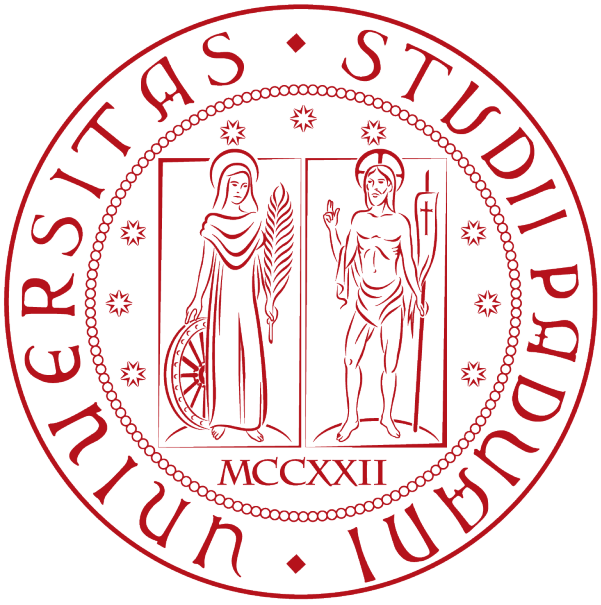


# QTComicStore

Alberto Giudice - 10 Luglio 2017

| Tipo Utente | Username | Password |
|---|---|---|
| Gestore | admin | admin |
| Cliente Base | base | base |
| Cliente Gold | gold | gold |

# Indice

# 1 Scopo del progetto

Il progetto **QtComicStore** è un'applicazione per l'inventario di una fumetteria, pensata sia per il suo gestore che per i clienti.
Nella fumetteria sono presenti vari articoli, che in particolare possono essere fumetti, DVD o action figure. Tali articoli possono essere aggiunti, cercati, modificati e rimossi dal gestore della fumetteria, che è unico. Il gestore può inoltre aggiungere, cercare, modificare e rimuovere utenti, che si suddividono in due categorie: clienti base e clienti gold. I clienti base possono solamente cercare gli articoli di loro interesse secondo vari filtri. I clienti gold possono, oltre a cercare, aggiungere gli articoli di loro interesse a una lista dei preferiti facile da consultare. Ogni utente può comunque modificare i propri dati personali, eccezion fatta per la tipologia di cliente a cui appartiene, modificabile solo dal gestore.

# 2 Compilazione ed esecuzione

Per compilare il progetto è sufficiente aprire il terminale all'interno della cartella principale contentente il file `Fumetteria.pro`, eseguire il comando `qmake` e successivamente il comando `make`. Una volta compilato per avviarlo è necessario il comando `./Fumetteria`. Assieme ai file `.h` e `.cpp` vengono consegnate due immagini nella cartella `.\images\` contenenti l'icona dell'applicazione e lo sfondo della schermata principale, oltre che i file `databasestuff.xml` e `databaseutenti.xml` contenenti degli utenti e degli articoli di esempio. Nel caso il file relativo al database degli utenti venisse cancellato è previsto che all'avvio venga ricreato con all'interno un gestore con le seguenti credenziali:

- **username**: admin
- **password**: admin

# 3 Struttura

## 3.1 Informazioni Generali

Il progetto si compone di due gerarchie principali: `User` e `NerdStuff`, entrambe modellate a livello di contenitore tramite la struttura `list` messa a disposizione dalla libreria STD e implementata tramite puntatori agli oggetti di base astratti delle due gerarchie. Inoltre l'untente Gold contiene al suo interno un'altra struttura `list`, con la quale durante l'esecuzione ha la possibile di creare una propria lista di articoli preferiti, visualizzabili in un'apposita finestra. Eventuale *Garbage* è gestito dalle funzioni di eliminazione e dai distruttori delle singole classi, ove necessario.
A livello di struttura il progetto è stato realizzato seguendo una divisione tra *business logic* e presentazione. Per tale motivo è stato adottato il pattern `Model-View-Controller`. In questo modo è possibile delegare alla `View` i controlli più semplici derivati ad esempio da campi dati vuoti, mentre il `Controller` si occupa di gestire tutte le interazioni tra parte grafica e parte logica. A tale scopo ogni classe è dotata degli opportuni metodi `get` e `set`.
L'interfaccia grafica è stata realizzata senza l'ausilio di tool come *QtDesigner* o affini. Al fine di evitare la ripetizione di codice le finestre si adattano al segnale che le ha richiamate o all'utente che le sta visualizzando, mostrando o nascondendo dinamicamente alcuni elementi. Per far ciò è stato utilizzato il **QGridLayout** per una disposizione

