Progettazione del Software

Gestione Biblioteca

Gruppo 14

- Andrea Di Vito
- Francesca Gaia Amato
- Aniello De Marinis
- Luigi De Falco

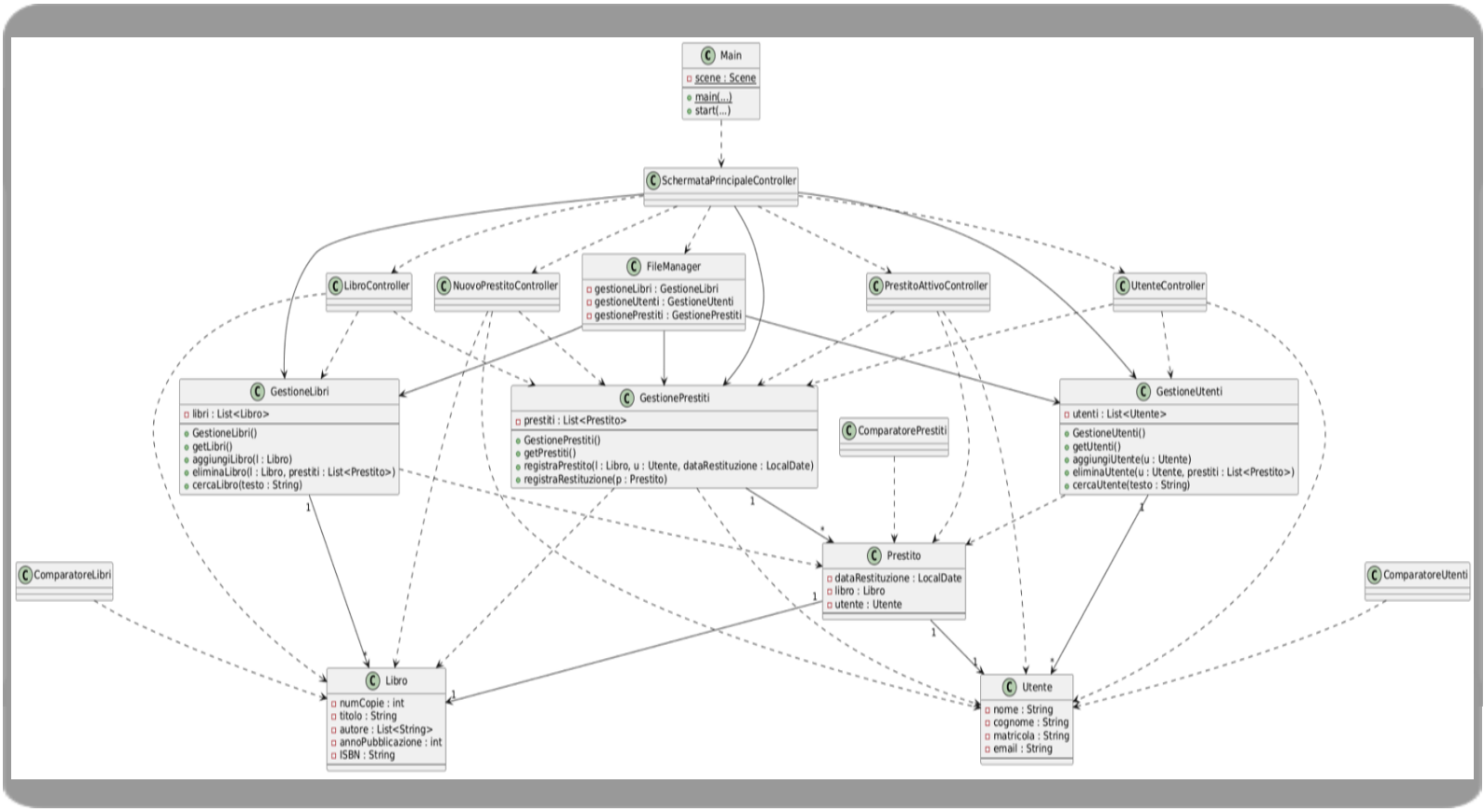
Indice

Introduzione

La fase di progettazione del software di gestione della biblioteca è stata guidata dai principi della programmazione **Object-Oriented (OO)** e focalizzata sulla creazione di un'architettura **modulare e facilmente manutenibile**. Le scelte progettuali sono state effettuate in accordo ai principi per una progettazione orientata agli oggetti (**S.O.L.I.D**), che favorisce l'**alta coesione** e il **basso accoppiamento**, e in modo da garantire un sistema di qualità.

