.DIEM

Università degli Studi di Salerno
Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata (DIEM)

Documentazione del Progetto

Sviluppo di un'applicazione Java per la gestione di una biblioteca
Corso di Ingegneria del Software- A.A. 2025/26

Studenti Gruppo 16:
Iannaccone Dana
Matricola: 0612709501
Nigro Laura
Matricola: 0612709525
Purcaro Andrea
Matricola: 0612709382
Zamosteanu Ionela
Matricola: 0612709590

Docente:
Prof. Nicola Capuano

1 Sommario

2 Descrizione dell'applicazione

2.1 Finalità del documento SRS

Il summenzionato gruppo, nell'atto della rassegna di un documento di Specifica dei Requisiti del Software (SRS), mira a definire, nella prima delle fasi dello sviluppo del progetto, i requisiti alla base dell'accordo con il fruitore dell'applicazione che verrà realizzata.
Seguiranno, dunque, rappresentazioni adottate mediante un linguaggio naturale, talvolta strutturato.

2.2 Descrizione ed obiettivi del programma

La richiesta circa la quale saremo operativi si riconduce all'esigenza da parte di un bibliotecario universitario di avere a disposizione un'applicazione atta alla gestione dei libri e degli utenti della biblioteca stessa. Con il programma si potrà tracciare ogni singolo libro e studente, per garantire una rispettosa osservanza delle scadenze, congiunta ad una gestione ordinata e sistematica dei prestiti e delle conseguenti restituzioni. La concessione di un libro si può avere solo previa disponibilità della copia desiderata, con il limite massimo di tre libri per studente nello stesso lasso di tempo. L'elenco dei prestiti attivi si accorderà alle date previste per ciascuna restituzione e, per chi non sarà ligio a quanto prestabilito, scatterà un messaggio di avviso di evidenziazione del ritardo.
La memorizzazione delle informazioni avverrà su un file preposto. Gli obiettivi ricercati fanno capo al creare un sistema che risulti affidabile, sicuro e che sia sicuramente d'ausilio a chi se ne servirà.

2.3 Attori coinvolti

Bibliotecario:
Di fatto, è l'unico fruitore dell'applicazione; egli si troverà a lavorare con libri tracciati ad hoc e studenti registrati, con la possibilità continuativa di effettuare operazioni di inserimento, ricerca, modifica e cancellazione proprio sui target indicati poc'anzi. Mediante delle regole opportunamente stabilite, questi provvederà anche alla gestione della concessione temporanea dei volumi e al riottenimento dei medesimi.

3 Categorizzazione dei Requisiti

3.1 Schema di categorizzazione

Affinché la categorizzazione dei requisiti risulti di interpretazione intuitiva e non equivocabile, lasciamo che segua la seguente legenda, che vede dei codici specifici, veri e propri acronimi, accordarsi al nome del requisito preso in esame:

- **IF: Funzionalità Individuali**
- **BF: Business Flow**
- **DF: Esigenze di dati e informazioni**
- **UI: Interfaccia Utente**
- **FC: Requisiti non funzionali**

3.2 Funzionalità individuali

- **IF-1:** Il sistema consente di inserire un libro alla volta nella piattaforma, specificandone titolo, autore, anno di pubblicazione, ISBN e numero di copie disponibili.
- **IF-2:** Il sistema permette di modificare i dati del singolo libro.
- **IF-3:** Il sistema fa sì che i dati del libro possano essere cancellati e che, quindi, esso possa essere rimosso dal catalogo.
- **IF-4:** Il sistema supporta la visualizzazione della lista dei libri ordinata per titolo.
- **IF-5:** Il sistema ammette la ricerca di un libro per titolo, autore o codice identificativo.
- **IF-6:** Il sistema favorisce l'inserimento dei dati del singolo studente nella piattaforma, specificandone nome, cognome, matricola ed e-mail istituzionale.
- **IF-7:** Il sistema permette di modificare i dati di uno studente registrato.
- **IF-8:** Il sistema consente di rimuovere uno studente dalla piattaforma, cancellandone i relativi dati.
- **IF-9:** Il sistema supporta la visualizzazione della lista degli studenti ordinata per cognome e nome.
- **IF-10:** Il sistema ammette la ricerca di uno studente registrato per cognome o matricola.
- **IF-11:** Il sistema attua la registrazione di un prestito appena effettuato in favore di uno studente.
- **IF-12:** Il sistema predispone la visualizzazione dell'elenco dei prestiti attivi, ordinati in accordo alla data prevista di restituzione.
- **IF-13:** Il sistema permette la ricerca di uno specifico prestito effettuato, mediante cognome o matricola dello studente e titolo o ISBN del libro preso in carico.
- **IF-14:** Il sistema mette a registro la restituzione di un libro precedentemente prestato.
- **IF-15:** Il sistema consente di salvare su file l'intero archivio della biblioteca, costituito, dunque, da libri, studenti e prestiti.

3.3 Business Flow

- **BF-1: Gestione libri**: Il bibliotecario può inserire, modificare e cancellare i dati dei libri, visualizzarne la lista ordinata per titolo e cercare un elemento in accordo a titolo medesimo, autore o codice identificativo.
- **BF-2: Gestione studenti**: Il bibliotecario può inserire, modificare e cancellare i dati degli studenti, visualizzarne la lista ordinata per cognome e nome ed effettuare una ricerca per cognome stesso o matricola.
- **BF-3: Gestione prestiti e restituzioni**: Il bibliotecario può registrare un prestito, selezionando uno studente e il libro scelto, del quale specificare la data di restituzione; questo attributo è fondamentale nella determinazione dell'ordine dell'elenco dei prestiti attivi, visualizzabile dal fruitore dell'applicazione e in cui poter evidenziare i ritardi. Un libro può essere prestato solo se ci sono copie disponibili, mentre è consentito un massimo di tre libri a testa. Quando lo studente riconsegnerà il libro, il bibliotecario effettuerà un'operazione di ricerca tale da individuare la persona e il libro riportatogli, per poi registrare la restituzione come cancellazione del prestito attivo.

3.4 Esigenze di dati e informazioni

- **DF-1: Dati libro**: Il sistema deve memorizzare titolo, lista degli autori (nome e cognome), anno di pubblicazione, codice identificativo univoco (ISBN) e numero di copie disponibili di ogni libro.
- **DF-2: Dati studente**: Il sistema deve memorizzare nome, cognome, matricola ed e-mail istituzionale di ogni studente. Le informazioni relative ai prestiti attiv**i** (libri attualmente in prestito e relative date di restituzione) non sono attributi dello studente, ma sono gestite dall'archivio adibito ai prestiti.
- **DF-3: Dati prestito**: Il sistema deve memorizzare la data in cui è avvenuto il prestito e la conseguente data di restituzione.

3.5 Interfaccia utente

- **UI-1:** Il sistema, dopo che è stata avviata l'applicazione, porta il bibliotecario su una schermata Home, dove visualizzare superficialmente le sezioni "Libri", "Studenti" e "Prestiti".
- **UI-2:** Il sistema, dopo che si è scelto di accedere alla sezione "Libri", illustra una schermata con le tre funzionalità relative possibili: "Nuovo libro", "Registro libri" e "Ricerca libro".
- **UI-3:** Il sistema, dopo che si è scelto di accedere alla sezione "Studenti", illustra una schermata con le tre funzionalità relative possibili: "Nuovo studente", "Registro studenti" e "Ricerca studente".
- **UI-4:** Il sistema, dopo che si è scelto di accedere alla sezione "Prestiti", illustra una schermata con le tre funzionalità relative possibili: "Nuovo prestito", "Registro prestiti attivi" e "Ricerca prestiti attivi".
- **UI-5:** Il sistema, dopo la pressione di "Nuovo libro" presso la schermata "Libri", consente di inserire un nuovo libro, mediante un form da compilare; ne verranno

configurati gli attributi fondamentali, come, appunto, titolo, autore/i, anno di pubblicazione, ISBN e numero di copie; se ne procederà alla conferma e, qualora occorrano errori nelle informazioni propagate, verrà segnalata l'anomalia.

- **UI-6:** Il sistema, dopo l'accesso alla schermata "Registro libri", consente di visualizzare un riepilogo dei libri disponibili tramite una lista ordinata per titolo; in caso di assenza della stessa, il sistema segnala l'errore.
- **UI-7:** Il sistema, nella sezione "Ricerca libro", consente la ricerca di uno specifico libro tramite una barra apposita e l'impostazione di un filtro. In caso di sua assenza, il sistema segnala l'errore.
- **UI-8:** Il sistema, presso la sezione "Ricerca libro", una volta selezionato l'elemento desiderato, presenta una schermata con due relative operazioni possibili: "Modifica libro" e "Rimuovi libro".
- **UI-9:** Il sistema, se pressato "Modifica libro", indirizza ad una schermata tale da visionare un form identico a quello visto per l'inserimento, potendone modificare alcuni o tutti i campi, per poi confermare quanto prodotto.
- **UI-10:** Il sistema, dopo la pressione di "Nuovo studente" presso la schermata "Studenti", consente l'inserimento di un nuovo studente tramite un apposito form da compilare; in esso, saranno indicati nome, cognome, matricola ed e-mail istituzionale della persona; in caso di errore nelle informazioni trasmesse, il sistema deve fare luce sull'inesattezza. Qualora, invece, lo studente sia già presente nel sistema, se ne ha una segnalazione.
- **UI-11:** Il sistema, dopo l'accesso alla schermata "Registro studenti", consente di visualizzare un elenco degli studenti registrati, per mezzo di una lista ordinata per cognome e nome; in caso di assenza della stessa, il sistema segnala l'errore.
- **UI-12:** Il sistema, presso la schermata "Ricerca studente", consente la ricerca di uno studente mediante una barra apposita e l'impostazione di un filtro. Individuato lo studente, è possibile, cliccando sullo stesso, modificarne o cancellarne i dati. Sarà segnalato un errore se l'operazione non sarà andata a buon fine.
- **UI-13:** Il sistema, presso la sezione "Ricerca studente", una volta selezionata la persona desiderata, presenta una schermata con due relative operazioni possibili: "Modifica studente" e "Rimuovi studente".
- **UI-14:** Il sistema, se pressato "Modifica studente", indirizza ad una schermata tale da visionare un form identico a quello visto per l'inserimento, potendone modificare alcuni o tutti i campi, per poi confermare quanto prodotto.
- **UI-15:** Il sistema, dopo che si è acceduti a "Nuovo prestito" dalla schermata "Prestiti", propone un form da compilare, al fine di inserire una nuova informazione: verranno inseriti l'ISBN del libro e la matricola dello studente; se a seguito della conferma vi saranno errori, ve ne sarà una segnalazione.
- **UI-16:** Il sistema, presso "Registro prestiti attivi", consente di visualizzare la lista dei prestiti attivi, ordinata per data prevista di restituzione, evidenziando quelli in ritardo o prossimi alla scadenza. Qualora l'elenco in questione sia privo di elementi, il sistema esibisce, di fatto, una lista vuota, insieme al messaggio "Nessun prestito attivo presente".
- **UI-17:** Il sistema, nella sezione "Ricerca prestiti attivi", consente la ricerca di un prestito tramite una barra apposita e l'impostazione di un filtro. Se non lo si rinviene, viene segnalato un errore.

- **UI-18:** Il sistema, dopo la ricerca andata a buon fine di un prestito, permette, nella sezione successiva “Chiudi prestito”, di segnalare la restituzione del libro in questione.

3.6 Requisiti non funzionali

- **FC-1:** L’interfaccia deve essere semplice e intuitiva.
- **FC-2:** Le prestazioni del sistema non devono degradare al crescere del numero di libri inseriti e studenti registrati.
- **FC-3:** Il sistema deve consentire di attuare facilmente modifiche alla lista dei libri e degli studenti, senza alterare i dati altrui preesistenti.
- **FC-4:** L’applicazione deve adottare un’architettura MVC, che sia, dunque, garante di una tripartizione tra Model (atto a gestire e memorizzare i dati relativi a studenti, libri e prestiti effettuati), View (vera e propria interfaccia grafica) e Controller (logica alla base del coordinamento tra Model e View).
- **FC-5:** Ogni volta che si ha una compilazione, che sia finalizzata ad una visualizzazione o ad una ricerca, bisogna evitare di lasciare vuoti dei campi, altrimenti l’obbligatorietà della definizione degli stessi impedirà di effettuare ulteriori operazioni.
- **FC-6:** Il salvataggio dell’archivio della biblioteca è sincronizzato alla chiusura dell’applicazione.

4 Priorità dei Requisiti

Presentiamo, ora, uno schema che, sinteticamente, mediante l'adozione dei codici definiti in precedenza, associa a ciascun requisito una duplice determinazione della relativa priorità: di fatto, una farà riferimento al <<business value>>, ossia il valore di importanza dato dal committente alla funzionalità desiderata, l'altra al rischio tecnico, ossia alla difficoltà di realizzazione di quanto indicato.

