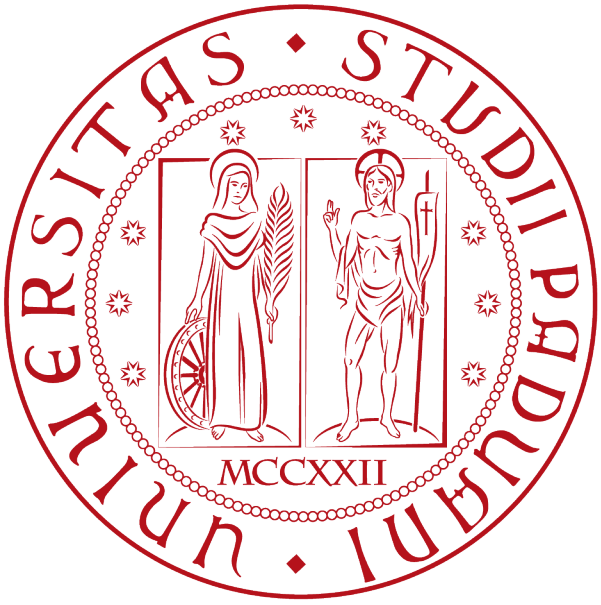

Relazione PAO

MangAnimeList

Kevin Silvestri - Matricola: 1094138 - Anno 2016/2017

Indice

1 Scopo del progetto

Il progetto si prefigge lo scopo di realizzare un programma che simuli un Database collaborativo dedicato ai fumetti e cartoni animati giapponesi, concedendo agli utenti specifici permessi. Gli utenti si dividono in Admin, Standard User e Pro User. In particolare si vogliono fornire le seguenti funzionalità:

1. Admin: aggiunta, rimozione e modifica degli articoli e degli utenti. La creazione degli utenti è riservata esclusivamente all'amministratore, che potrà così definire quali utenti possono dare un contributo e quali invece possono solo avere una funzione consultativa.
2. Standard User: consultazione di tutte le informazioni presenti nel database degli articoli e ricerca migliorata con l'aggiunta di altre due combo box. Lui è l'unico utente che può suddividere gli articoli secondo delle categorie predefinite di manga e di anime.
3. Pro User: consultazione di tutte le informazioni presenti nel database degli articoli e nel database degli utenti. Aggiunta, modifica ed eliminazione solo degli articoli.

1.1 Accesso

Accesso Amministratore:
Username: admin Password: admin
Accesso Standard User:
Username: kevin Password: kevin
Accesso Pro User:
Username pro Password: pro

1.2 Compilazione ed esecuzione

Per compilare il progetto è necessario eseguire il comando qmake MangAnimeList.pro, contenuto all'interno della cartella MangAnimeList. Tale file permetterà la generazione automatica tramite qmake del Makefile, e successivamente lanciare il comando make. Per l'esecuzione invece: ./MangAnimeList
Vengono consegnati anche i file ItemDatabase.xml e UserDatabase.xml come esempi di dati. L'applicazione verrà eseguita ugualmente anche nel caso in cui non dovessero essere presenti i file creando di default un utente amministratore con username admin e password admin.

2 Struttura

2.1 Contenitori

La scelta dei contenitori è ricaduta su QList<T>, un template di classe che rappresenta le liste e QMap<Key,T>, un template di classe che fornisce un dizionario basato su alberi rosso-neri. Entrambe le strutture sono presenti nella libreria di Qt.

2.1.1 QList

La preferenza di una lista è dovuta al fatto che, data la natura di Database del programma, è necessario che le classiche operazioni di inserimento e cancellazione siano più rapide possibili, in questo caso dunque in tempo O(1). Ho usato QList<T> sia per il database degli articoli (item_ database) che rappresenta una semplice lista di puntatori ad articolo, sia per il database degli utenti (user_ database), che rappresenta una lista di puntatori ad utente. Per quanto riguarda la gestione della memoria sono stati definiti dei metodi di rimozione.

