WhiskerWatcher

Progetto di Programmazione ad Oggetti

Riccardo Martinello
2009086

Anno Accademico 2024/2025

Indice

1 Introduzione al progetto

Il mio progetto WhiskerShelf consiste in un'interfaccia di gestione e visualizzazione di contenuti multimediali, ispirata ai moderni software di catalogazione. Questa applicazione permette di organizzare e tenere traccia di diversi tipi di media, con l'obiettivo di fornire una gestione centralizzata e intuitiva della propria collezione multimediale.

1.1 Tipologie di media

Il sistema supporta quattro tipi di media:

- **Articoli**: Permette di catalogare articoli scientifici o giornalistici, includendo informazioni come il nome del journal, il volume, il DOI e il numero di pagine.
- **Audio**: Gestisce contenuti audio come musica o podcast, tracciando durata, formato e album di appartenenza.
- **Libri**: Cataloga libri con relativi dettagli come ISBN, numero di pagine, editore e genere.
- **Film**: Archivia film e contenuti video, registrando informazioni come casa di produzione, durata, genere e budget.

1.2 Gestione dei media

Nell'applicazione, è possibile aggiungere, modificare o eliminare i media. Sul pannello principale sono elencati tutti i contenuti disponibili, permettendo una visione d'insieme della collezione. In alto si trova una barra di ricerca dei media, che consente di filtrare rapidamente i contenuti, mentre nel pannello è presente un pulsante per aggiungerne di nuovi, accessibile anche tramite shortcut Ctrl+A.
Selezionando un media specifico, si apre un pannello che mostra tutti i dettagli relativi all'elemento selezionato, inclusa l'immagine di copertina. Questo pannello include anche i pulsanti per modificare o eliminare il media, accessibili rispettivamente tramite le shortcut Ctrl+E e Ctrl+D.

1.3 Visualizzazione e organizzazione

Ogni media viene rappresentato con una card che mostra l'immagine di copertina (se presente) e le informazioni principali. Le immagini di copertina vengono automaticamente gestite dal sistema, che le copia in una cartella dedicata del progetto e mantiene i riferimenti usando percorsi relativi.

1.4 Barra di stato e controlli di gestione

In basso, una barra di stato fornisce feedback sull'azione in corso, messaggi di errore e raccomandazioni. Questo strumento è essenziale per garantire che l'utente sia sempre informato sulle operazioni effettuate e sui possibili problemi, come ad esempio la presenza di modifiche non salvate o errori durante il caricamento delle immagini di copertina.

Al di sopra della status bar si trovano tre pulsanti essenziali per la gestione della collezione di media:

- **Apri**: Per caricare una collezione di media da un file JSON esistente. Se ci sono modifiche non salvate, l'applicazione chiederà conferma prima di procedere. Shortcut Ctrl+O.
- **Salva**: Per salvare le modifiche alla collezione corrente nel file JSON attualmente aperto. Il pulsante viene automaticamente abilitato quando ci sono modifiche non salvate. Shortcut Ctrl+S.
- **Salva con nome**: Per salvare la collezione in un nuovo file JSON, permettendo di specificare un nome file diverso. Questo pulsante è disponibile solo quando ci sono media nella collezione. Shortcut Ctrl+Shift+S.

2 Descrizione del modello

2.1 Introduzione

Il modello logico del sistema è composto da due componenti principali: la gestione dei media, che include tutte le operazioni CRUD (creazione, lettura, aggiornamento e cancellazione, oltre alla ricerca), e la gestione delle risorse associate come le immagini di copertina.

2.2 Gestione dei Media

La gestione dei media comprende sia le classi che descrivono i media stessi, illustrate nel diagramma mostrato in Figura 1, sia diverse classi di servizio per convertire i media in formato JSON e viceversa, e per salvarli su file. Per queste operazioni, vengono utilizzati gli strumenti offerti da Qt, in particolare `QJsonObject`.