PRIORITÀ DEI REQUISITI			
Codice	**Descrizione sintetica**	**Business Value**	**Rischio Tecnico**
IF-1	Inserimento libro	**Alta**	**Basso**
IF-2	Modifica libro	**Alta**	**Basso**
IF-3	Cancellazione libro	**Alta**	**Basso**
IF-4	Visualizzazione lista libri ordinata	**Media**	**Basso**
IF-5	Ricerca libro (titolo/autore/ISBN)	**Alta**	**Medio**
IF-6	Inserimento studente	**Alta**	**Basso**
IF-7	Modifica studente	**Alta**	**Basso**
IF-8	Cancellazione studente	**Alta**	**Medio**
IF-9	Visualizzazione lista studenti ordinata	**Media**	**Basso**
IF-10	Ricerca studente (cognome/matricola)	**Alta**	**Medio**
IF-11	Registrazione prestito	**Alta**	**Alto**
IF-12	Visualizzazione prestiti ordinati	**Media**	**Medio**
IF-13	Ricerca prestito (cognome/matricola – titolo/ISBN)	**Media**	**Alto**

PRIORITÀ DEI REQUISITI			
IF-14	Registrazione restituzione	**Alta**	**Alto**
IF-15	Salvataggio archivio su file	**Media**	**Alto**
BF-1	Gestione libri	**Alta**	**Medio**
BF-2	Gestione studenti	**Alta**	**Medio**
BF-3	Gestione prestiti	**Alta**	**Alto**
DF-1	Dati libro	**Alta**	**Basso**
DF-2	Dati studente	**Alta**	**Medio/Alto**
DF-3	Dati prestito	**Alta**	**Medio**
UI-1	Schermata Home	**Alta**	**Basso**
UI-2	Sezione “Libri”	**Alta**	**Medio/Basso**
UI-3	Sezione “Studenti”	**Alta**	**Medio/Basso**
UI-4	Sezione “Prestiti”	**Alta**	**Medio/Basso**
UI-5	Schermata “Nuovo libro”	**Alta**	**Medio**
UI-6	Schermata “Registro libri”	**Media**	**Medio**
UI-7	Schermata “Ricerca libro”	**Alta**	**Medio/Alto**
UI-8	“Ricerca libro” e relativo layout	**Media**	**Medio/Alto**
UI-9	Sezione “Modifica libro”	**Alta**	**Medio**
UI-10	Schermata “Nuovo studente”	**Alta**	**Medio**

PRIORITÀ DEI REQUISITI			
UI-11	Schermata “Registro studenti”	**Media**	**Medio**
UI-12	Schermata “Ricerca studente”	**Alta**	**Medio/Alto**
UI-13	“Ricerca studente” e relativo layout	**Media**	**Medio/Alto**
UI-14	Sezione “Modifica studente”	**Alta**	**Medio**
UI-15	Schermata “Nuovo prestito”	**Alta**	**Medio**
UI-16	Schermata “Registro prestiti attivi”	**Media**	**Medio**
UI-17	Schermata “Ricerca prestiti attivi”	**Alta**	**Medio/Alto**
UI-18	Sezione “Chiudi prestito”	**Alta**	**Medio**
FC-1	Interfaccia semplice e intuitiva	**Alta**	**Basso**
FC-2	Scalabilità del sistema	**Media**	**Alto**
FC-3	Facilità di modifica senza alterare altri dati	**Alta**	**Medio/Alto**
FC-4	Architettura MVC	**Alta**	**Alto**
FC-5	Controllo obbligatorietà campi	**Media**	**Basso**
FC-6	Salvataggio archivio alla chiusura	**Media**	**Medio/Alto**

5 Casi d'Uso

UC-1: Inserimento di un libro

Nome: Inserimento Libro
Attore partecipante: Bibliotecario
Precondizioni:

- Il bibliotecario ha effettuato l'accesso alla piattaforma.
- Il bibliotecario conosce titolo, autore/i, anno di pubblicazione, ISBN e numero di copie disponibili del libro.

Postcondizioni:

- Il nuovo libro è registrato nel sistema ed è visibile nella lista dei libri.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Libri".
2. Il bibliotecario accede alla sezione "Nuovo libro".
3. Inserisce titolo, autore/i, anno di pubblicazione, ISBN e numero di copie.
4. Conferma l'inserimento.
5. Il sistema verifica la correttezza dei dati e salva il nuovo libro.
6. Il sistema aggiorna la lista dei libri.

Flusso di eventi alternativi:

5a. Le informazioni inserite non sono valide (campi vuoti, anno non numerico o superiore al corrente, ISBN non nel formato corretto).
- **5a.1** Il sistema mostra un messaggio di errore.
- **5a.2** L'esecuzione riprende dal passo 3.

5b. Il libro è già presente nel catalogo (ISBN duplicato).
- **5b.1** Il sistema notifica che il libro esiste già.
- **5b.2** L'esecuzione termina senza salvare.

UC-2: Modifica di un libro

Nome: Modifica Libro
Attore partecipante: Bibliotecario
Precondizioni:

- Il bibliotecario ha fatto l'accesso.
- Il libro da modificare è stato precedentemente inserito nel sistema.

Postcondizioni:

- Il libro è aggiornato con le nuove informazioni.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Libri".
2. Il bibliotecario accede alla sezione "Ricerca libro".
3. Inserisce un parametro di ricerca (titolo, autore, ISBN o anno).
4. Il sistema mostra il libro corrispondente.
5. Il bibliotecario seleziona il libro.
6. Il bibliotecario seleziona "Modifica libro"
7. Aggiorna uno o più campi (titolo, autore/i, anno, copie).
8. Conferma la modifica.
9. Il sistema salva le modifiche.

Flusso di eventi alternativi:

4a. Il libro non è presente nella lista.

4a.1 Il sistema segnala l'assenza del libro.

4a.2 L'esecuzione termina.

UC-3: Cancellazione di un libro

Nome: Cancellazione Libro

Attore partecipante: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Il libro è registrato nel sistema.

Postcondizioni:

- Il libro è rimosso dal catalogo.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Libri".
2. Il bibliotecario accede alla sezione "Ricerca libro".
3. Inserisce un parametro di ricerca (titolo, autore, ISBN o anno).
4. Il sistema mostra il libro corrispondente.
5. Il bibliotecario seleziona il libro.
6. Il bibliotecario seleziona "Rimuovi libro"
7. Il bibliotecario conferma l'eliminazione.
8. Il sistema elimina il libro e aggiorna la lista.

Flussi di eventi alternativi:

4a. Il libro non è presente nella lista.

4a.1 Il sistema notifica l'assenza.

4a.2 L'esecuzione termina.

UC-4: Visualizzare la lista dei libri ordinata per titolo

Nome: Visualizzazione Libri Ordinata

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- È presente almeno un libro nel sistema.

Postcondizioni:

- La lista dei libri viene visualizzata ordinata per titolo.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Libri".
2. Il bibliotecario accede alla sezione "Registro libri".
3. Il sistema ordina automaticamente i libri per titolo e li mostra.

Flusso di eventi alternativi:

2a. Nella lista non sono ancora presenti libri.

UC-5: Ricerca di un libro

Nome: Ricerca Libro

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Nel sistema sono salvate informazioni come titolo, autore, ISBN, anno.

Postcondizioni:

- Il bibliotecario visualizza il libro richiesto, se presente.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Libri".
2. Il bibliotecario accede alla sezione "Ricerca libro".
3. Inserisce un parametro di ricerca (titolo, autore, ISBN o anno).
4. Il sistema mostra il libro corrispondente.

Flusso di eventi alternativi:

4a. Nessun libro corrisponde ai criteri.

4a.1 Il sistema segnala che il libro non è presente.

4a.2 L'esecuzione può riprendere dal passo 3.

UC-6: Inserimento di uno studente

Nome: Inserimento Studente

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Lo studente fornisce nome, cognome, matricola ed e-mail istituzionale.

Postcondizioni:

- Il nuovo studente risulta registrato nel sistema.

Flusso di eventi:

1. Lo studente richiede un prestito per la prima volta.
2. Il bibliotecario apre la schermata "Studenti".
3. Il bibliotecario accede alla sezione "Nuovo studente".
4. Inserisce nome, cognome, matricola ed e-mail istituzionale.
5. Conferma l'inserimento.
6. Il sistema verifica la correttezza dei dati e salva il nuovo studente.
7. Il sistema aggiorna la lista degli studenti.

Flusso di eventi alternativi:

5a. Lo studente risulta già registrato (matricola duplicata).

5a.1 Il sistema segnala la presenza dello studente.

6a. Le informazioni non sono corrette o mancanti.

6a.1 Il sistema notifica l'errore.

6a.2 L'esecuzione riprende dal passo 4.

UC-7: Modifica di uno studente

Nome: Modifica Studente

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Lo studente è presente nel sistema.

Postcondizioni:

- I dati dello studente risultano modificati.

Flusso di eventi:

1. Il bibliotecario apre la schermata “Studenti”.
2. Il bibliotecario accede alla sezione “Ricerca studente”.
3. Inserisce un parametro di ricerca (cognome o matricola).
4. Il sistema mostra lo studente corrispondente.
5. Il bibliotecario seleziona lo studente.
6. Il bibliotecario seleziona “Modifica studente”
7. Aggiorna uno o più campi (nome, cognome, matricola ed e-mail istituzionale).
8. Conferma la modifica.
9. Il sistema salva le modifiche.

Flusso di eventi alternativi:

4a. Lo studente non esiste nel sistema.

4a.1 Il sistema notifica l’assenza.

UC-8: Rimozione di uno studente

Nome: Rimozione Studente

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l’accesso.
- Lo studente è presente nel sistema.

Postcondizioni:

- Lo studente è rimosso dalla piattaforma.

Flusso di eventi:

1. Il bibliotecario apre la schermata “Studenti”.
2. Il bibliotecario accede alla sezione “Ricerca studente”.
3. Inserisce un parametro di ricerca (cognome o matricola).
4. Il sistema mostra lo studente corrispondente.
5. Il bibliotecario seleziona lo studente.
6. Il bibliotecario seleziona “Rimuovi studente”.
7. Il bibliotecario conferma l’eliminazione.
8. Il sistema elimina lo studente e aggiorna la lista.

Flusso di eventi alternativi:

4a. Lo studente non è presente nel sistema.

4a.1 Il sistema segnala l’assenza.

UC-9: Visualizzare la lista degli studenti

Nome: Visualizzazione Studenti Ordinati

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l’accesso.
- Nel sistema sono presenti studenti registrati.

Postcondizioni:

- La lista degli studenti è visualizzata in ordine alfabetico.

Flusso di eventi:

1. Il bibliotecario apre la schermata “Studenti”.
2. Il bibliotecario accede alla sezione “Registro Studenti”.

3. Il sistema ordina e mostra gli studenti ordinati per cognome e nome.

Flusso di eventi alternativi:

3a. La lista non viene visualizzata per errore.

3a.1 Il sistema mostra un messaggio di errore.

UC-10: Ricercare uno studente

Nome: Ricerca Studente

Attore: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Le informazioni sullo studente (cognome, matricola) sono già registrate.

Postcondizioni:

- Il bibliotecario visualizza lo studente richiesto.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Studenti".
2. Il bibliotecario accede alla sezione "Ricerca studente".
3. Inserisce un parametro di ricerca (cognome o matricola).
4. Conferma l'inserimento.
5. Il sistema verifica la correttezza dei dati e mostra lo studente corrispondente.

Flusso di eventi alternativi:

5a. Lo studente non esiste nel sistema.

5a.1 Il sistema notifica l'assenza.

UC-11: Registrare un prestito

Nome: Registrazione Prestito

Attore: Bibliotecario

Precondizioni:

- Lo studente richiede un libro.
- Il bibliotecario ha effettuato l'accesso.
- Il libro è disponibile.
- Lo studente non ha superato il limite massimo di prestiti (3).

Postcondizioni:

- Il prestito è registrato.
- Le copie disponibili del libro si aggiornano.
- La lista prestiti dello studente si aggiorna.

Flusso di eventi:

1. Lo studente richiede il prestito di un libro.
2. Il bibliotecario apre la schermata "Prestiti".
3. Il bibliotecario accede alla sezione "Nuovo prestito".
4. Il bibliotecario inserisce i dati del prestito (studente interessato e libro richiesto).
5. Conferma l'inserimento.
6. Il sistema verifica la correttezza dei dati e salva il nuovo prestito.
7. Il sistema aggiorna il numero di copie disponibili del libro e la lista dei prestiti attivi dello studente.

Flusso di eventi alternativi:

6a. Le informazioni inserite non sono valide (campi vuoti, data in formato non valido o superiore alla corrente).

6a.1 Il sistema mostra un messaggio di errore.

6a.2 L'esecuzione riprende dal passo 4.

6b. Lo studente ha già 3 prestiti attivi.

6b.1 Il sistema nega il prestito.

6c. Il libro non è disponibile.

6c.1 Il sistema nega il prestito.

UC-12: Visualizzare prestiti attivi

Nome: Visualizzazione Prestiti Attivi

Attore partecipante: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.

Postcondizioni:

- La schermata mostra l'elenco dei prestiti attivi, ordinato per data di restituzione.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Prestiti".
2. Il bibliotecario accede alla sezione "Registro prestiti attivi".
3. Il sistema visualizza tutti i prestiti attivi ordinati per data prevista di restituzione.

Flusso di eventi alternativi:

3a. Il database non contiene prestiti attivi.

3a.1 Il sistema mostra una lista vuota con il messaggio "Nessun prestito attivo presente".

UC-13: Ricercare un prestito

Nome: Ricerca Prestiti Attivi

Attore partecipante: Bibliotecario

Precondizioni:

- Il bibliotecario ha effettuato l'accesso.
- Le informazioni sui prestiti sono registrate correttamente.

Postcondizioni:

- Il bibliotecario visualizza il prestito ricercato.

