

“Ingegneria del Software” 2024-2025

Docente: Prof. Angelo Furfaro

<Gestore Libreria>

Data	<06/09/2025>
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	Matricola	E-mail address
Maria Rita Ombres	239898	mbrmr04a45d086m@studenti.unical.it

Sommario

Non è stata trovata alcuna voce d'indice.

List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Gestione creazione oggetto Libro	22-08-25	22-08-25	Creazione oggetto Libro tramite Builder
Implementazione funzioni di ricerca ed ordinamento in base a criteri specifici	23-08-25	24-08-25	Utilizzo del pattern strategy con chain of Responsibility per la gestione di ricerca; Mentre per l'ordinamento solo Strategy
Persistenza della libreria e funzioni undo/redo	25-08-25	25-08-25	Implementazione del pattern Memento per rendere la libreria persistente; Aggiunta di funzioni undo e redo per funzioni di ripristino operazioni

A. Stato dell'Arte

Analizzate sistemi esistenti, prendete spunto da cio' che esiste

La gestione di cataloghi librari è un problema ampiamente affrontato sia in ambito accademico che commerciale. Esistono diversi software dedicati, che vanno da semplici strumenti di catalogazione personale fino a veri e propri sistemi gestionali per librerie ed e-commerce.

In contesti più semplici, è possibile trovare soluzioni basate su **fogli di calcolo** (es. Excel) utilizzati per catalogare libri con funzionalità minime di ordinamento e filtraggio. Sebbene accessibili, tali strumenti mancano di un'interfaccia dedicata e di una gestione strutturata dei dati.

Il progetto sviluppato in questa relazione si inserisce in questo panorama proponendo una soluzione intermedia: un **gestore di catalogo librario** pensato per scopi didattici, che mantiene semplicità d'uso e leggerezza ma al tempo stesso integra funzionalità tipiche dei software più evoluti, come la ricerca, l'ordinamento e la persistenza dei dati su file. Inoltre, sono stati adottati pattern software che rendono l'architettura estensibile e più vicina alle buone pratiche ingegneristiche rispetto alle soluzioni più basilari.

Per garantire una gestione semplificata ed efficiente all'utente viene fornita una GUI(Graphic User Interface) con la quale si possa interagire con l'applicazione e le sue funzionalità in maniera intuitiva ed efficiente.

B. Raffinamento dei Requisiti

A partire dai servizi minimali richiesti, raffinate la descrizione dei servizi offerti dal vostro applicativo. Descrivete anche i requisiti non funzionali.

A.1 Servizi (con prioritizzazione)

Descrivete in dettaglio i servizi offerti dal vostro Sistema, insieme a quelli che ritenete siano le soluzioni concettuali necessarie. In questa fase, non fate riferimento ad alcuna tecnologia specifica. Se volete, intervistate stakeholder e collezionate dati dal web o da altre sorgenti. Dovete acquisire una conoscenza avanzata dei problemi associati ai vostri servizi. Assegnate un ID a ciascun servizio. Priorizzate inoltre i servizi in base a due scale: importanza alta, media, bassa. Complessità alta, media, bassa.

Requisiti funzionali offerti dal sistema:

1. Interfaccia grafica costruita tramite Java Swing per facilitare l'utente nell'eseguire il programma; (importanza=alta, complessità=alta)
 2. Creazione di oggetti con gestione di inserimento attributi errati (importanza=alta, complessità=media)
 3. Possibilità di aggiornare in maniera dinamica la struttura garantendo l'evoluzione del catalogo, comprendendo aggiunte, rimozioni e modifiche varie; (importanza alta, complessità=alta);
 4. Operazioni di ricerca ed ordinamento in base a determinati attributi (importanza=media, complessità=alta)
 5. Gestione salvataggio e caricamento libreria su/da file CSV/JSON (importanza=media, complessità=media)
-

A.2 Requisiti non Funzionali

Elencare i requisiti non funzionali più importanti per il vostro Sistema

I requisiti non funzionali più importanti per Gestione Libreria sono:

- L'usabilità del sistema data l'interfaccia grafica semplificata;
- La robustezza: il sistema avvisa l'utente nel caso in cui non venissero rispettate le condizioni di implementazione per oggetti di tipo 'libro' e le operazioni ad esso associate;
- Il sistema è stato progettato per garantire duttilità oltre che robustezza: l'architettura è modulare, i pattern (Command, Memento, Facade, Builder, Strategy, Observer, Chain of Responsibility) sono stati adottati per separare le responsabilità e per ognuno è presente un test specifico per controllare il corretto funzionamento;
- Affidabilità: il sistema deve garantire il corretto funzionamento in materia di stabilità e soddisfare i requisiti anche in condizioni non convenzionali, con gestione degli errori tramite funzione di testing per ridurre i malfunzionamenti;
- Sicurezza: L'utente non ha accesso a tutti i dettagli implementativi del sistema, proteggendo il sistema da operazioni non valide;

A.3 Scenari d'uso dettagliati

Descrivere gli scenari più comuni, più interessanti, o più complicati d'uso dei vostri servizi.

Caso d'uso	Aggiungere libro
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> 1. L'utente inserisce i dati del nuovo libro 2. Il sistema controlla che i dati inseriti rispettino le condizioni 3. il sistema lo aggiunge direttamente in catalogo
Svolgimento alternativo	<ol style="list-style-type: none"> a. L'utente non inserisce i dati corretti (non segue i parametri giusti) b. Il sistema non inserisce il libro e manda un messaggio d'errore
Postcondizione	Il libro viene inserito come richiesto
Descrizione	L'utente inserisce un nuovo libro inserendo tutti i parametri

Caso d'uso	Rimuovere libro
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> 1. L'utente seleziona il libro da rimuovere; 2. Il sistema elimina il libro insieme ai rispettivi parametri dal catalogo
Postcondizione	Il libro selezionato viene rimosso come richiesto

Descrizione	L'utente seleziona dal catalogo il libro da rimuovere
--------------------	---

Caso d'uso	Modificare libro
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> 1. L'utente seleziona il libro al quale modificare i parametri 2. L'utente modifica i parametri che desidera mentre quelli inalterati rimangono uguali; 3. Il sistema aggiorna il catalogo con le modifiche effettuate
Postcondizione	Gli attributi del libro selezionato vengono modificati come richiesto
Descrizione	L'utente seleziona un libro che intende modificare

Caso d'uso	Ordinare catalogo tramite filtri
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> 1. L'utente sceglie il parametro per l'ordinamento del catalogo 2. Il sistema ordina gli elementi in base a tale parametro ed aggiorna la tabella
Postcondizione	Il catalogo dei libri è ordinato secondo i parametri scelti dall'utente
Descrizione	L'utente seleziona un parametro per l'ordinamento del catalogo e lo aggiorna

Caso d'uso	Ricerca tramite filtri
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> L'utente seleziona il filtro per il quale applicare la ricerca Il sistema mostra i risultati di tale ricerca
Postcondizione	Il catalogo libri mostra i risultati della ricerca effettuata
Descrizione	Il sistema mostra i risultati della ricerca, aggiornando il catalogo libri

Caso d'uso	Salvare catalogo su file CSV/JSON
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> L'utente richiede il salvataggio della libreria corrente in formato CSV/JSON Il sistema effettua il salvataggio e mostra il percorso dove è stato salvato
Postcondizione	Il catalogo libri viene salvato nel file CSV/JSON
Descrizione	Il sistema salva i dati su file in formato CSV o JSON

Caso d'uso	Caricare catalogo da file CSV/JSON
Tipo	primario
Svolgimento	<ol style="list-style-type: none"> L'utente seleziona il file sorgente di tipo CSV/JSON dal

	<p>quale prendere i libri</p> <ol style="list-style-type: none">2. Il sistema controlla la validità dei campi e successivamente li aggiunge3. Il catalogo viene aggiornato
Svolgimento alternativo	<ol style="list-style-type: none">a. Il sistema non riesce a caricare i dati correttamente (il file scelto dall'utente non è il formato atteso)b. L'utente viene avvisato tramite messaggio di errore e viene invitato a riprovare
Postcondizione	I libri vengono aggiunti come richiesto
Descrizione	Il sistema carica i dati da file CSV/JSON e li aggiunge alla libreria

A.4 Excluded Requirements

Descrivere i servizi eventualmente esclusi, e spiegare il perché

A.5 Assunzioni

<Briefly document, in this section, the most relevant requirement assumptions/decisions you had to make during your project>

Per il funzionamento del gestore libreria, non sarà possibile avere più di una tabella contenente il catalogo dei libri disponibili data la costruzione della Libreria come una Singleton, in modo tale da non avere più istanze della stessa libreria.

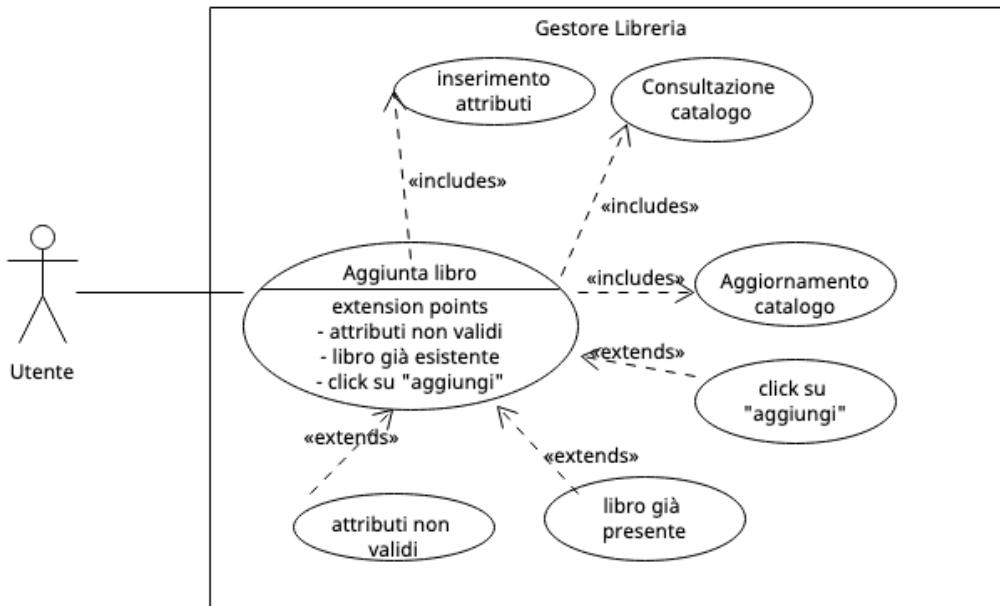
Per la funzione di aggiunta di un volume al catalogo libri, tali parametri devono rispettare delle condizioni:

- Due volumi si considerano uguali solo tramite codice ISBN, in modo tale che due versioni dello stesso libro possano coesistere; Se si prova a modificare o aggiungere un volume con codice ISBN già presente, il sistema non lo aggiungerà e manderà un messaggio di errore;
- Per valutare e recensire un libro il suo stato di lettura dev'essere *letto*, il codice ISBN deve iniziare con 978 e contenere 13 cifre in totale o il libro non verrà caricato nel catalogo.
- Per quanto riguarda le funzioni di undo/redo, queste ultime non riguarderanno lo stato dell'applicazione ma solo dello stato libreria. Non serviranno lo scopo di tornare indietro dopo aver ordinato il catalogo, ma solo nei casi in cui si va a modificarne lo stato(come aggiunta, modifica ecc...) poiché collegate strettamente con il pattern memento;
- Nella fase di inserimento libri da file CSV/JSON, i parametri dei libri presenti dovranno seguire lo stesso ordine della tabella:
titolo,autore,genere,valutazione,statuto di lettura e codice ISBN; Se non seguiranno tale ordine non verranno aggiunti in libreria poiché gli attributi dei libri non saranno considerati validi.

