Sistema Gestionale per Biblioteca

Documento di Progettazione

---

1. Come è Strutturato il Sistema

I Tre Moduli Principali

Abbiamo organizzato il progetto dividendolo in tre parti ben distinte, così è più facile lavorarci e modificarlo in futuro.

Il **Modulo Model** contiene tutti i dati e le regole del sistema. Dentro ci sono le classi che rappresentano i libri, gli utenti e i prestiti. La classe **BibliotecaManager** è quella che coordina tutto: gestisce le liste di dati in memoria e si occupa di salvarli su file quando serve.

Il **Modulo View** è tutta la parte visiva. Abbiamo usato file FXML per disegnare le schermate principali, mentre la classe **Finestre** crea al volo le finestre popup quando bisogna inserire dati nuovi (tipo quando aggiungi un libro o registri un utente).

Il **Modulo Controller** fa da tramite tra quello che vede l'utente e i dati veri e propri. La classe **interfacciaController** cattura i click dell'utente e chiama i metodi giusti per fare le operazioni richieste.

Perché abbiamo scelto questa struttura (Pattern MVC)

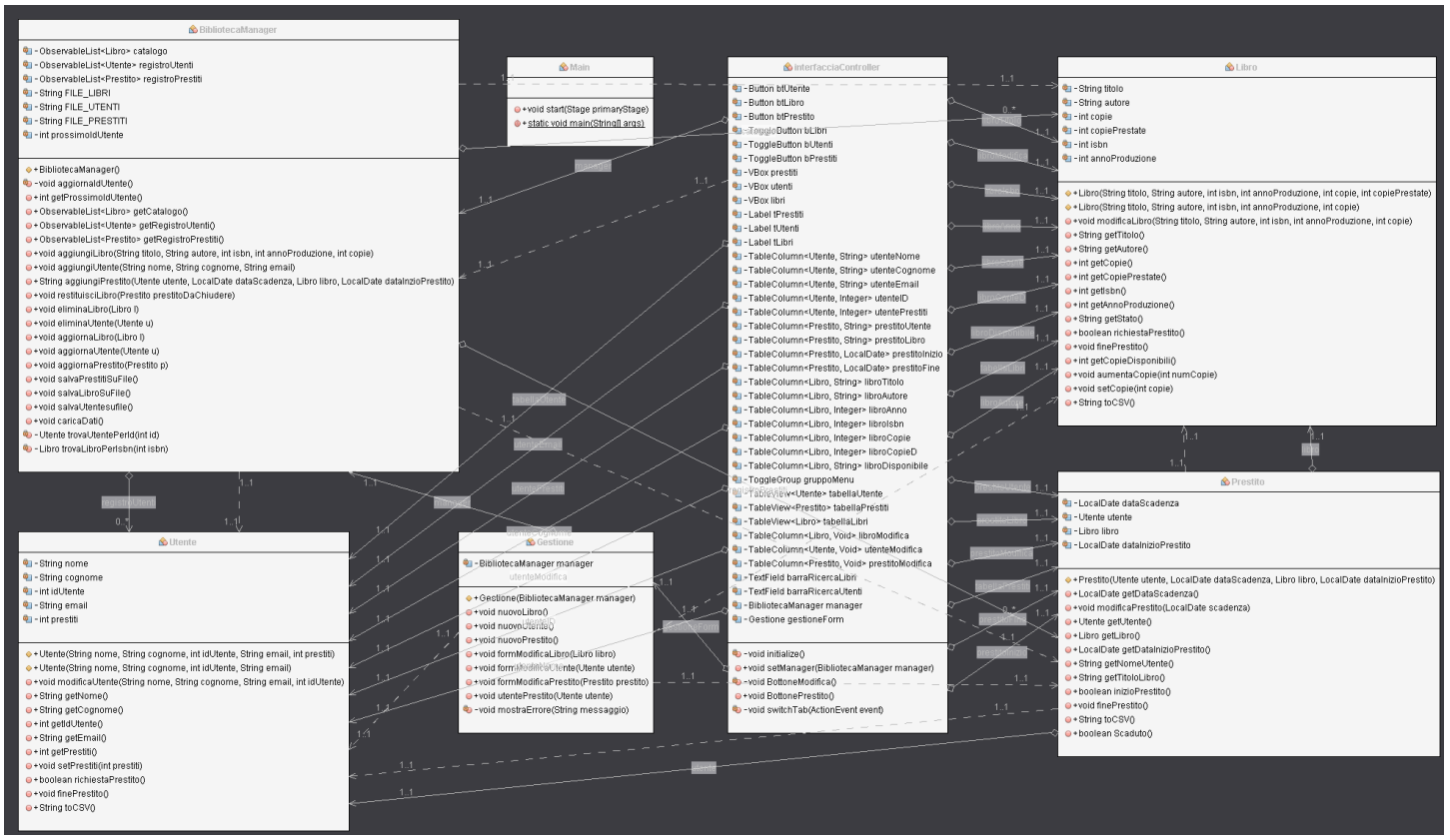
Abbiamo seguito il pattern **MVC** perché è perfetto per JavaFX. Funziona così:

- Il **Model** (**Libro**, **Utente**, **Prestito**, **BibliotecaManager**) tiene tutti i dati e controlla che le regole siano rispettate. Per esempio, **BibliotecaManager** controlla che non ci siano ISBN duplicati e assegna automaticamente gli ID agli utenti.

- La **View** è descritta nei file FXML per le schermate fisse, mentre la classe **Finestre** gestisce i popup.

- Il **Controller** (**interfacciaController**) prende le liste dalla **BibliotecaManager** e le mostra a schermo nelle tabelle, e richiama i metodi appositi quando l'utente fa qualcosa nell'interfaccia.

