Progetto di Programmazione ad Oggetti

A.A. 2022/2023

QTShop

Componenti

Bomben Filippo 2008461

Indice

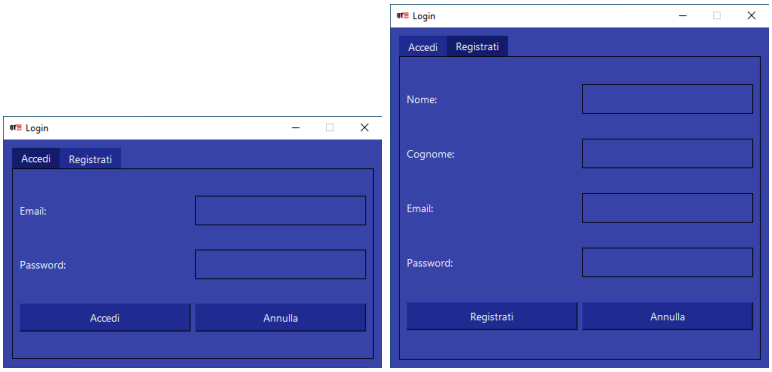
1 Introduzione

QTShop è un'applicazione che gestisce un negozio in stile Feltrinelli. All'interno del software a seconda di come si accede, si possono acquistare Film, Libri e Musica, oppure, se effettuato l'accesso come amministratore, è possibile gestirne il suo magazzino, aumentando la quantità disponibile di uno specifico oggetto, o tramite la creazione/eliminazione di altri. L'utente grazie agli acquisti può guadagnare punti che possono essere spesi per i premi, e che quindi non possono essere acquistati regolarmente. Ogni euro speso equivale ad 1 punto, il punto viene arrotondato per difetto nel caso i centesimi siano <50 e arrotondato per eccesso nel caso i centesimi siano $>= 50$.

1.1 Modifiche effettuate

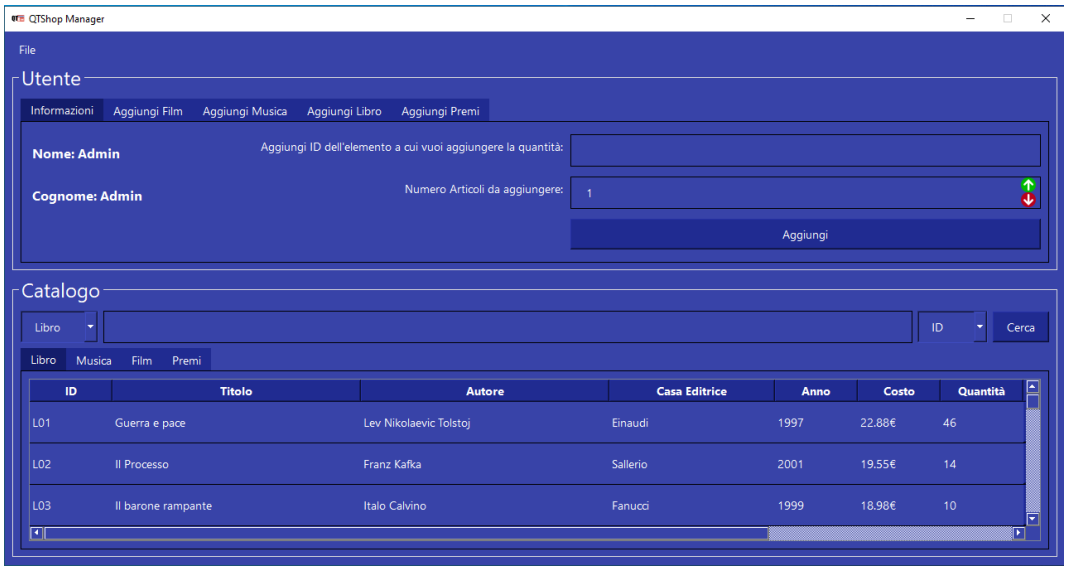
- Implementazione classe Contenitore, che gestisce un contenitore nativo per la gerarchia Oggetto.
- Sistemati i nomi delle classi Oggetti in Oggetto e Libri in Libro.
- Aggiunta una funzione virtuale nella gerarchia Oggetto, **virtual void stampaDettagli() const**, **virtual void updateDelOgg() const** e **virtual int updateQtaOgg() const**.
- Eliminazione del metodo virtuale getType() e pulizia del codice in Database
- Maggiore possibilità di ridimensionare l'applicazione.
- Maggiore personalizzazione della GUI con aggiunta di colori e icone.

