Relazione progetto “Biblioteca Virtuale” Anno accademico 2024/2025

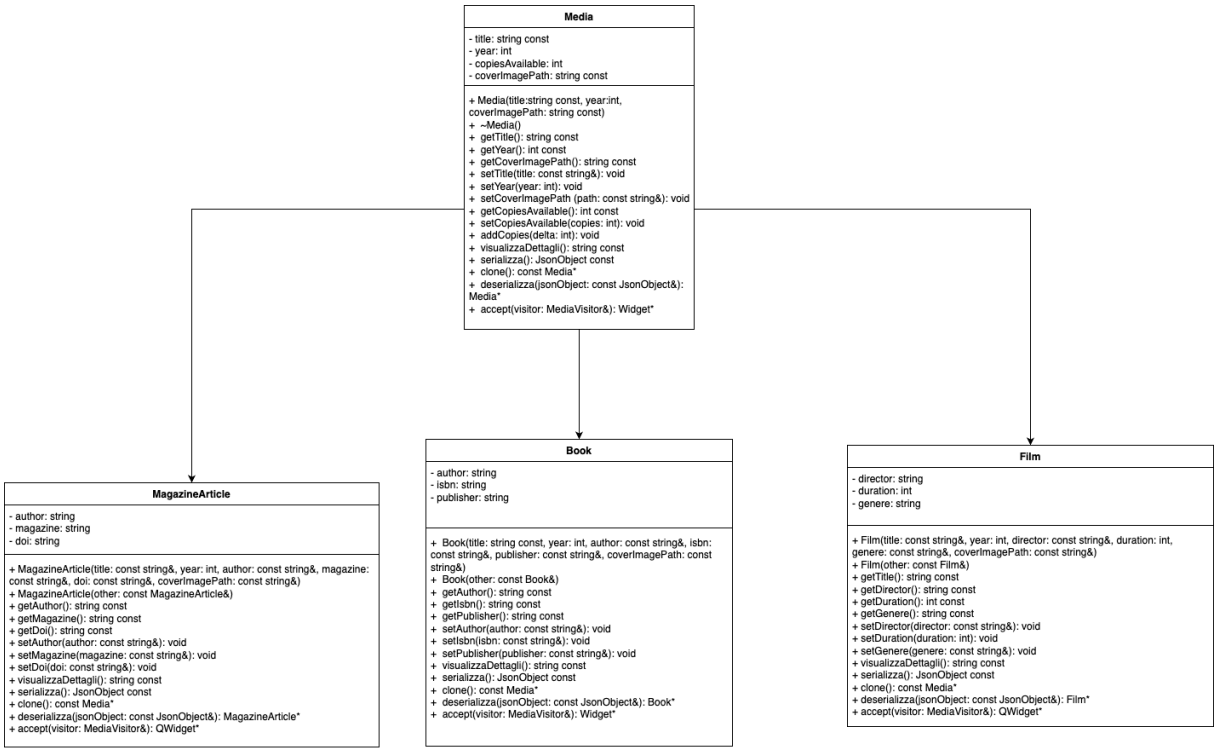
Andrea Masiero
Matricola: 2118057

Indice

1 Introduzione

Il progetto implementa un sistema per la gestione di una collezione multimediale (libri, film, articoli di rivista) con interfaccia grafica Qt. L’applicazione consente di:

- Accedere alla biblioteca virtuale attraverso una pagina di login.
- Caricare e salvare su file JSON l’intero catalogo.
- Visualizzare, filtrare e ricercare i media.
- Creare e modificare elementi con formulari dedicati.
- Gestire il numero di copie disponibili.



2 Classi principali del modello logico

Il progeto presenta una struttura formata da 4 classi (1 astratta e 3 concrete) Di seguito una panoramica delle entità principali emerse dal codice.

- **Media**
  - Classe base astratta per tutti i tipi di contenuto.
  - Attributi comuni: title, year, coverImagePath, copiesAvailable.
- **Sottoclassi concrete di Media**
  - Book: formata dai campi specifici author, isbn, publisher;
  - Film: formata dai campi specifici director, duration, genere;
  - MagazineArticle: formata dai campi specifici author, magazine, doi;

3 Polimorfismo

3.1 Visitor

- Pattern Visitor per la costruzione dei widget specifici per tipo.
- Dove: Media.h (accept), view/MediaVisitor.h, view/MediaWidgetVisitor.{h,cpp}.
- Uso: in view/MainWindow.cpp, media->accept(visitor) crea e popola dinamicamente la UI di anteprima/modifica in base al tipo (Book/Film/Article).
- Beneficio: separa la logica di presentazione per tipo, evitando if/switch sparsi e facilitando l'estensione.