2. Modello Statico



BibliotecaManager

Questa è la classe "principale". Gestisce tre liste osservabili (**catalogo**, **registroUtenti** e **registroPrestiti**). Poiché le liste sono osservabili, quando cambio qualcosa nella lista essa apparirà subito anche a schermo.

Quando avvii il programma, carica tutti i dati dai file CSV. Poi, ogni volta che aggiungi o modifichi qualcosa, salva automaticamente tutto su disco.

Il metodo **aggiungiLibro()** è furbo: se il libro esiste già, invece di fare un duplicato aumenta semplicemente il numero di copie. Il metodo **aggiungiPrestito()** invece controlla tutto quello che serve prima di registrare un prestito.

Libro

Ogni libro ha le informazioni base (titolo, autore, ISBN, anno) che non cambiano mai, più due contatori importanti: quante copie totali abbiamo e quante sono attualmente in prestito.

Il metodo **richiestaPrestito()** controlla se ci sono copie disponibili prima di permettere il prestito.

Utente

Rappresenta una persona iscritta. Oltre ai dati personali (nome, cognome, email), ogni utente ha un ID unico e un contatore che tiene traccia di quanti libri ha in prestito.

Abbiamo messo un limite di 3 prestiti contemporanei per utente, gestito dal metodo **richiestaPrestito()**.

Prestito

Questa classe collega un utente a un libro per un periodo specifico. Registra quando inizia il prestito e quando scade, mantenendo i riferimenti diretti sia all'utente che al libro.

interfacciaController

Si occupa di far funzionare la schermata principale. Collega le colonne delle tabelle ai dati delle classi (usando **PropertyValueFactory**) e gestisce quando l'utente clicca sulle schede in alto.

Gestione/Finestre

Crea tutti i form per inserire dati nuovi. Prima di passare le informazioni al Manager, controlla che tutto sia ok: campi non vuoti, email valida, numeri scritti correttamente.

Scelte di Design

Abbiamo cercato di tenere ogni classe focalizzata su una cosa sola. Per esempio, tutta la validazione degli input sta nella classe **Finestre/Gestione**, così il controller principale rimane pulito. Allo stesso modo, solo **BibliotecaManager** sa come funzionano i file CSV.

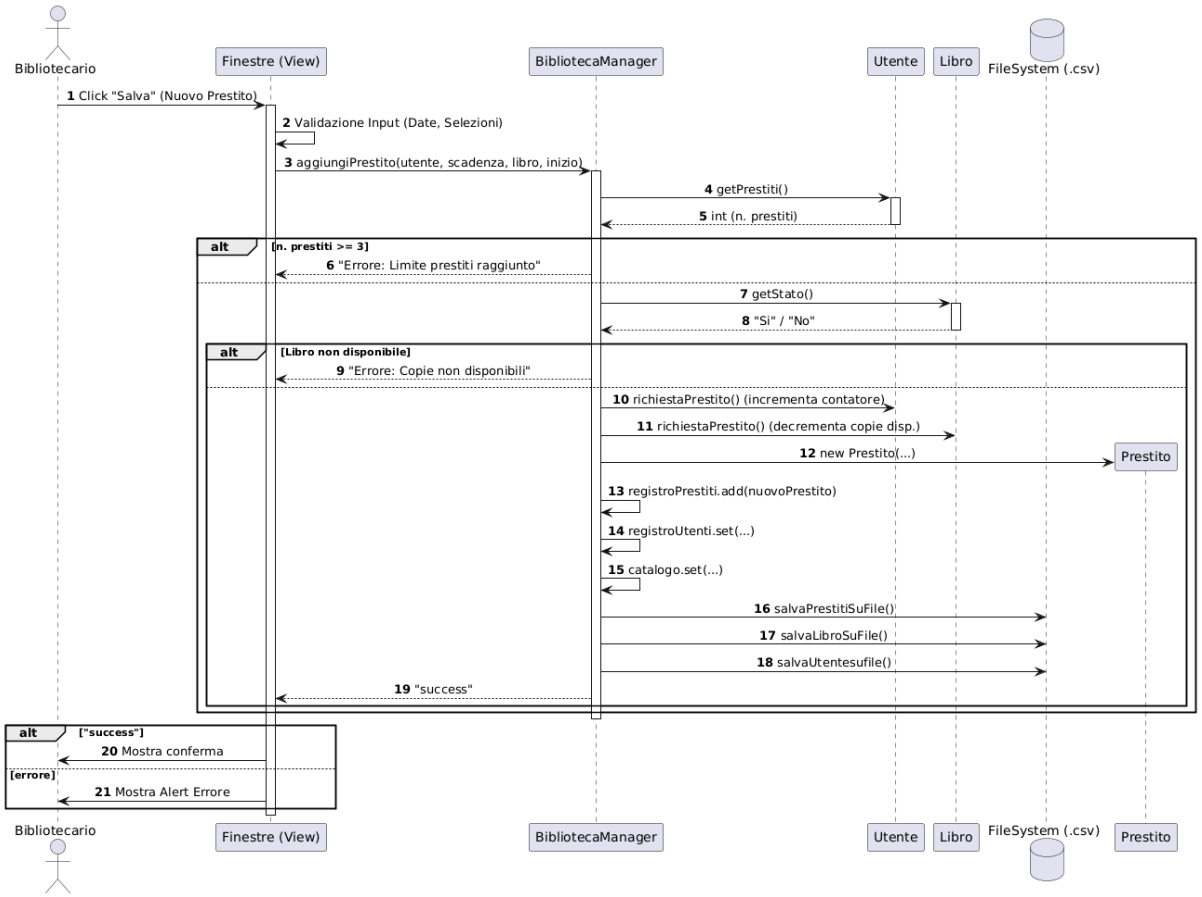
Le classi sono poco dipendenti tra loro: il controller usa solo i metodi pubblici del Manager, senza sapere come funziona internamente. Le entità (**Libro**, **Utente**) sono classi semplici che potrebbero essere usate anche in altri progetti.