1.2 Interazione GUI



Per usufruire del software si può accedere con due profili già preesistenti, user e admin, in alternativa ci si può registrare riempiendo i campi nella Tab "Registrati".

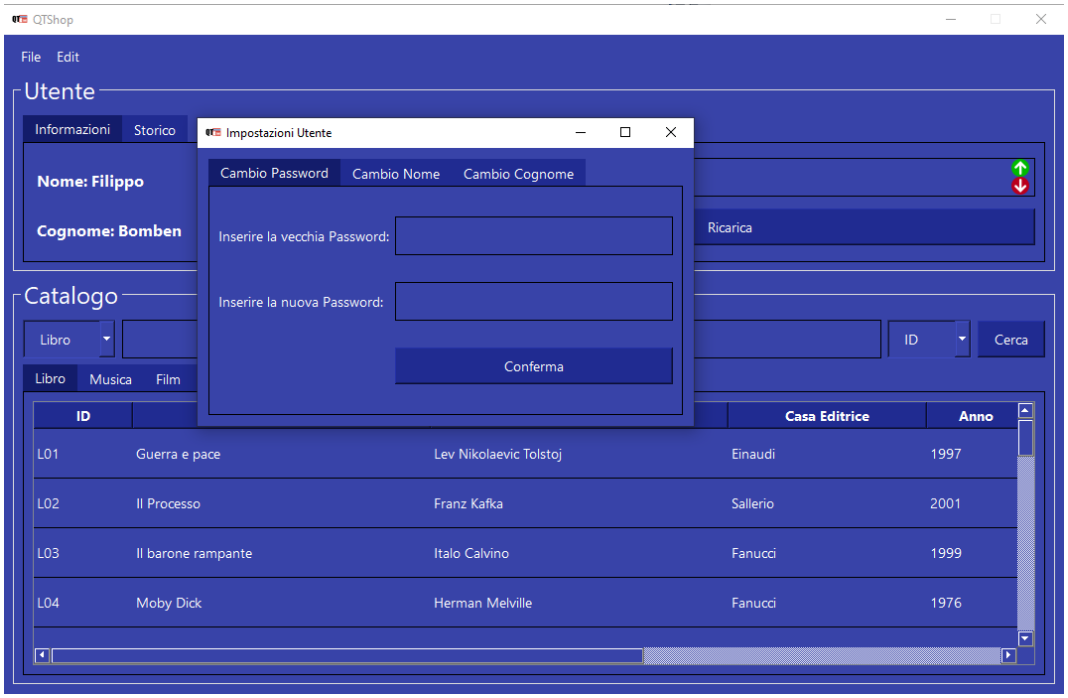
1.2.1 Interazione Admin



- Informazioni dell'utente e l'aggiunta di quantità per uno specifico elemento.
- Aggiunta di Film, Musica, Libri e Premi.
- Ricerca di elementi tramite search bar (vale anche per l'utente non admin), se lasciata vuota e premuto il pulsante "cerca" la tabella mostra tutti i risultati.
- Eliminazione dell'oggetto tramite pulsante, situato nell'ultima colonna di ogni riga.
- Tramite il menù superiore è possibile chiudere l'applicazione o tornare alla finestra di login.