2.1.2 QMap

Per quanto riguarda il QMap ho scelto di utilizzarlo per l'utente standard per il salvataggio degli articoli nelle categorie predefinite. Questa struttura mi permette di avere una corrispondenza univoca della Key (Cod. per gli articoli, Status per la categoria). QMap<key,data> permette di associare oggetti di tipo Key ad oggetti di tipo Data. Un oggetto QMap non può contenere più elementi con la stessa chiave, il che va bene per l'uso che devo farne dato che un articolo non può trovarsi contemporaneamente in più categorie predefinite.

2.2 Pattern Model-View-Controller

Per lo sviluppo ho deciso di seguire quanto più possibile una scelta architetturale del pattern MVC, così da separare l'implementazione logica da quella grafica. Nel Model, infatti, vengono gestiti direttamente i dati; nella View vengono visualizzati i dati contenuti nel model fornendo una rappresentazione grafica, occupandosi dell'interazione con gli utenti, così come semplici controlli di inserimento dati; nel Controller invece viene definita la logica di controllo.

3 Gerarchie di tipi

In seguito illustro le principali scelte progettuali effettuate per la realizzazione del modello dei dati del programma:

3.1 Gerarchia degli articoli

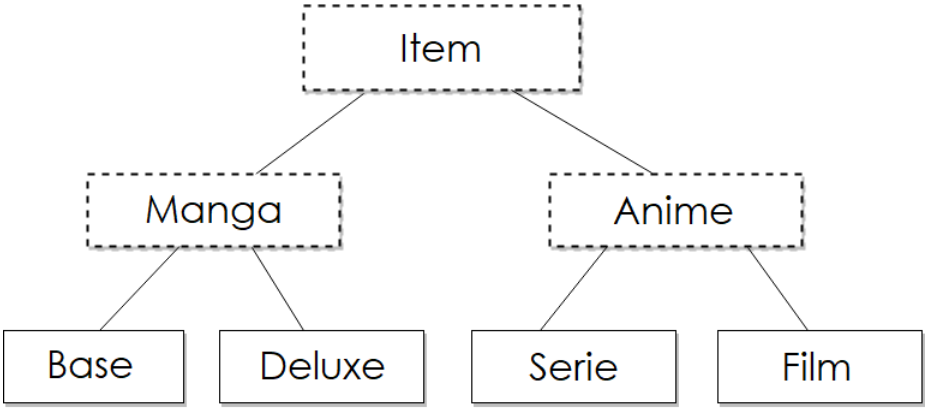

Figura 1: Gerarchia articoli

3.1.1 Classe base astratta Item

Tale gerarchia costituisce l'insieme degli articoli raccolti nel contenitore C. Si parte da un generico articolo nella classe base astratta Item, che è alla base della gerarchia. I campi dati sono:

- **originalName** : Nome originale dell'articolo.
- **englishName** : Nome inglese dell'articolo.
- **author** : Nome dell'autore.
- **releaseDate** : Data di uscita dell'articolo.
- **state** : Stato dell'articolo.
- **genre** : Genere di appartenenza dell'articolo.
- **rating** : Limitazione età del pubblico.

Un articolo viene identificato univocamente da un valore static int che verrà incrementato ogni volta che verrà creato un nuovo oggetto. L'astrazione del concetto di articolo è data dai seguenti metodi:

- virtual QString getTypeItem() const=0;
- virtual void saveItem(QXmlStreamWriter &)=0;
- virtual void loadItem(QXmlStreamReader &)=0;

Nella sezione Uso del Polimorfismo saranno analizzati in dettaglio. I campi dati comuni di ogni articolo vengono caricati con la funzione void loadCommonFieldsItem(QXmlStreamReader&) che viene richiamata nel metodo loadItem delle classi derivate concrete, e scritti nel file .xml con void saveCommonFieldsItem(QXmlStreamWriter&) che viene richiamata dal metodo saveItem delle classi derivate concrete. Successivamente vengono definite le due classi astratte Manga e Anime derivate da Item, che definiscono e distinguono la natura delle sottoclassi.