Il documento seguente illustra i **diagrammi delle classi**, **di sequenza e di package**, **valutazione dei livelli di coesione ed accoppiamento** e opportuni chiarimenti.

1. Diagramma delle Classi



1.1 Chiarimenti sul diagramma delle classi

Per la progettazione è stato seguito un approccio alla scomposizione di tipo **Object Oriented.** Il diagramma delle classi rappresenta un modello **statico** che descrive le classi del sistema, gli attributi e i metodi di ogni classe e le relazioni tra esse.

Nel diagramma sono rappresentate: la **classe Main,** le **classi per i modelli principali** (Libro, Utente, Prestito), seguite dalle **classi per la logica applicativa** (GestioneLibri, GestioneUtenti, GestionePrestiti) ed infine le **classi per la persistenza** (FileManager) e per **l'interfaccia grafica** (i vari Controller).

Le classi del modello **definiscono** le entità principali e **incapsulano** i relativi

dati e comportamenti. Ad esempio, la classe **Libro** rappresenta un singolo libro con le sue proprietà (*titolo*, *autori*, *anno*, *ISBN*, *copie disponibili*), mentre la gestione della collezione che comprende tutti i libri presenti in archivio è affidata alla classe **GestioneLibri.**

Il sistema presentato dimostra attributi di **qualità** (**Q.A.**) **interni**, quali manutenibilità e modularità, in quanto ogni classe è progettata per essere facilmente modificabile senza impatto su altre parti del sistema. Ad esempio, la gestione della lista di libri può essere modificata senza necessità di alterare la classe Libro. Ogni classe di gestione, infatti, conserva internamente una collezione di oggetti del modello e fornisce metodi specifici per manipolare tale collezione (come aggiungere, rimuovere, modificare e cercare libri).
Anche la **riusabilità** è garantita, infatti le classi Modello possono essere facilmente riutilizzabili in altri progetti.

I comparatori (ComparatoreLibri, ComparatoreUtenti, ComparatorePrestiti) sono classi progettate per implementare **criteri di ordinamento specifici**, senza modificare le classi del modello (Libro, Utente, Prestito) o le classi per la logica applicativa (GestioneLibri, GestioneUtenti, GestionePrestiti).

La classe FileManager è responsabile della **persistenza dei dati**.
Questo modulo è isolato dal resto e riceve dall'esterno le informazioni di cui ha bisogno. Tale separazione evita che le classi di gestione mescolino logica del dominio e logica di I/O, aumentando la manutenibilità del sistema.

Le classi Controller dell'**interfaccia grafica** sono associate ognuna ad una specifica schermata (le rispettive view FXML). I controller gestiscono gli eventi dell'utente, aggiornano le tabelle e invocano i metodi delle classi di logica applicativa (classi Gestione). Questo modello rispecchia il pattern **Model–View–Controller**.

2. Valutazione del livello di coesione

Classe	Livello di Coesione	Descrizione
Libro	Funzionale	La classe implementa le operazioni necessarie alla gestione della struttura dati che rappresenta un libro (metodi getter e setter)
Utente	Funzionale	La classe implementa le operazioni necessarie alla gestione della struttura dati che rappresenta un utente (metodi getter e setter)
Prestito	Funzionale	La classe implementa le operazioni necessarie alla gestione della struttura dati che rappresenta un prestito (metodi getter e setter)
ComparatoreLibri	Funzionale	La classe implementa l'interfaccia Comparator ridefinendo il metodo `compare` che consente di comparare i libri in accordo con il criterio di ordinamento richiesto.
ComparatoreUtenti	Funzionale	La classe implementa l'interfaccia Comparator ridefinendo il metodo `compare` che consente di comparare gli utenti in accordo con il criterio di ordinamento richiesto.
ComparatorePrestiti	Funzionale	La classe implementa l'interfaccia Comparator ridefinendo il metodo `compare` che consente di comparare i prestiti in accordo con il criterio di ordinamento richiesto.
GestioneLibri	Funzionale	La classe presenta tutti i metodi necessari per la creazione e la gestione di una lista di libri.
GestioneUtenti	Funzionale	La classe presenta tutti i metodi necessari per la creazione e la gestione di una lista di utenti.
GestionePrestiti	Funzionale	La classe presenta tutti i metodi necessari per la creazione e la gestione di una lista di prestiti.
LibroController	Funzionale	La classe implementa i metodi necessari alla gestione del tab Gestione Libri, i quali lavorano sugli stessi dati di input.
UtenteController	Funzionale	La classe implementa i metodi necessari alla gestione del tab Gestione Utenti, i quali lavorano sugli stessi dati di input.
PrestitoAttivoController	Funzionale	La classe implementa i metodi necessari alla gestione del tab Prestiti Attivi, i quali lavorano sugli stessi dati di input.
NuovoPrestitoController	Funzionale	La classe implementa i metodi necessari alla gestione del tab Nuovo Prestito, i quali lavorano sugli stessi dati di input.
SchermataPrincipaleController	Funzionale	La classe funge da contenitore dei vari tab e svolge la sola operazione di salvataggio su file dei dati dell'archivio.
FileManager	Funzionale	La classe presenta i metodi per salvare l'archivio su un file e caricare l'archivio da un file esterno. Si occupa di serializzare le liste di libri, utenti e prestiti in un file strutturato, assicurandosi che i dati salvati rispecchino gli ordinamenti correnti definiti dalle regole di visualizzazione.
Main	Funzionale	La classe contiene il metodo main che funge da entry point, istanziando i componenti necessari e lanciando l'interfaccia grafica per permettere al bibliotecario di iniziare l'interazione con il sistema.