Flusso di eventi:

1. Il bibliotecario apre la schermata "Prestiti".
2. Il bibliotecario accede alla sezione "Ricerca prestiti attivi".
3. Inserisce il criterio di ricerca (cognome o matricola dello studente per cui è attivo il prestito e titolo o ISBN del libro).
4. Il sistema mostra il risultato.

Flusso di eventi alternativi:

3a. Non risulta alcun prestito rispondente ai criteri di ricerca.

3a.1 Il sistema notifica l'assenza.

UC-14: Registrare la restituzione

Nome: Registrazione Restituzione

Attore partecipante: Bibliotecario

Precondizioni:

- Uno studente restituisce un libro.
- Il bibliotecario ha effettuato l'accesso.
- Il prestito risulta tra quelli attivi.

Postcondizioni:

- Il libro è segnato come restituito.
- Il numero di copie disponibili viene aggiornato.
- La lista dei prestiti attivi viene aggiornata.
- Lo studente non risulta più in possesso del libro restituito.

Flusso di eventi:

1. Lo studente restituisce un libro.
2. Il bibliotecario apre la schermata "Prestiti".
3. Il bibliotecario accede alla sezione "Ricerca prestiti attivi".
4. Inserisce il criterio di ricerca (cognome o matricola dello studente per cui è attivo il prestito e titolo o ISBN del libro).
5. Conferma l'inserimento.
6. Il sistema verifica la correttezza dei dati e mostra il prestito corrispondente.
7. Il bibliotecario seleziona il prestito.
8. Il bibliotecario seleziona "Chiudi".
9. Conferma la restituzione.
10. Il sistema salva la modifica.

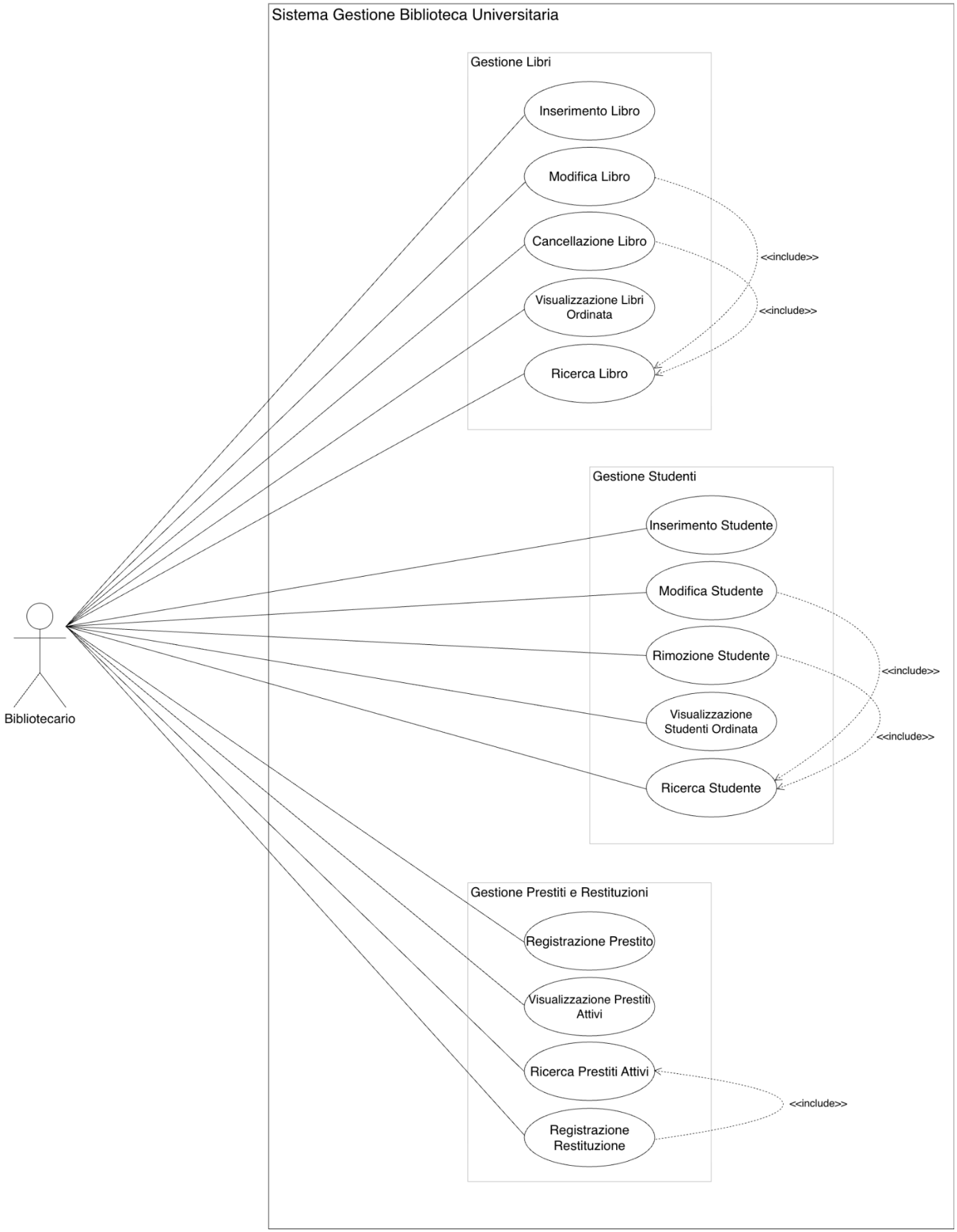
Flusso di eventi alternativi

6a. Il prestito non risulta nel sistema.

6a.1 Il sistema segnala l'incongruenza.

6 Diagramma dei Casi d'Uso

Dopo aver esaminato i casi d'uso afferenti al nostro documento e alle relative finalità, prospettiamo un'ulteriore esplicitazione, dovuta ad un'illustrazione grafica data dalla realizzazione di un diagramma specifico.
La composizione dello stesso non genera particolari dubbi nell'interpretazione, dal momento che si ha un solo attore coinvolto, il bibliotecario, che si trova a poter svolgere molteplici azioni, partizionate a seconda del business flow di riferimento. In particolare, abbiamo adottato un'architettura modulare rispetto alla gestione differita di libri, studenti e prestiti-restituzioni. Motiviamo la scelta della relazione <<include>>, all'indirizzo di ogni operazione di ricerca, in rapporto al fatto che quest'ultima fa da tappa obbligatoria nel percorso logico-computazionale che conduce alle effettuazioni di modifica e cancellazione.
Discorso praticamente identico vale per la terza macrofunzionalità, giacché una restituzione può essere concretizzata solo previa ricerca del prestito attivo.


Sistema Gestione Biblioteca Universitaria
Gestione Libri
Inserimento Libro
Modifica Libro
Cancellazione Libro
Visualizzazione Libri Ordinata
Ricerca Libro
<<include>>
<<include>>
Gestione Studenti
Inserimento Studente
Modifica Studente
Rimozione Studente
Visualizzazione Studenti Ordinata
Ricerca Studente
<<include>>
<<include>>
Gestione Prestiti e Restituzioni
Registrazione Prestito
Visualizzazione Prestiti Attivi
Ricerca Prestiti Attivi
Registrazione Restituzione
<<include>>
Bibliotecario

7 Matrice di Tracciabilità

Adoperiamo, infine, la seguente matrice, allo scopo di tenere traccia dei requisiti durante il processo di sviluppo.

TRACCIABILITÀ

Requisiti	Design	Codice	Test	Requisiti Correlati
IF-1	AggiungiLibro.fxml	ArchivioLibri.java/ AggiungiLibroController.java	testAggiungiLibroSuccesso(), testAggiungiLibroNull(), testAggiungiLibroDuplicato()	BF-1, DF-1, UI-2, UI-5, FC-5
IF-2	ModificaLibro.fxml	ArchivioLibri.java/ ModificaLibroController.java	testModificaLibroSuccesso(), testModificaLibroNonEsistente()	BF-1, DF-1, UI-8, UI-9, FC-3
IF-3	SelezionaLibro.fxml	ArchivioLibri.java/ SelezionaLibroController.java	testRimuoviLibroSuccesso(), testRimuoviLibroNonEsistente()	BF-1, DF-1, UI-8, UI-9
IF-4	RegistroLibri.fxml	ArchivioLibri.java/ RegistroLibriController.java	testGetLibriOrdinatiPerTitolo(), testGetTutti()	BF-1, DF-1, UI-6
IF-5	RicercaLibro.fxml	ArchivioLibri.java/ RicercaLibroController.java	testCercaPerIsbnEsistente(), testCercaPerIsbnNonEsistente(), testCercaPerIsbnNonValido(), testCercaPerTitolo(), testCercaPerAutore()	BF-1, DF-1, UI-7, UI-8
IF-6	AggiungiStudente.fxml	ArchivioStudenti.java/ AggiungiStudenteController.java	testAggiungiStudenteSuccesso(), testAggiungiStudenteMatricolaDuplicata()	BF-2, DF-2, UI-3, UI-10, FC-5

TRACCIABILITÀ				
IF-7	ModificaStudente.fxml	ArchivioStudenti.java/ ModificaStudenteController.java	testModificaStudenteSuccesso(), testModificaStudenteNonTrovato()	BF-2, DF-2, UI-12, UI-13, UI-14, FC-3
IF-8	SelezionaStudente.fxml	ArchivioStudenti.java/ SelezionaStudenteController.java	testRimuoviStudenteSuccesso(), testRimuoviStudenteNonTrovato()	BF-2, DF-2, UI-12, UI-13
IF-9	RegistroStudenti.fxml	ArchivioStudenti.java/ RegistroStudentiController.java	testGetTutti(), testGetStudentiOrdinati()	BF-2, DF-2, UI-11
IF-10	RicercaStudente.fxml	ArchivioStudenti.java/ RicercaStudenteController.java	testCercaPerCognomeTrovato(), testCercaPerCognomeParziale(), testCercaPerCognomeNonTrovato(), testCercaPerMatricolaTrovato(), testCercaPerMatricolaNonTrovata()	BF-2, DF-2, UI-12, UI-13
IF-11	AggiungiPrestito.fxml	ArchivioPrestiti.java/ AggiungiPrestitoController.java	testRegistraPrestitoSuccesso(), testLimiteTrePrestiti()	BF-3, DF-3, DF-1, DF-2, UI-4, UI-15, FC-5
IF-12	RegistroPrestiti.fxml	ArchivioPrestiti.java/ RegistroPrestitiController.java	testGetPrestitiPerDataOrdinati()	BF-3, DF-3, UI-16
IF-13	RicercaPrestito.fxml	ArchivioPrestiti.java/ RicercaPrestitoController.java	testCercaPrestitiAttivi()	BF-3, DF-3, UI-17, UI-18
IF-14	SelezionaPrestito.fxml	ArchivioPrestiti.java/ SelezionaPrestitoController.java	testRegistraRestituzione()	BF-3, DF-3, UI-18

TRACCIABILITÀ				
IF-15	Non visibile nel design	GestoreStatoBiblioteca.java/ ArchivioRepository.java/ StatoBiblioteca.java	--	FC-6
BF-1	gestioneLibri.controller	AggiungiLibroController.java/GestioneLibriController.java/ModificaLibroController.java/RegistroLibriController.java/RicercaLibroController.java/ SelezionaLibroController.java	--	IF-1, IF-2, IF-3, IF-4, IF-5, DF-1, UI-2, UI-5, UI-6, UI-7, UI-8, UI-9
BF-2	gestioneStudenti.controller	AggiungiStudenteController.java/GestioneStudentiController.java/ModificaStudenteController.java/RegistroStudentiController.java/RicercaStudenteController.java/SelezionaStudenteController.java	--	IF-6, IF-7, IF-8, IF-9, IF-10, DF-2, UI-3, UI-10, UI-11, UI-12, UI-13, UI-14
BF-3	gtrestionePrestiti.controller	AggiungiPrestitoController.java/GestionePrestitiController.java/RegistroPrestitiController.java/RicercaPrestitoController.java/ SelezionaPrestitoController.java	--	IF-11, IF-12, IF-13, IF-14, DF-1, DF-2, DF-3, UI-4, UI-15, UI-16, UI-17, UI-18
DF-1	AggiungiLibro.fxml	ArchivioLibri.java/ AggiungiLibroController.java	testAggiungiLibroSuccesso(), testAggiungiLibroNull(),testAggiungiLibroDuplicato()	IF-1, IF-2, IF-3, IF-4, IF-5, IF-11, BF-1, BF-3, UI-5, UI-6, UI-7, UI-9