3.1.2 Classe astratta Manga

La classe astratta Manga ha come campi dati:

- **dealer** : chi ha distribuito il manga.
- **chapters** : numero di capitoli del manga.
- **volumes** : numero di volumi del manga.

3.1.3 Classe Base e Deluxe

Alla classe Manga derivano Base e Deluxe. In queste classi vengono implementati i metodi virtuali puri. Un manga base viene ulteriormente esteso con il campo **reprint** che descrive il numero di ristampa del manga, mentre un manga deluxe rappresenta un particolare cofanetto del manga ed essendo un'edizione speciale, ha dei contenuti speciali:

- **variantCover** : indica se la copertina è in versione standard o nella variante.
- **poster** : indica se ci sono poster aggiuntivi.
- **actionFigure** : indica se nel cofanetto è incluso un modellino.
- **adhesives** : indica se nel cofanetto sono inclusi degli adesivi.

3.1.4 Classe astratta Anime

La classe astratta Anime ha come campi dati:

- **producer** : chi ha prodotto l'anime.
- **studio** : lo studio di produzione.
- **source** : indica da cosa ha preso spunto l'anime.
- **duration** : la durata in minuti dell'anime (degli episodi o del film).

3.1.5 Classe Serie e Film

Arriviamo alle classi concrete Serie e Film; la prima rappresenta una serie di episodi di un anime, perciò avrà due campi dati aggiuntivi che sono **episodes** che è il numero di episodi e **seasons** che è il numero di stagioni. Per quanto riguarda invece Film, che rappresenta un film di un anime, l'unico campo dati è **sceneAfterQueueTitles**, indica se alla fine del film è presente la scena dopo i titoli di coda.

3.2 Gerarchia degli utenti

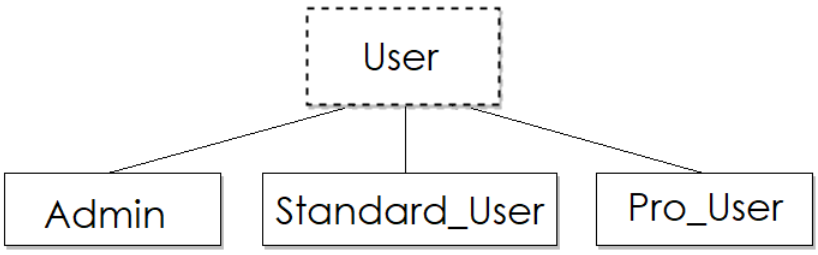

Figura 2: Gerarchia utenti

3.2.1 Classe base astratta User

Per quanto riguarda la caratterizzazione di un generico utente si è deciso di definire i seguenti campi dati di tipo QString:

- **username** : Un utente viene identificato univocamente dal suo username.
- **password** : Password dell'utente.
- **name** : Nome dell'utente.
- **surname** : Cognome dell'utente.

Tale classe è resa astratta dai seguenti metodi polimorfici:

- virtual QString getLabel() const =0;
- virtual bool AlterUsers() const =0;
- virtual bool AlterItems() const =0;
- virtual bool proFunctions() const =0;

I dati comuni di tutti gli Utenti vengono letti e scritti grazie ai metodi:

- virtual void saveCommonFieldsUser(QXmlStreamWriter &);
- virtual void loadCommonFieldsUser(QXmlStreamReader &);
- virtual void saveUser(QXmlStreamWriter &);

Verranno approfondite le funzionalità di questi metodi all'interno della sezione Uso del Polimorfismo.