Table 1: Valutazione del livello di Coesione

3. Valutazione del livello di accoppiamento

Classi	Livello di Accoppiamento	Descrizione
Prestito-Libro/Utente	Per dati	La classe Prestito mantiene riferimenti diretti alle istanze di Libro e Utente come attributi interni. Questo è un accoppiamento necessario per definire l'entità del prestito.
GestioneLibri-Libro	Per dati	La classe GestioneLibri, per soddisfare i requisiti di aggiunta o modifica, riceve dall'esterno solo i tipi di dato primitivi o semplici (Stringhe per titolo/autore, interi per anno/copie) strettamente necessari all'operazione. Non accede mai direttamente alle strutture dei chiamanti, né condivide variabili globali.
GestioneUtenti-Utente	Per dati	Le operazioni di GestioneUtenti chiamano i metodi pubblici di Utente che accettano parametri specifici (nome, cognome, matricola, email). La classe restituisce informazioni semplici o liste di oggetti sola-lettura per la visualizzazione, senza esporre la propria struttura interna (evitando l'accoppiamento per contenuti).
GestioneUtenti/Libri-GestionePrestiti	Per timbro	Le operazioni di rimozione di un libro e di un utente richiedono di verificare la presenza di un prestito attivo riferito all'oggetto da eliminare.
GestionePrestiti-Prestito	Per dati	La classe GestionePrestiti chiama i metodi pubblici di Prestito e riceve da quest'ultima solo i metodi getter necessari per l'associazione dell'utente e del libro.
GestionePrestiti-Libro/Utente	Per dati	La classe GestionePrestiti chiama il metodo pubblico per ottenere il numero di copie della classe Libro e per ottenere le informazioni dalla classe Utente su chi richiede il prestito.
ComparatoreLibri-Libro	Per dati	ComparatoreLibri usa i metodi getter pubblici messi a disposizione dalla classe Libro per ottenere i titoli da comparare.
ComparatoreUtenti-Utente	Per dati	ComparatoreUtenti usa i metodi getter pubblici messi a disposizione dalla classe Utente per ottenere i cognomi, nomi e matricole da comparare.
ComparatorePrestiti-Prestito	Per dati	ComparatorePrestiti usa i metodi getter pubblici messi a disposizione dalla classe Prestito per ottenere le date previste per la restituzione da comparare.
FileManager-GestioneLibri	Per timbro	FileManager usa i metodi pubblici di GestioneLibri nelle sue operazioni, per accedere agli attributi dei vari modelli da scrivere su un file.
FileManager-GestioneUtenti	Per timbro	FileManager usa i metodi pubblici di GestioneUtenti nelle sue operazioni, per accedere agli attributi dei vari modelli da scrivere su un file.
FileManager-GestionePrestiti	Per timbro	FileManager usa i metodi pubblici di GestionePrestiti nelle sue operazioni, per accedere agli attributi dei vari modelli da scrivere su un file.
LibroController-GestioneLibri	Per dati	Il controller raccoglie i dati dalla GUI e li passa ai metodi della classe GestioneLibri per la manipolazione o la ricerca. Si limita a invocare i metodi di interfaccia pubblica passando i parametri richiesti.
UtenteController-GestioneUtenti	Per dati	Il controller raccoglie i dati dalla GUI e li passa ai metodi della classe GestioneUtenti per la manipolazione o la ricerca. Si limita a invocare i metodi di interfaccia pubblica passando i parametri richiesti.
NuovoPrestitoController-Gestione(Libri/Utenti/Prestiti)	Per dati	Il controller raccoglie i dati dalla GUI e invoca metodi di verifica su GestioneLibri e GestioneUtenti prima di registrare un nuovo prestito tramite la GestionePrestiti.
SchermataPrincipaleController-FileManager	Per dati	Il controller della schermata principale raccoglie i dati dalla GUI e orchestra il salvataggio su file invocando il metodo salvaSuFile della classe FileManager e carica i vecchi salvataggi da file esterno con il metodo caricaDaFile.
***Controller-** *vari modelli* **(Libro/Utente/Prestito)**	Per dati	I Controller chiamano i metodi pubblici dei vari modelli per manipolarli tramite getter/setter.