1.2.2 Interazione User

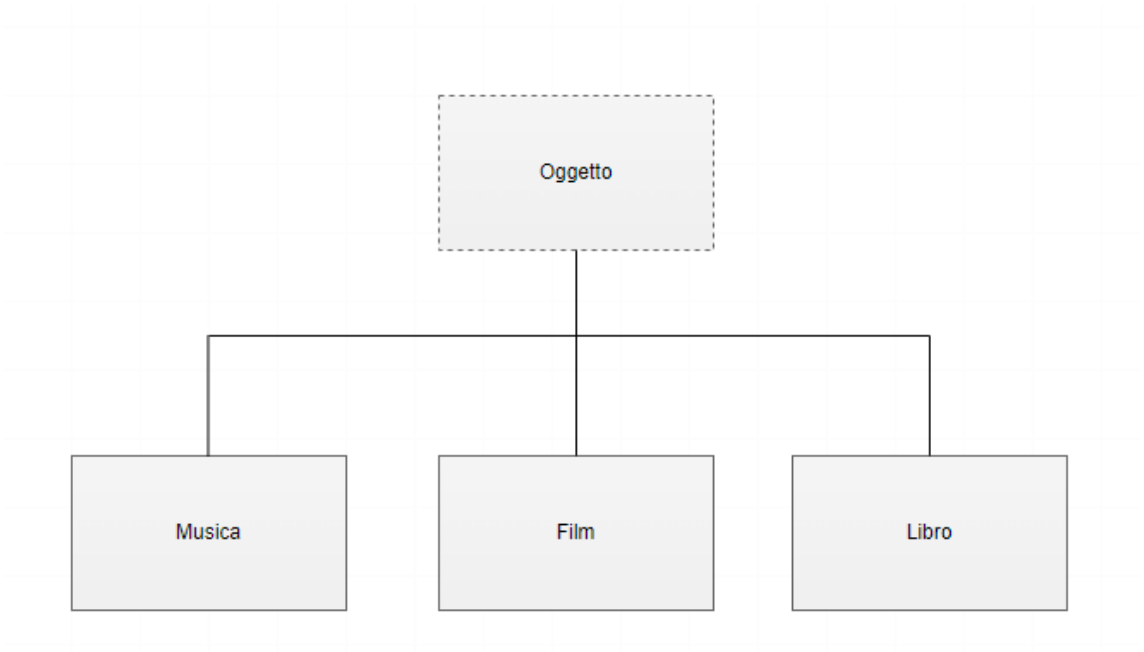
- Informazioni dell'utente e la possibilità di aggiungere soldi al proprio saldo.
- Visualizzare lo storico dell'utente, degli oggetti acquistati.
- Acquisto oggetti tramite pulsante situato nell'ultima colonna di ogni riga.
- Tramite il menù superiore, è possibile eseguire tutte le interazioni che ha anche l'admin, con l'aggiunta dell'eliminazione del proprio profilo, il cambio nome/cognome e il cambio password.


QTShop
File Edit
Utente
Informazioni Storico
Nome: Filippo
Cognome: Bomben
Ricarica
Impostazioni Utente
Cambio Password Cambio Nome Cambio Cognome
Inserire la vecchia Password:
Inserire la nuova Password:
Conferma
Catalogo
Libro
ID
Cerca
Libro Musica Film
ID
Casa Editrice
Anno
L01 Guerra e pace Lev Nikolaevic Tolstoj Einaudi 1997
L02 Il Processo Franz Kafka Sallerio 2001
L03 Il barone rampante Italo Calvino Fanucci 1999
L04 Moby Dick Herman Melville Fanucci 1976

2 Struttura

Il progetto si compone di due gerarchie: Oggetto e V_Main.
Oggetto è una classe astratta da cui derivano le classi Musica, Film e Libro, mentre la classe V_Main ha come classi derivate V_App_A (per la gestione della view per l'admin) e V_App_U (per la gestione della view dell'utente).
E' stato implementato un contenitore nativo all'interno della classe Contenitore. La struttura Nodo salva gli oggetti di tipo Oggetto all'interno del campo **data** e il campo dati **next** punta all'oggetto successivo del nodo. Ogni volta che un oggetto o un utente viene creato, il database viene aggiornato immediatamente tramite la classe Database, il quale gestisce le chiamate per aggiornare il contenitore nodo.
Il progetto sfrutta il pattern **MVC** (Model-View-Controller) con un unica classe Controller che fa comunicare la vista con il modello e viceversa

2.1 Gerarchia Oggetto



La gerarchia della classe Oggetto è composta da varie funzioni:

- Vari get e set di default per prende o modificare i campi dati privati.
- **virtual double getCosto() const**:

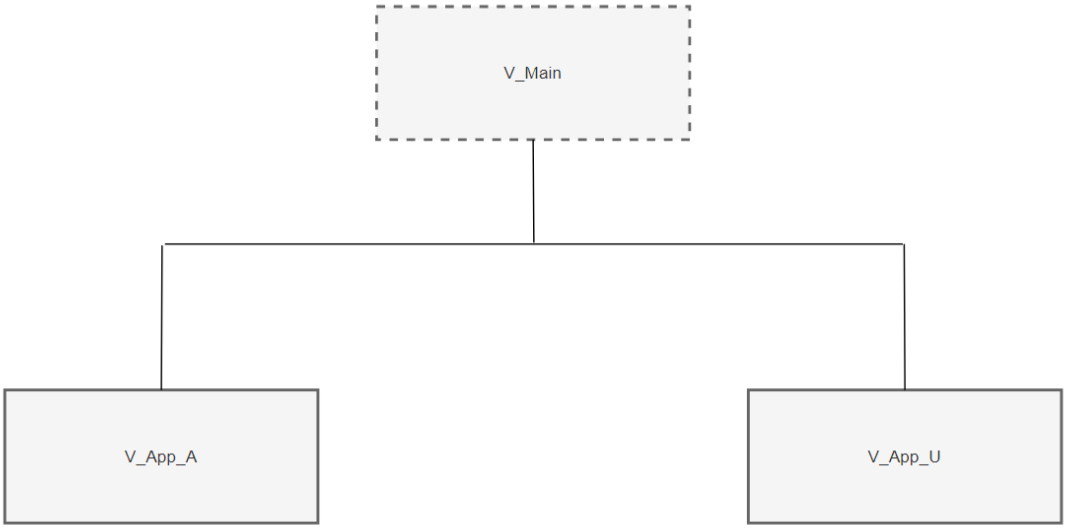
  ritorna la variabile costo, nelle classi derivate è presente un static double IVA che varia in base all'oggetto, in questa maniera il costo finale è compreso di IVA.
- **virtual void stampaDettagli() const**:

  Stampa i campi dati dell'oggetto invocato, mostrando tutte le informazioni necessarie.
- **virtual void updateDelOgg() const =0**

  Esegue la query per l'eliminazione di ogni diverso tipo di oggetto, in maniera da aggiornare il database costantemente
- **virtual int updateQtaOgg() const =0**

  Esegue la query per aggiornare nel database il campo "quantità" di ogni diverso tipo di oggetto.

2.2 Gerarchia V_Main



La gerarchia della classe V_Main è composta da varie funzioni:

- **void creaTable(bool admin)**:

  Non ho reso questa funzione virtuale pura per non avere la maggior parte del codice duplicato, ho quindi utilizzato una variabile bool per capire se la funzione creerà la view per l'admin o per l'user. In alternativa avrei reso la funzione virtuale. Questa funzione chiama a sua volta, **void insLib(QTableWidget *table, bool admin), void insFil(QTableWidget *table, bool admin), void insMus(QTableWidget *table, bool admin), void insPre(QTableWidget *table, bool admin)**, per popolare le tabelle.

- **virtual void creaMenu()**:

  Crea il Qmenu dell'applicazione aggiungendo nella view dell'user maggiori opzioni.

- **virtual void creaInfo() =0**:

  Funzione virtuale pura che crea la sezione Utente (QGroupBox) della view. Entrambi i QGroupBox hanno al suo interno delle QTabView per la gestione delle finestre.

- Funzioni per ricaricare il contenuto delle tabelle, in maniera che, se eliminati o aggiunti elementi, il controller possa chiamare la funzione e aggiornare la view.

- Slot e segnali, per quanto riguarda comprare o eliminare elementi dalla tabella. E' stata utilizzata una lambda function per passare i parametri allo slots, disponibile da C++11 e versioni superiori. Ho dovuto utilizzare questa soluzione per passare le informazioni.

2.3 Altre Classi

2.3.1 Database

La classe Database gestisce tutti i cambiamenti, degli oggetti, premi e utenti. Il controller quindi, comunica con il database per modificare la vista. Il costruttore di "Database" popola le liste e il contenitore prendendo i dati dalle tabelle del DB.

2.3.2 Messaggi

Messaggi è una classe con funzioni statiche che aprono dei QMessageBox, vengono chiamati all'interno delle altre classi per informare l'utente se l'operazione è andata a buon fine o meno.

2.3.3 Premi

La classe premi ha come campi dati:

- **id**: id del premio, QString sempre univoco, di base standard con la P come inizio (P...) e poi una cifra a crescere, non necessario però rispettare questo standard.
- **nome**: titolo QString dell'oggetto, può essere poster, portachiavi...
- **cPunti**: Il costo in Punti che l'utente deve utilizzare per poter comprare il Premio.
- **qta**: Quantità disponibile, se il premio ha una sola quantità e l'utente lo compra, viene eliminato automaticamente dalla tabella.

