UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Progetto Programmazione ad Oggetti

"BananoTECH-a"

Indice:

a. Dati del Gruppo

Studente: Giacomo Giora **Numero di matricola:** 2101094
Studente: Giacomo Nalotto **Numero di matricola:** 2101094

b. Introduzione

Il progetto consiste nello sviluppo di un'applicazione per la gestione di una biblioteca digitale. L'obiettivo principale è fornire un sistema per la catalogazione, la ricerca e la gestione di diversi tipi di media, quali film, libri, giochi da tavolo, riviste e vinili.

L'applicazione è stata progettata pensando all'utilizzatore come a un gestore di biblioteca (o strutture simili) e consente agli utenti autenticati di aggiungere nuovi media al catalogo, visualizzare i dettagli di ciascun elemento, modificare le informazioni esistenti e gestire la disponibilità delle copie. È stata posta attenzione alla modularità del codice, separando quindi il modello logico dall'interfaccia utente (GUI). Questo approccio facilita la manutenzione e un eventuale estensione del progetto, consentendo l'introduzione di nuovi tipi di media o di diverse strategie di persistenza senza alterare significativamente le altre componenti del sistema.

L'applicazione è sviluppata in C++ utilizzando il framework Qt per l'interfaccia grafica, garantendo una buona portabilità e un'esperienza utente intuitiva. La persistenza dei dati è gestita attraverso l'implementazione di strategie di I/O che supportano sia il formato JSON che XML.

Dopo aver inserito l'username ("admin") e la password ("admin") l'applicazione permette di selezionare il file da cui prelevare i dati del catalogo della biblioteca o, al contrario, di crearne uno nuovo. Una volta selezionata l'opzione si apre la MainWindow; nella parte centrale della finestra compare la lista completa di tutti i media appartenenti al catalogo della biblioteca, ognuno di questi può essere selezionato per avere una piccola anteprima che compare sul lato destro della schermata: questa presenta immagini di copertina e alcuni dati riassuntivi dei media (titolo, autore, anno, rating).
Inoltre nella finestra sono situati i pulsanti che permettono le principali funzionalità:
Salva: se al momento del login si seleziona un file già esistente esso viene sovrascritto con le nuove modifiche apportate
Salva come: selezionando questa opzione si viene indirizzati ad una finestra che permette di inserire il nome del file e la sua directory
Aggiungi: attraverso questo tasto l'utente aggiunge un nuovo media all'interno del catalogo
Modifica: permette di modificare i dati del media selezionato nella lista centrale dei media
Elimina: viene rimosso il media selezionato dal catalogo
Ricerca: barra testuale che permette di cercare un media per titolo all'interno del catalogo
Alla lista di media inoltre possono essere applicati dei filtri che appaiono sul lato sinistro della schermata, per attuare i filtri è necessario schiacciare il tasto **applica filtri** e per rimuoverli è presente l'apposito tasto **rimuovi filtri** (che rimuove eventualmente anche il testo inserito nella barra di ricerca)

c. Classi Principali del Modello Logico

Nel modello logico del progetto sono presenti 7 classi, di cui 1 astratta dalla quale sono derivate 5 classi per rappresentare i diversi tipi di media, più una classe che implementa le funzionalità di

gestione della biblioteca. Le classi principali sono le seguenti:

Media

La classe Media è la classe base astratta che rappresenta un generico elemento multimediale all'interno della biblioteca. In essa sono presenti le proprietà comuni a tutti i tipi di media, ovvero Id univoco per ogni media, Titolo, Autore, Genere, Lingua, Immagine, Disponibilità, Numero_copie, In_prestito, Collocazione e Rating.

La classe Media definisce anche metodi virtuali puri toJson e toXml per la serializzazione dei dati, che vengono implementati dalle classi derivate per gestire la conversione a Json o Xml di ciascun tipo di media.

Le seguenti classi ereditano da Media e aggiungono proprietà e comportamenti specifici per ciascun tipo di media:

- **Film** : Rappresenta un film. Aggiunge le proprietà come durata, cast.
- **Libro** : Rappresenta un libro. Include le proprietà isbn, editore e npagine (numero di pagine).
- **GiocoDaTavolo** : Rappresenta un gioco da tavolo. Contiene gli attributi ngiocatori (numero massimo di giocatori), durata (stima della durata in minuti), etaMinima (età minima consigliata) ed editore.
- **Rivista** : Rappresenta una rivista. Ha gli attributi editore, n_pagine, data_pubb (data di pubblicazione), e periodicità.
- **Vinile** : Rappresenta un disco in vinile. Include nTracce (numero di tracce) e durata.

Ogni classe derivata implementa i metodi toJson e toXml per serializzare le proprie proprietà specifiche, oltre a quelle ereditate dalla classe Media .

