano le classi concrete riguardanti gli artisti con l’utilizzo della *multiple inheritance,* in particolare “user_songwriter” chiude il diamante con “user_singer” e “user_writer” che derivano virtualmente da "user_artist” per far si che in ogni oggetto di tipo user_songwriter si abbia un solo sotto oggetto di tipo user_artist, evitando ambiguità.

6. Contenitore



Il contenitore “Container” è templetizzato a un parametro T e utilizzato come classe base astratta per le classi “vecUsers” (container istanziato con T = user*) e “vecSongs” (container istanziato con T = song*) concrete, che memorizzano rispettivamente tutti gli utenti registrati e tutte le canzoni inserite.
Il contenitore fornisce i metodi principali per l’inserimento, l’eliminazione sia dal fondo che da un indice preciso, ridefinizione dell’operatore di selezione, un iteratore standard e uno costante, metodi begin() e end() che ritornano un iteratore costante che punta al primo o all’ultimo elemento del contenitore rispettivamente.
Si è scelto di utilizzare un contenitore che operasse come un contenitore vector della libreria STL per rendere più veloci le operazioni di accesso agli elementi in tutto il contenitore, con l’utilizzo dell'operatore[ ], a discapito di rendere meno efficienti le operazioni di inserimento ed eliminazione degli elementi, che sono generalmente meno frequenti.

Come visto esiste inoltre una classe concreta “song” i quali oggetti, coerentemente con il nome, rappresentano canzoni create e inserite nell’oggetto contenitore di tipo vecSong.

7. Chiamate polimorfe

I distruttori della classe base *user* e del template *container* sono stati dichiarati virtuali, così da rendere virtuali anche i distruttori delle classi derivate.

I metodi virtuali puri ridefiniti nelle varie classi concrete derivanti da user sono stati utilizzati principalmente nel file *main_widget.cpp* per l'utilizzo corretto dei vari bottoni, infatti per l'esecuzione di diverse funzionalità è fondamentale capire se l'utente che ha eseguito l'accesso possa effettivamente compierle. Inoltre il metodo *clone()* ha permesso l'utilizzo del costruttore di copia in modo corretto per ogni tipo di utente sfruttando la covarianza tra i tipi di ritorno.

Anche il metodo virtuale *increaseSalary()* è utilizzato in *main_widget.cpp* ogni volta che l'utente artista inserisce una canzone, per aumentare il suo salario di una quota fissa che varia a seconda del tipo di artista (il *Plus* del songwriter = *Plus* del singer + il *Plus* del writer).

Nel *container* inoltre sono presenti i metodi virtuali puri

virtual bool addEnd(T) = 0 e virtual bool remove(string name) = 0

utilizzati in vecSongs e vecUsers per l'inserimento e la rimozione di elementi nel container, utilizzando il push_back(), pop() e pop_ind(unsigned int) rispettivamente, e la restituzione di un valore booleano per capire se l'operazione si è compiuta con successo.

Le restanti chiamate polimorfe riguardano l'utilizzo dei file di input e output.

8. Formati usati per i file di input/output

L'applicazione, per il salvataggio dei dati riguardanti gli utenti registrati e le canzoni caricate, utilizza due file contenuti nella cartella InputXML: userdata.xml e songdata.xml.

Si utilizzano vari metodi virtuali per la corretta importazione ed esportazione di dati sui due file xml, in particolare in *user* è presente il metodo

void exportXml(QXmlStreamWriter&) const

che permette l'esportazione dei dati inseriti dall'applicazione riguardante username e password dell'utente che si registra nel file di default (in questo caso userdata.xml).
Il metodo è virtuale perché overridato in *user_artist* per il salvataggio anche delle informazioni riguardanti etichetta discografica e salario.
Lo stesso metodo è presente anche in *song* (non virtuale) per inserire sul file di default (in questo caso songdata.xml) le informazioni riguardanti la nuova canzone caricata.

Nel container si utilizzano invece i seguenti metodi virtuali puri

virtual QString getStartLabelXml() const = 0 (restituisce i tag iniziali nei due file)
virtual QString getDefaultFile() const = 0 (restituisce il file di default)
virtual bool importXml() = 0 (per l'importazione di userdata.xml e songdata.xml)

utilizzati nelle classi derivate.
Nel file *exception handling* si utilizza la libreria QXmlStreamReader per la lettura negli importXml() dei vari tag, e nel caso di un errore nell'importazione di un utente (o canzone), lanciare un'eccezione e quindi saltare all'utente successivo (o canzone).

9. Suddivisione del lavoro

Il progetto è stato svolto con Samuele De Simone, matricola 1219399.

Personalmente mi sono occupato più della parte del modello e del contenitore, con le relative classi vecUsers, vecSongs e song. Ovviamente per lavorare a un buon ritmo molte cose sono state fatte insieme, modificando sezioni di codice a vicenda in caso di errori trovati nelle fasi di test, il tutto utilizzando Github come repository dei file.
Fondamentale è stato trovarci fisicamente per organizzare modello e vista, per confrontarsi più facilmente, discutere sulle parti più critiche riguardanti l'interfaccia della GUI e l'utilizzo in generale del framework Qt.

10. Ore di sviluppo

Analisi preliminare e Progettazione interfaccia : 4 ore

Studio Qt : 8 ore (tenendo conto del Tutorato)

Scrittura modello : 26 ore

Scrittura GUI : 12 ore

Testing e Debugging: 6 ore

L'ammontare delle ore supera le 50 ore sicuramente per la continua consultazione della documentazione di Qt riguardo librerie, classi e metodi che abbiamo utilizzato. Inoltre è successo più volte di dover riscrivere parti di codice per evitare ambiguità tra modello e vista o semplicemente per correggere errori non controllati nell'immediato nella fase di testing.