Abbiamo usato le **ObservableList** perché aggiornano automaticamente le tabelle quando cambiano i dati, senza dover ricaricare manualmente tutto ogni volta.

Tutti gli attributi delle classi sono privati e si accede solo tramite getter/setter o metodi specifici. Questo protegge i dati da modifiche sbagliate (per esempio impedisce che le copie in prestito diventino negative).

3. Modello Dinamico

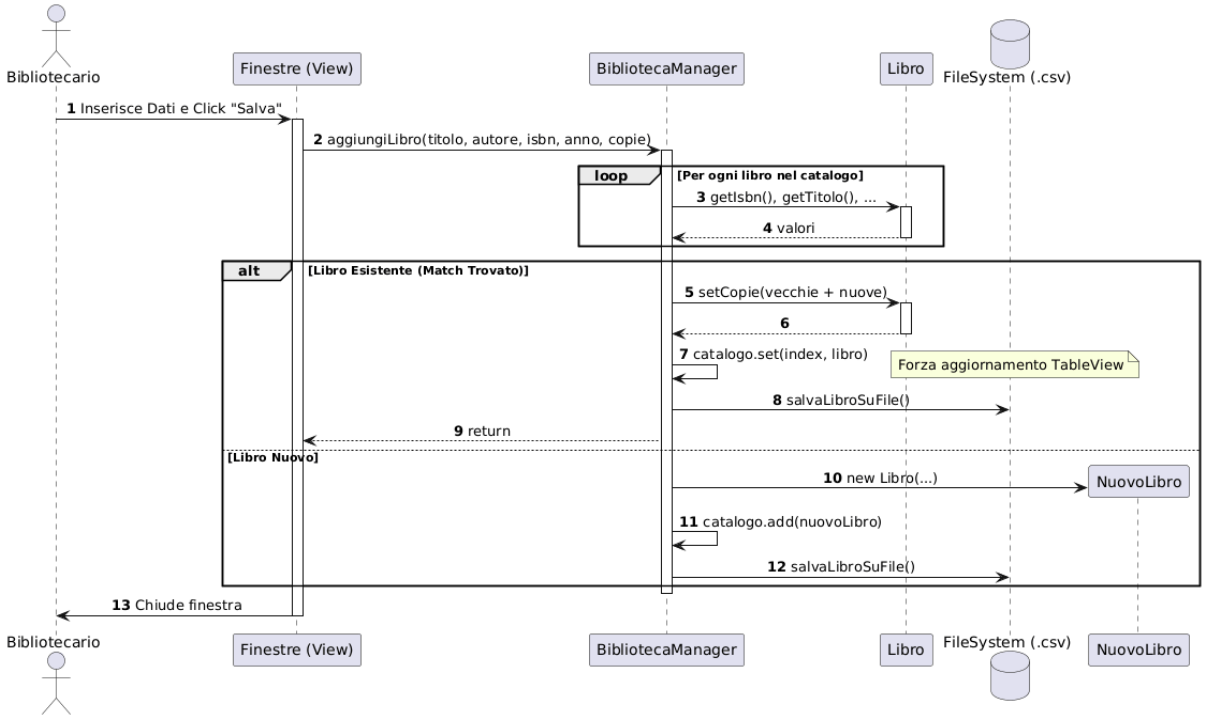
Registrare un Nuovo Prestito



Quando il bibliotecario vuole registrare un prestito, succede questo:

1. La classe **Finestre** raccoglie i dati (quale utente, quale libro)
2. **BibliotecaManager** fa i controlli necessari:
    - Chiede all'**Utente** quanti prestiti ha già attivi
    - Chiede al **Libro** se ci sono copie disponibili
3. Se va tutto bene, aggiorna i dati:
    - Aumenta il contatore prestiti dell'utente
    - Diminuisce le copie disponibili del libro
    - Crea il nuovo oggetto **Prestito**
4. Forza l'aggiornamento delle tabelle nell'interfaccia
5. Salva tutto sui file CSV