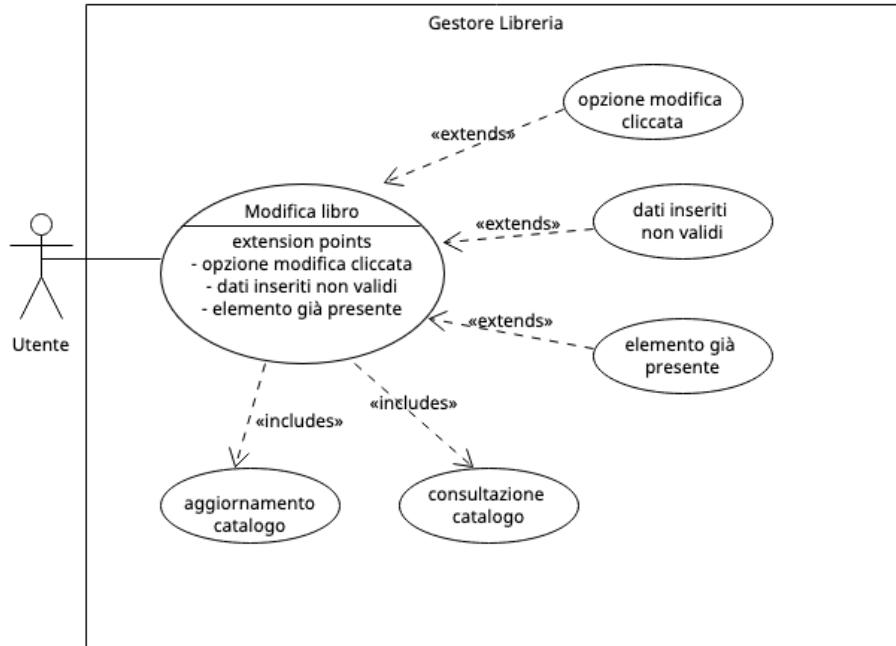
A.6 Use Case Diagrams

Vengono presentati qui di seguito, gli Use Case diagrams con le operazioni più significative del sistema, fino ad arrivare all'ultimo diagramma il quale rappresenta, in visione generale, tutte le operazioni viste nel loro complesso. Ogni caso d'uso verrà quindi descritto singolarmente con il suo scenario specifico, mentre in quello complessivo verranno omesse tutte le funzionalità.