Table 2: Valutazione del livello di Accoppiamento

4. Diagrammi di sequenza

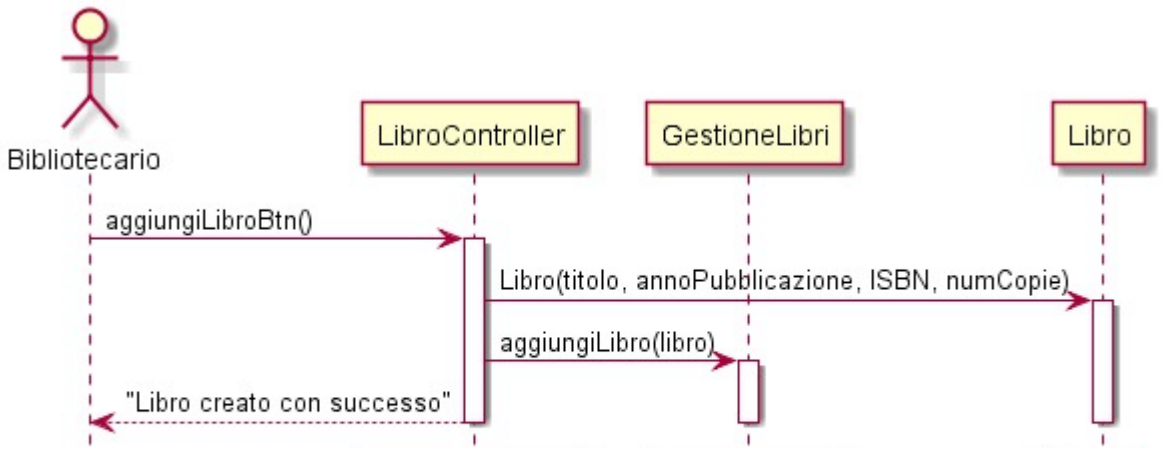


Diagramma 1: Interazione 1 - Aggiungi Libro

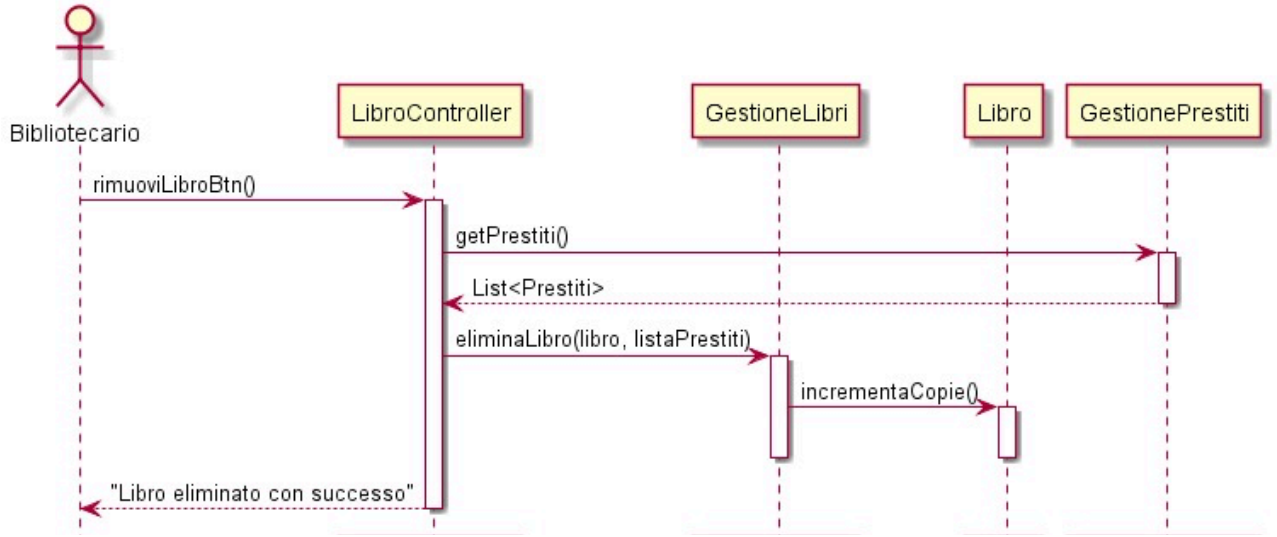


Diagramma 2: Interazione 2 - Elimina Libro

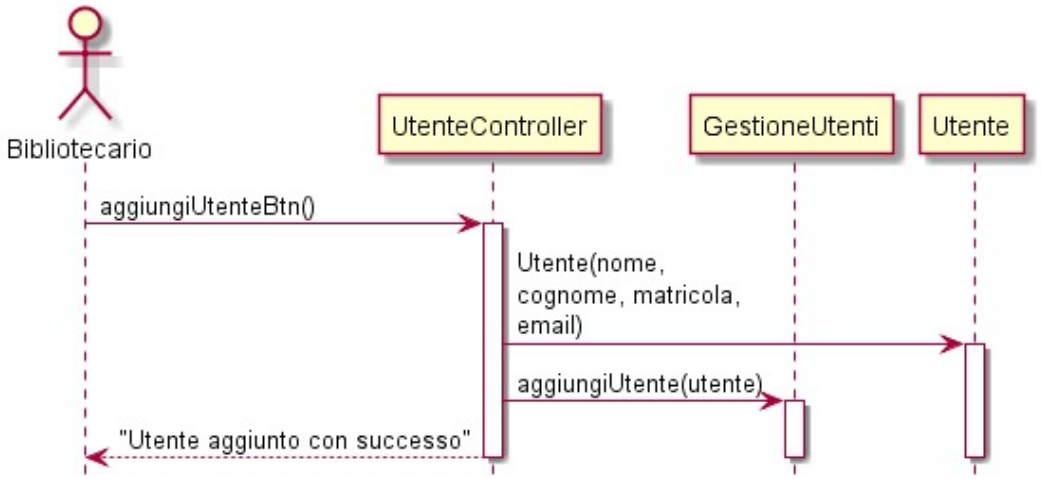


Diagramma 3: Interazione 3 - Aggiungi Utente

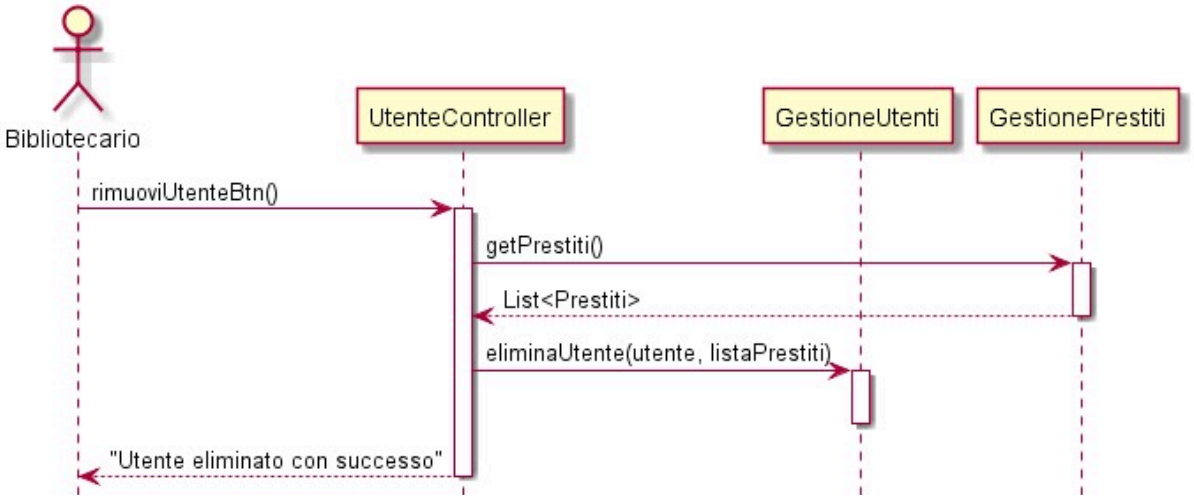


Diagramma 4: Interazione 4 - Elimina Utente

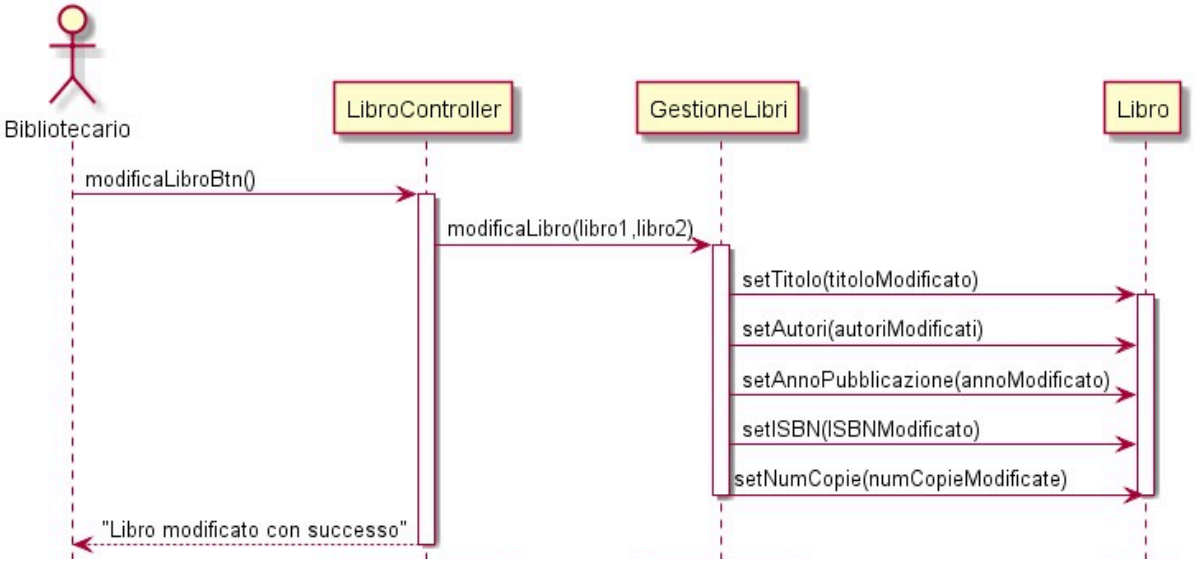


