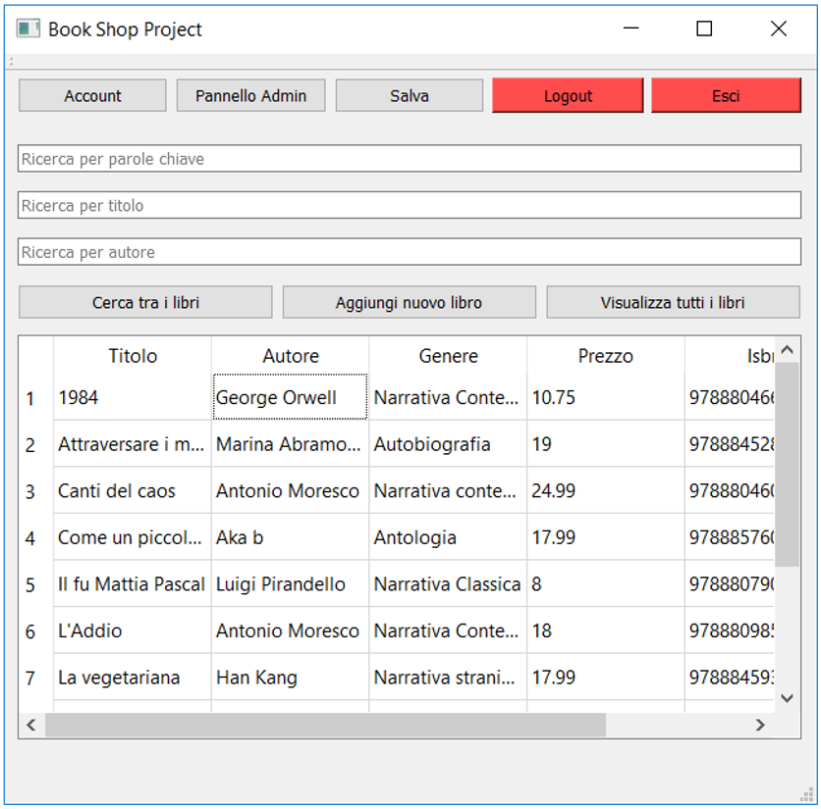
Riccardo Patanè (Matricola n. 1125514)


# Programmazione ad Oggetti

## Relazione del progetto

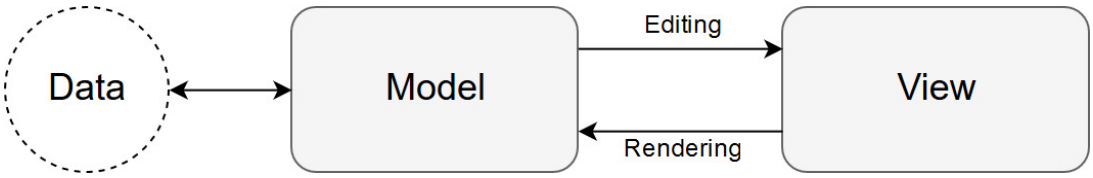



Sommario:

# Scopo del progetto

Il *BookShopProject* si prefigge lo scopo essere un'applicazione di supporto per la gestione di una libreria. Ogni utente, una volta effettuato il login, può cercare tra i libri disponibili (sfruttando diverse modalità) e per ogni libro visualizzarne le informazioni.
Ad un utente abilitato è permesso inserire, modificare e cancellare i libri presenti nel catalogo. Questo ci permette di avere una visione completa e aggiornata di tutti i libri.

Per far ciò ho deciso di  adottare la *Model View Architecture*, permettendo così un' astrazione tra *model* e *view.* Questa separazione agevola la localizzazione di possibili errori e semplifica implementazioni e modifiche future.




# Descrizione delle gerarchie e dei tipi utilizzati

Nel progetto è presente una gerarchia che riunisce le diverse tipologie di utenti e che ha come base la classe astratta *User*. I libri sono rappresentati tramite l' uso di una sola classe non avendo notato caratteristiche che potessero rendere utile una gerarchia.