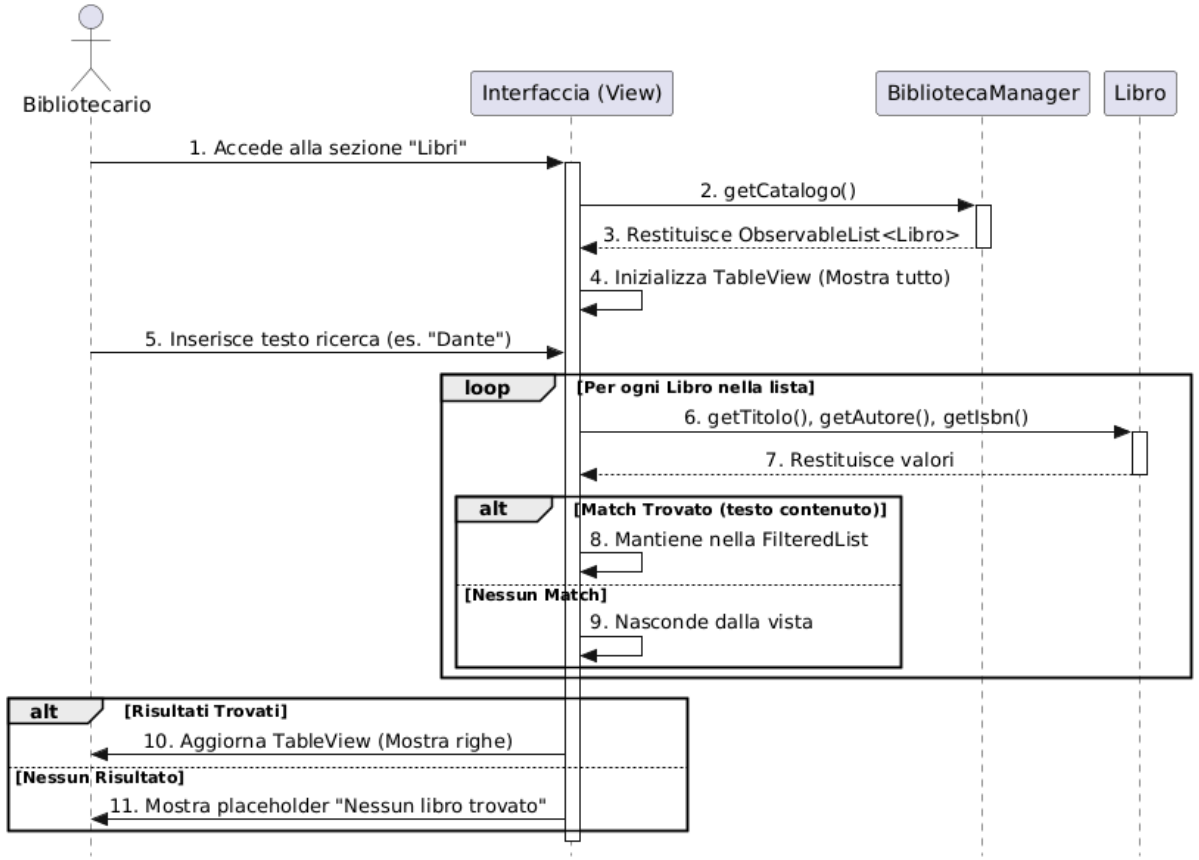
Aggiungere un Libro



Il metodo **aggiungiLibro()** è intelligente perché gestisce i duplicati:

1. Prima cerca nel catalogo se quel libro esiste già (confronta titolo, autore, ISBN)
2. **Se lo trova**: non crea un duplicato, ma aumenta solo il numero di copie e aggiorna la riga nella tabella
3. **Se non lo trova**: crea un nuovo libro e lo aggiunge alla lista
4. In entrambi i casi, alla fine salva tutto su file

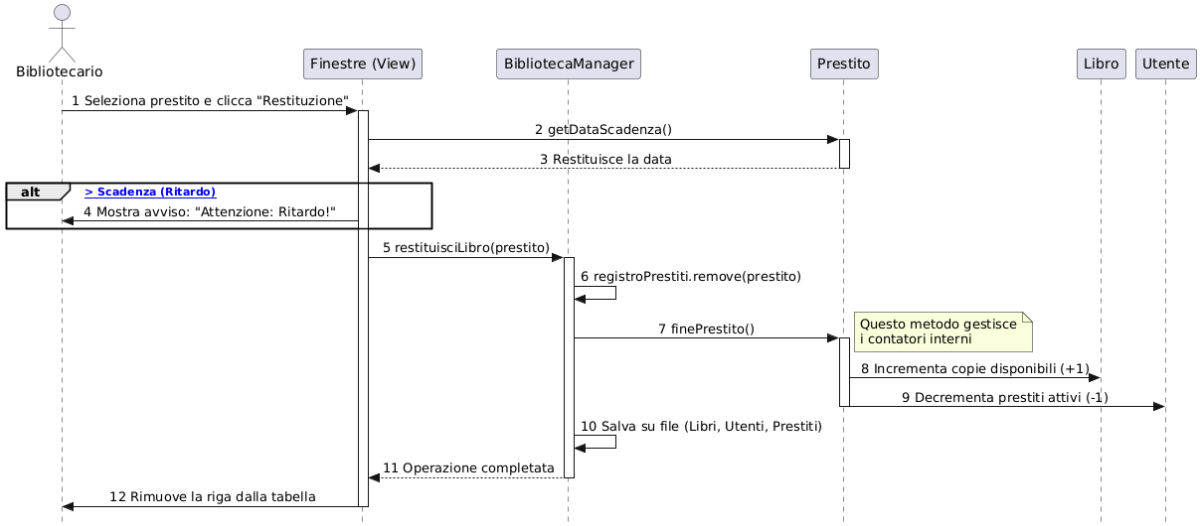
Ricerca di un libro



Quando il bibliotecario inizia a scrivere qualcosa nella barra di ricerca (per esempio "Dante"), succede questo:

1. L'interfaccia grafica chiede al **BibliotecaManager** l'elenco completo di tutti i libri presenti tramite il metodo **getCatalogo()**. Questo serve per avere la lista aggiornata su cui lavorare.
2. Il sistema non ha bisogno di ricaricare il file CSV ogni volta che scrivi una lettera. Lavora direttamente sulla lista che ha in memoria (la ObservableList), che è molto più veloce.
3. Per ogni libro della lista, il programma controlla se le parole scritte corrispondono al Titolo, all'Autore o al codice ISBN. Per farlo usa i metodi getter della classe Libro (come **getTitolo()** o **getIsbn()**).
4. Se trova una corrispondenza, il libro rimane visibile in tabella.
5. Se invece la ricerca non dà risultati (nessun libro contiene quelle parole), la tabella si svuota e appare il messaggio "Nessun libro trovato" per avvisare l'utente.