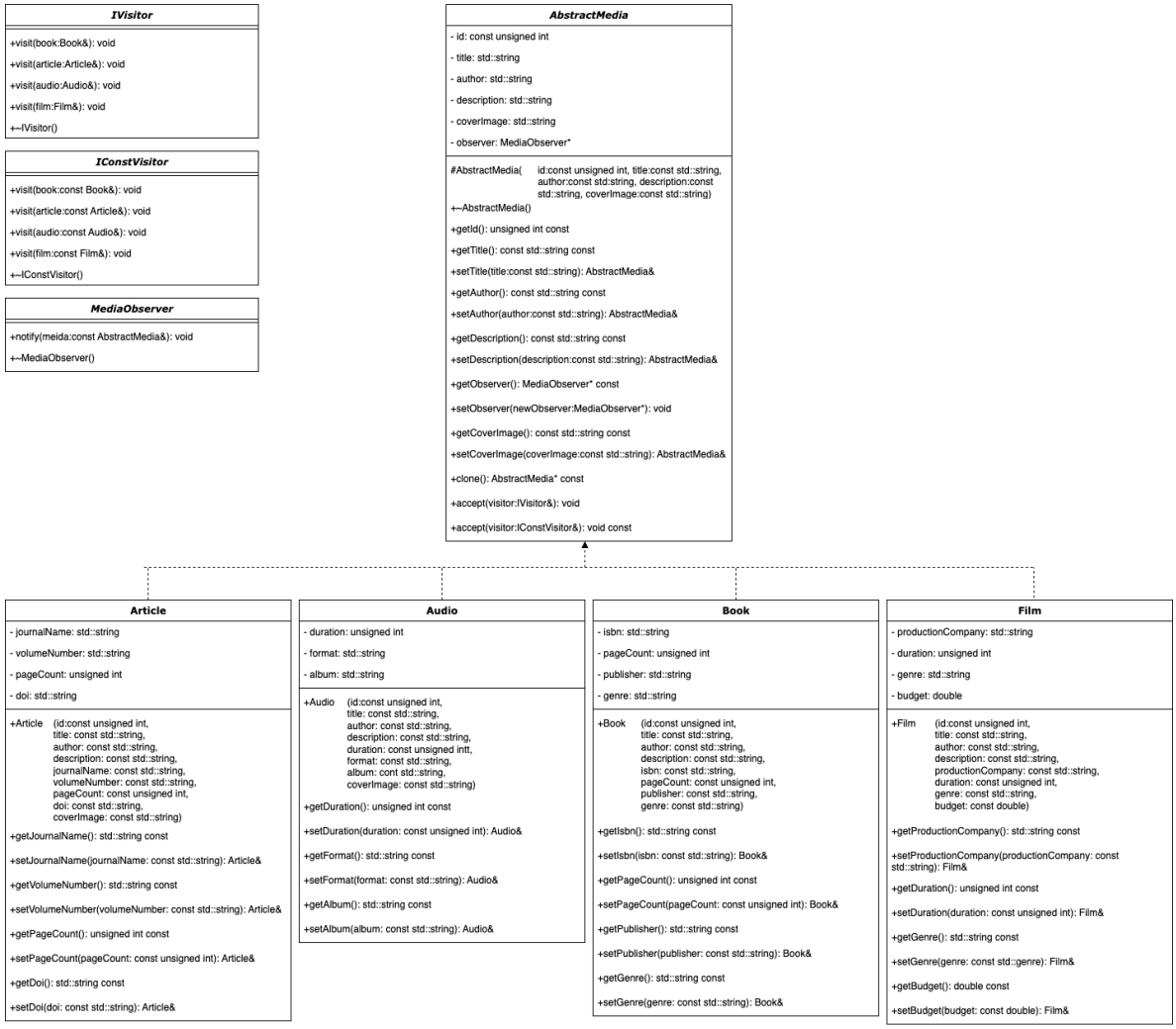

Figura 1: Diagramma delle Classi del Modello

Alla base del modello c'è una classe astratta `AbstractMedia`, che contiene le informazioni comuni a tutti i media, come l'identificatore univoco, il titolo, la descrizione, l'autore e il percorso dell'immagine di copertina (oltre a un puntatore a `MediaObserver`). Sono implementati i metodi getter e setter per gestire queste informazioni.

Successivamente, ci sono quattro classi concrete che derivano da `AbstractMedia`:

- `Article`: rappresenta un articolo scientifico o giornalistico.