TRACCIABILITÀ				
DF-2	AggiungiStudente.fxml	ArchivioStudenti.java/ AggiungiStudenteController.java	testAggiungiStudenteSuccesso(), testAggiungiStudenteMatricolaDuplicata()	IF-6, IF-7, IF-8, IF-9, IF-10, IF-11, BF-2, BF-3, UI-10, UI-11, UI-12, UI-14
DF-3	AggiungiPrestito.fxml	ArchivioPrestiti.java/ AggiungiPrestitoController.java	testRegistraPrestitoSuccesso(), testLimiteTrePrestiti()	IF-11, IF-12, IF-13, IF-14, BF-3, UI-15, UI-16, UI-17, UI-18
UI-1	BibliotecaInterfaccia.fxml	Main.java	--	FC-1, UI-2, UI-3, UI-4
UI-2	GestioneLibri.fxml	GestioneLibriController.java	--	BF-1, UI-1, UI-5, UI-6, UI-7
UI-3	GestioneStudenti.fxml	GestioneStudentiiController.java	--	BF-2, UI-1, UI-10, UI-11, UI-12
UI-4	GestionePrestiti.fxml	GestionePrestitiController.java	--	BF-3, UI-1, UI-15, UI-16, UI-17
UI-5	AggiungiLibro.fxml	AggiungiLibroController.java	--	IF-1, DF-1, BF-1, FC-5
UI-6	RegistroLibri.fxml	RegistroLibriController.java	--	IF-4, DF-1, BF-1, FC-1
UI-7	RicercaLibro.fxml	RicercaLibroController.java	--	IF-5, BF-1, UI-8, FC-1
UI-8	SelezionaLibro.fxml	SelezionaLibroController.java	--	IF-2, IF-3, IF-5, BF-1, UI-7, UI-9
UI-9	ModificaLibro.fxml	ModificaLibroController.java	--	IF-2, DF-1, BF-1, FC-3, FC-5
UI-10	AggiungiStudente.fxml	AggiungiStudenteController.java	--	IF-6, DF-2, BF-2, FC-5
UI-11	RegistroStudenti.fxml	RegistroStudentiController.java	--	IF-9, DF-2, BF-2, FC-1
UI-12	RicercaStudente.fxml	RicercaStudenteController.java	--	IF-10, BF-2, UI-13, FC-1

TRACCIABILITÀ				
UI-13	SelezionaStudente.fxml	SelezionaStudenteController.java	--	IF-7, IF-8, IF-10, BF-2, UI-12, UI-14
UI-14	ModificaStudente.fxml	ModificaStudenteController.java	--	IF-7, DF-2, BF-2, FC-3, FC-5
UI-15	AggiungiPrestito.fxml	AggiungiPrestitoController.java	--	IF-11, DF-3, BF-3, FC-5
UI-16	RegistroPrestiti.fxml	RegistroPrestitiController.java	--	IF-12, DF-3, BF-3, FC-1
UI-17	RicercaPrestito.fxml	RicercaPrestitiController.java	--	IF-13, BF-3, UI-18, FC-1
UI-18	SelezionaPrestito.fxml	SelezionaPrestitiController.java	--	IF-13, IF-14, BF-3, DF-3
FC-1	BibliotecaInterfaccia.fxml	BibliotecaInterfacciaController.java	--	UI-1
FC-2	Non visibile nel design	--	--	--
FC-3	ModificaLibro.fxml/ModificaStudente.fxml	ModificaLibroController.java/ModificaStudenteController.java	--	DF-1,DF-2
FC-4	view/controller/model/persistence	view.*/controller.*/model.*/persistence.*	--	DF-1,DF-2,DF-3
FC-5	AggiungiLibro.fxml/ModificaLibro.fxml/AggiungiStudente.fxml/AggiungiLibro.fxml/AggiungiPrestito.fxml	AggiungiLibroController.java/ModificaLibroController.java/AggiungiStudenteController.java/AggiungiLibroController.java/AggiungiPrestitoController.java	--	IF-1,IF-2,IF-6,IF-7,IF-11
FC-6	Non visibile nel design	--	--	--

8 Finalità del documento di progettazione

Dopo aver provveduto a ragionare a tutto tondo sui requisiti afferenti alla nostra richiesta progettuale, presentiamo, ora, un nuovo dossier, finalizzato alla fase di design.
In primis, motivata la scelta architetturale del sistema, procederemo ad una scomposizione in moduli dello stesso, arrivando a realizzare un diagramma dei package identificati. Costruiremo, poi, i diagrammi delle classi coinvolte nella nostra applicazione e delle quali implementeremo uno scheletro. Illustreremo, ancora, dei diagrammi di sequenza per le interazioni maggiormente significative, oltre ad una serie di ulteriori immagini opportunamente commentate e motivate. Tenteremo di fare luce su tutte le scelte effettuate, puntando a rientrare nei vincoli previsti. Infine, rilasceremo anche la documentazione delle interfacce pubbliche, servendoci di Doxygen.
In particolare, al fine di visionare al meglio proprio questo aspetto, è consigliato recarsi, mediante https://github.com/purcaroandrea/g16_projectIS.git, presso quanto pubblicato sul nostro GitHub, dove risulteranno consultabili le classi annotate e il Doxyfile associato.

9 Architettura del sistema

9.1 Scelta architetturale

Il nostro sistema si articolerà su una tripartizione logico-funzionale che provveda a curare separatamente gli aspetti legati alla gestione dei dati, alla presentazione visiva e al loro coordinamento. Questa scelta è in linea con quanto dichiarato nel nostro documento SRS, in particolare in relazione agli obiettivi perseguiti di scalabilità e manutenibilità, oltre che al già dichiarato intento di adottare un'architettura MVC (Model-View-Controller; si vedano FC-2, FC-3 e FC-4).
Nella fattispecie, valutiamo sinteticamente quanto offertoci da ciascun livello:

- il Model, in quanto gestore dei dati fondamentali in gioco e della rispettiva memorizzazione, si fa carico delle classi di dominio dell'applicazione (quindi Libro, Studente e Prestito); inoltre, ottempera alla necessità di caricare e salvare su file l'archivio della biblioteca; gestisce, pertanto, classi di servizio, come ArchivioLibri, ArchivioStudenti e ArchivioPrestiti. Vengono soddisfatti, dunque, i requisiti relativi alle esigenze di dati e informazioni (DF-1, DF-2, DF-3) e alla persistenza del sistema (IF-15, FC-6). Si occupa, quindi, dello stato applicativo e delle regole di business. Questo livello è indipendente dall'interfaccia grafica e può essere riutilizzato o esteso senza impattare sugli altri strati.
- la View, che si compone delle varie schermate (in JavaFX) riferite alla gestione di libri, studenti e prestiti, si rapporta alle azioni interattive propugnate dal bibliotecario, consentendo di visualizzare a schermo le conseguenze delle operazioni da questi effettuate. Vengono espletati, dunque, i requisiti confacentisi all'interfaccia utente (UI-1, …, UI-9), con la risultanza di semplicità e intuitività della medesima.
- il Controller (qui abbiamo ControllerLibri, ControllerStudenti e ControllerPrestiti), invece, come anticipato, media tra View e Model: esso riceve degli eventi dalla prima, li traduce in chiamate al secondo e applica regole di dominio, come controlli di

validazione e aggiornamenti consequenziali; la View sarà poi aggiornata in base all'esito delle operazioni. Si ha una stretta relazione tra questo livello e i business flow (BF-1, BF-2, BF-3), che abbiamo definito nel documento di specifica dei requisiti.

Questa struttura consente, in definitiva, di apportare modifiche all'interfaccia utente, senza temere stravolgimenti imprevisti sui dati inseriti e sulla logica di business; ammette, poi, l'estensione del dominio applicativo, con la possibilità di intervenire sul modello e sui dati da esso configurati; centralizza le funzionalità di caricamento e salvataggio dell'archivio, garantendo persistenza e assenza di rischio di inconsistenze; assicura maggiore testabilità per le classi di Model e Controller, oltre che possibili sostituzioni della View, come per un eventuale passaggio futuro ad un'interfaccia web.

9.2 Descrizione dei moduli principali

I moduli, altresì definiti come package, costituiscono la struttura portante dell'architettura del sistema e rappresentano il primo livello di organizzazione logica dell'applicazione. La loro suddivisione non è casuale, ma risponde alla necessità di garantire la separazione delle responsabilità prevista dal pattern architetturale MVC, adottato come riferimento per l'intero progetto. Coerentemente con questo modello, ogni package è progettato per svolgere un insieme ben definito di compiti e per collaborare con gli altri attraverso interfacce chiare e controllate, evitando sovrapposizioni di ruoli o dipendenze indesiderate.
L'obiettivo di una scomposizione modulare efficace è quello di ottenere alta coesione e basso accoppiamento, ossia che, rispettivamente, all'interno di uno stesso modulo ci sia un forte legame tra ciò che vi è incluso, ma si abbia una minima interdipendenza tra i diversi package. Forti di un approccio pienamente orientato agli oggetti, possiamo dunque analizzare nel dettaglio la struttura dei package e le responsabilità che ciascuno di essi assume all'interno dell'architettura complessiva.

9.2.1 Package biblioteca

Rappresenta l'entry-point dell'intera applicazione, ergendosi a modulo atipico; infatti, a differenza degli altri package, che si focalizzano su specifiche aree funzionali, esso si pone come livello di coordinamento generale dell'intero sistema, governando il flusso comprensivo dell'applicazione e garantendo che vi si possa navigare tra le diverse sezioni in maniera coerente e lineare.
Al suo interno sono presenti due componenti fondamentali:

- **Main**, che avvia l'applicazione, inizializza l'interfaccia grafica e prepara le strutture principali necessarie al funzionamento del sistema;
- **BibliotecaInterfacciaController**, ovvero il controller principale dell'interfaccia, responsabile della gestione delle schermate, della ricezione degli input dell'utente e dell'instradamento delle richieste verso i controller specifici dei sottosistemi (libri, studenti, prestiti).

Questo package non contiene logica di dominio, né operazioni legate alla gestione dei dati: il suo ruolo è puramente di orchestrazione. È qui che viene coordinata la comunicazione tra l'interfaccia grafica e i moduli applicativi, assicurando che ogni

richiesta dell'utente venga indirizzata al componente corretto e che la transizione tra le diverse funzionalità avvenga in modo fluido.

9.2.2 Package view

Il package view costituisce il livello di presentazione dell'applicazione e raccoglie tutte le interfacce grafiche rivolte all'utente. È suddiviso in quattro sottopackage (view.libri, view.studenti, view.prestiti e view.gestioneBiblioteca), ciascuno dedicato a una specifica area funzionale del sistema.
Le classi contenute in questo modulo si occupano esclusivamente di mostrare i dati e raccogliere gli input dell'utente, delegando ogni operazione applicativa ai rispettivi controller. In accordo con il pattern MVC, le View non contengono logica di dominio, né accedono direttamente ai modelli: fungono da ponte tra l'utente e i controller, garantendo un'interazione chiara e un basso accoppiamento con il resto del sistema.

9.2.3 Package gestione.Libri

Il package gestione.libri raccoglie le componenti dedicate alla gestione del catalogo bibliografico. Include la classe di dominio (Libro) e l'archivio corrispondente (ArchivioLibri), responsabile della memorizzazione, ricerca e organizzazione dei volumi. Il controller (LibriController) coordina le operazioni richieste dall'interfaccia, applicando le regole di business, come il controllo dell'unicità dell'ISBN. Il package presenta un'elevata coesione interna e interagisce con gli altri moduli solo quando necessario, mantenendo un basso accoppiamento.

9.2.4 Package gestione.Studenti

Il package gestione.studenti gestisce l'anagrafica degli studenti registrati nel sistema. Comprende la classe di dominio (Studente) e l'archivio dedicato (ArchivioStudenti), che offre funzionalità di inserimento, modifica, rimozione e ricerca. Il controller (StudentiController) si occupa della validazione dei dati e dell'univocità della matricola, fungendo da intermediario tra View e Model. Il package è altamente coeso e mantiene dipendenze minime verso gli altri moduli, limitate ai casi in cui è necessario verificare i prestiti associati a uno studente.

9.2.5 Package gestione.Prestiti

Il package gestione.prestiti rappresenta il nucleo della logica applicativa relativa ai prestiti. Contiene la classe Prestito, che modella una singola operazione di prestito, e ArchivioPrestiti, che gestisce la collezione dei prestiti attivi e storici, applicando i vincoli di dominio (come il limite di tre prestiti attivi per studente). Il controller (PrestitiController) coordina le operazioni di prestito e restituzione, interagendo con i moduli dei libri e degli studenti. Pur essendo il package più interconnesso, mantiene un accoppiamento controllato e una coesione molto elevata.

9.2.6 Package persistence

Questo modulo è, chiaramente, dedicato alla gestione della persistenza dei dati dell'applicazione. Include componenti come ArchivioRepository, responsabile del

salvataggio e del caricamento delle collezioni, StatoBiblioteca, che rappresenta lo stato complessivo del sistema, e GestoreStatoBiblioteca, dedito a manipolare proprio il salvataggio e il caricamento dello stato della biblioteca stessa. Questo modulo è indipendente dalla logica applicativa e interagisce principalmente con i controller, garantendo un basso accoppiamento e permettendo di sostituire o estendere le modalità di persistenza senza incidere sugli altri package.