Sono presenti due classi "contenitori" *BookList* e *UserList che tramite una std::list* gestiscono rispettivamente i libri presenti nel catalogo e gli utenti registrati.
*BookList* e *UserList si occupano della lettura e scrittura su file, dell'inserimento, della cancellazione e della modifica di ogni singolo elemento che contengono.*

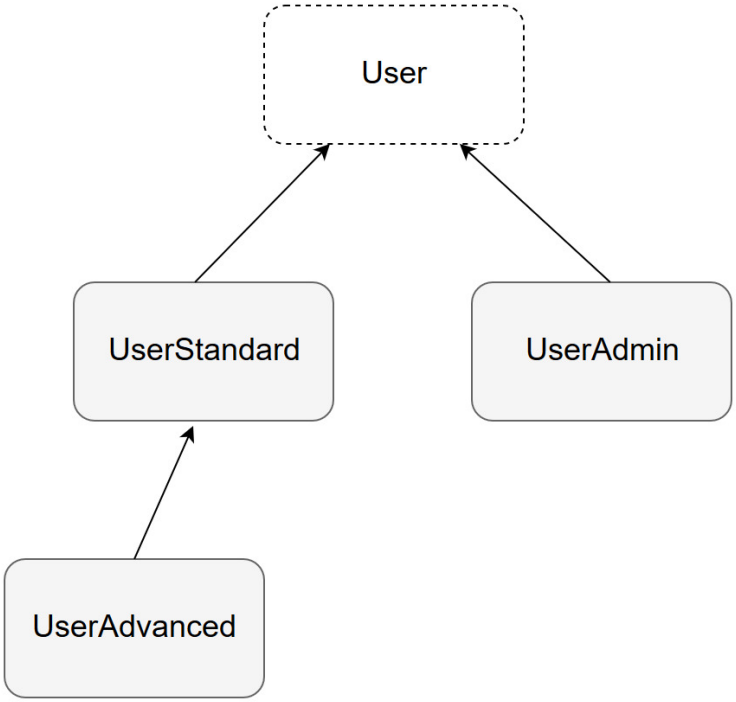
## Descrizione della classe *Book*

La classe Book modella il concetto di libro sfruttando 9 campi dati:

- titolo
- autore
- genere
- trama
- ISBN
- numero delle pagine
- anno
- numero di copie presenti
- prezzo

La classe ha a disposizione l'overloading dell'operatore di uguaglianza, un metodo che genera la stringa per il salvataggio su file del libro, un costruttore che sfrutta la stringa per costruire il libro.

# Descrizione della gerarchia User




La gerarchia degli utenti ha come radice la classe astratta *User* contenente *nickname* e *password*.
Da *User* derivano *UserStandard* e *UserAdmin*, entrambe le classi sovrascrivono le funzioni virtuali che gestiscono i permessi.
Da UserStandard deriva UserAdvanced che sovrascrive la funzione virtuale canWrite() così da abilitare il permesso di modifica.

Schema con i permessi di ogni classe utente

| | Lettura | Scrittura | Admin |
|---|---|---|---|
| UserStandard | ✔ | ✘ | ✘ |
| UserAdvanced | ✔ | ✔ | ✘ |
| UserAdmin | ✔ | ✔ | ✔ |

*UserStandard* è un utente che ha solo il diritto di visionare i libri disponibili nel negozio (esempio: account di un cliente).
*UserAdvanced* è un utente che ha il diritto di visionare e modificare il catalogo del negozio (esempio: account di un commesso o di un magazziniere).
*UserAdmin* è un utente che ha il diritto di visionare e modificare il catalogo del negozio, inoltre può vedere e modificare gli altri account presenti (esempio: direttore della libreria o responsabile del personale).

Ogni utente ha il permesso di modificare il proprio nickname e la propria password.
Nessun utente (UserAdmin compreso) può modificare i propri permessi.