Biblioteca

La classe Biblioteca è il contenitore principale che gestisce una collezione di oggetti Media. Essa fornisce le funzionalità per aggiungere, rimuovere, cercare e modificare i media presenti nella biblioteca. Contiene un vettore di puntatori a Media , consentendo di gestire polimorficamente i diversi tipi di media attraverso un'unica interfaccia,e vari metodi che implementano le funzionalità di:

- Aggiunta e rimozione di media.
- Ricerca di media
- Gestione della disponibilità delle copie (prestito e restituzione).

IOStrategy

La classe IOStrategy è un'interfaccia astratta che permette di operare in input/output. Essa possiede due metodi virtuali puri:

- salvaSuFile : Salva lo stato della biblioteca su un file.
- caricaDaFile: Carica lo stato della biblioteca da un file.

Questo design pattern Strategy consente di cambiare facilmente il meccanismo di persistenza dei dati senza modificare la logica principale della classe Biblioteca .

Il progetto implementa due concrete strategie di I/O:

- **JsonIO** : Implementa l'interfaccia IOStrategy per la persistenza dei dati in formato JSON. I metodi mediaToJson e jsonToMedia sono responsabili della conversione tra oggetti Media (e le loro classi derivate) e QJsonObject .
- **XmlIO** : Implementa l'interfaccia IOStrategy per la persistenza dei dati in formato XML. I metodi mediaToXml e xmlToMedia gestiscono la conversione tra oggetti Media e QDomElement .

d. Gestione polimorfismo

L'applicazione del polimorfismo si manifesta principalmente in due aree chiave: la gestione dei diversi tipi di Media e la strategia di persistenza dei dati.

Polimorfismo nella Gerarchia Media

La classe base Media dichiara due metodi virtuali puri: toJson e toXml. Questi metodi sono intrinsecamente polimorfi e rappresentano un esempio significativo di polimorfismo, obbligando le classi derivate a fornire la propria implementazione specifica. Tali metodi permettono:

1. **Serializzazione dati specifici**: ogni classe derivata (Film , Libro , GiocoDaTavolo , Rivista , Vinile) ha attributi unici che devono essere serializzati in modo appropriato. Implementando toJson e toXml in ciascuna sottoclasse, è possibile gestire la serializzazione di questi attributi specifici in modo coerente con il formato JSON o XML, senza che la classe Biblioteca debba conoscere i dettagli interni di ogni tipo di media.

2. **Gestione unificata nella Biblioteca** : la classe Biblioteca possiede una collezione di puntatori a Media. Quando la Biblioteca deve salvare i dati, può iterare su questa collezione e chiamare media->toJson(jsonObj) o media->toXml(elemento, doc) su ciascun oggetto indipendentemente dal tipo del media. Grazie al polimorfismo, la chiamata al metodo corretto (quello della classe derivata specifica) viene risolta a runtime. Questo rende il sistema facilmente estensibile a nuovi tipi di media in futuro, senza la necessità di modificare il codice esistente della Biblioteca .

Polimorfismo nella Strategia di I/O

Un altro esempio non banale di polimorfismo si trova nell'implementazione del pattern Strategy per la persistenza dei dati. La classe astratta IOStrategy definisce l'interfaccia per le operazioni di salvataggio e caricamento tramite i metodi salvaSuFile e caricaDaFile.

Le classi concrete JsonIO e XmlIO implementano queste interfacce, permettendo:

1. **Flessibilità nella Scelta del Formato**: l'applicazione può scegliere dinamicamente la strategia di I/O da utilizzare (JSON o XML) a runtime, senza modificare il codice che invoca le operazioni di salvataggio/caricamento. La classe Biblioteca non ha bisogno di sapere se sta salvando in JSON o XML, le basta un puntatore a IOStrategy per delegare l'operazione.

2. **Estensibilità**: se in futuro si volesse aggiungere un nuovo formato di persistenza basterebbe creare una nuova classe che eredita da IOStrategy e implementa i metodi salvaSuFile e caricaDaFile . Non sarebbe necessario modificare alcuna parte del codice esistente della Biblioteca o delle classi Media .

e. Persistenza dei Dati

La persistenza dei dati è stata implementata adottando il design pattern Strategy, che permette di astrarre il meccanismo di salvataggio e caricamento dei dati dalla logica principale della Biblioteca. Questo approccio garantisce che l'applicazione possa supportare diversi formati di archiviazione senza modificare il codice che gestisce la collezione di media. In questo caso sono stati implementati i formati Json e Xml per salvataggio e caricamento dei media, nelle rispettive classi elencate nella sezione “Classi principali del modello logico”.

f. Funzionalità aggiuntive

Oltre alle funzionalità di base di catalogazione e persistenza dei media, sono incluse diverse funzionalità aggiuntive che migliorano l'usabilità e l'interazione con l'applicazione.