9.3 Responsabilità e dipendenza dei package

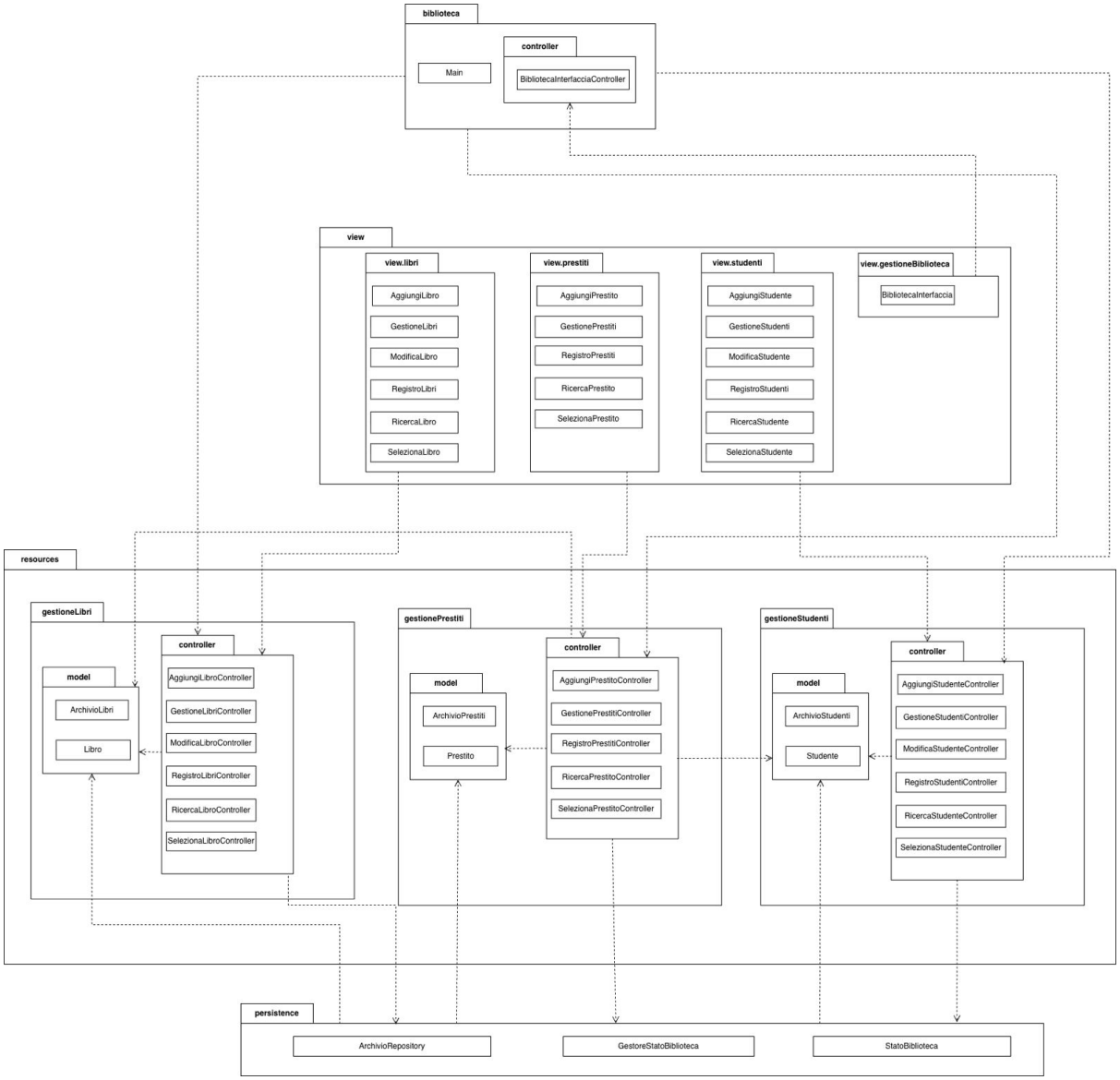
Modulo / package	**Responsabilità principali**	**Dipendenze in uscita (usa...)**	**Dipendenze in ingresso (usato da...)**
biblioteca	Entry-point dell'applicazione; avvio del sistema; creazione e configurazione dei componenti principali; collegamento tra controller, view e layer di gestione/persistenc e.	gestione.libri. controller, gestione.stud enti.controlle r, gestione.pre stiti.controller	Nessuno (entry-point)
gestione.libri. model	Classi dominio libri: Libro, ArchivioLibri; rappresentazione dei dati e regole base sull'entità libro.	Nessuna (dipende solo da Java standard)	gestione.libri.controll er, persistence (tramite ArchivioRepository / StatoBiblioteca) gestione.prestiti
gestione.libri. controller	Coordinamento delle operazioni sui libri; mediazione tra view dei libri e modello (Libro, ArchivioLibri).	gestione.libri. model, persistence	biblioteca, view.libri,
gestione.stud enti.model	Classi dominio studenti: Studente, ArchivioStudenti; gestione struttura dati e vincoli base (es. unicità matricola).	Nessuna (dipende solo da Java standard)	gestione.studenti.co ntroller, persistence (tramite ArchivioRepository / StatoBiblioteca), gestione.prestiti
gestione.stud enti.controller	Coordinamento delle operazioni sugli studenti; mediazione tra view.studenti e	gestione.stud enti.model, persistence	biblioteca, view.studenti

	modello (Studente, ArchivioStudenti).		
gestione.prestiti.model	Classi dominio prestiti: Prestito, ArchivioPrestiti; rappresentazione dei prestiti attivi e storici e regole di base sull'entità prestito.	Nessuna (dipende solo da Java standard)	gestione.prestiti.controller, persistence (tramite ArchivioRepository / StatoBiblioteca), gestione.prestiti
gestione.prestiti.controller	Gestione del ciclo di vita dei prestiti; applicazione dei vincoli (max prestiti, disponibilità copie); coordinamento tra prestiti, libri e studenti.	gestione.prestiti.model, gestione.libri.model, gestione.studenti.model, persistence	biblioteca, view.prestiti
view.libri	Interfacce grafiche per la gestione dei libri; raccolta input utente e inoltro delle richieste al controller dei libri.	gestione.libri.controller	Nessuno
view.studenti	Interfacce grafiche per la gestione degli studenti; raccolta input e inoltro al controller studenti.	gestione.studenti.controller	Nessuno
view.prestiti	Interfacce grafiche per la gestione dei prestiti; dialogo con il controller prestiti e lettura dati di libri e studenti.	gestione.prestiti.controller	Nessuno
persistence	Gestione salvataggio/caricamento degli archivi tramite ArchivioRepository e dello stato complessivo della biblioteca tramite GestoreStatoBiblioteca, con rappresentazione mediante StatoBiblioteca.	gestione.libri.model, gestione.studenti.model, gestione.prestiti.model	gestione.libri.controller, gestione.studenti.controller, gestione.prestiti.controller

9.4 Diagramma dei package

Archiviamo questa sezione con un'illustrazione che, in modo sintetico, restituisca una visione d'insieme circa le connessioni tra i moduli principali del sistema. Il diagramma dei package evidenzia, difatti, le dipendenze architetturali attraverso frecce direzionali che indicano il verso dell'utilizzo: ogni freccia parte dal package che dipende e punta verso quello utilizzato.
Nella parte inferiore dell'immagine che segue, è collocato il package **persistence**, che viene utilizzato da tutti i controller applicativi e, al contempo, si serve dei model associati (ciò si ha per i package di gestione). La sua posizione riflette il ruolo trasversale e passivo nella gestione dello stato del sistema. Sul versante opposto, in seconda linea, troviamo il package view, suddiviso in quattro sottosezioni (view.libri, view.studenti, view.prestiti, view.gestioneBiblioteca), ciascuna delle quali dipende esclusivamente dal modulo di gestione corrispondente e, nello specifico, dal relativo controller.
Al centro della struttura si colloca il package gestionePrestiti, che assume un ruolo di snodo: esso dipende, circa il proprio controller, sia da gestioneLibri.model che da gestioneStudenti.model, in quanto ogni operazione di prestito coinvolge inevitabilmente un libro e uno studente. I tre moduli di gestione sono a loro volta utilizzati dal package biblioteca, che funge da orchestratore generale e coordina l'interazione tra interfaccia e logica applicativa.
Nel complesso, il diagramma conferma la presenza di un flusso architetturale ben definito, privo di dipendenze circolari e coerente con i principi di separazione dei ruoli. Non si evidenziano interazioni dirette tra persistence e view, a conferma della corretta applicazione del pattern MVC e della modularità del sistema.

.

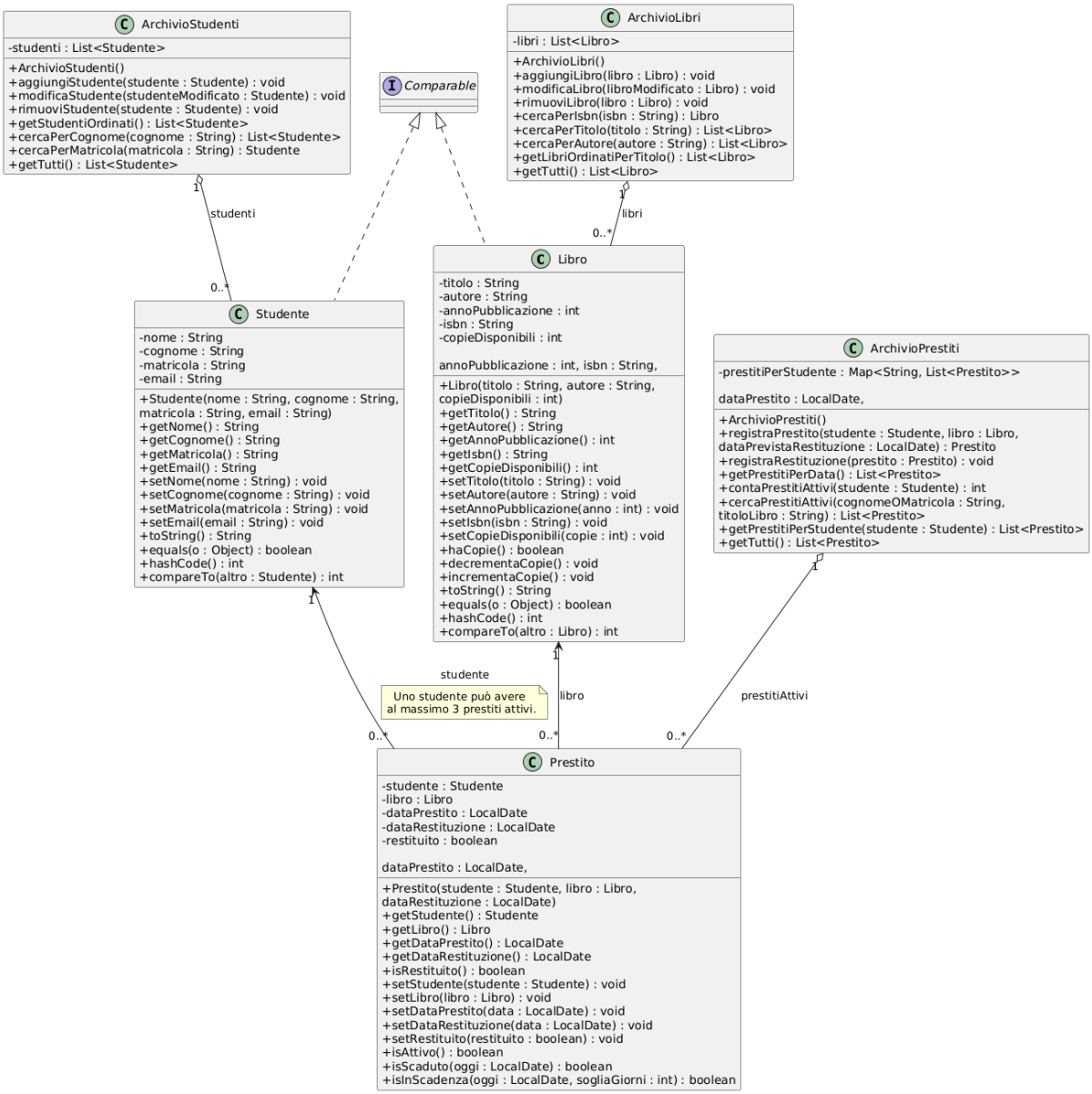


10 Modello statico

10.1 Panoramica e diagramma delle classi

Il modello statico del sistema rappresenta la struttura concettuale e organizzativa dell'applicazione di gestione della biblioteca universitaria, descrivendo in modo formale le entità principali, i loro attributi, il comportamento pubblico esposto e le relazioni che intercorrono tra di esse.
Il diagramma delle classi che segue, a tal proposito, fornisce una visione d'insieme dell'intero dominio applicativo, evidenziando come i concetti fondamentali (Studente, Libro e Prestito) siano modellati e gestiti attraverso gli archivi dedicati (ArchivioStudenti, ArchivioLibri, ArchivioPrestiti).
L'obiettivo del modello statico è garantire un'illustrazione chiara, coerente e facilmente estendibile dell'intero sistema, assicurando un elevato livello di coesione interna alle classi e un basso accoppiamento tra i diversi componenti.

Il diagramma evidenzia, inoltre, alcuni vincoli di business fondamentali presenti nei requisiti, come il limite massimo di tre prestiti attivi per studente e il controllo delle copie disponibili per ciascun libro.



10.2 Descrizione delle principali classi

Le classi descritte di seguito costituiscono il nucleo del modello di dominio del sistema. Esse rappresentano gli elementi fondamentali per la gestione dei libri, degli studenti e dei prestiti, e definiscono le strutture dati e le operazioni che supportano i casi d'uso descritti nel documento SRS.

10.2.1 Classe Libro

Responsabilità

Rappresenta un volume presente nel catalogo della biblioteca e racchiude tutte le informazioni necessarie alla sua identificazione e disponibilità.

È l'unità informativa su cui si basano le operazioni di inserimento, modifica, ricerca e prestito dei libri.

Attributi principali:

- titolo: String
- autore: String
- annoPubblicazione: int
- isbn: String
- copieDisponibili: int

Metodi pubblici principali:

- Costruttore con tutti i campi obbligatori.
- Metodi getter/setter per ogni attributo.
- boolean haCopie() – indica se sono ancora presenti, in biblioteca, copie prestabili.
- void decrementaCopie() / void incrementaCopie() – aggiornano il numero di copie disponibili, a seguito di prestiti e restituzioni.
- Metodi standard toString(), equals(Object) e hashCode().
- Implementazione dell'interfaccia Comparable<Libro> tramite compareTo(Libro altro), che definisce l'ordinamento naturale (per titolo) utilizzato da ArchivioLibri per produrre la lista ordinata, come descritto nel documento SRS.