- `Audio`: rappresenta un contenuto audio.
- `Book`: rappresenta un libro.
- `Film`: rappresenta un film o contenuto video.

Ogni classe ha attributi specifici per il tipo di media, come il DOI e il nome del journal per gli articoli, la durata e il formato per gli audio, l'ISBN e l'editore per i libri, o la casa di produzione e il budget per i film.

2.3 Operazioni CRUD e Ricerca

Le operazioni CRUD e di ricerca sono implementate nelle classi dell'interfaccia grafica associate ai pulsanti che controllano queste operazioni. L'aggiunta di un media utilizza una finestra di dialogo personalizzata che permette di inserire i principali campi dati comuni (tipo di media, titolo, descrizione, autore) e i campi specifici per il tipo di media selezionato. È inoltre possibile selezionare un'immagine di copertina, che viene automaticamente copiata nella cartella del progetto.

La modifica di un media utilizza una finestra di dialogo simile, che precompila i campi con i valori esistenti, consentendo all'utente di apportare le modifiche necessarie. Entrambe le finestre di dialogo includono meccanismi di controllo dell'input specifici per ciascun campo di testo e validazione dei dati numerici. L'eliminazione di un media è una semplice operazione di rimozione dal vettore dei media, preceduta da una finestra di conferma per evitare cancellazioni accidentali.

Per la ricerca, è stato sviluppato un algoritmo che calcola la sottostringa comune più lunga tra i titoli dei media e la stringa di ricerca. La pertinenza è determinata dalla lunghezza di questa sottostringa. I risultati della ricerca vengono memorizzati in una `QMultiMap`, ordinati per pertinenza. Successivamente, un ciclo attraverso la mappa aggiorna i widget del pannello dei media con i risultati della ricerca.

2.4 Generazione delle Immagini

Il sistema include una gestione sofisticata delle immagini di copertina. Quando un'immagine viene selezionata durante la creazione o modifica di un media, viene:

- Copiata nella cartella `images` del progetto
- Rinominata con un timestamp per evitare conflitti
- Salvata con un percorso relativo nel media
- Ridimensionata automaticamente per la visualizzazione nelle card e nei dettagli

3 Polimorfismo

L'uso principale del polimorfismo nel nostro sistema si riferisce alla gestione dei diversi tipi di media attraverso la gerarchia `AbstractMedia` e l'implementazione dei design pattern Visitor e Observer.

Il pattern Visitor viene utilizzato in particolare per:

- La serializzazione JSON dei media, attraverso `JsonVisitor` che permette di convertire i diversi tipi di media in formato JSON e viceversa
- La creazione delle etichette specifiche per tipo nel widget dei dettagli, gestita da `AboveImageLabelsCreatorVisitor`
- L'impostazione delle icone e dei tipi nei widget dei media, attraverso `SetTypeAndIconOfMediaWidgetVisitor`

Il pattern Observer viene implementato attraverso `MediaObserver`, che permette ai widget dell'interfaccia di essere notificati quando:

- Un media viene modificato, per aggiornare la visualizzazione
- Un'immagine di copertina viene cambiata, per aggiornare le preview

- Il contenuto del media viene alterato, per abilitare il salvataggio

La combinazione di questi pattern permette di mantenere una netta separazione tra il modello dei dati (`AbstractMedia` e le sue sottoclassi) e la loro visualizzazione, consentendo di gestire in modo polimorfico sia le operazioni sui dati che l'aggiornamento dell'interfaccia.

Ho implementato diversi visitor specializzati per gestire il polimorfismo nella persistenza dei dati in file JSON e per la creazione delle etichette nel widget dei dettagli e nel pannello dei media. Ogni visitor implementa metodi specifici per ciascun tipo di media, permettendo di personalizzare il comportamento in base al tipo concreto.

Segnalo anche la creazione della classe astratta `AbstractDialogueWindow`, che fornisce un'interfaccia comune per le finestre di dialogo di creazione e modifica dei media.