3.2.2 Classe Standard_ User, Pro_ User e User_ Admin

Nel caso di Pro_ User e User_ Admin non si aggiungono campi dati ulteriori a quelli già ereditati, mentre in Standar_ User viene creato il campo **listItem** di tipo QMap<int,QString> che conterrà gli articoli scelti dall'utente con il relativo status (es. Da leggere). Tutte le classi implementano i metodi virtuali già citati con le dovute differenziazioni di permessi e Standar_ User ovviamente ha nuovi metodi relativi all'estrazione dati dal QMap. La scelta di diversificare le view per tipologia di utenza è stata definita marcando cosa si aspetterebbe di fare un dato utente. L'amministratore ha il compito di gestire (aggiungere/modificare/eliminare) gli articoli e gli utenti. L'utente standard ha la possibilità di registrarsi direttamente dalla schermata di login/registrazione ed ha la possibilità di suddividere gli articoli nel database nelle categorie predefinite per tenere traccia delle sue preferenze (es. vuole trovare subito quelli che si era segnato di leggere). L'utente pro ha la possibilità di gestire (aggiungere/modificare/eliminare) gli articoli e solo di visualizzare gli utenti. Tutti hanno la possibilità di modificare i propri dati utente e hanno una ricerca base. L'utente standard però ha la possibilità di raffinare meglio la ricerca.

4 Uso del polimorfismo

4.1 Uso del polimorfismo nella gerarchia utenti

Nella gerarchia degli utenti vengono usati i seguenti metodi polimorfici:

- virtual QString getLabel() const =0; : metodo che, quando implementato opportunamente nella classe derivata restituisce un QString che identifica la tipologia di appartenenza dell'utente;
- virtual bool AlterUsers() const =0; : metodo che, quando implementato opportunamente nella classe derivata, restituisce un booleano che permette di definire se un dato utente può modificare, aggiungere od eliminare altri utenti.
- virtual bool AlterItems() const =0; : metodo che, quando implementato opportunamente nella classe derivata, restituisce un booleano che permette di definire se un dato utente può modificare, aggiungere od eliminare degli articoli.
- virtual bool proFunctions() const =0; : metodo che una volta implementato restituisce un valore booleano: se true permette di poter cercare e visualizzare gli utenti nel database all'utente pro.
- virtual void saveCommonFieldsUser(QXmlStreamWriter &); : metodo che si occupa del salvataggio su file dei campi dati comuni.
- virtual void saveUser(QXmlStreamWriter &); : metodo utile per la scrittura su file dei campi dati, è importante perché richiama la label specifica dell'utente che si vuole inserire nel database (serve per l'apertura e la chiusura dei tag) con il metodo getLabel() e tramite il metodo saveCommonFieldsUser() salva i campi dati comuni.
- virtual void loadCommonFieldsUser(QXmlStreamReader &); : metodo utile per il caricamento da file.

Oltre all'uso del polimorfismo, nella classe User è stato reso virtuale il distruttore per non creare memory leak. Tale metodo è fondamentale affinché non venga lasciato garbage nella memoria. Alla distruzione di un utente viene invocato il distruttore standard della classe concreta dell'utente specifico.

4.2 Uso del polimorfismo nella gerarchia articoli

In riferimento alla gerarchia articoli vale lo stesso discorso sulla gestione della memoria della gerarchia utenti. In seguito vengono elencati i metodi polimorfici usati nella gerarchia degli articoli:

- virtual QString getTypeItem() const=0; : metodo virtuale puro che, una volta implementato, restituisce una QString contenente il tipo dell'articolo. Viene utilizzato al fine di non usare eccessivamente il dynamic_ cast.
- virtual void saveItem(QXmlStreamWriter &)=0; : metodo virtuale puro che, una volta implementato, si occupa del salvataggio dei dati dell'articolo. Esso è presente in ogni classe concreta derivata da Item e richiama il metodo void saveCommonFieldsItem(QXmlStreamWriter &) per il salvataggio dei campi comuni: englishName, author, releaseDate, state, genre, rating.
- virtual void loadItem(QXmlStreamReader &)=0; : metodo virtuale puro che si occupa del caricamento dei dati dell'articolo, esso verrà implementato in ogni sottoclasse derivata concreta e richiama il metodo void loadCommonFieldsItem(QXmlStreamReader &) per il caricamento dei campi dati comuni: englishName, author, releaseDate, state, genre, rating.