2.3.4 Utente

Utente è una classe con i campi dati, nome, cognome, email (univoco), password, saldo e punti. La classe oltre ad avere le classi funzioni get e set presenta le seguenti funzioni:

- **void addSaldo(const double& aSaldo)**: Aggiunge il valore della QDoubleSpinBox nel campo dati saldo.
- **void sotSaldo(const double& sSaldo)**: Sottrae il costo dell'oggetto acquistato al campo dati saldo.
- **void addPunti(const unsigned short int& aPunti)**: Aggiunge i punti.
- **void sotPunti(const unsigned short int& sPunti)**: Sottrae i punti

2.3.5 Storico

Lo Storico utilizza nel private il campo dato idU, che sarebbe l'email dell'utente e i campi dati id, titolo e costo per salvare informazioni dell'oggetto. Nello storico vengono salvati gli acquisti di Oggetto e non dei Premi, grazie all'idU la view carica correttamente solo gli acquisti fatti dall'utente della sessione corrente.

3 Istruzioni di compilazione

Per compilare il progetto bisogna spostarsi da terminale all'interno della cartella principale contenente il file QTShop.pro e lanciare i seguenti comandi:

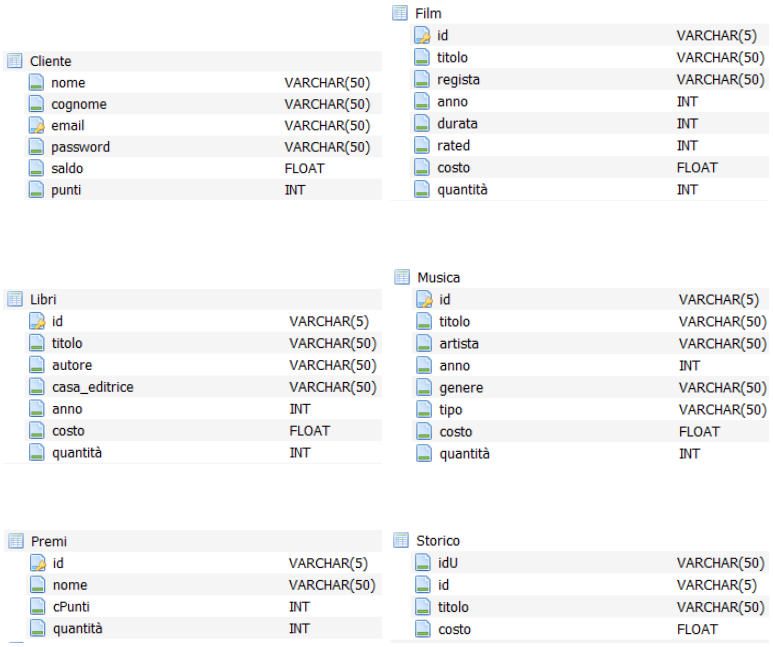
- qmake QTShop.pro
- make
- ./QTShop

Per accedere all'applicazione sono stati creati di default due profili, admin, non eliminabile, e user, volendo si può eliminare e crearne uno nuovo.

Email	**Password**
user	user
admin	admin

4 Database e persistenza dati

Per la creazione del Database è stato utilizzato SQLite, all'interno della cartella del progetto il database.sqlite carica i dati necessari. Le Tabelle sono strutturate nel seguente modo:



Per la popolazione delle tabelle ho utilizzato parzialmente `https://www.mockaroo.com/`.

5 Controlli

L'applicazione chiaramente fa interagire l'utente con l'interfaccia. Ci sono però dei controlli che fa e che comunica all'utente, ad esempio, l'utente non può chiaramente acquistare oggetti se non ha un saldo sufficiente, allo stesso modo per l'acquisto dei premi. L'admin invece non potrà creare oggetti con una quanità minore di 1, infatti la QSpinBox non potra scendere sotto 1, altri controlli vengono fatti su QSpinBox e QDoubleSpinBox settando valori massimi e minimi, per le date non si può andare prima dell'anno 1000 e dopo il 2023. Anche i QLineEdit se non compilati in maniera corretta segnaleranno errori con relativa spiegazione.

6 Descrizione ore richieste e informazioni tecniche

Attività	**Ore impiegate**
Analisi e idea progetto	5 ore
Apprendimento della libreria Qt	8 ore
Scrittura codice	29 ore
Debugging	6 ore
Testing	3 ore
Relazione	3 ore

Sono state superate le ore richieste di poco, sopratutto per capire il funzionamento della libreria Qt e risolvere qualche problema con la comunicazione con il database. Alla fine sono state impiegate 54 ore più 30min/1 ora per la popolazione delle tabelle.

Sistema Operativo	Windows 10
G++	11.2.0
Qt	6.2.4
Qt Creator	9.0.1