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)

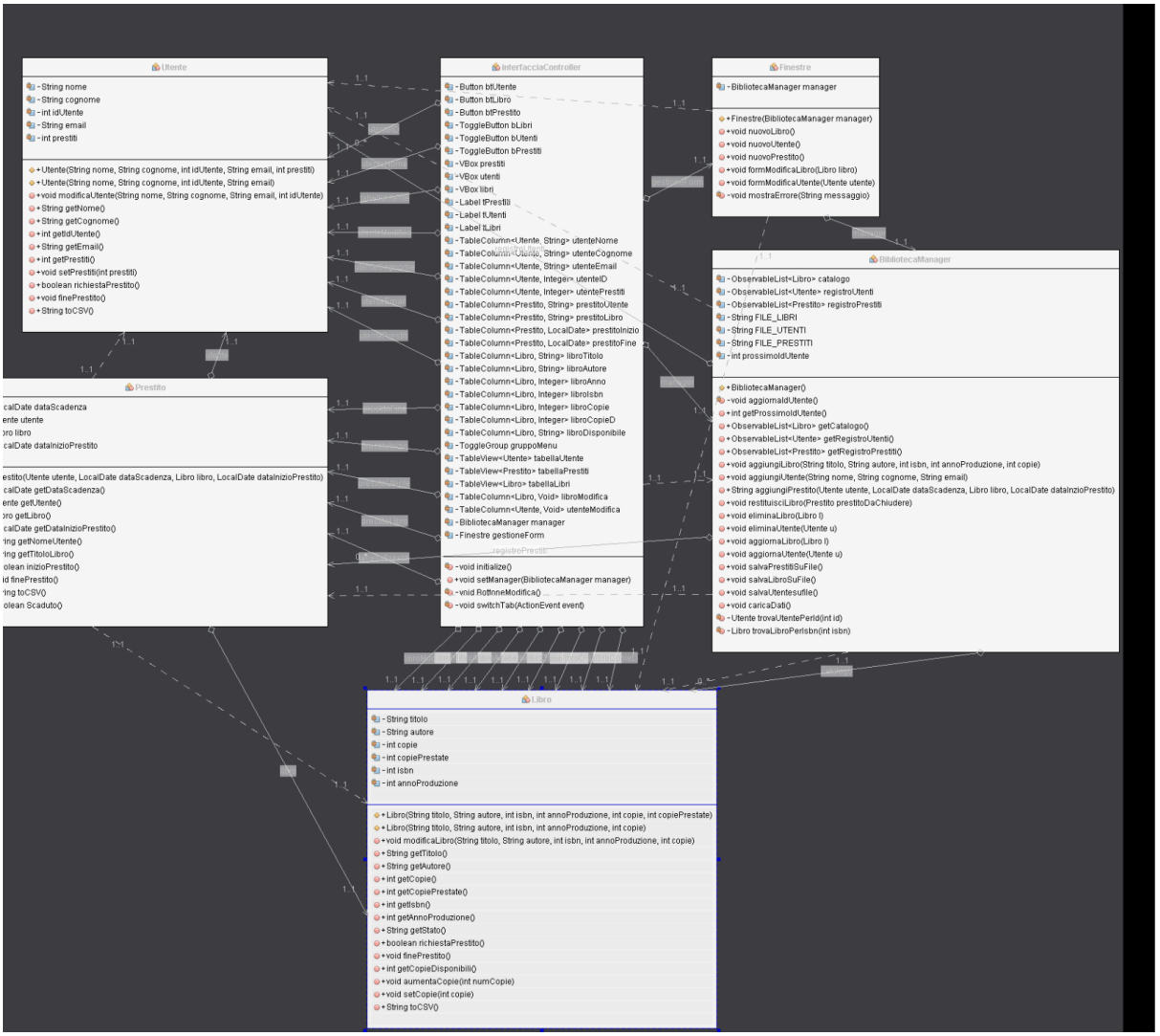
Abbiamo seguito il pattern **MVC** perché è perfetto per JavaFX. Funziona così:

- 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.

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**, così il controller principale rimane pulito. Allo stesso modo, solo **BibliotecaManager** sa come funzionano i file CSV.