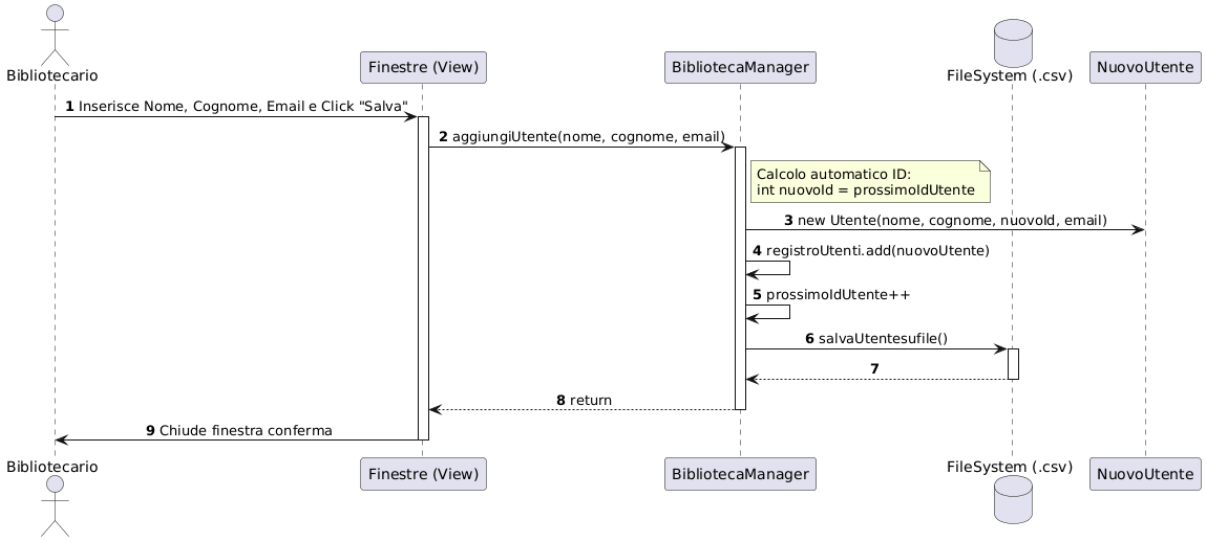
Restituzione Libro (prestito)



Quando un utente riporta un libro in biblioteca:

1. **Controllo preliminare:** Appena il bibliotecario clicca sul tasto di restituzione, l'interfaccia controlla subito la data di scadenza del prestito. Se la data di oggi ha superato la scadenza, appare un avviso (pop-up) che dice "Attenzione: Libro restituito in ritardo", così il bibliotecario lo sa.
2. **Chiamata al Manager:** Se si procede, l'interfaccia passa il comando **restituisciLibro** al **BibliotecaManager**.
3. **Aggiornamento Dati:** Il Manager fa tre cose importanti:
    - Toglie quel prestito dalla lista dei prestiti attivi (così sparisce dalla tabella).
    - Chiama il metodo **finePrestito** sull'oggetto stesso. Questo è un metodo che va automaticamente ad aumentare di 1 le copie disponibili del libro (perché è tornato sullo scaffale) e diminuisce di 1 il numero di libri che quell'utente ha in carico.
4. **Salvataggio:** Alla fine, il Manager sovrascrive tutti i file CSV (Libri, Utenti e Prestiti) per essere sicuro che queste modifiche restino salvate anche se si chiude il programma.

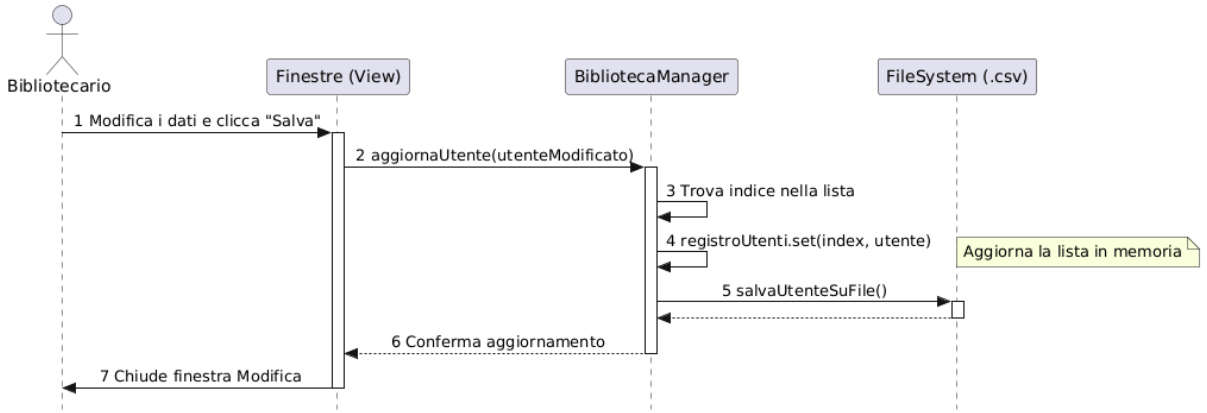
Registrare un Nuovo Utente



Quando registri un utente nuovo:

1. **Finestre** raccoglie nome, cognome ed email
2. Il controller passa i dati a **BibliotecaManager**
3. Il Manager calcola automaticamente un ID univoco (usando un contatore che parte dall'ID più alto già esistente)
4. Crea il nuovo oggetto **Utente** con quell'ID
5. Lo aggiunge alla lista (la tabella si aggiorna da sola)
6. Salva la lista aggiornata su file