Relazioni:

- La classe Libro è coinvolta in un'associazione unidirezionale molti-a-uno proveniente da Prestito: ogni prestito fa riferimento a un singolo libro, mentre un libro può comparire in più prestiti nel tempo, compatibilmente con il numero di copie disponibili.
- Gli oggetti della classe Libro sono aggregati in ArchivioLibri, che mantiene la collezione di tutti i volumi registrati nel sistema. L'archivio non è responsabile della creazione né del ciclo di vita dei libri, ma ne gestisce esclusivamente la memorizzazione e la ricerca.

10.2.2 Classe Studente

Responsabilità

Modella uno studente registrato nel sistema, cioè un potenziale beneficiario dei servizi della biblioteca. Raccoglie i dati anagrafici e identificativi utilizzati nelle operazioni di ricerca e nella tracciabilità dei prestiti.

Attributi principali:

- nome: String
- cognome: String
- matricola: String
- email: String

Metodi pubblici principali:

- Costruttore che inizializza tutti gli attributi obbligatori.
- Metodi *getter/setter* per ogni attributo.
- Metodo standard toString().
- Metodi equals(Object) e hashCode() basati sulla matricola.

- Implementazione di Comparable<Studente> (mediante compareTo(Studente altro)), con ordinamento per cognome e nome, utile per la visualizzazione della lista studenti ordinata.

Relazioni:

- La classe Studente partecipa ad un'associazione unidirezionale molti-a-uno proveniente da Prestito: ogni prestito è riferito a un singolo studente, mentre uno studente può risultare associato a più prestiti attivi nel tempo, nel rispetto del vincolo di dominio che limita a tre il numero massimo di prestiti contemporaneamente attivi.
- Gli oggetti della classe Studente sono aggregati in ArchivioStudenti, che mantiene la collezione completa degli studenti registrati nel sistema. L'archivio non gestisce la creazione né il ciclo di vita degli studenti, ma si occupa esclusivamente della loro memorizzazione, ricerca e organizzazione.

10.2.3 Classe Prestito

Responsabilità

Rappresenta una singola operazione di prestito di un libro in favore di uno studente, attiva o conclusa. Contiene le informazioni necessarie alla determinazione delle scadenze, al controllo dei ritardi e all'aggiornamento della disponibilità dei volumi.

Attributi principali:

- studente: Studente – destinatario del prestito.
- libro: Libro – volume concesso.
- dataPrestito: LocalDate – data di inizio del prestito.
- dataRestituzione: LocalDate – data prevista per la restituzione.
- restituito: boolean – indica se il prestito è stato chiuso.

Metodi pubblici principali:

- Costruttore che inizializza tutti gli attributi.
- Metodi getter/setter per tutti gli attributi.
- boolean isAttivo() – restituisce vero se il prestito non è ancora restituito.
- boolean isScaduto(LocalDate oggi) – verifica se la data prevista di restituzione è già trascorsa.
- boolean isInScadenza(LocalDate oggi, int sogliaGiorni) – consente di individuare prestiti prossimi alla scadenza, come richiesto per l'evidenziazione dei prestiti in ritardo o imminenti.

Relazioni:

- La classe Prestito è il punto di origine di due associazioni unidirezionali molti-a-uno: ogni oggetto Prestito fa riferimento a un singolo oggetto della classe *Studente* e a un singolo oggetto della classe *Libro*.
- Gli oggetti della classe Prestito sono aggregati all'interno di ArchivioPrestiti, che gestisce la collezione completa dei prestiti registrati nel sistema. L'archivio non controlla la creazione né il ciclo di vita dei prestiti, ma ne cura esclusivamente la memorizzazione, l'organizzazione e le operazioni di ricerca e aggiornamento.

10.2.4 Classe ArchivioLibri

Responsabilità

Gestisce l'insieme dei libri presenti nel sistema: inserimento, modifica, cancellazione, ricerca e ordinamento ne sono le operazioni relative. È l'unico punto del modello autorizzato a manipolare direttamente la lista di Libro, garantendo l'integrità dei dati e l'unicità dell'ISBN.

Attributi principali:

- libri: List<Libro> – collezione dei libri registrati.

Metodi pubblici principali:

- ArchivioLibri() – inizializza la lista interna.
- void aggiungiLibro(Libro libro) – inserisce un nuovo libro dopo i controlli sull'ISBN.
- void modificaLibro(Libro libroModificato) – aggiorna i dati di un libro esistente.
- void rimuoviLibro(Libro libro) – rimuove un libro dal catalogo.
- Libro cercaPerIsbn(String isbn) – ricerca un libro tramite il codice identificativo.
- List<Libro> cercaPerTitolo(String Titolo) – ricerca un libro in base ad un titolo passato come parametro.
    - List<Libro> cercaPerAutore(StringAutore) – ricerca un libro in base ad un autore passato come parametro.
- List<Libro> getLibriOrdinatiPerTitolo() – restituisce la lista ordinata, sfruttando Comparable<Libro>.
- List<Libro> getTutti() – restituisce tutti i libri presenti.
-

Relazioni:

- La classe ArchivioLibri aggrega oggetti della classe Libro, mantenendo la collezione completa dei volumi registrati nel sistema. L'archivio non è responsabile della creazione né del ciclo di vita dei libri, ma ne gestisce esclusivamente la memorizzazione, l'organizzazione e le operazioni di ricerca, fungendo da contenitore logico della loro presenza nel sistema.

10.2.5 Classe ArchivioStudenti

Responsabilità

Si occupa della gestione dell'elenco degli studenti, traducentesi in inserimento, modifica, rimozione e ricerca per cognome o matricola, oltre alla possibilità di visualizzare la lista ordinata.

Attributi principali:

- studenti: List<Studente> – collezione degli studenti registrati.

Metodi pubblici principali:

- ArchivioStudenti() – inizializza la lista interna.
- void aggiungiStudente(Studente studente) – inserisce un nuovo studente, controllando l'univocità della matricola.

- void modificaStudente(Studente studenteModificato) – aggiorna i dati di uno studente.
- void rimuoviStudente(Studente studente) – elimina uno studente dal sistema.
- List<Studente> getStudentiOrdinati – restituisce la lista ordinata sfruttando Comparable<Studente>.
- List<Studente> cercaPerCognome(String cognome) – ricerca per cognome.
- Studente cercaPerMatricola(String matricola) – ricerca puntuale per matricola.
-

Relazioni:

- La classe ArchivioStudenti aggrega oggetti della classe Studente, mantenendo la collezione completa degli studenti registrati nel sistema. L'archivio non è responsabile della creazione né del ciclo di vita degli studenti, ma ne gestisce esclusivamente la memorizzazione, l'organizzazione e le operazioni di ricerca, fungendo da contenitore logico della loro presenza nel sistema.

10.2.6 Classe ArchivioPrestiti

Responsabilità

È il componente centrale per la gestione dei prestiti. Mantiene l'elenco dei prestiti attivi e passati, applica i vincoli di business, in particolare il limite di tre prestiti attivi per studente e il controllo delle copie disponibili, e fornisce viste ordinate e filtri di ricerca utilizzati dall'interfaccia.

Attributi principali:

- prestitiPerStudente: Map<String, List<Prestito>> – indicizza i prestiti per matricola dello studente, facilitando il conteggio e il recupero rapido dei prestiti attivi.

Metodi pubblici principali:

- ArchivioPrestiti() – inizializza le strutture interne.
- Prestito registraPrestito(Studente studente, Libro libro, LocalDate dataRestituzione)
  - crea un nuovo prestito,
  - verifica che lo studente non abbia superato il limite di tre prestiti,
  - controlla la disponibilità del libro e, in caso positivo, decrementa le copie disponibili.
- void registraRestituzione(Prestito prestito) – segna il prestito come restituito, aggiorna la disponibilità del libro e rimuove il prestito dall'elenco degli attivi.
- List<Prestito> getPrestitiPerData() – restituisce i prestiti ancora attivi ordinati per data prevista di restituzione.
- int contaPrestitiAttivi(Studente studente) – restituisce il numero di prestiti attivi per uno studente passato come parametro.
- List<Prestito> cercaPrestitiAttivi(String cognome, String matricola, String titoloLibro) – ricerca combinata per studente e libro.
- List<Prestito> getPrestitiPerStudente(Studente studente) – restituisce tutti i prestiti di uno studente, utile anche per controllare il numero di prestiti attivi.
  - List<Prestito> getTutti() – restituisce tutti i prestiti attivi.

Relazioni:

- La classe ArchivioPrestiti aggrega oggetti della classe Prestito, mantenendo la collezione completa dei prestiti registrati nel sistema. L'archivio non è responsabile della creazione né del ciclo di vita dei prestiti, ma ne gestisce esclusivamente la memorizzazione, l'organizzazione e le operazioni di ricerca e aggiornamento, fungendo da contenitore logico delle informazioni relative ai prestiti attivi e storici.

10.3 Scelte progettuali: coesione, accoppiamento e principi adottati

Il modello statico adottato per l'applicazione riflette una serie di scelte progettuali orientate alla chiarezza strutturale, alla manutenibilità del codice e alla corretta separazione delle responsabilità. Le classi principali (Libro, Studente e Prestito) rappresentano entità del dominio con caratteristiche e comportamenti ben definiti, mentre gli archivi dedicati (ArchivioLibri, ArchivioStudenti e ArchivioPrestiti) svolgono il ruolo di contenitori logici e gestori delle rispettive collezioni. Questa distinzione netta tra dati e loro gestione costituisce uno degli elementi cardine dell'intero modello.

Dal punto di vista della **coesione**, ogni classe presenta un insieme di responsabilità omogenee e chiaramente circoscritte. La classe Libro si occupa esclusivamente della rappresentazione dei dati bibliografici e della gestione delle copie disponibili; Studente modella i dati anagrafici e identificativi, senza inglobare logiche relative ai prestiti; Prestito incapsula la relazione tra studente e libro, insieme alle informazioni temporali e allo stato dell'operazione. Gli archivi, a loro volta, mantengono un'elevata coesione interna, poiché si limitano alla memorizzazione, ricerca e organizzazione delle rispettive entità, senza assumere responsabilità estranee al loro ruolo.

Per quanto riguarda l'**accoppiamento**, il sistema è progettato per mantenere dipendenze ridotte e ben controllate. Le associazioni tra le classi sono unidirezionali: un oggetto Prestito conosce lo studente e il libro coinvolti, ma né Studente né Libro mantengono riferimenti ai prestiti. Questa scelta evita dipendenze circolari e riduce il rischio di inconsistenze, oltre a semplificare la gestione del ciclo di vita degli oggetti. Analogamente, gli archivi non dipendono gli uni dagli altri e non condividono strutture dati interne: ciascuno opera in modo autonomo sulla propria collezione, contribuendo a mantenere un basso livello di accoppiamento tra i componenti del sistema.

Le scelte progettuali adottate rispettano, inoltre, diversi **principi di buona progettazione orientata agli oggetti**. Il **Single Responsibility Principle (SRP)** è applicato in modo rigoroso: ogni classe svolge un unico compito ben definito, evitando sovrapposizioni di responsabilità. Il **principio dell'incapsulamento** è rispettato grazie alla protezione degli attributi e all'accesso controllato tramite metodi getter e setter. L'uso dell'interfaccia Comparable<T> per le classi Libro e Studente consente di definire un ordinamento naturale senza introdurre dipendenze aggiuntive, favorendo l'estensibilità del sistema in conformità con l'**Open/Closed Principle (OCP)**. Infine, la scelta di centralizzare la gestione dei prestiti in

ArchivioPrestiti evita duplicazioni di informazioni e garantisce una visione coerente dello stato del sistema, in linea con il principio di **Information Hiding**.
Nel complesso, il modello statico risulta quindi **coerente, modulare e facilmente estendibile**, capace di supportare in modo efficace i requisiti funzionali del sistema e di costituire una base solida per le fasi successive di implementazione e manutenzione.