3.2 Visitor/Raccolta per filtri

- Raccolta filtrata per tipo sfruttando dispatch sul tipo dinamico.
- Dove: view/Biblioteca.{h,cpp} con collectMediaByType(...) e view/MediaCollectorVisitor.{h,cpp} (visitor dedicato alla selezione), richiamata da MainWindow.cpp in getFilteredMedia().
- Uso: consente di ottenere liste per soli libri/film/articoli senza conoscere a compile-time i tipi concreti.
- Beneficio: incapsula la logica di filtro per tipo, mantenendo il codice della UI pulito.

3.3 Dettagli polimorfici

- Metodo virtuale sovrascritto per mostrare i dettagli specifici.
- Dove: Media.h (es. virtual visualizzaDettagli()), override in model/Book.h, model/Film.h, model/MagazineArticle.h.
- Uso: in MainWindow.cpp -> showMediaDetails() chiama selectedMedia->visualizzaDettagli() e ottiene descrizioni diverse per tipo.
- Beneficio: un'unica chiamata espone comportamenti diversi e coerenti con il tipo dinamico.

3.4 Serializer/Factory (JSON)

- Creazione e salvataggio polimorfi degli oggetti.
- Dove: persistence/JsonSerializer.{h,cpp} insieme a view/MediaFactory.{h,cpp}.
- Uso: in caricamento, in base a un campo "type" nel JSON, MediaFactory istanzia Book/Film/MagazineArticle e JsonSerializer popola i campi specifici; in salvataggio, JsonSerializer scrive i campi peculiari del tipo concreto.
- Beneficio: disaccoppia la persistenza dal tipo concreto, permettendo estensioni senza toccare la UI.

3.5 Azioni di modifica specifiche per tipo

- Gestione runtime di proprietà specifiche per sottoclasse.
- Dove: MainWindow.cpp -> editMedia(): dopo aver applicato modifiche comuni (title, year, cover, copies), esegue rami specifici (Book/Film/Article) per aggiornare attributi propri (autore/ISBN/editore; regista/durata/genere; autore articolo/rivista/DOI).
- Uso: il comportamento varia profondamente in base al tipo dinamico dell'oggetto selezionato.
- Beneficio: mantiene coesistenza di regole di modifica omogenee e specifiche per tipo.

4 Persistenza dei dati

Nel progetto la persistenza è centralizzata nella classe JsonSerializer, che legge e scrive l'intera Biblioteca su file JSON. L'UI (MainWindow) invoca esplicitamente le operazioni di caricamento/salvataggio e gestisce eventuali errori tramite messaggi.

4.1 Caricamento (deserialize)

- In base al file JSON selezionato dall'utente (o a quello di default), JsonSerializer legge il file e ricostruisce l'insieme dei Media nella Biblioteca.
- Meccanismo: il JSON contiene per ogni elemento un campo di tipo (es. Book/Film/MagazineArticl in base a questo, si istanzia dinamicamente il sottotipo corretto e si popolano i campi specifici.
- Integrazione UI:
  - loadLibrary(): apre un file picker e passa il path a JsonSerializer::caricaBiblioteca.
  - loadDefaultLibrary(): tenta data/bibliotecas/biblioteca_esempio.json; in alternativa usa un backup (ultima_sessione.json).
- Error: le eccezioni (FileNotFoundException, JsonParseException, BibliotecaException) sono catturate da MainWindow e mostrate con QMessageBox.

4.2 Salvataggio (serialize)

- Le modifiche apportate in memoria (aggiunta, modifica, eliminazione) vengono rese persistenti scrivendo l'intero stato corrente della Biblioteca su JSON.
- Meccanismo: JsonSerializer::salvaBiblioteca serializza tutti i Media, includendo il loro tipo e i campi specifici del sottotipo; il file viene riscritto completamente.
- Integrazione UI:
  - Salvataggio manuale: menu "Salva Biblioteca" chiama saveLibrary() con QFileDialog per scegliere il percorso.
  - Salvataggio automatico: alla chiusura dell'app (closeEvent) viene creato/aggiornato ultima_sessione.json come backup della sessione.

4.3 Modifica/Eliminazione

- La modifica/eliminazione avviene in memoria (sulla Biblioteca); quando l'utente salva o alla chiusura, l'intero contenuto aggiornato viene serializzato su file.
- Non è necessario "cercare e sostituire" il singolo oggetto nel JSON: si rigenera il file dalla rappresentazione in memoria, riducendo ambiguità su duplicati.

4.4 Gestione duplicati

Poiché si serializza l'intero stato corrente, non si basa sul solo titolo per identificare univocamente un elemento nel file. L'oggetto "giusto" è quello selezionato/modificato in memoria; la persistenza riflette esattamente la collezione aggiornata.

4.5 Vantaggi e estendibilità

- Aggiungere un nuovo tipo di Media richiede solo estendere la logica di (de)serializzazione per quel tipo (creazione e campi specifici), senza cambiare i flussi UI.
- La riscrittura completa del JSON mantiene consistenza tra memoria e disco ed evita errori di aggiornamento parziale.