Modifica Utente già esistente

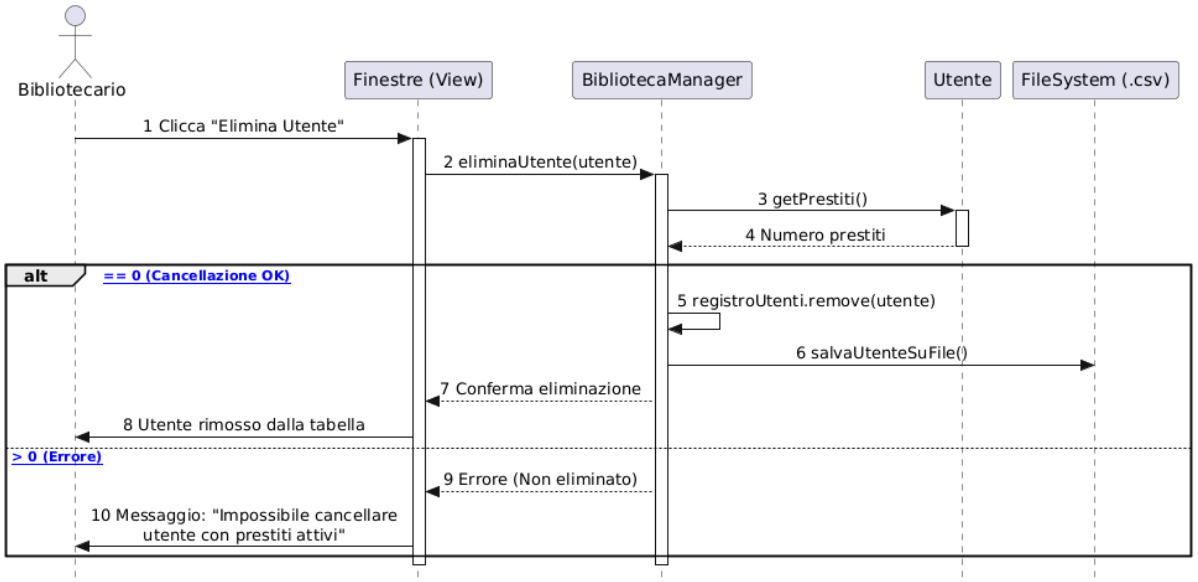


Quando il bibliotecario deve correggere i dati di una persona:

1. Dalla schermata di modifica, l'interfaccia raccoglie i nuovi dati inseriti e li passa al **BibliotecaManager** tramite il metodo **aggiornaUtente**.

2. Il Manager cerca dove si trova quell'utente specifico all'interno della lista (il registroUtenti) e sostituisce la vecchia versione con quella nuova modificata.
3. Subito dopo, chiama il metodo **salvaUtenteSuFile** per sovrascrivere il file CSV. Così, anche se chiudiamo e riapriamo il programma, le modifiche rimangono salvate.

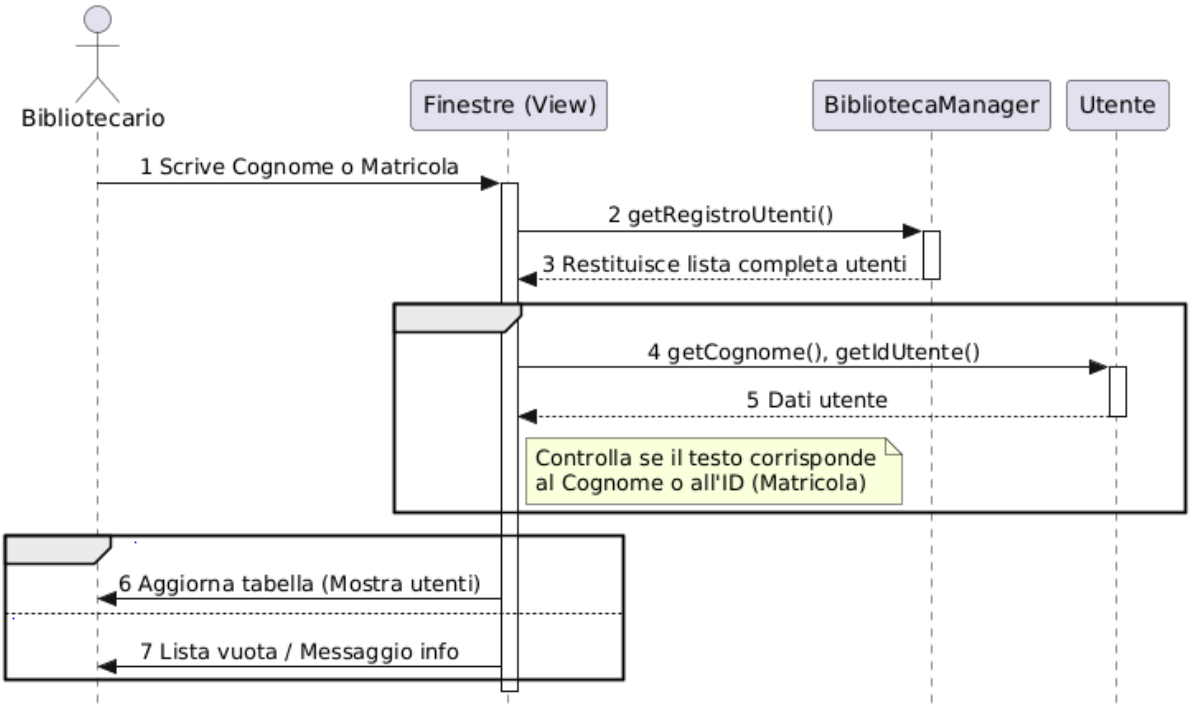
Cancellazione Utente



L'eliminazione di un utente è un'operazione delicata, quindi il sistema fa un controllo prima di procedere:

1. Quando si clicca su elimina, il **BibliotecaManager** riceve il comando **eliminaUtente**.
2. Prima di cancellare, il Manager "interroga" l'oggetto **Utente** chiedendo: "Quanti libri hai in prestito?" (usando il metodo **getPrestiti**).
3. **Scenario A (Nessun prestito):** Se la risposta è 0, l'utente viene rimosso dalla lista e il file CSV viene aggiornato. L'utente sparisce dalla tabella.
4. **Scenario B (Ha prestiti):** Se l'utente ha ancora dei libri da restituire (prestiti > 0), il Manager blocca tutto. Non cancella nulla e dice all'interfaccia di mostrare un
5. messaggio di errore. Questo evita di perdere traccia dei libri che sono ancora in giro.

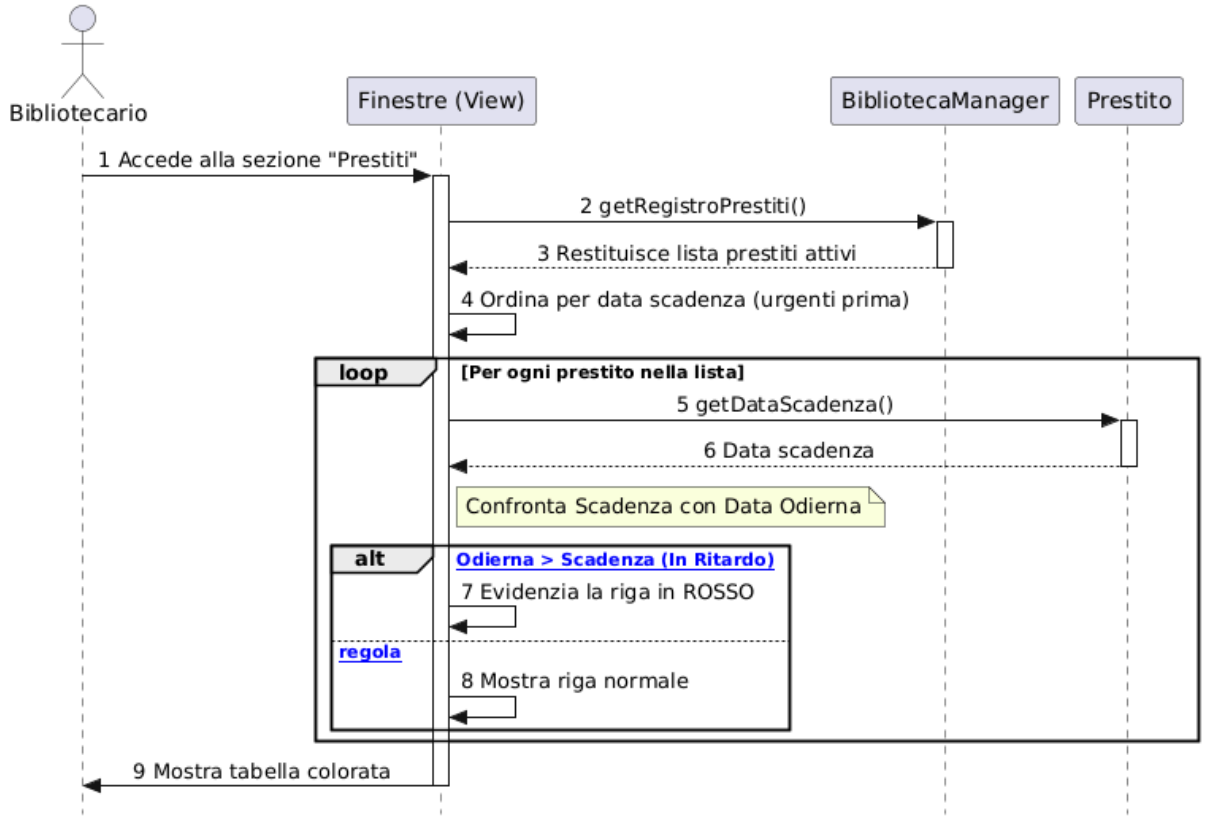
Ricerca di un Utente



Quando il bibliotecario ha bisogno di trovare un utente, il funzionamento è questo:

1. Il bibliotecario inizia a digitare nel campo di ricerca il cognome della persona o il suo numero di matricola (l'ID univoco).
2. L'interfaccia grafica si rivolge al **BibliotecaManager** e chiama il metodo **getRegistroUtenti()**. Questo serve per farsi dare l'elenco completo di tutte le persone registrate fino a quel momento.
3. Il sistema analizza uno per uno gli utenti della lista. Per ogni utente, usa i metodi come **getCognome()** o **getIdUtente()** per vedere se c'è una corrispondenza con quello che il bibliotecario sta scrivendo.
4. Man mano che scrivi, la tabella sullo schermo elimina automaticamente i nomi che non c'entrano nulla, lasciando visibili solo le persone che corrispondono alla ricerca.
5. Se cerchi una persona che non esiste, la tabella diventa vuota per farti capire che non ci sono risultati.