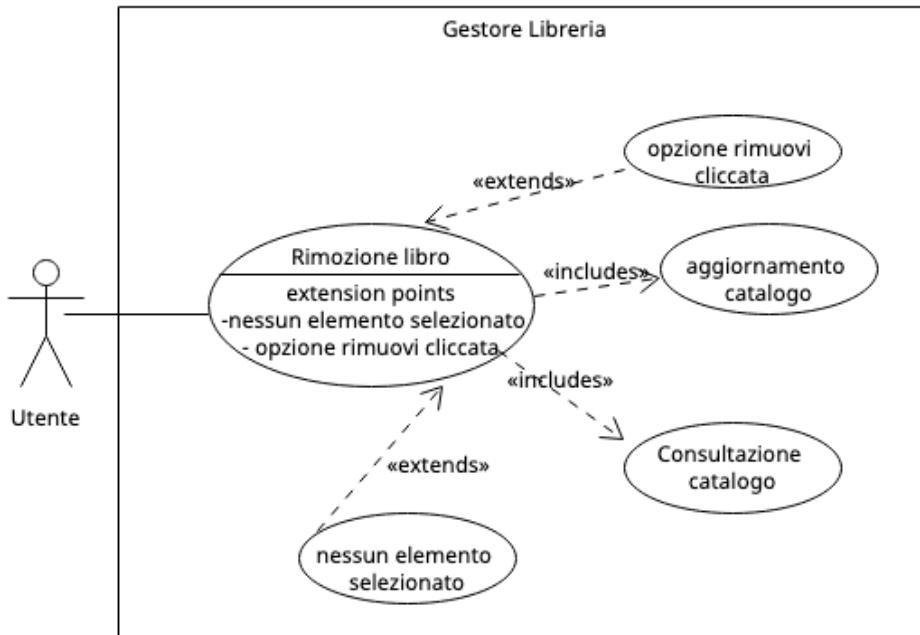
1. Aggiunta di un libro



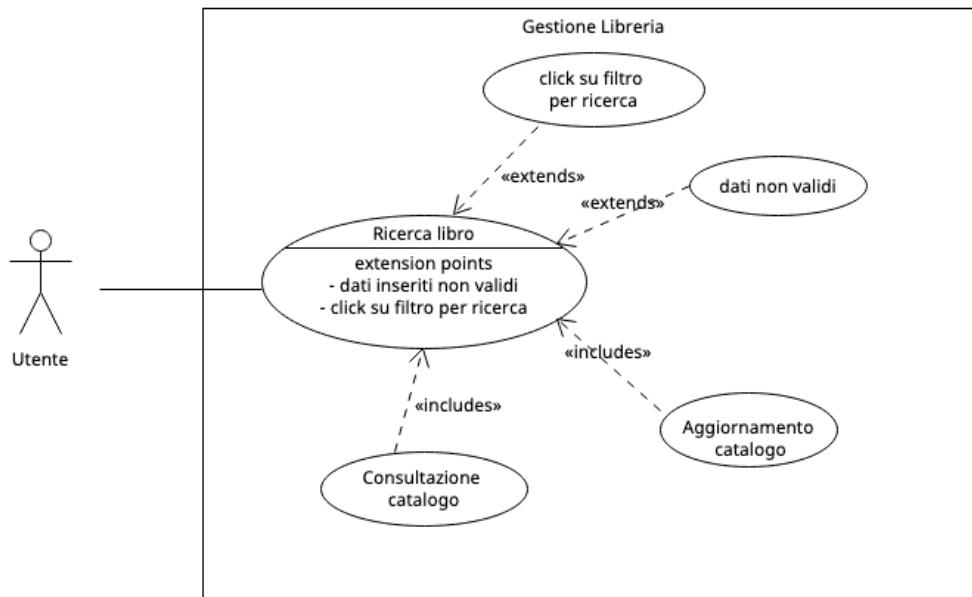
2. Modifica dell'oggetto libro



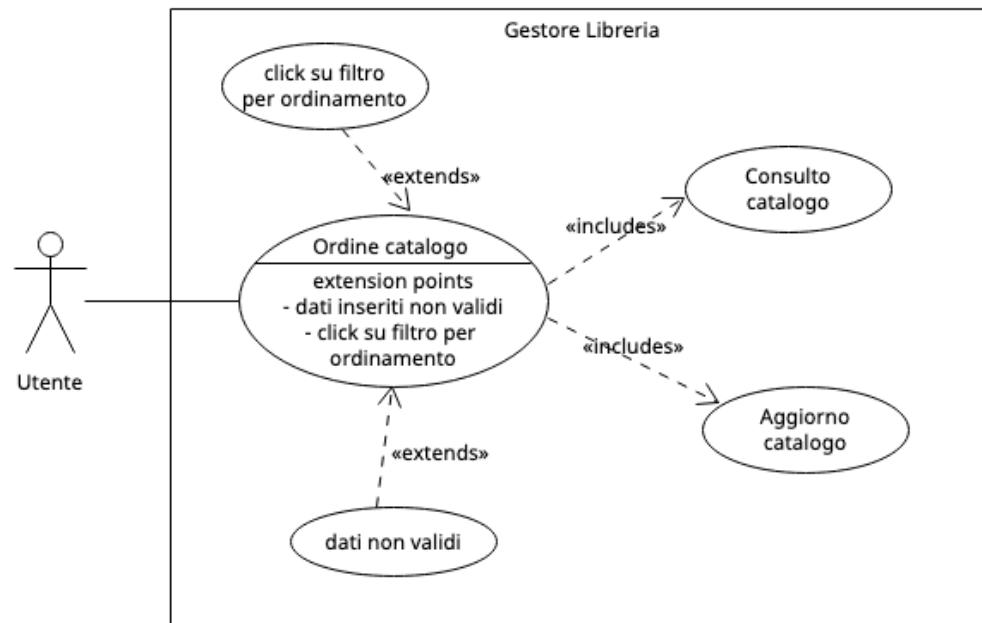
3. Rimozione di un libro



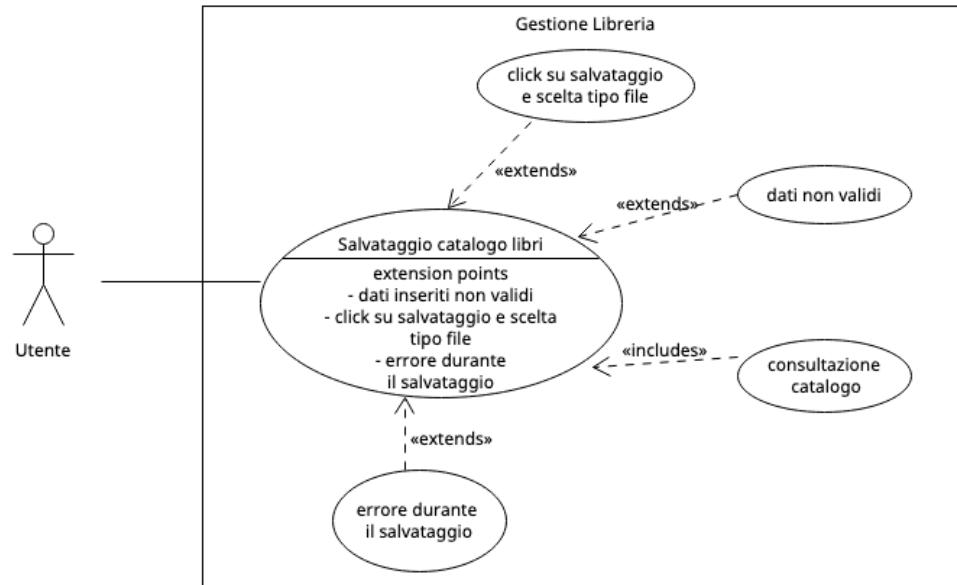
4. Ricerca di un libro tramite filtri



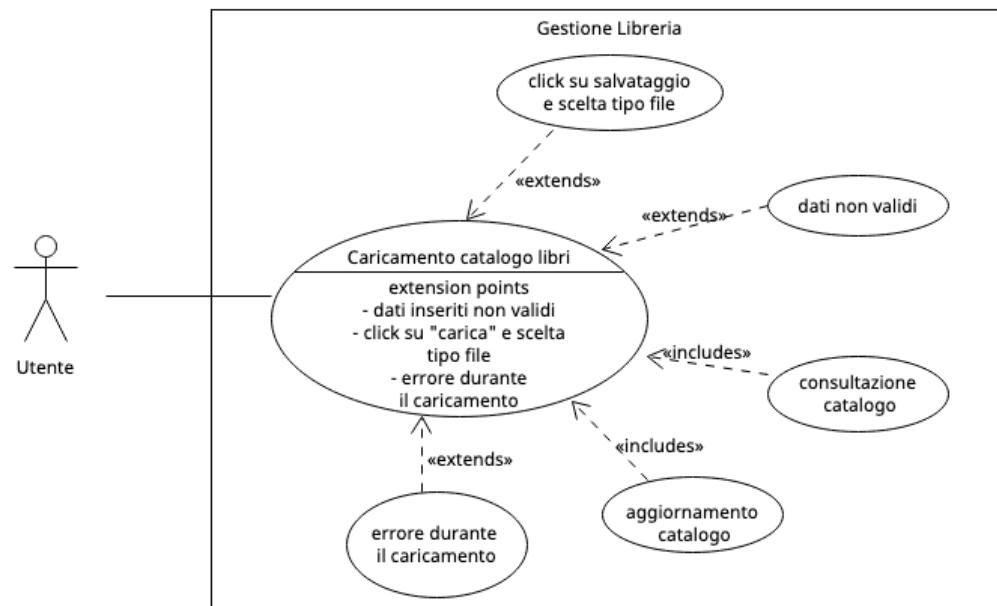
5. Ordine catalogo tramite filtro



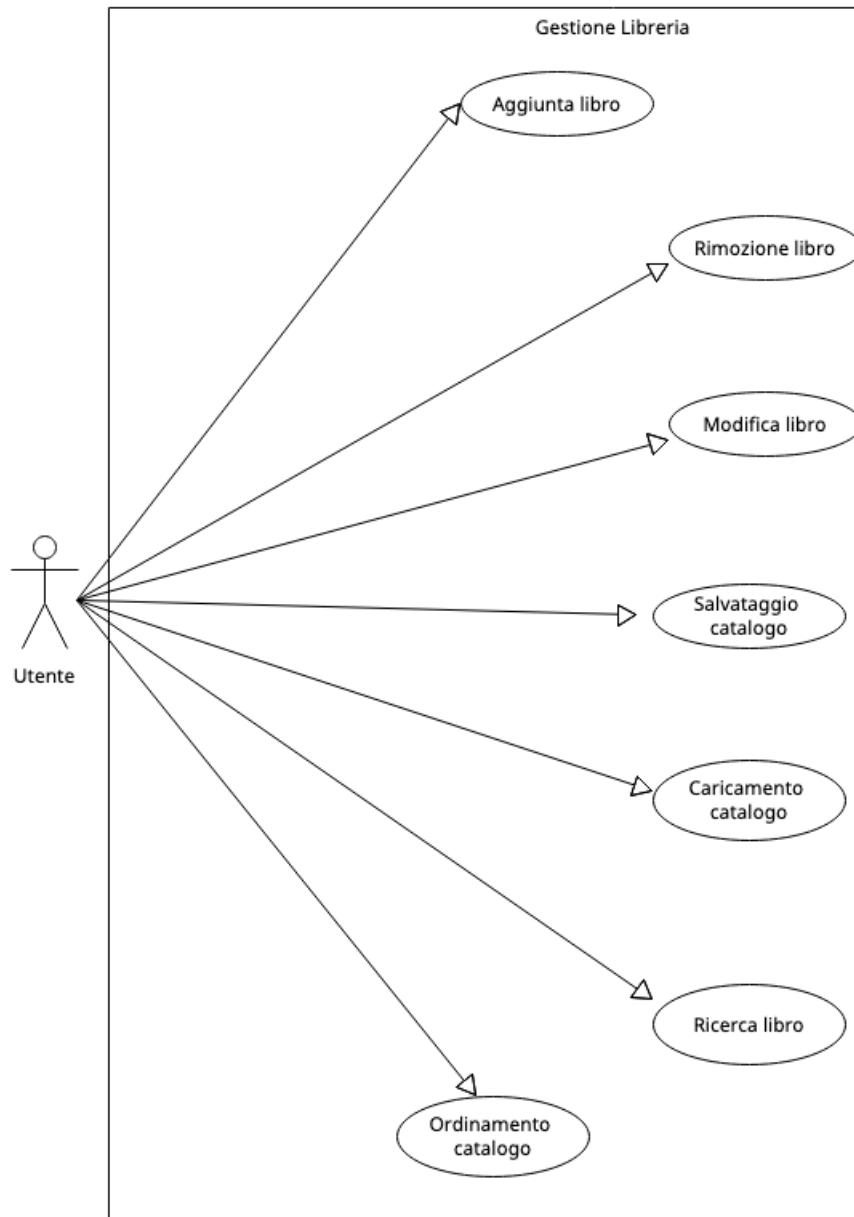
6. Salvataggio su file del catalogo



7. Caricamento libreria da file



Use Case Diagram di Gestore Libreria:



C. Architettura Software

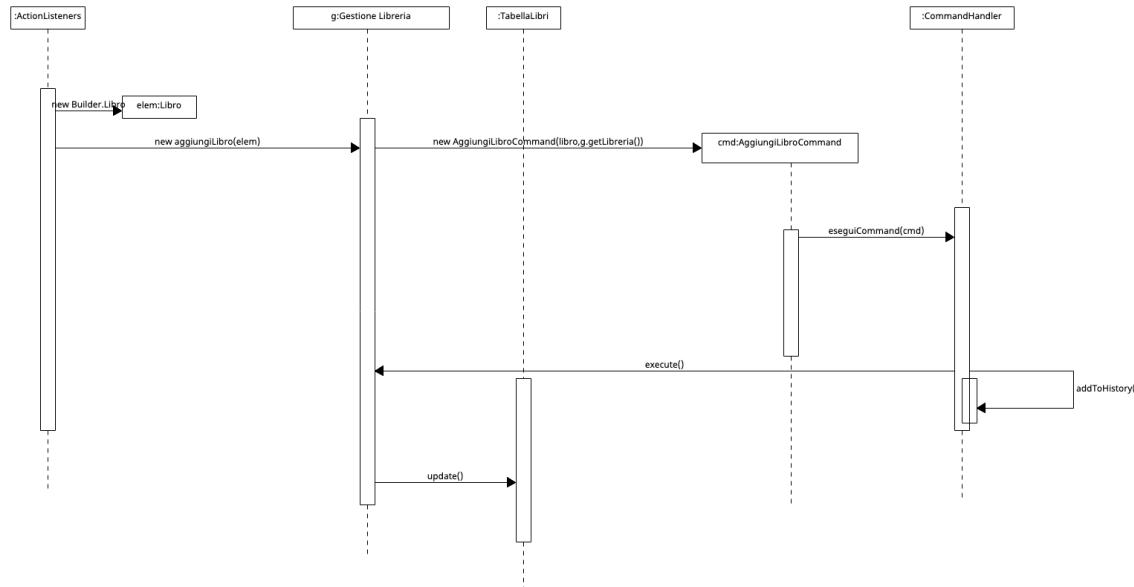
<IF RELEVANT, Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, and their related Sequence Diagrams >

C.1 The static view of the system: Component Diagram

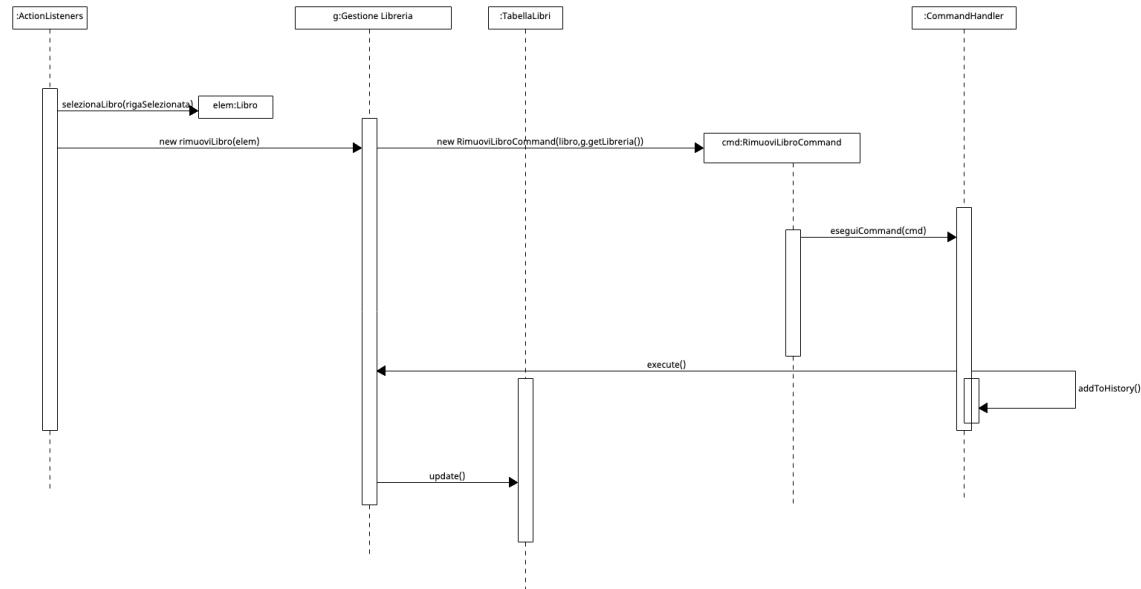
C.2 The dynamic view of the software architecture: Sequence Diagram

Per determinare la vista dinamica del sistema, si utilizzano i diagrammi di interazione. In tali diagrammi rientrano i sequence diagram e gli activity diagram, i quali di seguito descriveranno le funzioni principali del programma e la sua dinamicità.