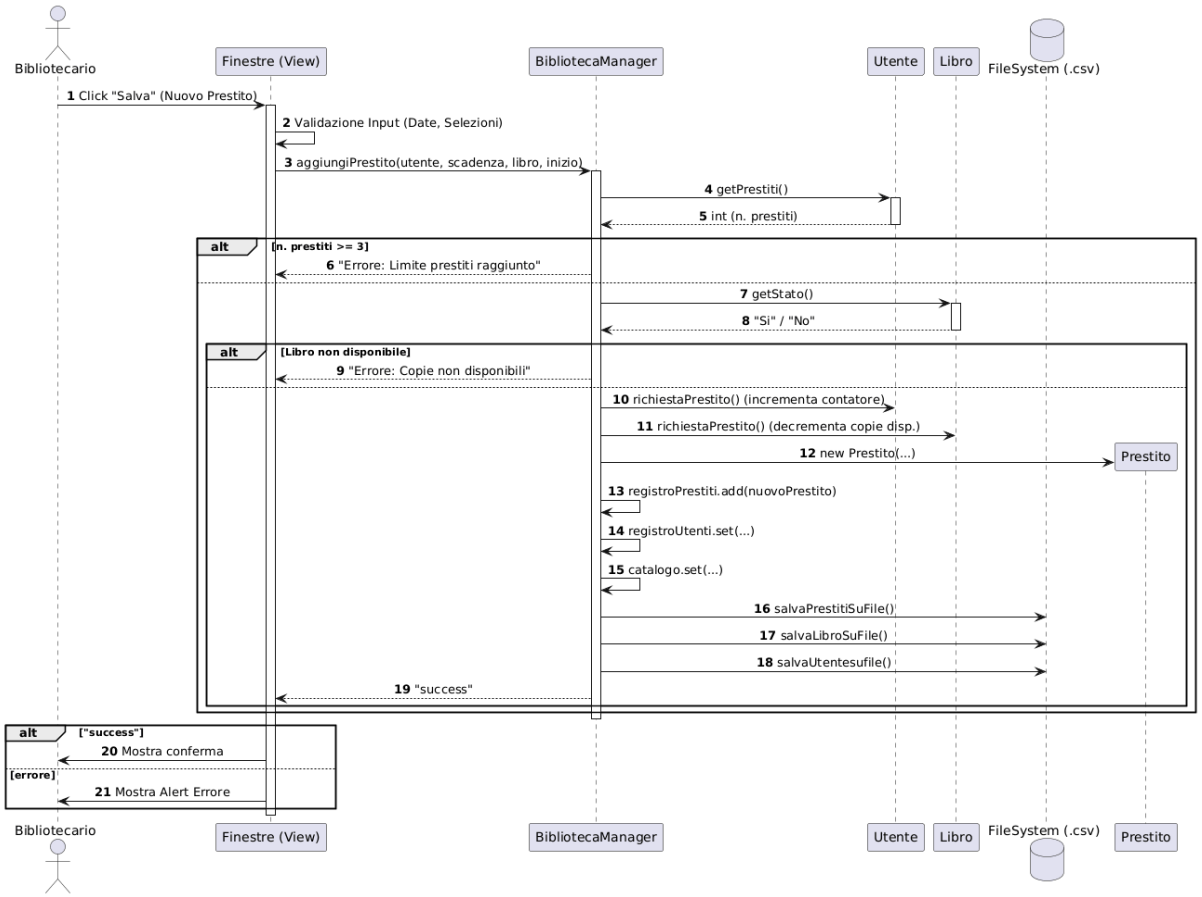
Le classi sono poco dipendenti tra loro: il controller usa solo i metodi pubblici del Manager, senza sapere come funziona internamente. Le entità (**Libro**, **Utente**) sono classi semplici che potrebbero essere usate anche in altri progetti.

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)