11 Modello dinamico

11.1 Interazioni tra classi

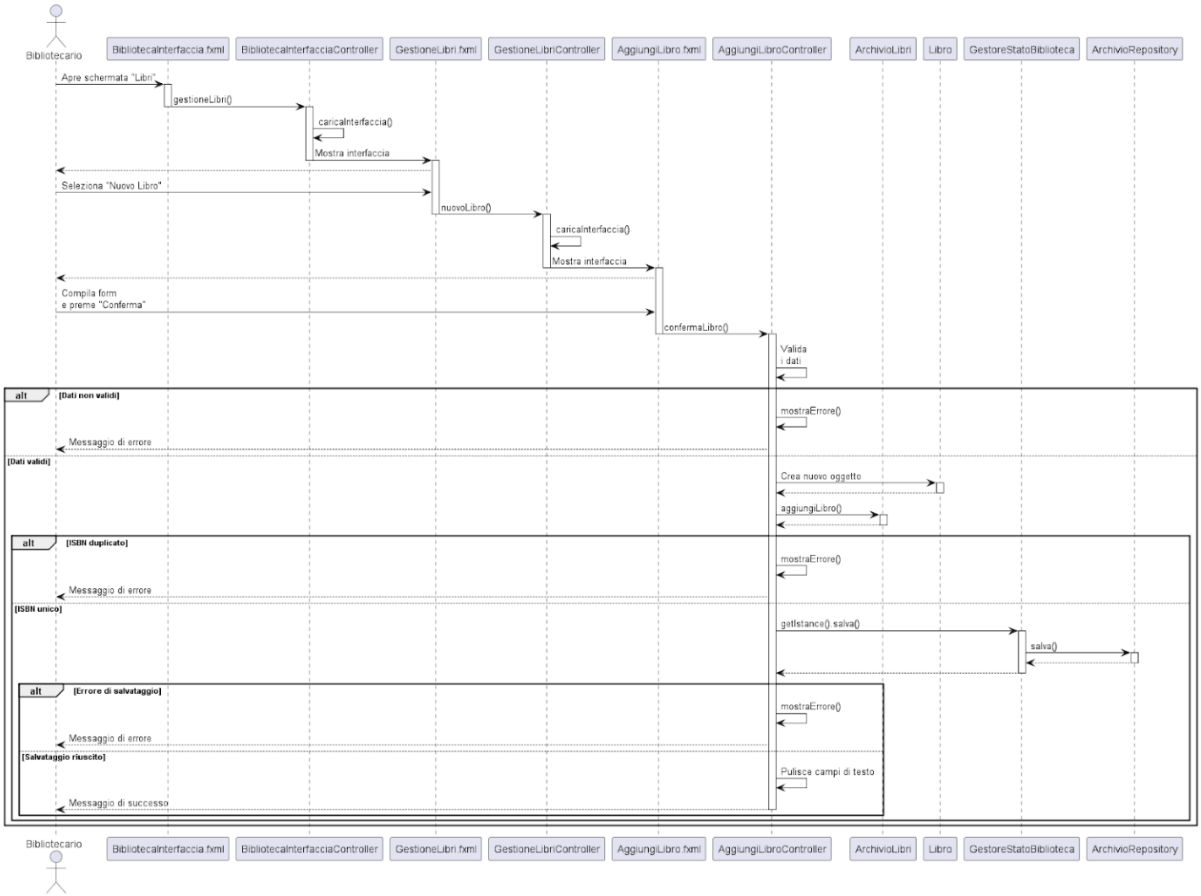
In questa sezione vengono documentate le interazioni tra le classi per i casi d'uso ritenuti più significativi, in modo da mostrare come il modello statico venga effettivamente utilizzato a runtime.
Sono stati modellati, dunque, i seguenti casi d'uso:

- **UC-1: Inserimento di un libro**
- **UC-10: Ricercare uno studente**
- **UC-11: Registrare un prestito**
- **UC-14: Registrare la restituzione**

Per ciascuno di essi viene presentato un **diagramma di sequenza UML** (in PlantUML), congiuntamente ad una breve descrizione delle responsabilità delle classi coinvolte.

UC-1: Inserimento di un libro

Diagramma di sequenza:



Obiettivo: Inserire nel sistema un nuovo libro, specificandone titolo, autore, anno di pubblicazione, ISBN e numero di copie disponibili; aggiornare la lista dei libri.

Classi coinvolte:

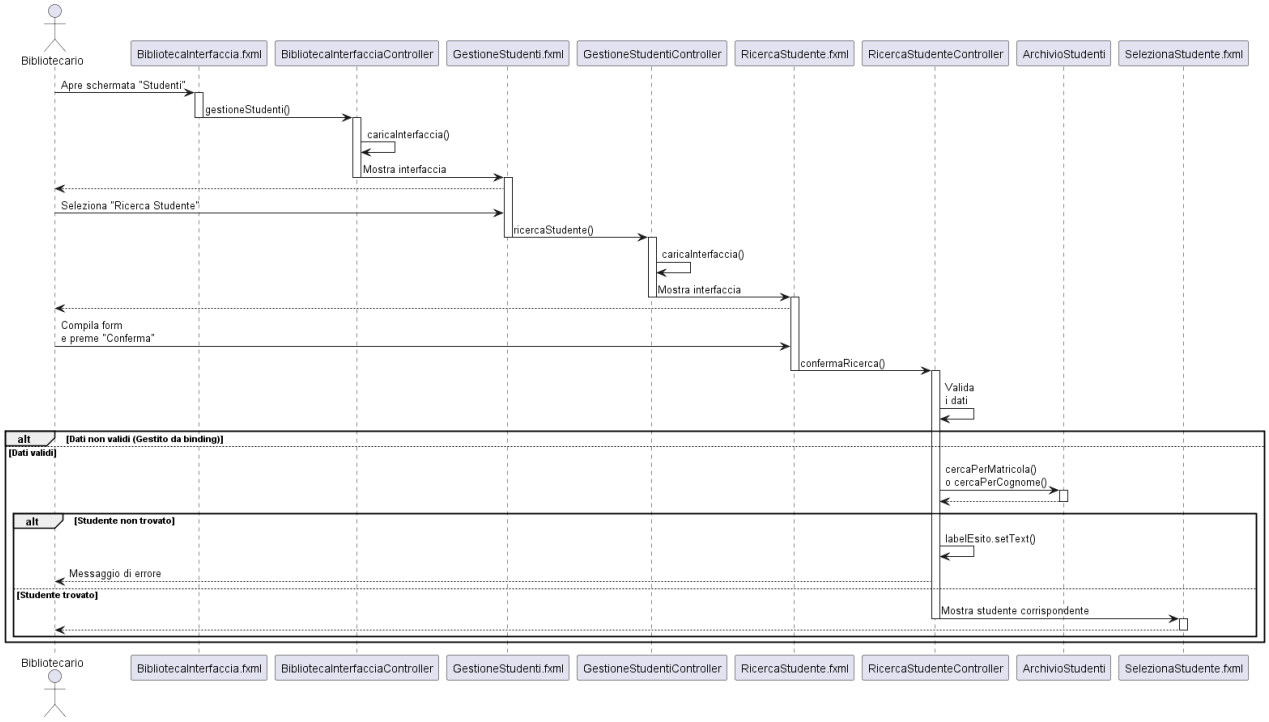
- **BibliotecaInterfaccia.fxml, GestioneLibri.fxml, AggiungiLibro.fxml**: rappresenta le interfacce con cui il bibliotecario interagisce.
- **BibliotecaInterfacciaController, GestioneLibriController, AggiungiLibroController**: coordina l'operazione di registrazione del prestito.
- **ArchivioLibri**: gestisce la collezione di oggetti Libro.
- **Libro**: classe di dominio che rappresenta un singolo oggetto Libro.
- **GestoreStatoBiblioteca**, **ArchivioRepository**: gestiscono il salvataggio persistente dello stato.

Commento al diagramma:

1. Il bibliotecario apre la schermata "Libri" e seleziona "Nuovo Libro".
2. Il bibliotecario compila il form della AggiungiLibro.fxml e preme il pulsante di conferma.
3. La view invia i dati al AggiungiLibroController,, che si occupa di:
   - **3a.** Validare i dati (campi vuoti, anno non numerico o superiore al corrente, ISBN non nel formato corretto).
     - **3a.1** Se le informazioni inserite non sono valide, il controller mostra un messaggio di errore.
4. Se le verifiche hanno esito positivo, AggiungiLibroController:
   - **4a.** Crea un nuovo oggetto Libro.
   - **4b.** Aggiorna la lista dei libri, invocando aggiungiLibro() su ArchivioLibri.
     4b.1 Se il metodo aggiungiLibro() fallisce (ad esempio per ISBN duplicato) il controller mostra un messaggio di errore.
   - **4c.** Invoca getInstance() su GestoreStatoBiblioteca per scrivere lo stato aggiornato su file.
     4c.1 Se il salvataggio fallisce, il controller mostra un messaggio di errore.
     4c.2 Se il salvataggio riesce, il controller mostra un messaggio di successo.

UC-10: Ricercare uno studente

Diagramma di sequenza:



Obiettivo: Ricercare uno studente registrato nel sistema, specificandone cognome o matricola.

Classi coinvolte:

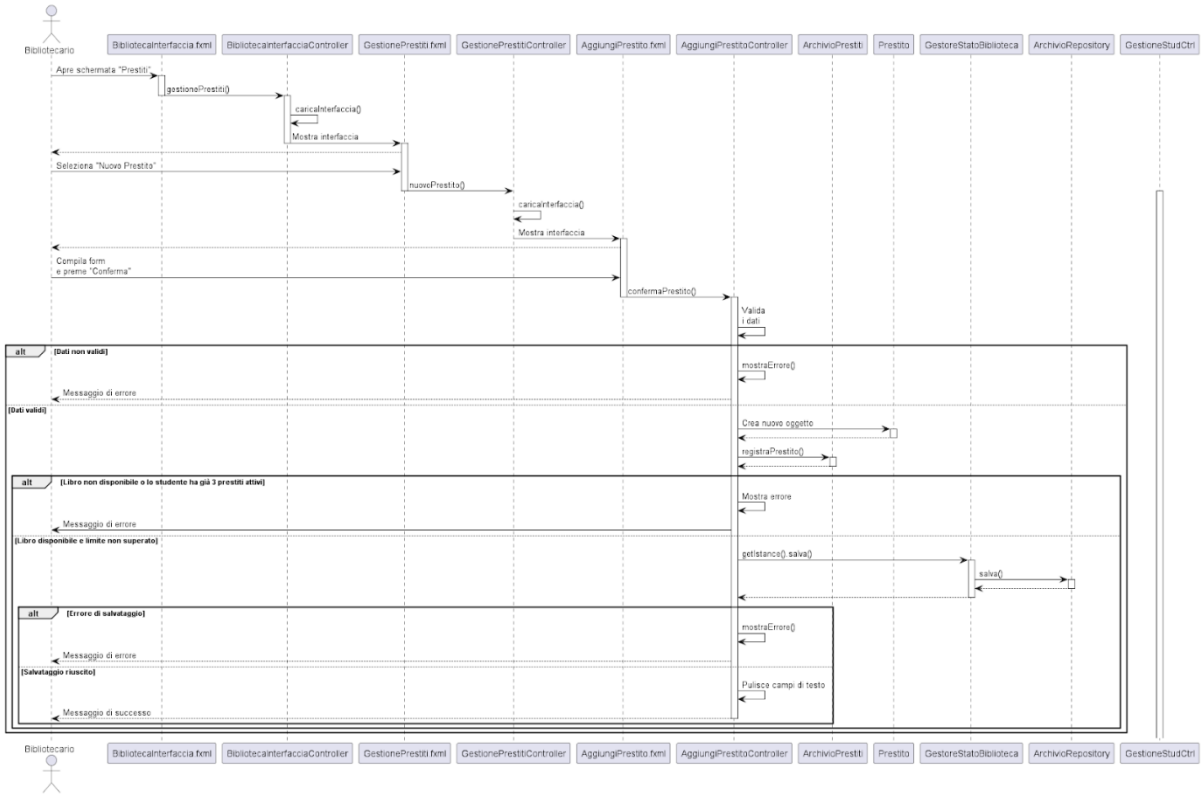
- **BibliotecaInterfaccia.fxml, GestioneStudenti.fxml, RicercaStudente.fxml, SelezioneStudente.fxml**: rappresentano le interfacce con cui il bibliotecario interagisce.
- **BibliotecaInterfacciaController, GestioneStudentiController, RicercaStudenteController**: ricevono l'evento di conferma, validano i dati, dialogano con l'archivio.
- **ArchivioStudenti**: gestisce la collezione di oggetti Studente.

Commento al diagramma:

1. Il bibliotecario apre la schermata "Studenti" e seleziona "Ricerca Studente".
2. Il bibliotecario compila il form della RicercaStudenti.fxml e preme il pulsante di conferma.
3. La view invia i dati a RicercaStudentiController.
4. L a validazione dei dati è gestita dal binding.
5. Se la verifica ha esito positivo, il controller verifica che lo studente sia già registrato nel sistema, invocando cercaPerCognome() o cercaPerMatricola() su ArchivioStudenti.
   5a. Se lo studente non esiste, il controller mostra un messaggio di errore.
6. Se le verifiche hanno esito positivo, il controller richiede a SelezioneStudente.fxml di mostrare lo studente corrispondente.

UC-11: Registrare un prestito

Diagramma di sequenza:



Obiettivo: Registrare un prestito, specificandone studente interessato, libro e data del prestito, e rispettando i seguenti vincoli:

- Lo studente non ha superato il limite massimo di prestiti attivi (3).
- È disponibile almeno una copia del libro.

Aggiornare le copie disponibili del libro e la lista prestiti attivi dello studente.

Classi coinvolte:

- **BibliotecaInterfaccia.fxml, GestionePrestiti.fxml, AggiungiPrestito.fxml**: rappresentano le interfacce con cui il bibliotecario interagisce.
- **BibliotecaInterfacciaController, GestionePrestitiController, AggiungiPrestitoController**: coordinano l'operazione di registrazione del prestito.
- **ArchivioPrestiti**: contiene la collezione di oggetti Prestito.
- **Prestito**: classe di dominio che rappresenta un singolo oggetto Prestito.
- **GestoreStatoBiblioteca**, **ArchivioRepository**: gestiscono il salvataggio persistente dello stato.