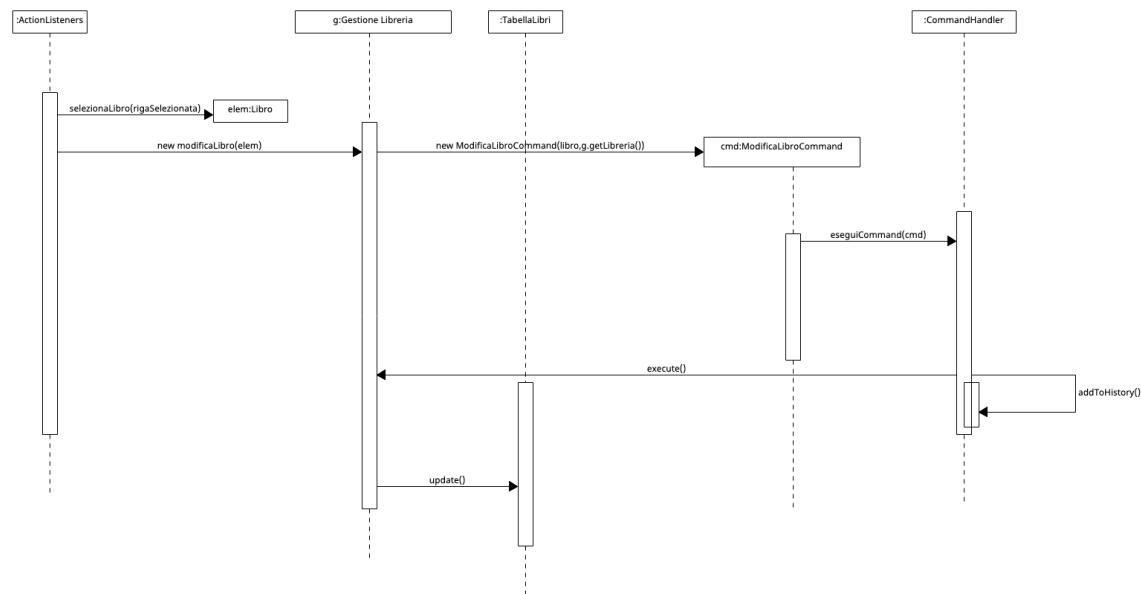
1. Aggiunta libro



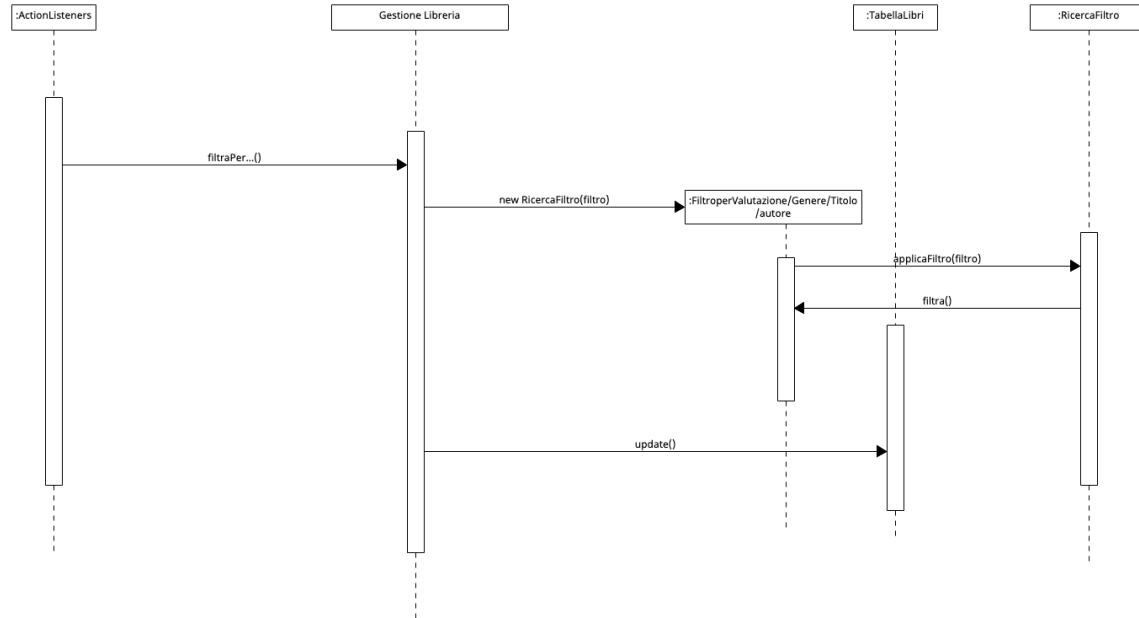
2. Rimozione libro



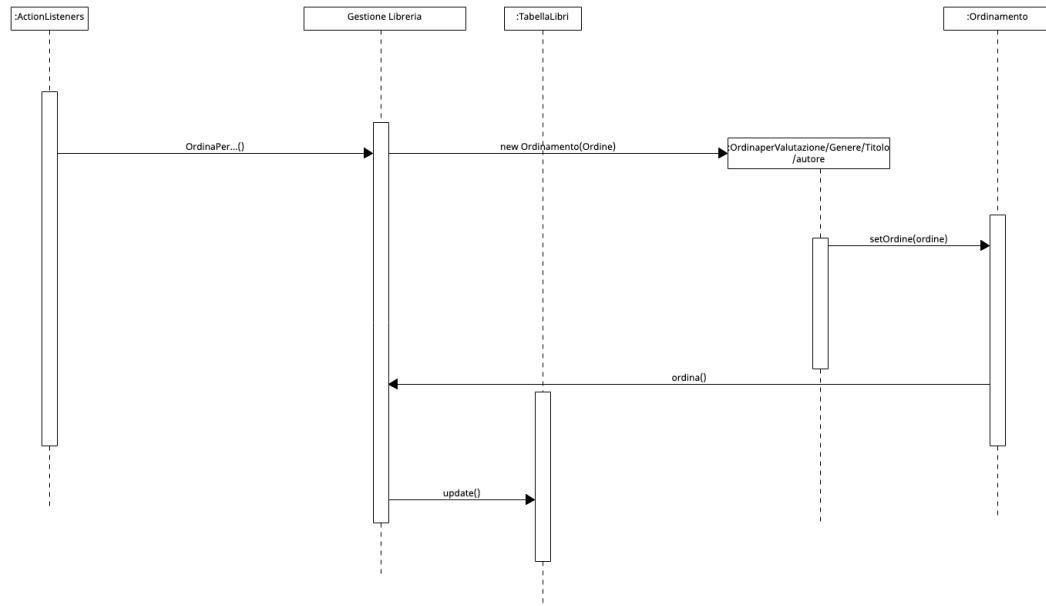
3. Modifica libro



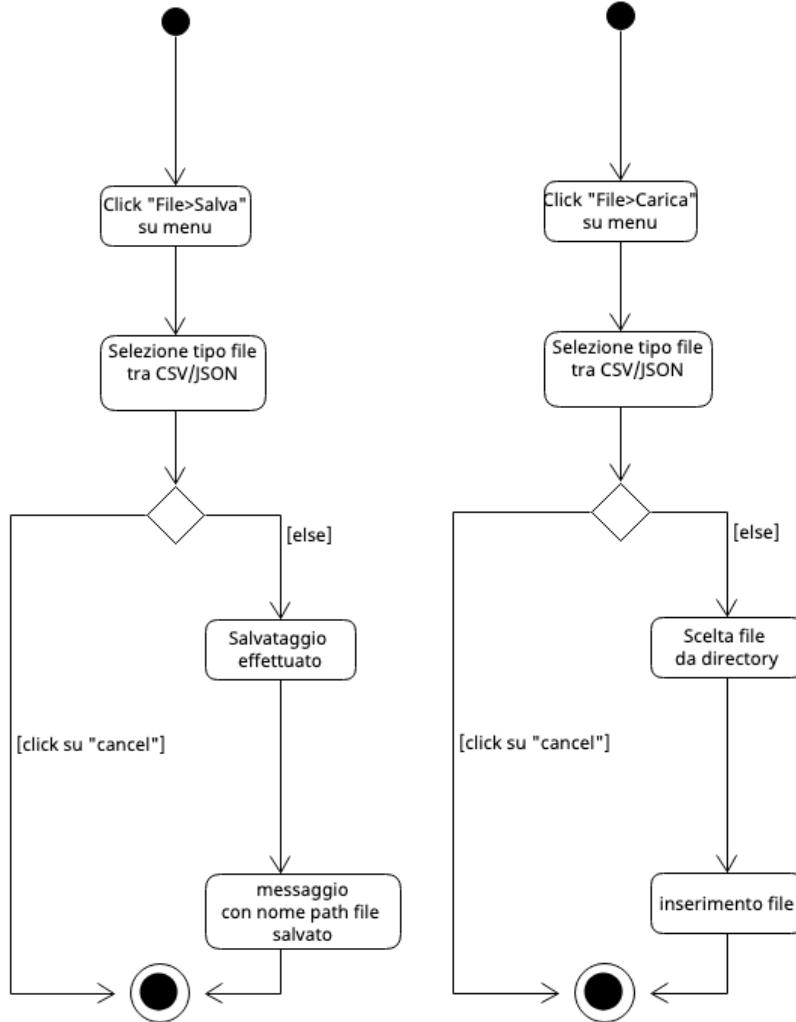
4. Ricerca libro



4. Ordinamento libreria



5. Salvataggio e Caricamento libreria



D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

Definite le sorgenti di dati a voi necessarie per realizzare i servizi di cui sopra. Modellate tali dati tramite un ER o similari. Specificate se e quali di tali dati sono già forniti da applicativi esistenti.