4.3 Uso del polimorfismo nella View

La distruzione dei QWidget viene gestita chiamando QWidget::setAttribute(Qt::WA_ DeleteOnClose), poiché Qt stessa si occupa della gestione della memoria.

5 Manuale utente

Una volta avviato il programma verrà visualizzata la LoginWindow.

Qui ci si può registrare diventando un utente standard oppure accedere con le credenziali già in possesso. Una volta eseguito l'accesso verrà visualizzata la MainWindow.

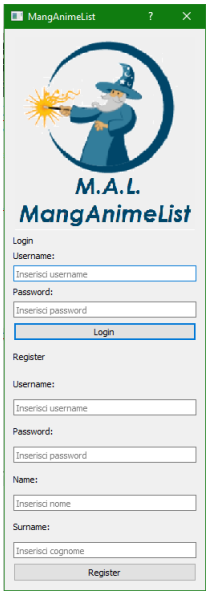

Figura 3: Finestra di login/registrazione

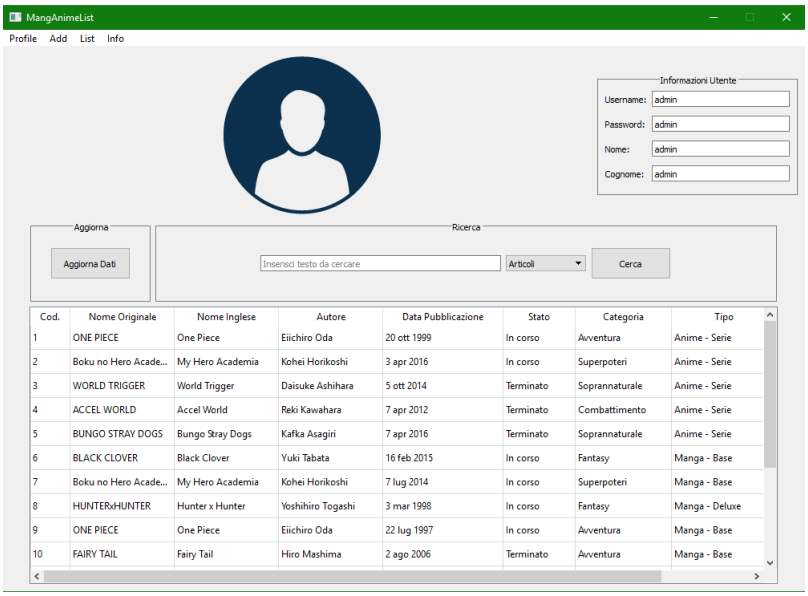

Figura 4: Finestra principale - Accesso con Admin

In questa finestra è disponibile un menù con 4 voci Profile, Add, List e Info. Cliccando sulla prima voce si aprirà un menù a tendina con:

- Settings, in cui sarà possibile modificare le proprie informazioni.
- Log Out, per effettuare il logout.

La seconda voce è Add (che non sarà visibile per l'utente standard) con la possibilità di aggiungere sia un User, sia un Item; per l'utente pro solo aggiunta Item. La terza voce è List dove si possono richiamare diverse liste nella schermata. Ci sono 4 liste:

- All Users: (non visibile all'utente standard) per visualizzare tutti gli utenti.
- All Items: per visualizzare tutti gli articoli.
- All Manga: per visualizzare tutti i manga.
- All Anime: per visualizzare tutti gli anime.

Infine, la quarta e ultima voce è Info, dove cliccando su About M.A.L. apparirà una finestra contenente informazioni sul programma.

Per arrivare alla consultazione/modifica/eliminazione (se permesse) basterà cliccare su List e poi sulla voce che più si avvicina a quello che si cerca oppure utilizzare la barra di ricerca e si apriranno le rispettive tabelle con tutte le informazioni salvate, e successivamente fare doppio click sulla riga della tabella interessata. La ricerca semplice è composta da una barra di testo, una combo box (Utenti/Articoli/Manga/Anime) ed il pulsante Cerca. Vicino alla Ricerca c'è il pulsante Aggiorna che permette di ricaricare i dati mostrati dalla pagina. Andrebbe usato subito dopo un inserimento/modifica/eliminazione di un utente o di un articolo per aggiornare i dati visualizzati.