# Descrizione sul utilizzo del codice polimorfo

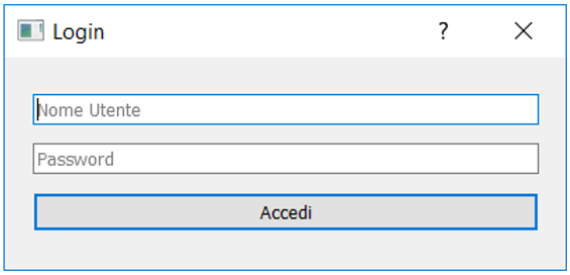
Nella gerarchia degli utenti sono presenti le seguenti funzioni virtuali:

- `User* clone() const`
  è una funzione virtuale pura che da User si estende in tutta la gerarchia tramite delle funzioni covarianti, viene sfruttato l'overriding per restituire un puntatore a una classe derivata rispetto a User
- `bool isAdmin() const`
  è una funzione virtuale pura che da User si estende in tutta la gerarchia. Tramite l'overriding viene gestito il permesso di essere amministratore.
- `bool canWrite() const`
  è una funzione virtuale pura che da User si estende in tutta la gerarchia. Tramite l'overriding viene gestito il permesso di modificare il catalogo
- `std::string fUsertString() const`
  è una funzione virtuale pura che da *User* si estende in tutta la gerarchia. Tramite l'overriding vengono generate diverse stringhe che verranno sfruttate dalla classe *UserList* per memorizzare i dati dell'utente e la sua tipologia sul file .txt .

# Manualistica per l'utente

Vengono forniti 2 file userfile.txt e bookfile.txt da inserire nella cartella che contiene l' eseguibile. I file contengono  rispettivamente un numero sufficiente di utenti e di libri da effettuare dei test.

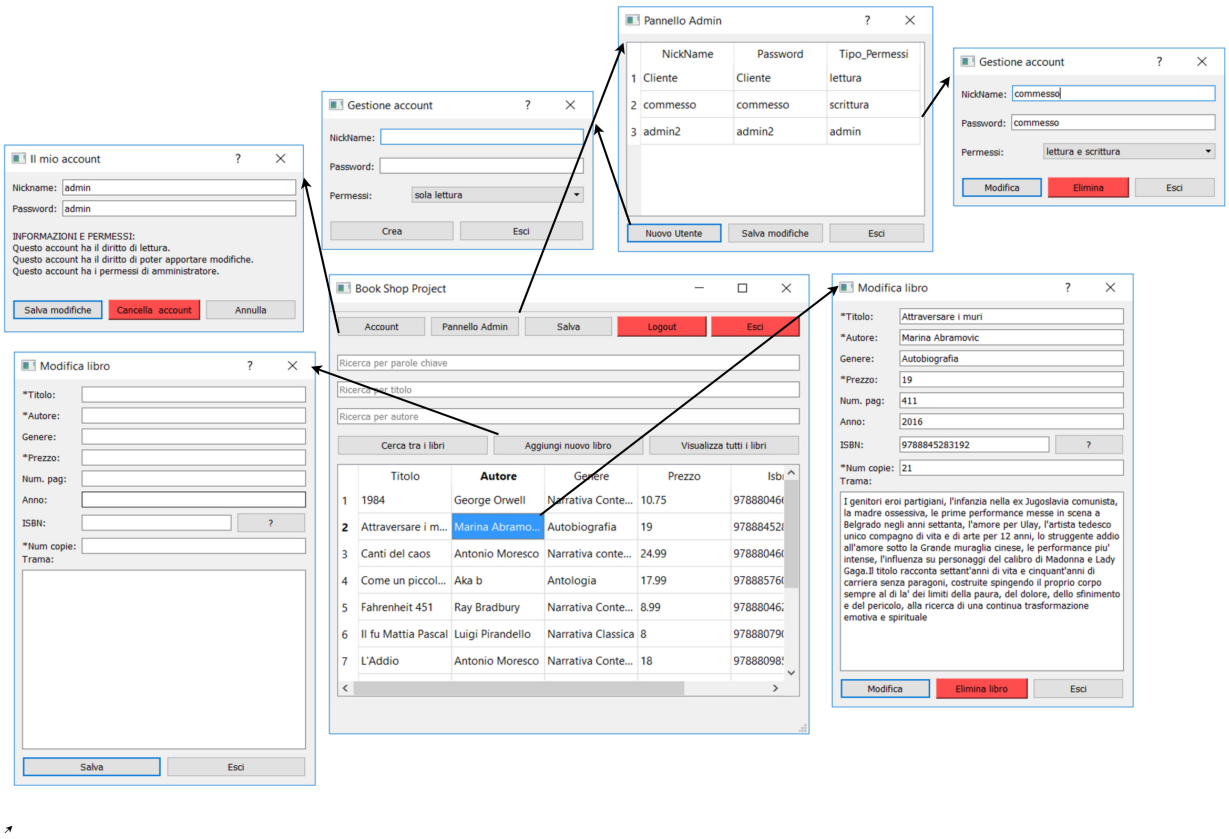
All'avvio il programma tenta di leggere il file *userfile.txt . S*e il file non è presente oppure è vuoto viene generato un amministratore di default con il quale si dovrà fare il login.
**Le credenziali per il login sono  admin, admin anche nel caso in cui il file non sia presente.**
Nel caso in cui sia presente il file si dovrà fare il login con le credenziali di uno degli utenti riportati.