La persistenza della libreria viene implementata da parte dell'utente, eseguendo il salvataggio o il caricamento su determinati tipi di file. L'utente può scegliere se salvare il file in formato .csv o in formato .json ed in maniera rapida viene salvata in memoria. Il sistema d'altro canto, completato il salvataggio, mostra all'utente un messaggio di conferma con il path file di destinazione del file salvato. Per quanto riguarda il caricamento di libri presenti su file, sempre .csv e .json, va rispettato il vincolo di ordine d'inserimento dei parametri. L'utente se inserisce i dati in ordine errato, il sistema annulla il caricamento dato che non riesce ad effettuare i controlli di validazione previsti. In caso di caricamento riuscito, la tabella dei libri si aggiorna mostrando tutti i libri presenti nella propria libreria.

E. Scelte Progettuali (Design Decisions)

<Document here the 5 most important design decisions you had to take. You can use both a textual or a diagrammatic specification.>

Di seguito in elenco si trovano le 5 decisioni progettuali principali attuate al fine di garantire il soddisfacimento dei requisiti per la creazione di un gestore di libreria.

1. **Facade:** Per determinare un'interazione con la libreria che sia, per l'utente usabile, e quindi con un'interfaccia molto intuitiva ho implementato il design pattern Facade. Tale pattern si occupa di riunire tutte le operazioni

nascondendo la logica interna e consentendo al resto del sistema di accedere in maniera unificata al programma rendendolo anche meno complesso da gestire;

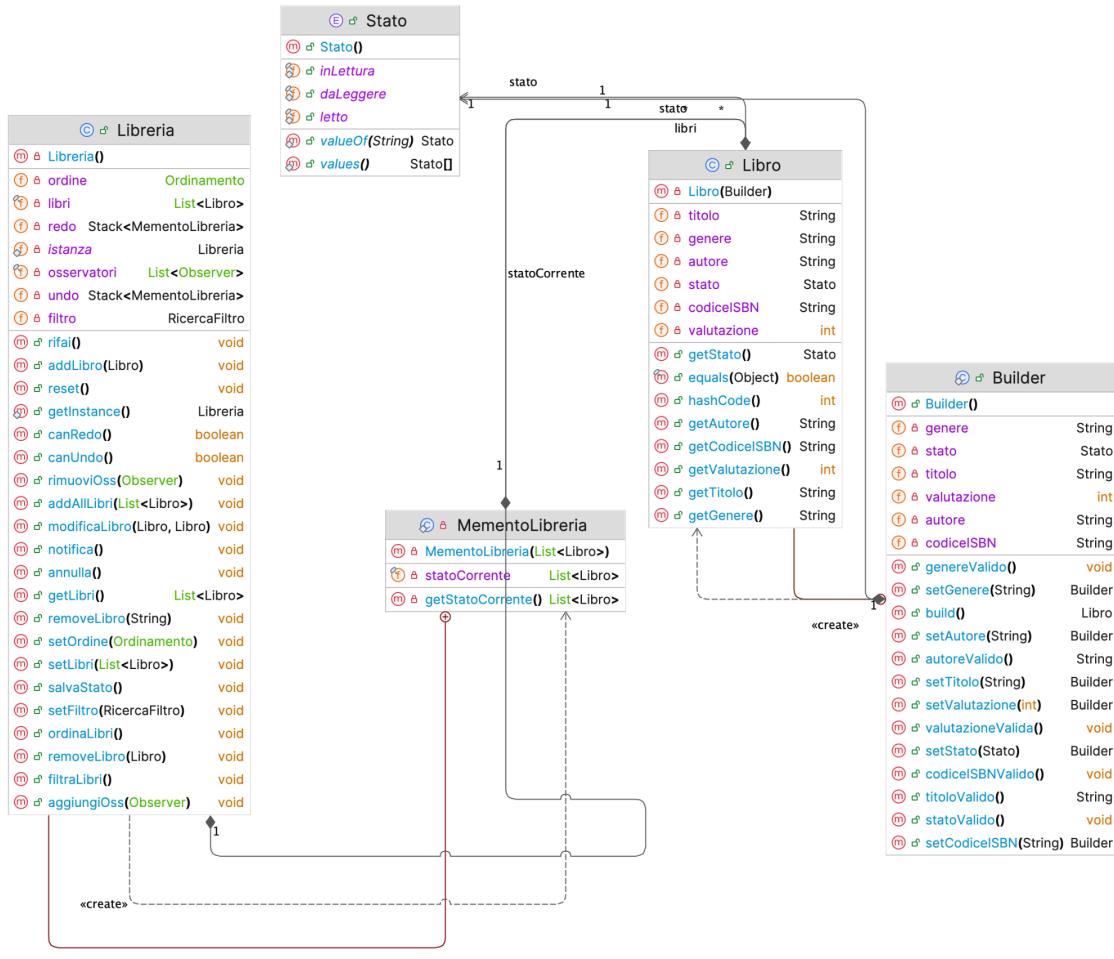
2. **Command:** Il pattern Command è stato utilizzato per rappresentare le azioni dell'utente come oggetti indipendenti. Operazioni come aggiungere, rimuovere o modificare un libro vengono incapsulate in comandi specifici (AggiungiLibroCommand, RimuoviLibroCommand, ModificaLibroCommand) in modo da migliorare notevolmente l'usabilità del prodotto. In questo modo, le azioni diventano gestibili in maniera uniforme e possono essere facilmente integrate con il sistema di undo/redo offerto dal Memento.
3. **Memento:** Il pattern Memento è stato scelto per implementare le operazioni di undo e redo, oltre che per garantire la persistenza dello stato della libreria. Ogni modifica eseguita (aggiunta, rimozione o modifica di un libro) può generare un “memento” che conserva lo stato precedente, permettendo così di annullare o ripristinare operazioni. Questo approccio è utile sia per l'usabilità del sistema sia per eventuali esigenze di salvataggio e recupero della libreria.
4. **Strategy e Chain of Responsibility:** Il pattern Strategy rappresenta una scelta fondamentale per quanto riguarda la gestione del progetto. Tale pattern è stato utilizzato in due maniere diverse: da solo ed insieme a Chain of Responsibility.
Nel suo utilizzo singolo viene adottato per due scopi principali: la gestione dei file ed ordinamento tramite filtri.
Per la gestione dei file consente di scegliere dinamicamente diverse strategie di caricamento e salvataggio in modo da mantenere la libreria indipendente dai formati utilizzati(CSV/JSON).
Per l'ordinamento, invece, ha permesso di definire strategie differenti per filtrare o ordinare il catalogo dei libri in base ai suoi attributi: titolo, autore, genere ecc...
Infine, la combinazione di Strategy e Chain of Responsibility è stata adottata per la ricerca con filtri multipli. Il pattern Strategy definisce i singoli criteri di filtro (per autore, per genere, per valutazione, ecc.), mentre la Chain of Responsibility consente di concatenare più filtri in sequenza, applicandoli progressivamente alla lista dei libri. Questo approccio rende il sistema di ricerca flessibile, modulare ed estendibile, permettendo di costruire ricerche avanzate in maniera dinamica.
5. **Observer:** Il pattern Observer è stato impiegato per consentire l'aggiornamento automatico delle interfacce grafiche quando cambia lo stato della libreria. Garantendo che tutti i cambiamenti si riflettessero

anche a livello visivo per l'utente. Ad esempio, l'aggiunta o la rimozione di un libro deve riflettersi immediatamente nella tabella dei libri. In questo scenario Libreria agisce come Observable, mentre i componenti grafici come Observer, ricevendo notifiche e aggiornandosi senza dover essere gestiti manualmente.

F. Progettazione di Basso Livello

Per questioni di praticità utilizzo l'IDE dove ho svolto il progetto come ausilio per descriverne le caratteristiche in modo da analizzare la progettazione a basso

livello del programma.

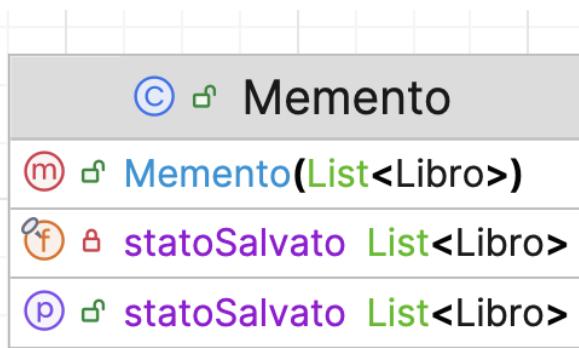


- Builder ed oggetti Libro, Libreria ed enum Stato:

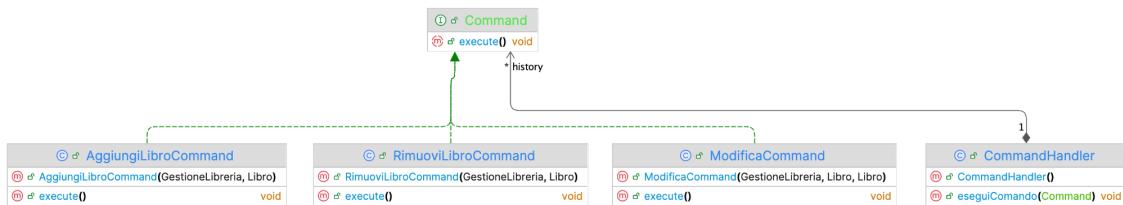
Il pattern Builder è stato utilizzato per la creazione degli oggetti Libro. La classe possiede infatti diversi attributi (titolo, autore, codice ISBN, genere, valutazione, stato di lettura) e un costruttore tradizionale risulterebbe poco leggibile e incline a errori. Con il Builder si possono impostare gradualmente i valori desiderati, ottenendo codice più chiaro, flessibile ed estendibile, senza obbligare a passare ogni parametro in un unico costruttore. Per la costruzione della libreria si è scelto di rendere tale una **Singleton**. Tale scelta è stata fatta per soddisfare la consistenza dei dati e centralizzare la gestione del catalogo, questo assicura che durante l'esecuzione del programma esista sempre e solo un'istanza condivisa della libreria, a cui tutti i componenti possono accedere senza rischiare duplicazioni o conflitti.

L'enum Stato viene utilizzata per la facilità nel descrivere lo stato di lettura di ogni singolo libro in modo chiaro e standardizzato, evitando l'uso di stringhe libere che potrebbero portare a incoerenze. In questo modo ogni libro può trovarsi in uno dei soli stati previsti (ad esempio *da leggere*, *in lettura*, *letto*), garantendo maggiore uniformità nella gestione e semplicità nelle operazioni di confronto e filtraggio.

- **Memento:** Dato il suo scopo di implementare le funzioni di undo/redo in modo tale da garantire persistenza nella libreria le sue implementazioni sono direttamente incluse nella classe libreria.