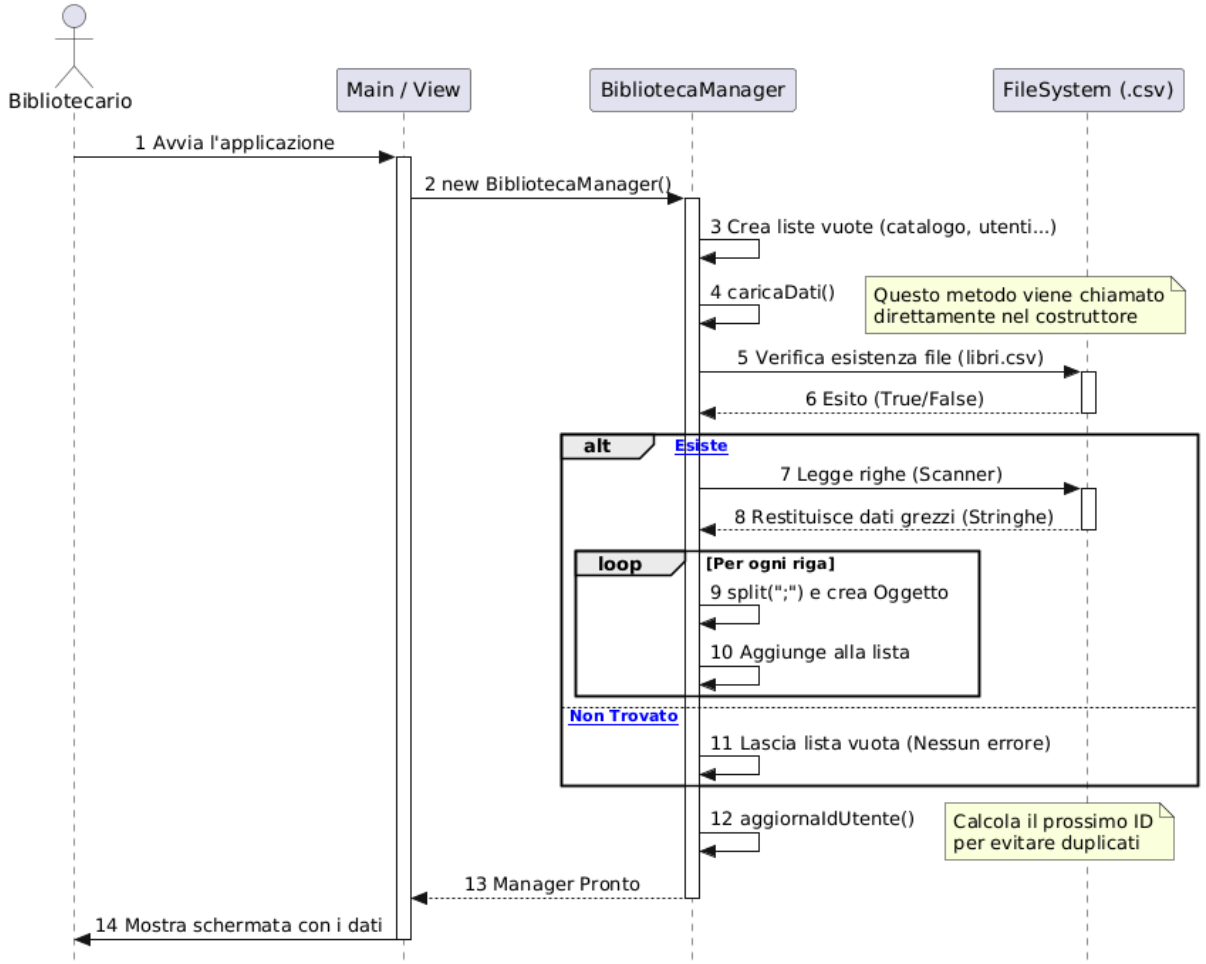
Visualizzazione Prestiti e Ritardi



Quando il bibliotecario apre la schermata dei prestiti per controllare la situazione, il programma fa questo:

1. L'interfaccia chiede al **BibliotecaManager** l'elenco di tutti i prestiti che sono ancora aperti (quelli che non sono stati restituiti) usando il metodo **getRegistroPrestiti()**.
2. Appena ricevuta la lista, il programma la riordina automaticamente, mettendo in cima quelli che scadono prima, così sono più visibili.
3. Poi inizia un controllo automatico, riga per riga. Per ogni prestito, il programma guarda la data di scadenza (tramite **getDataScadenza**) e la confronta con la data di oggi.
4. Se scopre che la data di oggi è successiva alla scadenza (quindi il libro doveva essere già tornato), colora quella riga della tabella di rosso.
5. Alla fine mostra al bibliotecario la tabella completa: a colpo d'occhio, tutte le righe rosse sono i libri in ritardo su cui bisogna intervenire.

Caricamento dati all’ avvio



Cosa succede all’avvio:

1. Quando si lancia l'app, la prima cosa che viene creata è il **BibliotecaManager**.
2. Nel momento stesso in cui nasce (nel suo costruttore), il Manager esegue subito il metodo **caricaDati()**.
3. Per ogni tipo di dato (Libri, Utenti, Prestiti), il sistema controlla se esiste il file CSV nella cartella.
4. **Se il file c'è:** Il programma usa uno Scanner per leggere il file riga per riga. Ogni riga viene "spezzata" dove c'è il punto e virgola (lo split) e trasformata in un oggetto vero e proprio (un Libro o un Utente) che viene messo nella lista.
5. **Se il file non c'è (es. primo avvio):** Il programma non si blocca e non dà errori. Semplicemente lascia le liste vuote, pronto per il primo inserimento.
6. Una cosa importante che fa alla fine è **aggiornaIdUtente()**: il sistema controlla qual è l'ID più alto tra gli utenti caricati (es. 5) e si prepara a dare il numero successivo (6) al prossimo nuovo iscritto.
7. Solo quando tutto questo è finito, l'interfaccia appare a video con tutti i dati già pronti.
8.

4. L'Interfaccia Grafica