4 Persistenza dei Dati

Per garantire la persistenza dei dati, utilizzo il formato JSON. Tutte le informazioni della collezione sono memorizzate in un unico file che contiene un array di oggetti media. Questi oggetti sono principalmente semplici coppie chiave-valore, e la serializzazione delle sottoclassi è gestita aggiungendo un attributo `type`. La gestione del file JSON introduce un ulteriore livello di gestione dei dati, separato dal vettore presente nella `MainWindow`.

Nella classe `JsonRepository`, viene creata una `std::map` che effettua una copia profonda (clonazione) dei media presenti nel vettore e li inserisce nel file JSON. Questa classe è anche responsabile della lettura dei dati dal file JSON, clonandoli nuovamente per inserirli nel vettore della `MainWindow`.

Un esempio della struttura del file JSON è fornito insieme al codice, contenente un media per ciascuna tipologia, per illustrare le diverse strutture in modo chiaro.

5 Funzionalità Implementate

Le funzionalità implementate possono essere suddivise in due categorie principali: funzionali ed estetiche.

5.1 Funzionalità Funzionali

- Gestione di quattro tipi diversi di media con campi specifici
- Gestione automatica delle immagini di copertina
- Conversione e salvataggio dei dati in formato JSON
- Sistema di ricerca interattivo
- Validazione dei campi in input

5.2 Funzionalità Estetiche e dell'Interfaccia Utente

- Utilizzo di icone nella toolbar per le operazioni principali
- Layout responsive con pannello principale e area dettagli
- Barra di stato per feedback all'utente
- Scorciatoie da tastiera intuitive (Ctrl+A per "Add", Ctrl+E per "Edit", Ctrl+D per "Delete", Ctrl+O per "Open", Ctrl+S per "Save")
- Preview delle immagini di copertina con ridimensionamento automatico
- Verifica delle modifiche non salvate prima della chiusura
- Aggiornamento in tempo reale dei risultati di ricerca
- Card interattive per la visualizzazione dei media
- Evidenziazione del media selezionato

6 Rendiconto delle ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	5	6
Sviluppo del codice del modello	15	18
Studio del framework Qt	8	10
Sviluppo del codice della GUI	15	20
Test e debug	4	8
Stesura della relazione	3	4
Totale	**50**	**66**

6.1 Resoconto delle Attività

Le mie stime iniziali erano in linea con le 50 ore previste, ma il tempo effettivo di sviluppo è stato leggermente superiore a causa di alcune sfide incontrate durante l'implementazione.

La prima fase del progetto ha riguardato la creazione di una gerarchia delle classi per i diversi tipi di media e la progettazione del design dell'applicazione utilizzando Figma. Questo mi ha permesso di avere una chiara visione delle funzionalità da implementare e dell'esperienza utente desiderata.

La complessità di alcune funzionalità, come la gestione delle immagini di copertina e il sistema di ricerca interattivo, ha richiesto particolare attenzione nella fase di sviluppo. Una delle sfide principali è stata l'implementazione di un sistema robusto per la gestione dei file immagine, che includesse la copia automatica nella cartella del progetto e l'uso di percorsi relativi per garantire la portabilità. Ho anche dedicato molto tempo a perfezionare l'interfaccia utente, curando in particolare il layout responsive e l'aspetto visivo delle card dei media.

Il modello dei dati ha richiesto più tempo del previsto per implementare correttamente il pattern Visitor per la serializzazione JSON e il pattern Observer per l'aggiornamento dell'interfaccia. Ho dedicato particolare attenzione alla validazione dei campi specifici per tipo di media e alla gestione delle modifiche non salvate.

Nella fase di test e debug, uno degli ostacoli maggiori è stato risolvere gli errori causati dai diversi ambienti di sviluppo. Dopo aver sviluppato l'applicazione su macOS con Clang, ho riscontrato incompatibilità con l'ambiente Linux fornito nella macchina virtuale, in particolare per quanto riguarda la gestione dei percorsi dei file e la codifica delle stringhe. Dopo numerose correzioni, l'applicazione ha funzionato correttamente anche su Linux e successivamente è stata testata anche su Windows.

La stesura della relazione ha richiesto la creazione di un diagramma delle classi con Draw.io per illustrare la gerarchia dei media e i pattern di design utilizzati, oltre alla scrittura in LaTeX per una formattazione professionale del documento.