Relazione Progetto Biblioteca Virtuale

Simone Buccolieri - 2113183

Aprile 2025

1 Introduzione

Il progetto realizzato consiste in un'applicazione desktop per la gestione di una biblioteca multimediale, sviluppata utilizzando il framework Qt in linguaggio C++. L'obiettivo principale è fornire un'interfacica semplice e intuitiva per la gestione completa di varie tipologie di media quali libri, film e musica, consentendo operazioni quali l'aggiunta, la modifica, la visualizzazione dettagliata, la prenotazione e la restituzione dei singoli elementi.

L'applicazione prevede inoltre la gestione degli utenti, che possono registrarsi, autenticarsi, e monitorare direttamente dalla propria area riservata i propri prestiti attivi. È stata posta particolare attenzione alla modularità e all'estensibilità del codice, attraverso l'adozione di tecniche di programmazione orientata agli oggetti e l'impiego intensivo del polimorfismo per gestire le diverse tipologie di media.

L'utilizzo di Qt ha permesso di realizzare un'interfaccia grafica user-friendly e responsive, che rende l'applicazione facilmente utilizzabile anche da utenti meno esperti. Il progetto mira inoltre a semplificare e automatizzare il processo di gestione e persistenza dei dati, attraverso l'utilizzo di file JSON per il salvataggio e il caricamento delle informazioni relative a utenti e media della biblioteca.

Nei capitoli successivi della relazione si descrivono dettagliatamente le scelte progettuali adottate, l'architettura logica dell'applicazione e gli aspetti tecnici più rilevanti dell'implementazione.

2 Descrizione delle classi principali del modello logico

Il progetto **Biblioteca** è strutturato seguendo il paradigma della programmazione orientata agli oggetti, con particolare attenzione all'organizzazione del modello logico. Di seguito viene fornita una panoramica delle classi principali:

2.1 ItemModel

`ItemModel` è la classe base astratta che rappresenta un generico elemento della biblioteca (libro, film, musica). Contiene attributi comuni quali titolo, autore, quantità disponibile, anno di rilascio, genere e icona. Definisce inoltre metodi di accesso e modifica per tali attributi. È la base per l'ereditarietà delle classi derivate.

2.2 BookModel, FilmModel, MusicModel

Queste tre classi derivano da `ItemModel` e rappresentano rispettivamente libri, film e dischi musicali. Ognuna aggiunge attributi specifici:

- `BookModel`: editore, ISBN, numero di pagine, lingua.
- `FilmModel`: regista, durata, valutazione, lingua.
- `MusicModel`: album, durata, etichetta discografica, formato.

Tali classi ridefiniscono metodi dove necessario per gestire correttamente le peculiarità di ogni tipo di media.

2.3 BibliotecaModel

`BibliotecaModel` gestisce l'insieme degli oggetti `ItemModel` presenti nella biblioteca. Fornisce metodi per aggiungere, eliminare, cercare e modificare gli elementi, oltre alla gestione della persistenza tramite salvataggio e caricamento da file JSON.

2.4 UserModel

`UserModel` rappresenta l'insieme degli utenti registrati. Ogni utente è associato a un nome utente, una password e una mappa dei prestiti attivi (associando l'ID dell'elemento alla quantità presa in prestito).

2.5 Controller

- `BibliotecaController`: gestisce la logica di alto livello per la biblioteca, come prenotazioni, restituzioni, modifiche e accesso agli elementi.
- `UserController`: gestisce la registrazione, autenticazione e aggiornamento degli utenti.

Entrambi i controller fungono da intermediari tra il modello logico e l'interfaccia grafica, mantenendo separazione tra logica e presentazione.

2.6 View

- `LoginView`, `RegisterView`, `MainView`, `ShowItem`, `EditItem`, `AccountView`: finestre e schermate grafiche basate su `QWidget`, connesse ai controller per eseguire operazioni specifiche.

Ciascuna view è responsabile della presentazione dei dati e della gestione delle interazioni utente, secondo il pattern MVC (Model-View-Controller).