Diagramma 5: Interazione 5 - Modifica Libro

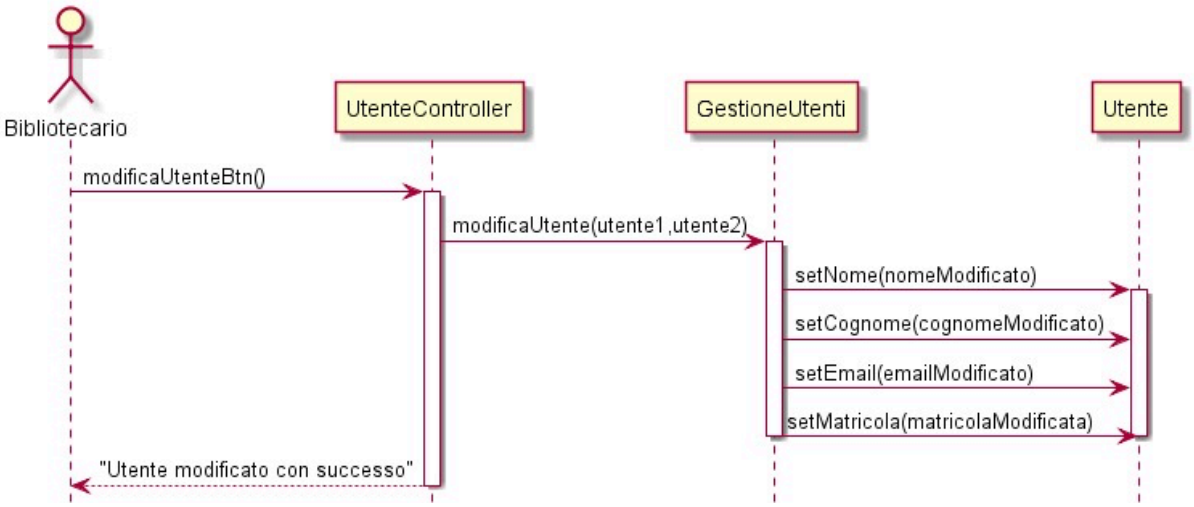


Diagramma 6: Interazione 6 - Modifica Utente

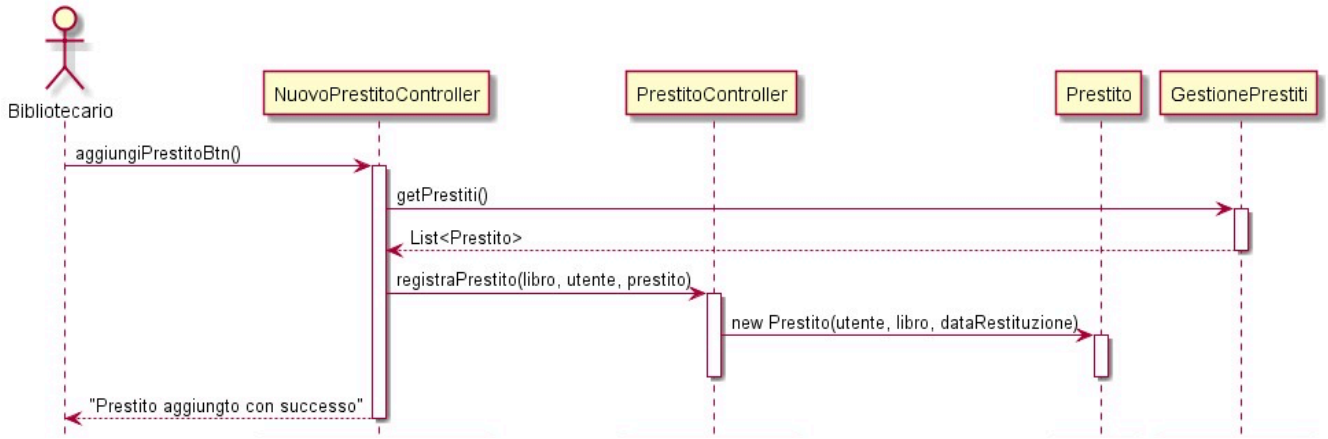


Diagramma 7: Interazione 7 - Aggiungi Prestito

Diagramma 8: Interazione 8 - Rimuovi Prestito

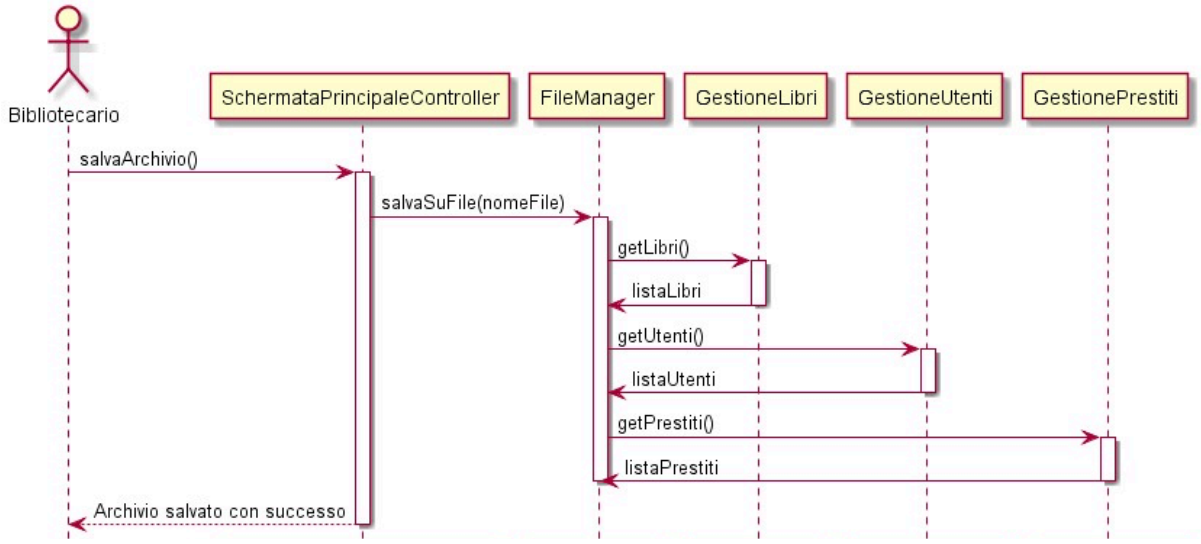


Diagramma 9: Interazione 9 - Salva Archivio

5. Diagramma dei Package

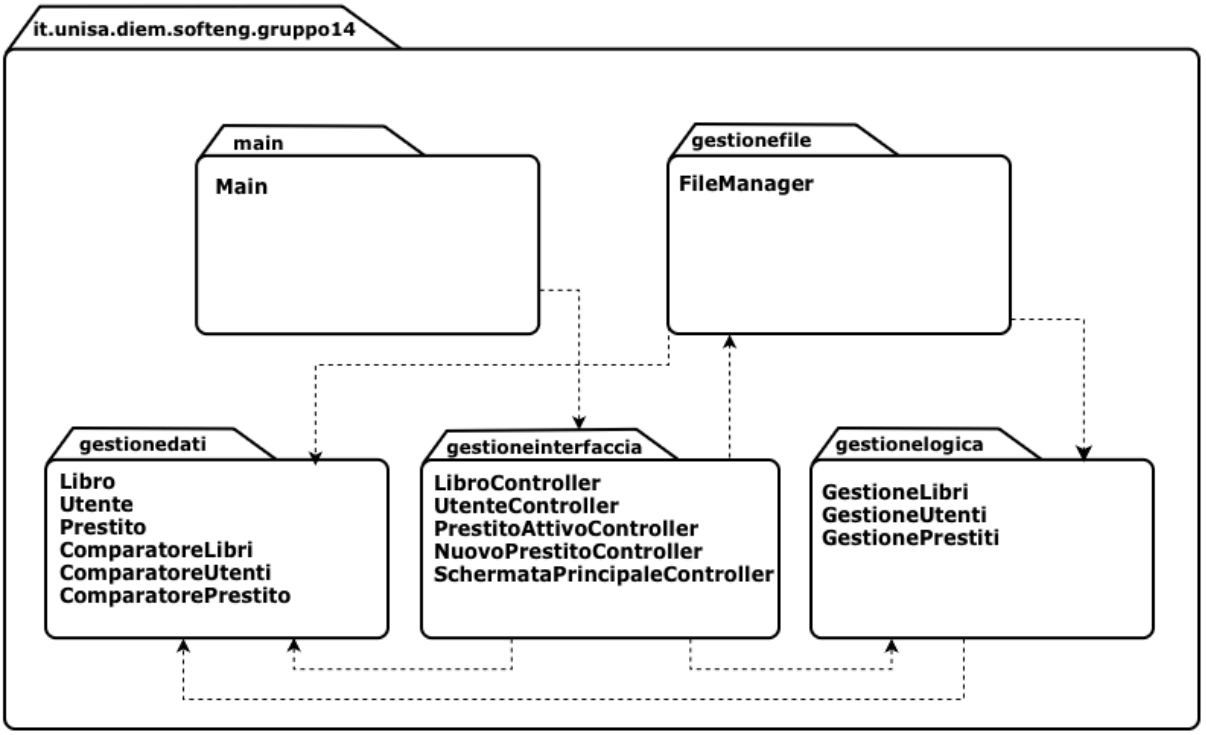


Diagramma 10: Diagramma dei Package