- **Command:** Utilizzato per gestire in maniera autonoma le operazioni sui libri del gestore libreria, la sua implementazione si costruisce in questo modo:

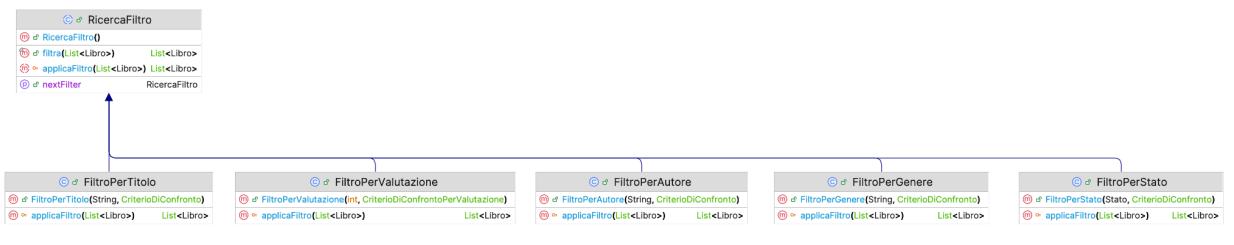


- **Strategy e Chain of Responsibility per la ricerca tramite filtri:** Il pattern **Strategy** è stato adottato per definire in maniera modulare i differenti criteri di ricerca e filtraggio applicabili ai libri presenti nella libreria. Ogni criterio (ad esempio filtraggio per autore, per genere, per valutazione o per stato di lettura) è implementato come una strategia distinta, consentendo di mantenere separata e indipendente la logica di ciascun filtro. Ciò

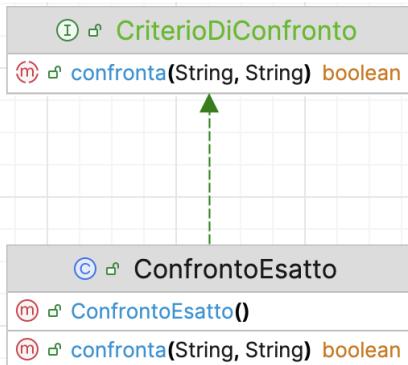
garantisce un'elevata flessibilità, in quanto è possibile aggiungere o modificare criteri senza incidere sul funzionamento complessivo del sistema.

Parallelamente, il pattern **Chain of Responsibility** è stato impiegato per permettere l'applicazione sequenziale di più filtri, collegandoli tra loro in una catena. In questo modo, ogni richiesta di ricerca può essere elaborata da una serie di strategie concatenate, che filtrano progressivamente l'insieme dei libri fino a ottenere i risultati corrispondenti a tutti i criteri impostati.

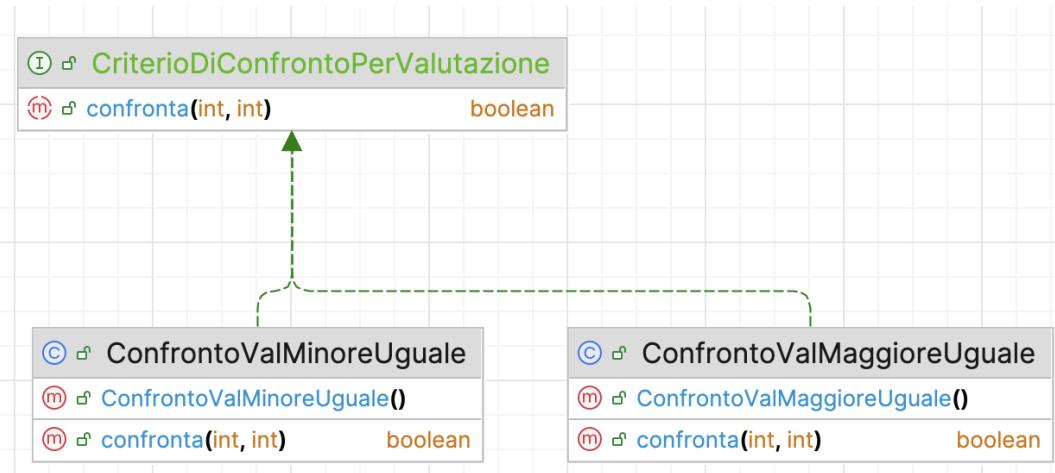
La combinazione dei due pattern risulta particolarmente efficace: lo Strategy assicura la modularità e la sostituibilità dei singoli criteri di filtraggio, mentre la Chain of Responsibility consente di gestire dinamicamente sequenze di filtri multipli. Questo approccio rende il sistema di ricerca **flessibile, estendibile e facilmente adattabile** alle esigenze dell'utente, garantendo al contempo un'implementazione ordinata e coerente dal punto di vista architetturale.



Per implementare le funzioni di filtraggio ho utilizzato dei criteri di confronto, i quali garantissero la correttezza delle varie operazioni:



Nello specifico, ho implementato dei confronti particolari soprattutto per la gestione di filtro per valutazioni. In questo modo si possono controllare valutazioni con parametri come maggiore o uguale/minore o uguale per filtrare la lista di libri disponibili.



- **Observer:** Il pattern **Observer** è stato adottato per gestire in maniera automatica e coerente l'aggiornamento delle componenti dell'interfaccia grafica a seguito di modifiche apportate alla libreria. La necessità di notificare più elementi contemporaneamente (ad esempio la tabella dei libri o altre viste collegate) rende poco pratico l'utilizzo di chiamate dirette, poiché aumenterebbe il livello di accoppiamento tra la logica applicativa e la presentazione.

Attraverso l'implementazione dell'Observer, la classe Libreria svolge il ruolo di **soggetto osservato** (observable), mentre le varie interfacce grafiche e componenti correlate assumono il ruolo di **osservatori** (observers). Ogni volta che lo stato della libreria viene modificato, ad esempio con l'aggiunta, la rimozione o la modifica di un libro, viene inviata automaticamente una notifica a tutti gli osservatori registrati, che provvedono ad aggiornarsi di conseguenza.

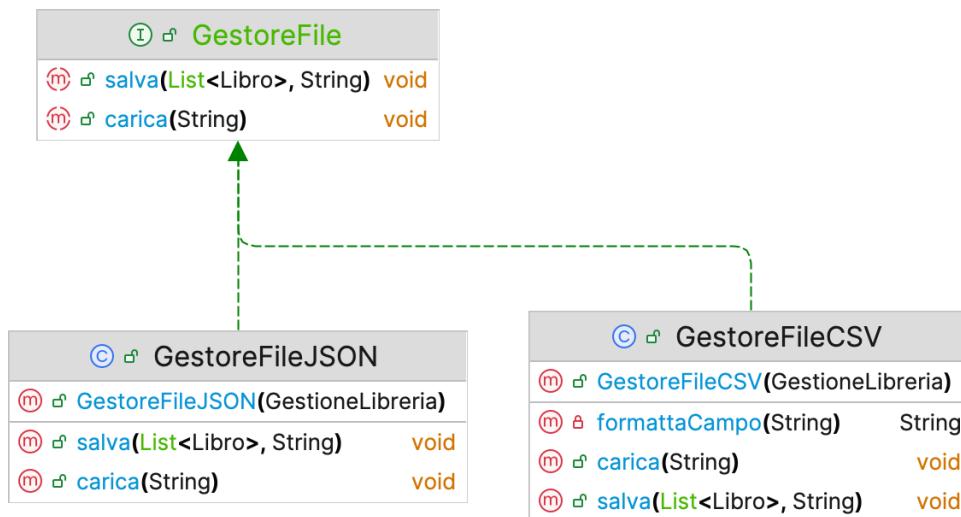
L'adozione di questo pattern garantisce un'architettura **disaccoppiata e scalabile**, in cui la logica di gestione dei dati e la loro rappresentazione grafica rimangono indipendenti. In questo modo è possibile estendere il numero di osservatori senza dover modificare la logica interna della libreria, ottenendo così un sistema più modulare, riusabile e semplice da manutenere.



- Strategy:
 - per la gestione file:

Poiché il sistema deve supportare diversi formati di persistenza, come CSV e JSON, era necessario predisporre un meccanismo che permettesse di estendere o modificare tali funzionalità senza introdurre rigidità o duplicazione di codice.

Attraverso Strategy, le varie modalità di input/output vengono implementate come strategie indipendenti (SalvataggioCSV, SalvataggioJSON, ecc.), tutte aderenti a un'interfaccia comune. In questo modo, la logica principale della libreria non dipende da un formato specifico, ma delega l'operazione alla strategia appropriata, scelta dinamicamente in base al contesto o alle preferenze dell'utente.



- per l'ordinamento del catalogo:
Il pattern **Strategy** è stato adottato anche per la gestione degli ordinamenti applicabili ai libri della libreria. La necessità di ordinare i dati secondo criteri differenti (ad esempio per titolo, per autore, per valutazione o per genere) richiede un approccio che mantenga la logica di ordinamento modulare e facilmente estendibile.

Ogni criterio di ordinamento è implementato come una strategia distinta che aderisce a un'interfaccia comune. In questo modo, l'algoritmo di ordinamento utilizzato dalla libreria rimane indipendente dal criterio specifico, delegando alla strategia selezionata la responsabilità di stabilire la modalità di confronto tra i libri.

Questa scelta progettuale consente di aggiungere o modificare criteri di ordinamento senza incidere sulla logica principale del sistema, garantendo così un'architettura **flessibile, riusabile e manutenibile**, in grado di adattarsi con semplicità alle esigenze dell'utente.



-Facade e GUI: Il pattern **Facade** è stato utilizzato per centralizzare e semplificare l'accesso alle numerose funzionalità offerte dalla libreria, come l'aggiunta, la rimozione, la ricerca, il salvataggio e il caricamento dei libri. Senza un'interfaccia unificata, la gestione diretta di tutte queste operazioni da parte della GUI comporterebbe un elevato grado di complessità e un forte accoppiamento tra la logica applicativa e la presentazione.