3 Utilizzo non banale del polimorfismo

Nel progetto *Biblioteca*, il polimorfismo è stato sfruttato in modo non banale principalmente nella gestione degli oggetti **ItemModel** e delle sue sottoclassi **BookModel**, **FilmModel** e **MusicModel**. Questo approccio ha permesso di realizzare funzionalità diverse a seconda del tipo specifico di media gestito, senza dover duplicare codice o appesantire la logica delle classi controller e delle viste.

3.1 Metodi polimorfi

- **Visualizzazione dei dati**: nelle interfacce grafiche, la classe `ShowItem` utilizza `dynamic_cast` per riconoscere a runtime il tipo concreto dell'oggetto `ItemModel*` passato. In base al tipo (libro, film o musica), vengono mostrati campi specifici come ISBN, regista, etichetta discografica, ecc.
- **Salvataggio dei dati**: durante la serializzazione in JSON nella classe `BibliotecaModel`, è implementata una logica polimorfa per salvare attributi differenti a seconda della sottoclasse effettiva di `ItemModel` (ad esempio, `publisher` per i libri o `duration` per film e musica).
- **Modifica degli oggetti**: nella classe `EditItem`, la visualizzazione e la modifica dei campi avviene riconoscendo dinamicamente il tipo concreto dell'item, permettendo così di editare informazioni pertinenti al tipo (ad esempio, numero pagine solo per i libri).

3.2 Valore aggiunto del polimorfismo

L'utilizzo del polimorfismo ha portato numerosi vantaggi progettuali:

- **Estendibilità**: aggiungere nuovi tipi di media in futuro richiederèbbe solo l'implementazione di una nuova sottoclasse di `ItemModel`, senza modificare il codice esistente.
- **Manutenzione facilitata**: il codice è più ordinato, evitando lunghe sequenze di `if` o `switch` sulla tipologia di oggetto.
- **Separazione delle responsabilità**: ogni sottoclasse gestisce i propri dati specifici, rispettando il principio di singola responsabilità (SRP).
- **Efficienza nello sviluppo**: le viste (come `ShowItem` ed `EditItem`) sono state sviluppate una volta sola, sfruttando il comportamento polimorfo per adattarsi ai diversi tipi di oggetti.

4 Metodo di persistenza dei dati

Il progetto *Biblioteca* utilizza il formato JSON per la persistenza dei dati, sfruttando le classi offerte da Qt come `QJsonDocument`, `QJsonObject`, `QJsonArray` e `QFile`. Questa scelta garantisce leggibilità, portabilità e facilità di manipolazione dei dati.

4.1 Salvataggio dei dati

Il salvataggio degli oggetti della biblioteca avviene attraverso la serializzazione delle istanze di `ItemModel` e delle sue sottoclassi (`BookModel`, `FilmModel`, `MusicModel`) in oggetti `QJsonObject`. Ogni oggetto serializzato include attributi comuni (ID, titolo, autore, quantità, anno di rilascio, genere, icona) e attributi specifici in base al tipo di media. Questi oggetti vengono aggregati in un `QJsonArray`, che viene poi scritto su file utilizzando `QJsonDocument`.

4.2 Caricamento dei dati

All'avvio dell'applicazione, i dati vengono caricati leggendo il contenuto dei file JSON. Il contenuto viene interpretato come un `QJsonArray`, e per ogni oggetto JSON viene determinato il tipo di media attraverso il campo `tipologia`. In base a questo, viene istanziata la classe corrispondente (`BookModel`, `FilmModel`, `MusicModel`) e popolata con i dati deserializzati.

4.3 Persistenza degli utenti

Analogamente, le informazioni sugli utenti vengono salvate e caricate utilizzando il formato JSON. Ogni utente è rappresentato da un oggetto JSON contenente il nome utente, la password e una mappa dei prestiti attivi. Questi dati vengono gestiti dalla classe `UserModel`.