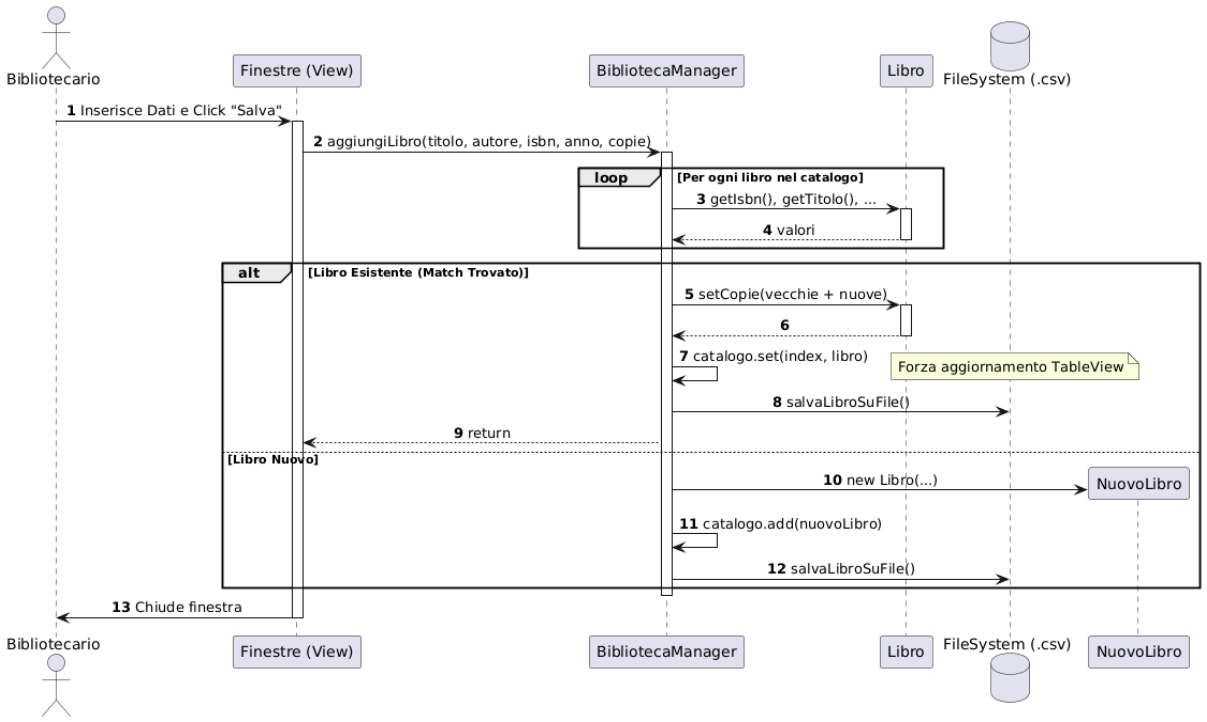
2. **BibliotecaManager** fa i controlli necessari:

    - 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

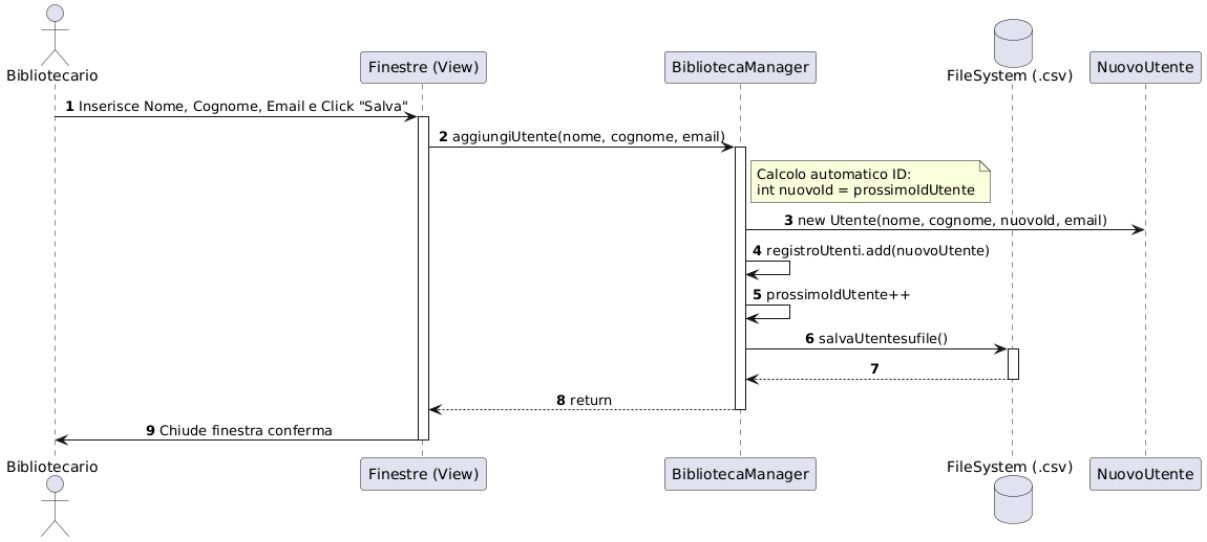
Aggiungere un Libro



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

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

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

4. L'Interfaccia Grafica