5.1 Manuale utente Amministratore

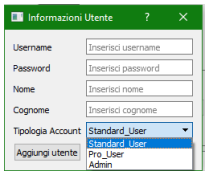

Figura 5: Aggiunta di un utente

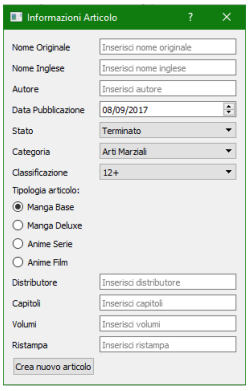

Figura 6: Aggiunta di un articolo

L'admin ha il compito di gestire (inserire/modificare/eliminare) utenti e articoli.

Nota: Se l'amministratore crea un utente, quest'ultimo può effettuare l'accesso e modificare la propria password. Nel momento in cui l'amministratore va a modificare i campi dell'utente nel pannello di modifica, la password verrà oscurata.

5.2 Manuale utente Standard

Per l'utente standard, il cui scopo è quella di consultare il database, sono state implementate funzionalità aggiuntive che sono:

- La ricerca migliorata che permette di ottenere risultati più precisi e specifici grazie a 2 ulteriori combo box (Stato e Categoria).
- La possibilità di suddividere gli articoli in 6 diverse categorie predefinite che permettono all'utente di tenere traccia dei propri manga ed anime (per esempio dei manga che ha già letto).

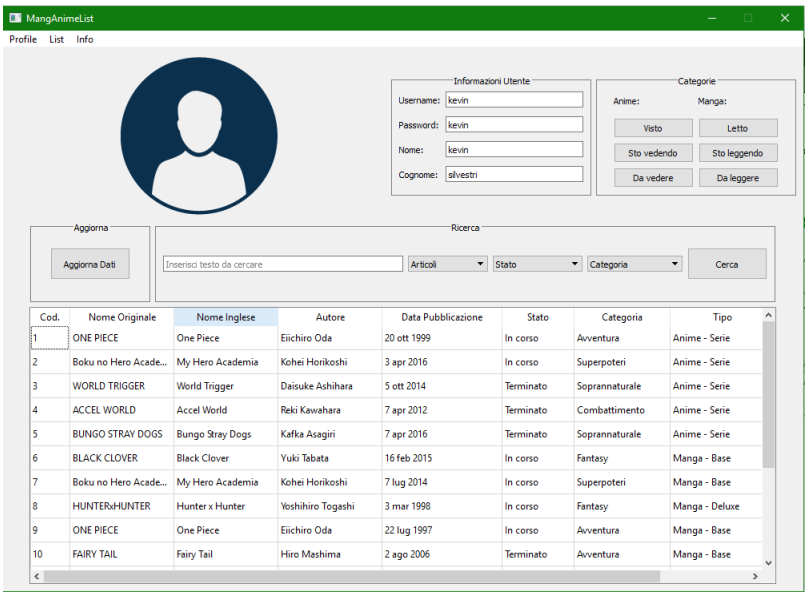

Figura 7: Gerarchia utenti

5.3 Manuale utente Pro

L'utente pro ha il compito di gestire (inserire/modificare/eliminare) gli articoli. Per quanto riguarda la gestione degli utenti può solo consultare la lista e visualizzarne le informazioni.

6 Indicazioni conclusive

6.1 Impegno temporale

Progettazione Modello ed Interfaccia Grafica: 7-8 h
Codifica Modello ed Interfaccia Grafica: circa 52 h
Debugging: 4-5 h

6.2 Informazioni tecniche

Sistema operativo: Windows 10 Home
Versione Qt: Qt Creator 4.2.2 based on Qt 5.8.0
Compilatore: MSVC 2015 32 bit