Solo dopo che il login ha avuto successo avviene la lettura di bookfile.txt .
Nel caso in cui il file non esista il catalogo sarà privo di libri, questo sarà segnalato da una finestra di dialogo. Se il file è presente verranno visualizzati tutti i libri disponibili.

L' assenza dei file userfile.txt e bookfile.txt non compromettono il programma.

Una volta effettuato il login si accede alla schermata principale.
Nella pagina successiva è presente uno schema che riepiloga le funzionalità e le schermate principali.




Questa è la schermata completa,a cui solo l' admin riesce ad accedere.
Se un utente non ha i permessi di amministratore non visualizza il tasto *Pannello Admin.*
Se un utente non ha i permessi di scrittura non visualizza il tasto *Aggiungi nuovo libro,* inoltre il *doppio click su un libro genera una finestra che mostra tutte le info del libro (fatta eccezione per la quantità disponibile) ma non c' è possibilità di modificare o cancellare il libro.*

*T*asti della schermata principale:

- Il tasto *Salva* salva le modifiche del catalogo e della lista utenti su file.
- *Logout* riporta alla fase di autenticazione l' utente.
- *Esci* conclude l' esecuzione del programma.
- *Cerca tra i libri* ha tre modalità di ricerca.
    1. La prima casella verifica la presenza delle parole inserite in un qualsiasi campo
    2. La seconda permette la ricerca per titolo
    3. L' ultima effettua la ricerca per autore

  La lista risultante è l' unione delle tre ricerche. Ogni campo non compilato non verrà considerato.
- *Visualizza tutti i libri* mostra tutti i libri disponibili.

# File consegnati

main.cpp
BookShopProject.pro
accountuserwindow.io
accountwindow.io
adminwindow.io
bookwindow.io
loginwindow.io
mainwindow.io
bookfile.txt
userfile.txt
relazione.pdf

/Model

- book.h
- book.cpp
- booklist.h
- booklist.cpp
- user.h
- user.cpp
- useradmin.h
- useradmin.cpp
- useradvanced.h
- useradvanced.cpp
- userlist.h
- userlist.cpp
- userstandard.h
- userstandard.cpp

/View

- accountuserwindow.h
- accountuserwindow.cpp
- accountwindow.h
- accountwindow.cpp
- adminwindow.h
- adminwindow.cpp
- bookwindow.h
- bookwindow.cpp
- loginwindow.h
- loginwindow.cpp
- mainwindows.h
- mainwindows.cpp

## Descrizione ore di lavoro

| | |
|---|---|
| Progettazione: | 4h |
| Progettazione grafica: | 2h |
| Scrittura codice: | 22h |
| Gestione file: | 3h |
| Scrittura GUI: | 20h |
| Test e debug: | 5h |
| Documentazione: | 5h |
| **TOTALE:** | **61h** |

## Ambiente di sviluppo

Sviluppato su:

- Qt Creator 4.2.1
- Based on Qt 5.8.0
- Windows 10 (10.0)

Testato su:

- Qt Creator 3.5.1
- Based on Qt 5.5.1
- Ubuntu 16  (16.04.2 LTS)
- t34.torre.math.unipd.it