5 Funzionalità aggiuntive

5.1 Pagina di login

Ho inserito una pagina di login che permette di accedere alla biblioteca virtuale inserendo l'sername e la passsword corretti. La pagina di login si apre appena avviata l'applicazione e sono già preinseriti le credenziali per accedere alla biblioteca, però si possono registrare anche delle nuove credenziali con l'apposito pulsante e verranno salvate nel file users.json.

5.2 Moduli dinamici per inserimento e modifica

I moduli di inserimento/modifica sono costruiti in modo dinamico in base al tipo di media, combinando:

- campi comuni (titolo, anno, copertina, copie disponibili);
- campi specifici (libro/film/articolo) con layout pulito e coerente. I form di modifica vengono precompilati con i dati correnti, così l'utente interviene solo sugli attributi da cambiare.

5.3 Ricerca e filtri in tempo reale

La visualizzazione supporta:

- filtro per tipo tramite una tendina dedicata (Tutti, Libri, Film, Articoli);
- ricerca case-insensitive per ogni caratteristica del media (titolo, genere, ISBN,...). I risultati si aggiornano immediatamente e la status bar mostra il numero di elementi trovati, rendendo l'esplorazione rapida anche con molte voci.

5.4 Valore aggiunto

- Interfaccia unica con griglia di anteprime cliccabili e dettagli contestuali.
- Scorciatoie da tastiera e toolbar/menu per azioni rapide.
- Conferme e messaggi chiari per evitare errori accidentali.
- Salvataggio automatico dell'ultima sessione per continuità d'uso.

6 Rendicontazione ore previste e svolte

Ho impiegato più tempo del previsto, arrivando a 60 ore totali: hanno inciso la cura dei dettagli dell'interfaccia, alcuni ripensamenti per allinearmi ai vincoli e diverse rifiniture legate a percorsi dei file e integrazione tra componenti.

6.1 Modellazione progetto

Previste 4h, effettive 4h.

- Definizione della gerarchia Media (Book, Film, MagazineArticle), dei campi comuni e specifici.
- Scelta del Visitor per separare la costruzione della UI dai tipi concreti.
- Stesura dei flussi principali (caricamento iniziale, ricerca, filtri).

6.2 Implementazione modello logico + pattern

Previste 10h, effettive 12h.

- Implementazione delle classi del dominio con metodi polimorfici e accettazione del visitor.
- Integrazione del MediaFactory per l'istanziazione per tipo durante la deserializzazione.
- Aggiustamenti iterativi per allineare naming e interfacce (es. getIsbn), e gestione delle copie disponibili.
- Piccole estensioni per supportare filtri e dettagli per tipo.

6.3 Interfaccia grafica Previste

14h, effettive 22h.

- Costruzione della MainWindow con griglia di anteprime, filtri, barra di ricerca e pannello dettagli.
- Visitor dei widget in doppia modalità (display/edit) con form dinamici per tipo e precompilazione dei campi in modifica.
- Gestione selezione copertine, anteprima immagini e percorsi relativi.
- Dialoghi dedicati per creazione/modifica e login.
- Tempo extra su layout, stati UI, messaggi e coerenza visiva tra schermate.

6.4 Persistenza (JSON)

Previste 6h, effettive 10h.

- Lettura/scrittura completa della collezione con JsonSerializer, includendo i campi specifici per tipo.
- Scelta di riscrivere l'intero file per garantire consistenza con lo stato in memoria.
- Gestione del caricamento automatico della libreria di esempio e backup di sessione.
- Tempo aggiuntivo per risolvere percorsi su macOS (applicationDirPath vs currentPath) e allineamento cartelle dati.

6.5 Gestione segnali/slot Previste

4h, effettive 6h.

- Collegamenti per ricerca in tempo reale, aggiornamento griglia, apertura dettagli, conferme e CRUD.
- Integrazione dei dialoghi (Add/Edit/Login) con la finestra principale e aggiornamenti dello schermo.
- Sincronizzazione degli stati della status bar e dei controlli di filtro dopo le operazioni.

6.6 Bugfix e rifiniture

Previste 2h, effettive 6h.

- Correzioni su nomi di metodi e incoerenze tra header/implementazioni (es. getIsbn).
- Fix dei percorsi per caricamento automatico e apertura dialoghi nelle cartelle corrette.
- Prevenzione di errori sui layout (clear e ricostruzione griglia) e controlli su puntatori nulli.
- Uniformazione messaggi all'utente, gestione conferme e piccoli miglioramenti di usabilità.

Totale: previste 40h, effettive 60h. La differenza è dovuta soprattutto alla cura dell'interfaccia (moduli dinamici, stati UI e immagini), all'integrazione della persistenza con percorsi portabili e a rifiniture.