Interfaccia Utente Grafica (GUI)

L'applicazione è dotata di un'interfaccia grafica sviluppata con il framework Qt. La GUI è strutturata in diverse pagine e widget per facilitare la navigazione e la gestione dei media:

- **Pagina di Login (`LoginPage.h` , `LoginPage.cpp`)**: Permette l'autenticazione dell'utente, garantendo un accesso controllato all'applicazione.
- **Pagina Principale (`MainPage.h` , `MainPage.cpp`)**: La schermata principale mostra l’elenco dei media presenti nella biblioteca, include funzionalità di ricerca e filtro per aiutare l'utente a trovare rapidamente i media desiderati, e alla selezione di un elemento dalla lista permette di visualizzare rapidamente le informazioni principali di tale media.
- **Pagine di Aggiunta/Modifica (`AddPage.h` , `AddPage.cpp` , `ModifyPage.h` , `ModifyPage.cpp`)**: Form dedicati per l'inserimento di nuovi media e la modifica di quelli esistenti. Queste pagine si adattano dinamicamente al tipo di media selezionato, presentando i campi specifici per Film, Libro, GiocoDaTavolo, Rivista e Vinile.
- **Pagina Approfondimento(`DetailsPage.h` , `DetailsPage.cpp`)**: Visualizza le informazioni dettagliate di un singolo media, inclusi tutti gli attributi specifici del suo tipo. Inoltre, include le opzioni per la gestione delle copie (prestito/restituzione.
- **Widget Specifici per Media (`Widgets/FilmWidget.h` , `Widgets/LibroWidget.h` , ecc.)**: Componenti riutilizzabili che incapsulano la logica di visualizzazione e input per i campi specifici di ciascun tipo di media. Abbiamo utilizzato questa versione modulare per semplificare la gestione dell'interfaccia.

Gestione delle Immagini

L'applicazione supporta l'associazione di un'immagine di copertina a ciascun media. Le immagini vengono caricate e visualizzate all'interno dell'interfaccia utente, migliorando l'aspetto visivo e l'identificazione rapida dei media. La gestione dei percorsi delle immagini è integrata nel processo di serializzazione/deserializzazione, assicurando che le immagini siano correttamente associate ai media anche dopo il salvataggio e il caricamento dei dati.

Gestione delle Copie e Disponibilità

Il sistema tiene traccia del numero totale di copie di ciascun media (`numero_copie`) e del numero di copie attualmente in prestito (`in_prestito`). Questo permette di calcolare la `disponibilita` di un media e di gestire le operazioni di prestito e restituzione.

Validazione Input

Le pagine di aggiunta e modifica dei media includono meccanismi di validazione dell'input per garantire che i dati inseriti dall'utente siano corretti e coerenti. Ad esempio, se viene inserito un media con titolo, autore e anno uguali a quelli di un media già esistente in biblioteca, compare un avviso che notifica la possibile duplicazione. Tale avviso è presente anche nel caso in cui si tenta di modificare un media già esistente inserendo titolo, autore e anno uguali a quelle di un altro media.

g. Rendicontazione delle ore

	Ore previste	Ore effettive
Realizzazione disegno iniziale	1 ora 30 minuti	2 ore
Pianificazione classi	1 ora	2 ore
Stesura codice e test del modello logico	10 ore	12 ore
Implementazione persistenza dei dati (JSon e XML)	5 ore	(contate in stesura modello logico)
Pianificazione classi per GUI	1 ora 30 minuti	-
Stesura codice e test GUI	17 ore	30 ore
Correzione bug	2 ore	15 ore
Stesura relazione	2 ore	2 ore
TOTALE	40 ORE	63 ORE

Motivazioni del superamento delle ore

Fino alla fase di stesura del codice per il modello logico compresa siamo riusciti a stare quasi perfettamente nei tempi prestabiliti prima dell'inizio del progetto, grazie soprattutto alla buona pianificazione effettuata. Invece, non avendo pianificato in anticipo le classi per l'interfaccia grafica (causa inesperienza e non conoscenza del framework Qt) ci siamo ritrovati a progredire molto più lentamente, sia nella creazione effettiva delle classi, a causa di ripensamenti o cambiamenti strutturali, sia soprattutto nella correzione dei bug derivanti dalle varie funzionalità implementate.

h. Suddivisione delle Attività tra i Membri del Gruppo

Abbiamo avuto modo di lavorare spesso insieme in presenza in modo tale da riuscire a confrontarsi in tempo reale sulle funzioni da implementare e sul codice da scrivere in generale. Per questo motivo non c'è stata una netta suddivisione delle attività tra i due membri, ma piuttosto un continuo scambio di opinioni e libertà nel gestire il lavoro sul progetto. Per fare ciò abbiamo fatto uso di GitHub per tracciare i file del progetto e la lista di cose da fare tramite il file Readme. Inoltre grazie a questo approccio è stato possibile per ognuno di noi lavorare in locale e, in caso di bug importanti, riportare il codice ad una versione precedente stabile.

i. Sviluppi futuri

L'applicazione si potrebbe prestare a eventuali sviluppi futuri e miglioramenti, come per esempio:

- gestione di utenti con profili personalizzati e storico prestiti
- integrazione con database esterni

- possibilità di richiedere un media a un’altra biblioteca
- funzionalità di ricerca avanzate