5.1 Chiarimenti sul diagramma dei Package

Dipendenze tra pacchetti:

1. Pacchetto **main:**
   Dipende da: *gestioneinterfaccia*.
   La **classe Main** nel pacchetto main è il punto d'ingresso dell'applicazione, interagisce con il controller della schermata principale per avviarla.

2. Pacchetto **gestionefile:**
   Dipende da: *gestionedati*, *gestionelogica*.
   La **classe FileManager** nel pacchetto gestionefile è la classe incaricata di gestire il caricamento e la scrittura dei file. Per fare ciò, ha bisogno di interagire con i dati e la logica di applicazione.

3. Pacchetto **gestioneinterfaccia:**
   Dipende da: *gestionedati*, *gestionelogica, gestionefile*.
   I **controller** dell'interfaccia nel pacchetto gestioneinterfaccia devono interagire con i dati e gli opportuni metodi messi a disposizione dalla logica applicativa. Inoltre per poter salvare e caricare i dati su/da file interagisce con la classe FileManager.

4. Pacchetto **gestionelogica:**
   Dipende da: *gestionedati*.
   Questo pacchetto contiene la **logica applicativa** per gestire libri, utenti e prestiti, quindi ha una dipendenza naturale dal pacchetto gestionedati, che fornisce le classi di modello da manipolare.

5. Pacchetto **gestionedati:**
   Non dipende da nessun altro pacchetto.
   Questo pacchetto contiene le **classi di modello** (come Libro, Utente, Prestito) e i comparatori, quindi non ha dipendenze da altri pacchetti.