ordinata di questi ultimi. La `View` si occupa inoltre anche della distruzione degli oggetti della GUI, poiché da essa creati.

## 3.2 Gerarchia Utenti

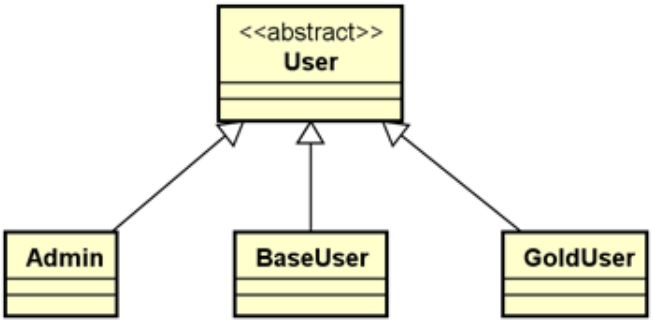


Figura 1: Gerarchia utenti

### 3.2.1 Base astratta *User*

Rappresenta il prototipo di utente, da cui vengono derivati pubblicamente tutti gli utenti concreti.
• `virtual QString getType() const=0`: restituisce una QString contenente il tipo dell'utente, con il quale vengono impostati alcuni elementi grafici della GUI. Inoltre è utilizzato dove possibile per evitare dei cast dinamici o statici;
• `virtual bool hasStarred()const=0`: restituisce un booleano che è *true* se l'utente possiede una lista di articoli preferiti e la relativa visualizzazione nella GUI, *false* altrimenti;
• `virtual bool hasUserList()const=0`: restituisce un booleano che è *true* se l'utente può visualizzare la lista dei clienti della fumetteria nella GUI, *false* altrimenti;
• `virtual bool hasManagement()const=0`: restituisce un booleano che è *true* se l'utente ha i privilegi per gestire la fumetteria, ossia può aggiungere, rimuovere e modificare utenti e articoli, *false* altrimenti;
• `virtual void saveUser(QXmlStreamWriter&)`: metodo virtuale necessario al salvataggio dei dati relativi agli utenti nel database, in base al tipo di utente richiama inizialmente in modo esplicito la funzione data dalla base `User` e poi aggiunge in modo virtuale ulteriori istruzioni;

I preferiti di un utente Gold non sono salvati su file, dunque vengono persi alla terminazione del programma. Ciò è ragionevole in quanto tale lista serve come promemoria personale durante una visita alla Fumetteria.
Le classi `Admin`, `BaseUser` e `GoldUser` si occupano di implementare i metodi virtuali della base, conferendo ad ogni utente diversi permessi.

Nel dettaglio:

| **Ruolo** | **Funzionalità** |
|---|---|
| Gestore | • ricerca di articoli e utenti secondo filtri<br>• aggiunta, modifica e rimozione di articoli e utenti<br>• modifica dei propri dati utente |

| | |
|---|---|
| Cliente Gold | • ricerca di articoli secondo filtri<br>• aggiunta e rimozione di articoli alla lista dei preferiti<br>• modifica dei propri dati utente |
| Cliente Base | • ricerca di articoli secondo filtri<br>• modifica dei propri dati utente |