4.4 Gestione dei file

L'applicazione offre la possibilità di selezionare i file JSON da utilizzare per la biblioteca e gli utenti attraverso una finestra di dialogo all'avvio. I percorsi dei file selezionati vengono memorizzati in variabili globali, rendendoli accessibili a tutte le componenti dell'applicazione che necessitano di accedere o modificare i dati persistenti.

5 Funzionalità aggiuntive implementate

Il progetto include diverse funzionalità extra rispetto alla gestione base della biblioteca:

- **Autenticazione utente:** È presente un sistema di login e registrazione, che permette di accedere in modo sicuro e riservato. Le credenziali (username e password) vengono salvate in un file JSON e sono riutilizzabili ad ogni avvio del programma. Ogni utente ha inoltre la possibilità di registrarsi autonomamente tramite un'apposita interfaccia.
- **Controllo dei privilegi:** Solo utenti autorizzati (ad esempio personale amministrativo) possono modificare o eliminare elementi presenti nella biblioteca. Gli utenti normali possono solamente visualizzare, prenotare e restituire i media.
- **Gestione avanzata dei prestiti:** Tramite una sezione dedicata nell'Account View, gli utenti possono monitorare i propri prestiti correnti e restituire i libri direttamente dall'interfaccia, semplificando l'interazione con il sistema.
- **Persistenza dei dati:** Ogni modifica (nuove registrazioni, modifiche ai prestiti, aggiornamenti agli item) viene salvata automaticamente in file JSON, assicurando la conservazione dei dati anche tra sessioni diverse.

6 Rendicontazione delle ore

Il progetto è stato inizialmente pianificato prevedendo un totale di circa 40 ore di lavoro. Tuttavia, durante lo sviluppo effettivo, il tempo necessario per la realizzazione delle funzionalità, la gestione della persistenza dei dati e l'implementazione dell'interfaccia grafica si è rivelato superiore rispetto alle stime iniziali.

Il totale delle ore effettivamente svolte è stato di circa 50 ore, suddivise nelle seguenti attività principali:

- Analisi dei requisiti e progettazione dell'architettura: 8 ore
- Implementazione delle classi modello (ItemModel e derivate): 10 ore
- Realizzazione dell'interfaccia grafica (Qt Widgets): 12 ore
- Gestione della persistenza dei dati tramite file JSON: 8 ore
- Implementazione dell'autenticazione e gestione utenti: 7 ore
- Testing, debugging e rifinitura del progetto: 5 ore

Il lavoro si è concentrato in particolare sull'integrazione del sistema di login, sulla gestione dei prestiti personalizzati per utente e sulla realizzazione di interfacce user-friendly attraverso Qt, che hanno richiesto più tempo del previsto.

7 Conclusioni

Il progetto sviluppato ha consentito di mettere in pratica molteplici concetti chiave della programmazione orientata agli oggetti, tra cui l'uso avanzato del polimorfismo, la gestione della persistenza dei dati e l'integrazione di una interfaccia grafica tramite Qt.

L'implementazione del sistema di autenticazione, della gestione dei prestiti personalizzati e della possibilità di modificare e visualizzare gli elementi della biblioteca ha fornito funzionalità complete e realistiche, simili a quelle di un'applicazione gestionale reale.

Inoltre, l'approfondimento delle tecniche di caricamento dinamico dei file JSON ha permesso di creare un sistema riutilizzabile e scalabile, migliorando la qualità del progetto. L'utilizzo del polimorfismo in maniera non banale, attraverso metodi come `display()` ridefiniti per ogni sottoclasse di `ItemModel`, ha dimostrato l'efficacia di queste tecniche nella riduzione della complessità del codice.

Nonostante le difficoltà incontrate durante lo sviluppo, il progetto ha raggiunto tutti gli obiettivi prefissati, offrendo un'applicazione stabile, funzionale e facilmente estendibile. L'esperienza complessiva si è rivelata altamente formativa, sia dal punto di vista tecnico sia metodologico.