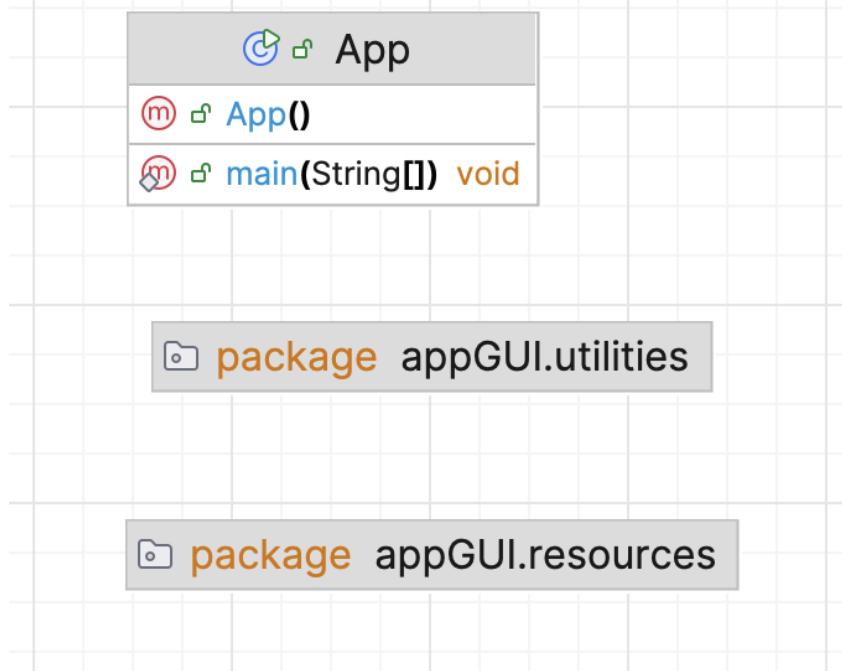
Attraverso il Facade (GestioneLibreria), la GUI interagisce con un'unica interfaccia semplificata, senza doversi preoccupare dei dettagli interni o delle dipendenze tra i diversi componenti della libreria. Ciò consente di ridurre la complessità della GUI, migliorare la leggibilità del codice e favorire la manutenzione futura.

Inoltre, l'uso del Facade garantisce una maggiore **modularità e disaccoppiamento**, poiché eventuali modifiche alla logica interna della libreria

possono essere gestite senza impattare direttamente la struttura della GUI, rendendo il sistema più stabile e coerente dal punto di vista architetturale.



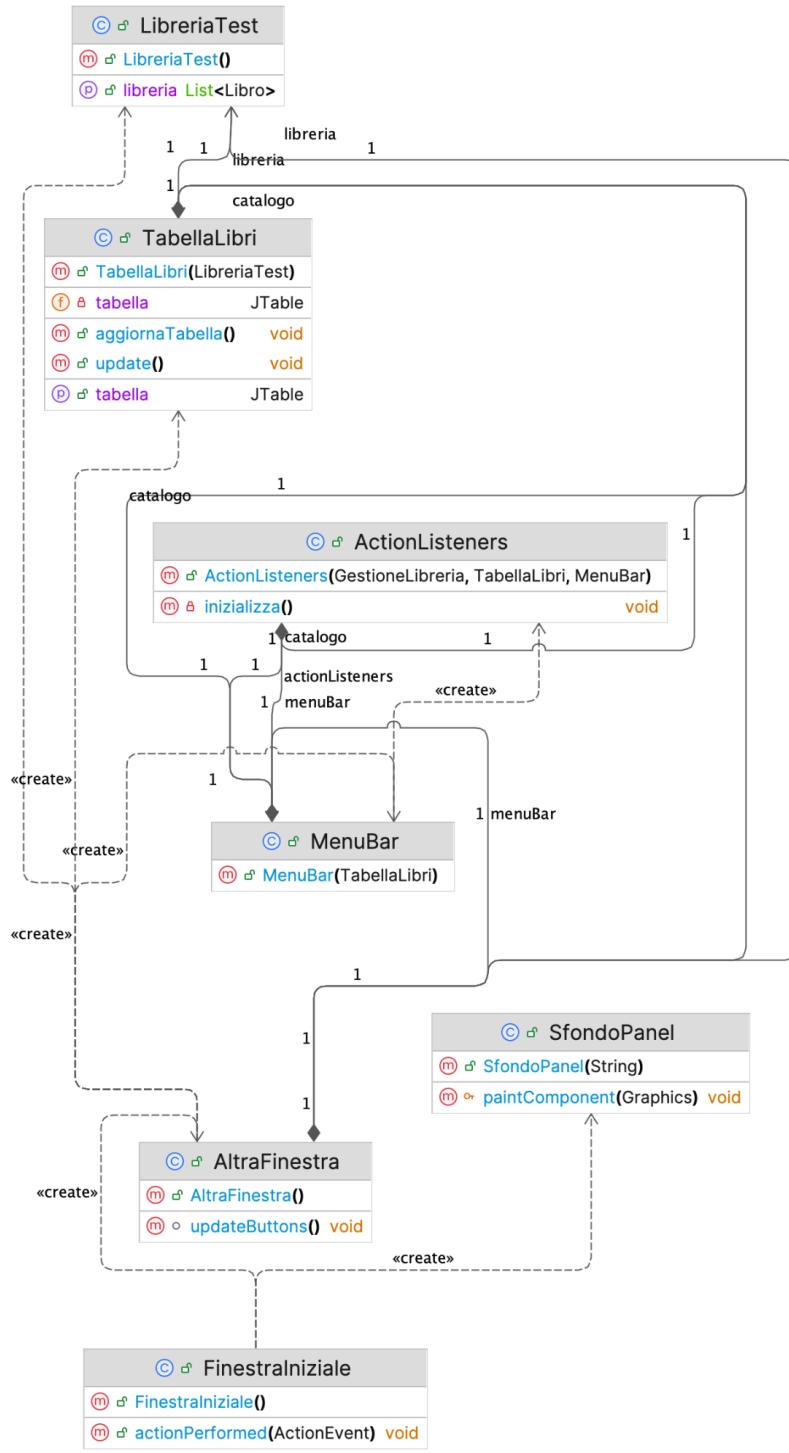
La **GUI**(Graphic User Interface) è stata strutturata in questo modo:



A partire dal package AppGUI è possibile trovare la classe App, la quale in maniera molto intuitiva e semplice permette all'utente di accedere direttamente al programma senza vedere nessuna implementazione interna.

Gli altri package invece rappresentano il “dietro le quinte” per la costruzione dell’interfaccia utente:

- il package resources, come si evince dal titolo, contiene tutti i file di “risorsa” utili per la decorazione del programma(sfondi, icone...);
- il package utilities è la parte fondamentale: al suo interno sono presenti tutte le classi con le loro rispettive funzioni che svolgono nell’interfaccia. Al suo interno è contenuta anche una classe chiamata LibreriaTest la quale rappresenta una lista pronta di libri da poter utilizzare come test del programma;

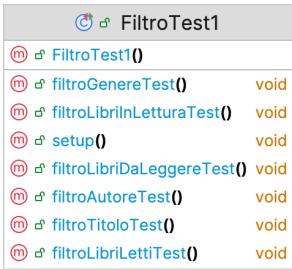


Oltre a tutte le funzionalità del programma, ai fini di manutenibilità e affidabilità è stato implementato un package dedicato contenente le procedure di testing. Per ogni funzionalità principale del sistema è stata creata una classe di test, la quale include una libreria di prova con alcuni volumi predefiniti e metodi di test specifici per verificare il corretto funzionamento delle operazioni associate (come aggiunta, rimozione, modifica, ordinamento e filtraggio dei libri, caricamento e salvataggio dei dati, e gestione dello stato di lettura).

I test sono stati realizzati utilizzando **JUnit 5** in ambiente **IDE IntelliJ**, garantendo l'esecuzione automatica e ripetibile delle verifiche. Questa struttura consente di individuare rapidamente eventuali regressioni o malfunzionamenti, assicurando una maggiore **stabilità, manutenibilità e affidabilità** del sistema durante lo sviluppo e l'evoluzione futura.



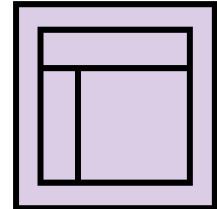
Rispettivamente nel package StrategyTest avremo:



G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

<Report in this section how the architectural and low level design you produced satisfies the FRs and the NFRs>

Il progetto di **Gestione Libreria** è stato sviluppato tenendo conto sia dei requisiti funzionali, cioè le funzionalità che l'utente deve poter utilizzare, sia dei requisiti non funzionali, che riguardano la qualità e l'organizzazione del sistema.



Requisiti funzionali:

- **Gestione dei libri**

Il sistema permette di aggiungere, modificare e rimuovere libri in maniera semplice e intuitiva. L'aggiunta avviene tramite la creazione di un oggetto Libro con il pattern **Builder** e il suo inserimento nella libreria attraverso il Facade, che coordina l'aggiornamento della lista interna e della GUI. La modifica dei libri è gestita da appositi pannelli e ActionListener che aggiornano sia la struttura dati sia la visualizzazione, mentre la rimozione garantisce la cancellazione completa e la notifica agli observer.

- **Filtri e ricerche**

L'utente può filtrare i libri in base a valutazione, genere o stato di lettura grazie al pattern **Strategy** e **Chain of Responsibility**, che consente di applicare dinamicamente strategie diverse senza modificare il codice principale. Questo rende i filtri flessibili e facilmente estendibili.

- **Ordinamento dei libri**

Il sistema supporta l'ordinamento dei libri per titolo, autore o valutazione, permettendo all'utente di visualizzare la lista in base alle proprie esigenze.

- **Persistenza dei dati**

I libri possono essere salvati e caricati da file CSV, mantenendo intatte tutte le informazioni principali (titolo, autore, ISBN, genere, valutazione, stato di lettura). In questo modo, i dati non vengono persi tra una sessione e l'altra.

- **Interfaccia utente**

La GUI permette di visualizzare i libri in una tabella aggiornata in tempo reale grazie al pattern **Observer**, migliorando l'interattività. L'utente riceve conferme tramite JOptionPane per le operazioni di aggiunta, modifica o eliminazione, aumentando la chiarezza e l'usabilità.

Requisiti non funzionali:

- **Manutenibilità**

La struttura modulare del progetto, organizzata in pacchetti e classi dedicate, insieme all'uso dei pattern di progettazione (Facade, Strategy, Observer, Command), facilita l'implementazione di nuove funzionalità senza alterare il codice esistente.

- **Scalabilità**

Il sistema è progettato per gestire un numero crescente di libri senza perdere efficienza. Filtri, ordinamenti e operazioni sulle liste sono implementati in modo dinamico, permettendo di adattarsi facilmente a volumi maggiori di dati.

- **Usabilità**

La GUI è chiara e intuitiva: i pulsanti, gli spinner e le finestre di conferma guidano l'utente nelle operazioni principali senza necessità di conoscere il funzionamento interno del sistema.