Tabella 1: Funzionalità per utente

## 3.3 Gerarchia Articoli

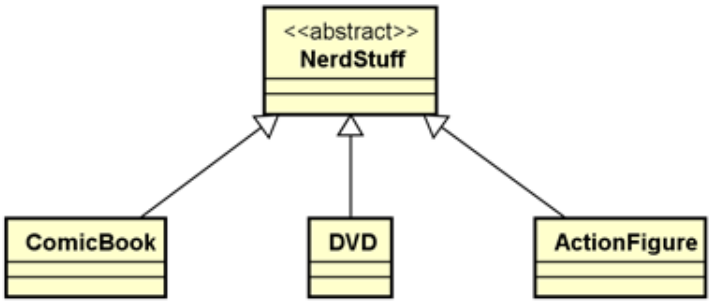


Figura 2: Gerarchia articoli

### 3.3.1 Base astratta *NerdStuff*

Rappresenta il prototipo di articolo, da cui vengono derivati pubblicamente tutti gli articoli concreti.
• `virtual QString getType() const=0`: restituisce una QString contenente il tipo dell'articolo, con il quale vengono impostati alcuni elementi grafici della GUI. Inoltre è utilizzato dove possibile per evitare dei cast dinamici o statici;
• `virtual double getPrice() const =0`: restituisce un double rappresentante il prezzo finale dell'articolo desiderato, calcolato in maniera diversa in base alla tipologia esatta di articolo che la richiama;
• `virtual void saveStuff(QXmlStreamWriter &)`: metodo virtuale necessario al salvataggio dei dati relativi agli articoli nel database, in base al tipo di articolo richiama inizialmente in modo esplicito la funzione data dalla base `NerdStuff` e poi aggiunge in modo virtuale ulteriori istruzioni;
• `virtual void loadStuff(QXmlStreamReader &)`: metodo virtuale necessario alla lettura dei dati relativi agli articoli nel database, in base al tipo di articolo richiama inizialmente in modo esplicito la funzione data dalla base `NerdStuff` e poi aggiunge in modo virtuale ulteriori istruzioni;
Inoltre è presente un campo dati statico che rappresenta l'IVA.

### 3.3.2 Classe concreta *ComicBook*

La classe `ComicBook` rappresenta un fumetto e ha i seguenti campi dati:

- `title`: rappresenta il titolo del fumetto;

- `author`: rappresenta l'autore del fumetto;
- `comicType`: rappresenta la categorie di fumetto (comics americano, italiano, franco-belga o manga giapponese);
- `volumeNumber`: rappresenta il numero di volume del fumetto, può andare da 0 a 9999;
- `pageNumber`: rappresenta il numero di pagine del fumetto, può andare da 0 a 9999;

### 3.3.3 Classe concreta *Dvd*

La classe `Dvd` rappresenta un film in DVD e ha i seguenti campi dati:

- `title`: rappresenta il titolo del film;
- `produce`: rappresenta il regista del film;
- `dvdType`: rappresenta la categorie di film (animazione o live-action);
- `filmLength`: rappresenta la durata del film, può andare da 0 a 300 minuti;
- `isAgeRes`: rappresenta se il film è o meno vietato ai minori;

### 3.3.4 Classe concreta *ActionFigure*

La classe `ActionFigure` rappresenta una action figure (statuina/collezionabile) e ha i seguenti campi dati:

- `name`: rappresenta il nome del prodotto;
- `manufacturer`: rappresenta la casa produttrice;
- `size[3]`: array che rappresenta le tre dimensioni spaziali del prodotto in cm, tramite un intero che va da 0 a 200;

In base alle dimensioni la funzione `QString getDimension() const` restituisce una stringa che determina se la statuina è Piccola, Media o Grande.

# 4 Manuale GUI

Nel caso in cui i file non venissero trovati viene visualizzato un messaggio informativo. Dopo aver inserito le credenziali, se corrette, l'utente visualizza la finestra principale dalla quale se è di tipo admin visualizza nella barra in alto sia le opzioni utente che la gestione del database, con aggiunta di Utenti e di Articoli. L'admin può ricercare utenti e articoli e modificarli cliccando due volte sopra la riga dell'oggetto su cui intende effettuare le modifiche, e poi modificando i campi dati nella finestra che ne risulta. Nella modifica degli articoli non è possibile modificare il tipo di articolo o l'ID in inventario, poiché in tal caso si tratterebbe proprio di un oggetto diverso e andrebbe creato come nuovo.

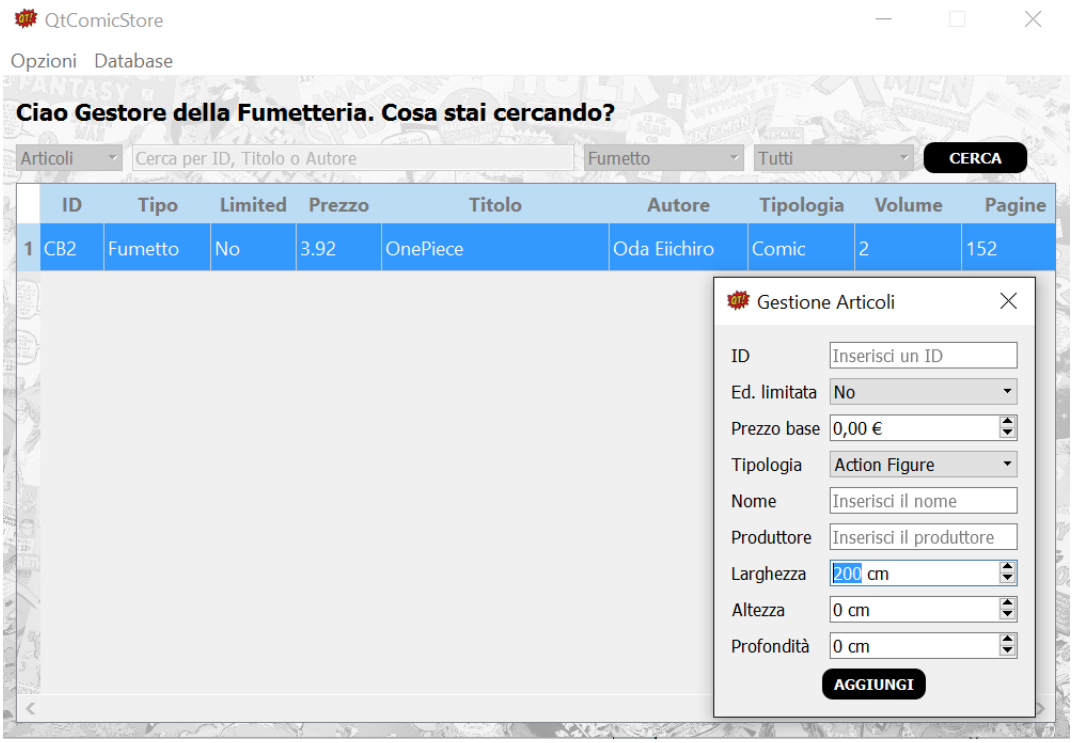


Figura 3: View Gestore

Per gli utenti Gold invece non è presente la gestione del database nella barra in alto, ma nelle opzioni oltre alla modifica dei propri dati utente e al logout è presente la gestione dei preferiti. Per aggiungere un oggetto ai preferiti l'utente Gold (che non può cercare utenti) deve cliccare due volte sopra la riga dell'oggetto di interesse, dopodiché confermare l'aggiunta nell'apposita finestra. Per rimuovere invece un elemento dalla lista dei preferiti deve aprirla tramite la voce nel menu Opzioni e fare doppio click sulla riga di interesse, dopodichè confermare nella finestra che si presenta l'eliminazione. Se vengono eliminati tutti gli elementi la lista dei preferiti viene chiusa in automatico.