- **Affidabilità**

La persistenza dei dati tramite file CSV garantisce che le informazioni non vadano perse. Le operazioni di lettura e scrittura gestiscono correttamente eventuali errori, assicurando un comportamento stabile del programma.

- **Portabilità**

Essendo realizzato in Java puro, il progetto può essere eseguito su qualsiasi macchina con JVM compatibile, senza dipendere da librerie esterne complesse.

Appendix. Prototype

<Provide a brief report on your prototype, and especially: information on what you have implemented, how the implementation covers the FR and NFR, how the prototypes demonstrates your project correctness with respect to the FR and NFR. You may add some screenshots to describe what required above. Be ready to show your prototype during the oral examination>

Si allegano qui di seguito, screenshots con lo scopo di illustrare il programma dal punto di vista dell'utente che accede alla sua libreria personale.



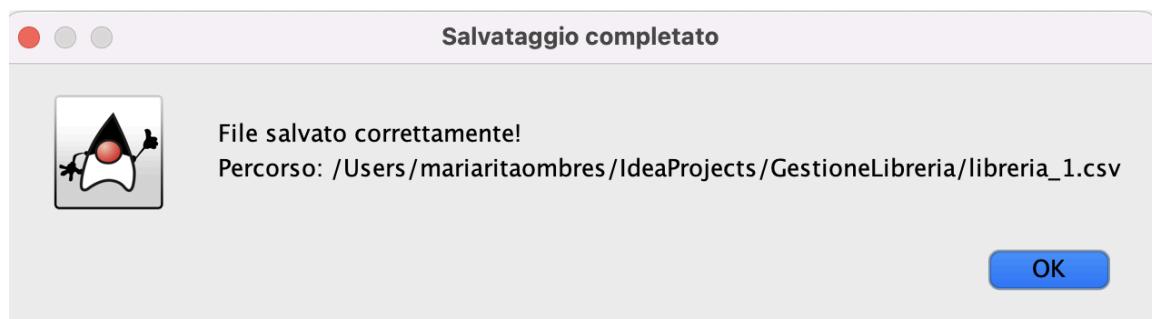
Esploriamo il catalogo:

Titolo	Autore	Genere	Valutazione	Stato lettura	ISBN
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto	9788807898624
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0	inLettura	9788807897453
Fu sera e fu mat...	Ken Follett	Storico	3	letto	9788804752370

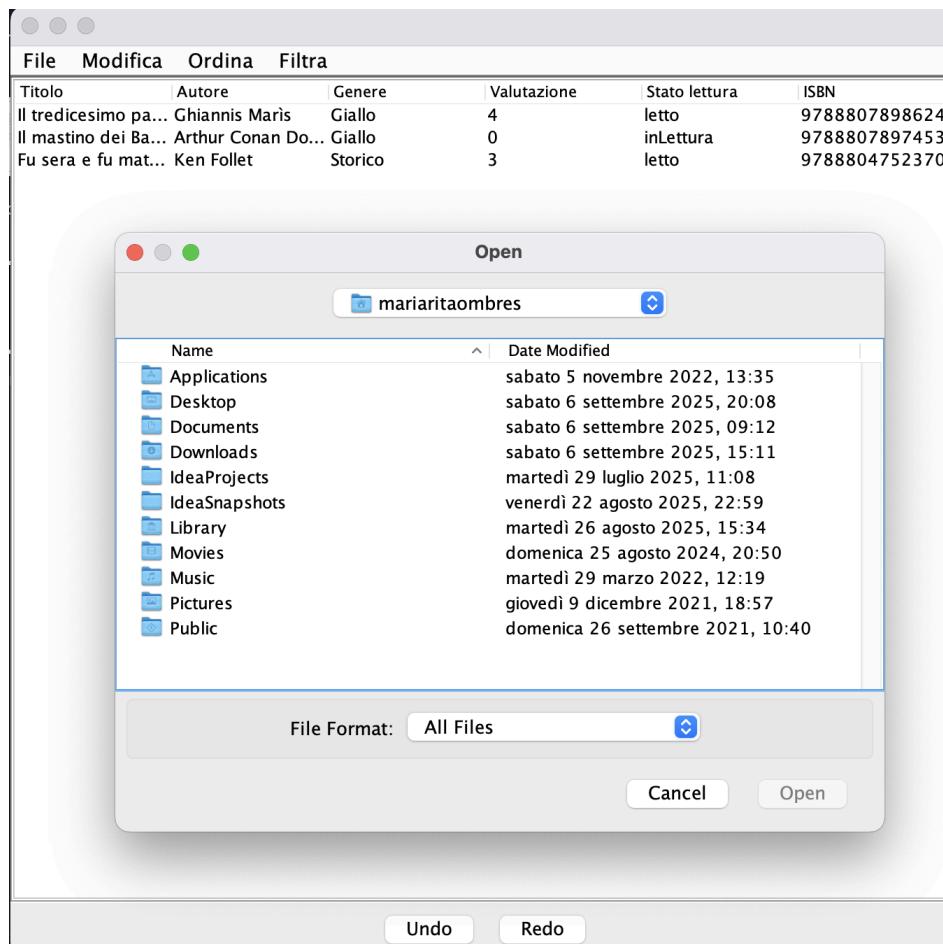
Esempio: salvataggio file

Salva come: ► CSV
Carica da file: ► ... Conan Do...
Modifica Ordina Filtra Esci

(stessa cosa per carica da file:)



Proviamo a caricare un file:



(Ovviamente accetta solo il formato scelto durante la selezione per il caricamento)

Proviamo a modificare un libro:

Titolo	Modifica libro	Genere	Valutazione	Stato lettura	ISBN
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto	9788807898624
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0	inLettura	9788807897453
Fu sera e fu mattina	Rimuovi libro	Storico	3	letto	9788804752370

Modifica libro

Titolo:
Fu sera e fu mattina

Autore:
Ken Follett

ISBN:
9788804752370

Genere:
Storico

Stato:
letto

Valutazione:
3

Cancel OK

nel caso in cui non avessimo selezionato nessun libro:

Titolo	Autore	Genere	Valutazione	Stato
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0	inLettura
Fu sera e fu mat...	Ken Follett	Storico	3	letto



Proviamo ad aggiungere e rimuovere un volume:

Titolo	Autore	Genere	Valutazione
Il tredicesimo pa...	Ghannis Maris	Giallo	4
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0
Fu sera e fu mat...	Ken Follet	Storico	3



Nella rimozione, come nella modifica, se non selezioniamo nessun libro il programma ci segnalera' errore.

Ricordo che in questo caso i pulsanti di undo/redo ci vengono utili dato che regolano lo stato della libreria e quindi l'aggiunta, la modifica e la rimozione di un libro e non le operazioni che si effettuano nell'applicazione (come tornare indietro dopo una ricerca dei volumi/ordinamento libri...)

Esempio ordinamento: per titolo

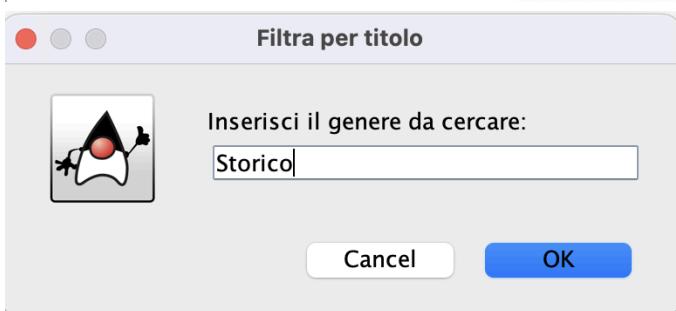
File	Modifica	Ordina	Filtra	Titolo	Autore	Valutazione	Stato lettura	ISBN
Il tredicesimo pa...	Ghannis Maris	Giallo		Il tredicesimo pa...	Ghannis Maris	Giallo	letto	9788807898624
Il mastino dei Ba...	Arthur Conan Do...	Giallo		Il mastino dei Ba...	Arthur Conan Do...	Giallo	inLettura	9788807897453
Fu sera e fu mat...	Ken Follet	Storico		Fu sera e fu mat...	Ken Follet	Storico	letto	9788804752370

Il catalogo diventa:

Titolo	Autore	Genere	Valutazione	Stato lettura	ISBN
Fu sera e fu mat...	Ken Follet	Storico	3	letto	9788804752370
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0	inLettura	9788807897453
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto	9788807898624

O filtraggio per genere:

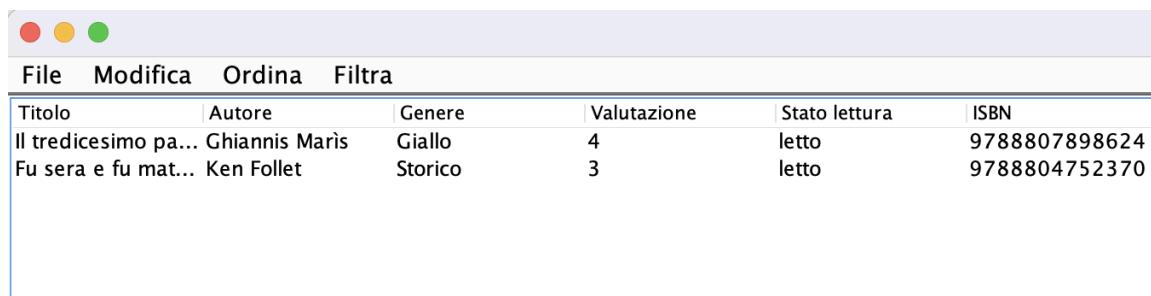
Titolo	Autore	Genere	Valutazione	Stato lettura	ISBN
Fu sera e fu mat...	Ken Follet	Storico	3	letto	9788804752370
Il mastino dei Ba...	Arthur Conan Do...	Giallo	0	inLettura	9788807897453
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto	9788807898624



Titolo	Autore	Genere	Valutazione	Stato lettura	ISBN
Fu sera e fu mat...	Ken Follet	Storico	3	letto	9788804752370

o per valutazione:





A screenshot of a software application window. At the top is a menu bar with "File", "Modifica", "Ordina", and "Filtra". Below the menu is a table with the following data:

Titolo	Autore	Genere	Valutazione	Stato lettura	ISBN
Il tredicesimo pa...	Ghiannis Maris	Giallo	4	letto	9788807898624
Fu sera e fu mat...	Ken Follett	Storico	3	letto	9788804752370

Una volta terminato l'utilizzo possiamo uscire tramite File>Esci



Il quale, come la schermata iniziale, ci chiederà la conferma per uscire.