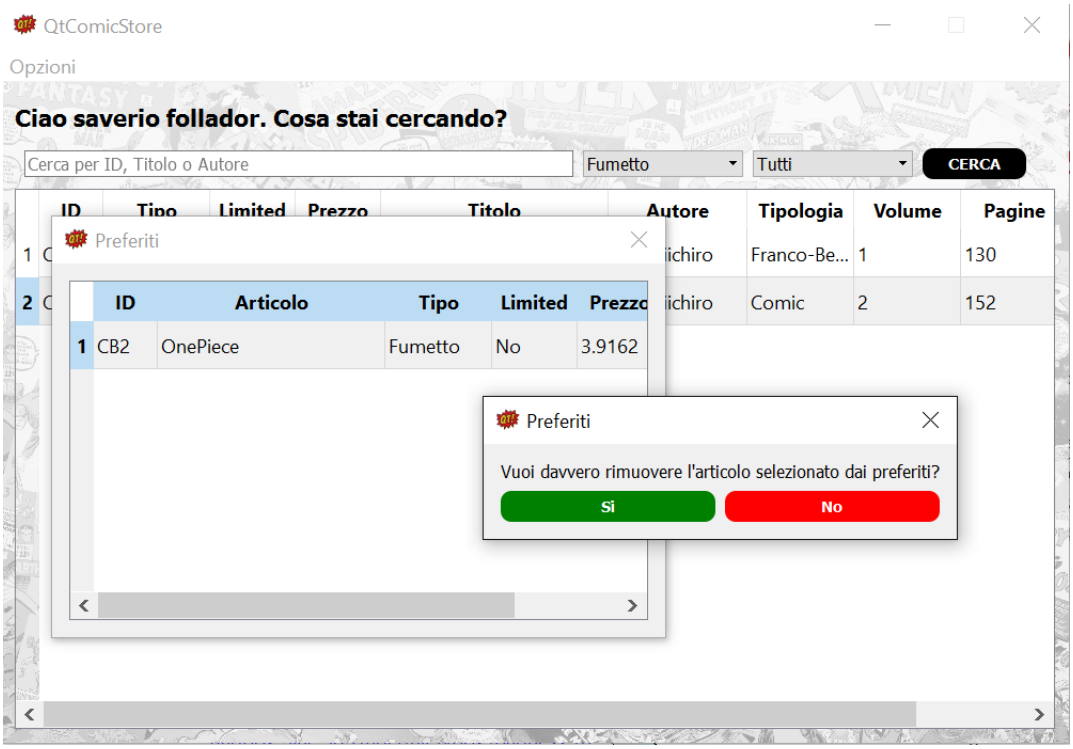


Figura 4: View Gold

Per gli utenti Base è presente solo la ricerca degli articoli e la modifica del proprio profilo nel menu Opzioni, oltre che ovviamente il logout.


Figura 5: View Base

Nell'inserimento dei dati degli utenti vengono fatti alcuni controlli con espressioni regolari al fine di permettere l'inserimento di sole lettere e spazi nei campi **nome**, **cognome**,

per **username** è permesso l'inserimento anche di numeri, mentre per **password** non avvengono controlli.

Nell'inserimento degli articoli vengono fatti alcuni controlli con espressioni regolari, variati in base al tipo di articolo inserito e ai suoi campi e coerente con il tipo di inserimento richiesto. Di particolare importanza è l'identificativo **ID**, che accetta solo sottostringhe di stringhe nella forma `ID0000`, con due lettere rigorosamente maiuscole seguite al massimo da quattro cifre da 0 a 9.

# A Indicazioni conclusive

## A.1 Impegno temporale

- **Progettazione model e GUI**: 8 ore
- **Realizzazione model e GUI**: 43 ore
- **Debugging e Test**: 7 ore
- **Totale**: 58 ore

## A.2 Informazioni tecniche

- **Sistema Operativo**: Windows 10 Pro
- **Compilatore**: MinGW 5.3.0 (32bit)
- **Versione Qt**: Qt 5.7