Commento al diagramma:

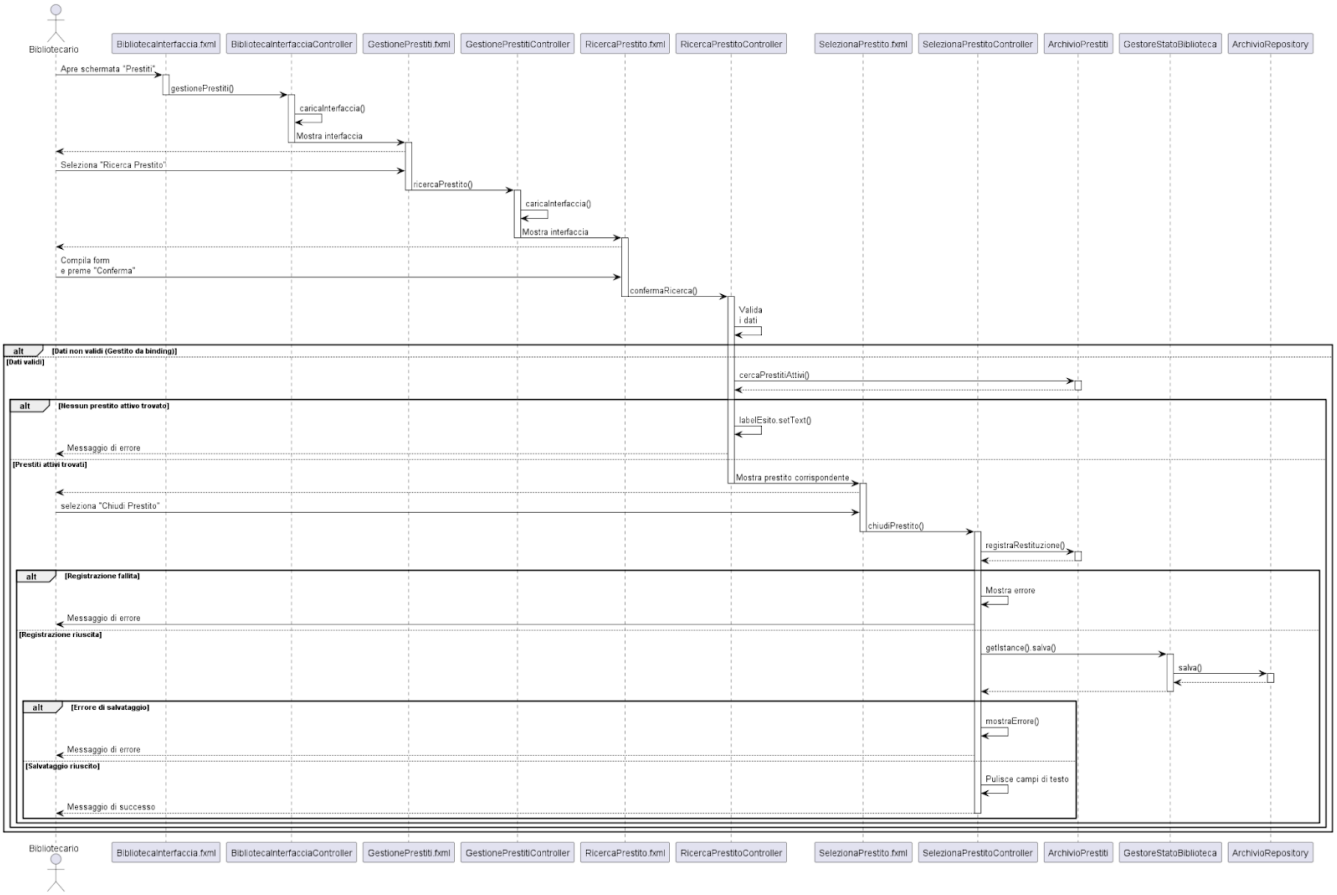
1. Il bibliotecario apre la schermata "Prestiti" e seleziona "Nuovo Prestito".
2. Il bibliotecario compila il form di AggiungiPrestito.fxml e preme il pulsante di conferma.
3. La view invia i dati a AggiungiPrestitoController, che si occupa di:
   3a. Validare i dati (campi vuoti, data in formato non valido o superiore alla corrente).

3a.1 Se le informazioni inserite non sono valide,il controller mostra un messaggio di errore.

4. Se la verifica ha esito positivo, AggiungiPrestitoController:
   4a. Crea un nuovo oggetto Prestito.
   4b. Invoca registraPrestito() su ArchivioPrestiti.
5. ArchivioPrestiti si occupa di:
   5a. Verificare che il libro abbia copie disponibili e che lo studente non abbia superato il limite massimo di prestiti attivi.
   5a.1 Se il libro non è disponibile o se lo studente ha già 3 prestiti attivi, il controller mostra un messaggio di errore.
   5b. Aggiornare il numero di copie disponibili del libro e la lista prestiti attivi dello studente.
6. Se tutte le verifiche hanno esito positivo AggiungiPrestitoController, invoca getInstance() su GestoreStatoBiblioteca per scrivere lo stato aggiornato su file.
   6a. Se il salvataggio fallisce, il controller mostra un messaggio di errore.
   6b. Se il salvataggio riesce, il controller mostra un messaggio di successo.

UC-14: Registrare la restituzione

Diagramma di sequenza:



Obiettivo: Registrare la restituzione di un libro, aggiornare il numero di copie disponibili del libro e la lista dei prestiti attivi dello studente.

Classi coinvolte:

- **BibliotecaInterfaccia.fxml, GestionePrestiti.fxml, RicercaPrestito.fxml, SelezionePrestito.fxml**: rappresentano le interfacce con cui il bibliotecario interagisce.
- **BibliotecaInterfacciaController, GestionePrestitiController, RicercaPrestitoController, SelezionaPrestitoController**: coordinano l'operazione di chiusura del prestito.
- **ArchivioPrestiti**: gestisce la collezione di oggetti Prestito.
- **GestoreStatoBiblioteca**, **ArchivioRepository**: gestiscono il salvataggio persistente dello stato.

Commento al diagramma:

1. Il bibliotecario apre la schermata "Prestiti" e seleziona "Ricerca Prestiti Attivi".
2. Il bibliotecario inserisce il criterio di ricerca del prestito (cognome o matricola dello studente per cui è attivo il prestito e titolo del libro) nel form di RicercaPrestito.fxml e preme il pulsante di conferma.
3. La view invia i dati RicercaPrestitiController.
4. La validazione dei dati è gestita dal binding.
5. Se le informazioni inserite sono valide, il controller verifica che il prestito risulti nel sistema, invocando cercaPrestitiAttivi() su ArchivioPrestiti.
   5a. Se il prestito non esiste, il controller mostra un messaggio di errore.
6. Se la verifica ha esito positivo, il controller richiede a SelezionePrestito.fxml di mostrare il prestito corrispondente.
7. Il bibliotecario preme il pulsante "Chiudi Prestito".
8. La view invia la richiesta a SelezionaPrestitoController, che registra la restituzione, invocando chiudiPrestito().
9. ArchivioPrestiti, tramite registraRestituzione(), si occupa di:
   9a. Segnare il prestito come restituito.
   9a.1 Se la registrazione fallisce, il controller mostra un messaggio di errore.
   9b. Aggiornare il numero di copie disponibili del libro e la lista prestiti attivi dello studente.
10. Se la registrazione ha esito positivo, SelezionaPrestitiController invoca getInstance() su GestoreStatoBiblioteca per scrivere lo stato aggiornato su file.
    10a. Se il salvataggio fallisce, il controller mostra un messaggio di errore.
    10b. Se il salvataggio riesce, il controller mostra un messaggio di successo.

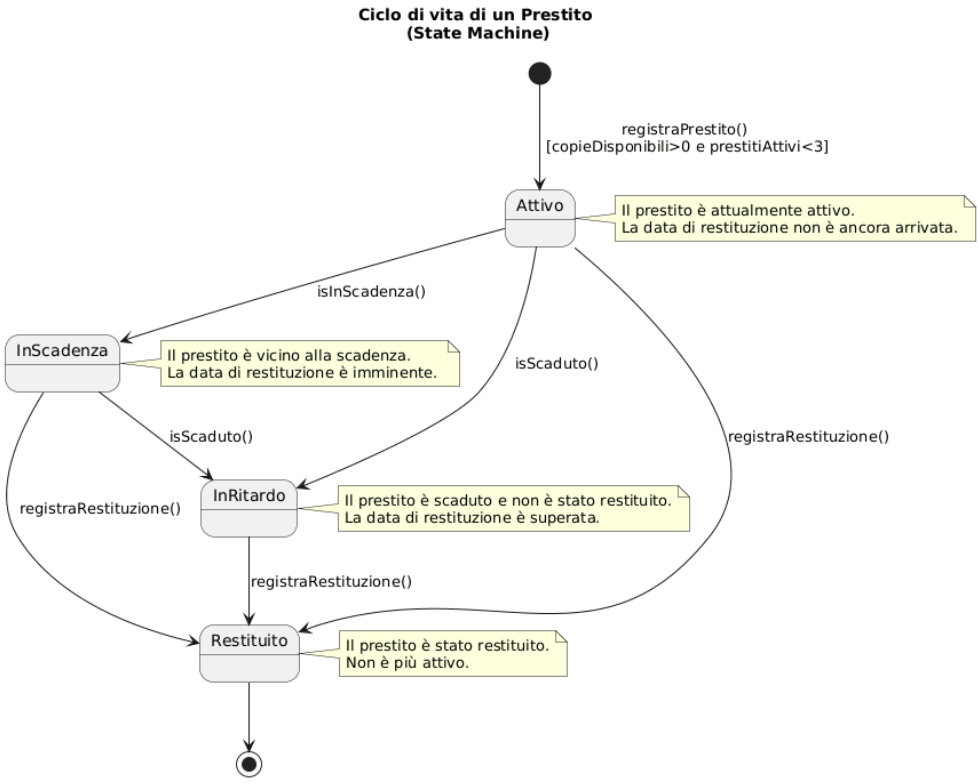
11.2 Macchina a stati del Prestito

Come ulteriore diagramma UML, ci riserviamo di illustrare un diagramma di macchina a stati, nella specificità dello studio dello stato del generico oggetto Prestito.
Gli stati principali che esso può assumere sono:

- **Attivo** – il prestito è stato registrato e non è ancora terminato, ossia ancora non è avvenuta una restituzione.
- **Restituito** – il libro è stato riconsegnato.

- **inScadenza** – il prestito deve essere terminato entro una certa soglia di giorni, altrimenti scatterà l'evidenziazione di un ritardo.
- **inRitardo** – il prestito non è stato terminato entro la data prevista della restituzione del libro, quindi se ne ha una segnalazione.
  Questi ultimi due stati sono da intendersi come proprietà derivate, poiché dipendenti unicamente dalla data di uno specifico giorno corrente e dalla data prevista per la riconsegna del libro. I metodi isScaduto() e isInScadenza(), a tale scopo, sono fortemente esemplificativi.

Lo stato iniziale, punto di partenza del ciclo, è rappresentato da un cerchio nero pieno, dove non si hanno attività o trigger (eventi che innescano potenziali cambiamenti). Tuttavia, nel momento in cui vogliamo rendere il nostro oggetto Prestito **Attivo**, nell'ambito di una "entry", al trigger **registraPrestito()** si subordina la guardia (ossia la condizione) **copieDisponibili > 0 && prestitiAttivi < 3.** Acceduti allo stato **Attivo** come conseguenza delle due situazioni, da esso sono 3 le ramificazioni possibili: il trigger **isInScadenza()** potrebbe portarci allo stato **InScadenza**, mentre, da questo o dalla proprietà primigenia, mediante **isScaduto()**, si effettua una transizione presso **InRitardo.**
Si può approdare allo stato **Restituito** da tutte e 3 le situazioni appena appurate, ogni volta per grazia del trigger **registraRestituzione()**. Una volta restituito il prestito, il diagramma viene ultimato da un cerchio nero, bordato di bianco, segnante uno stato finale, precludente ulteriori attività e transizioni.



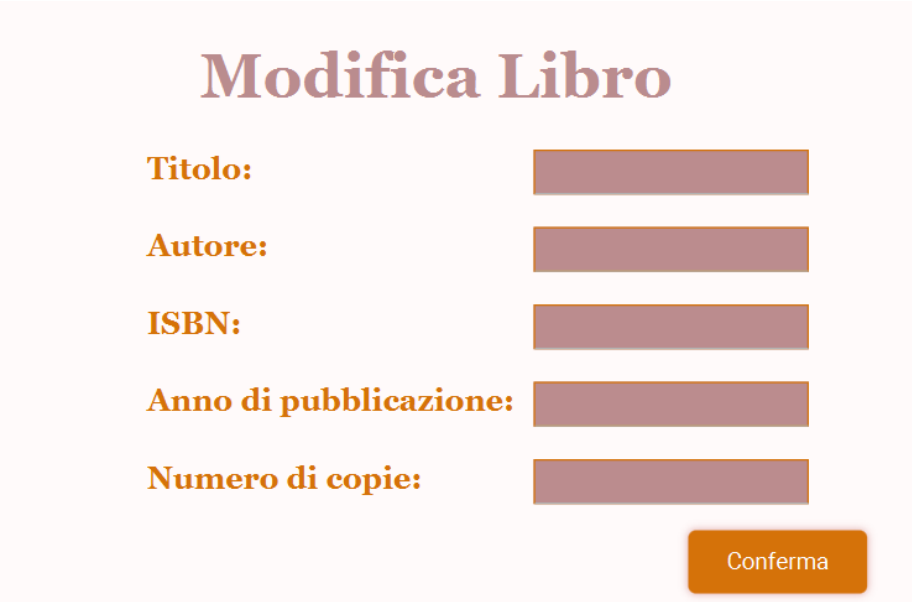
12 Design dell’interfaccia utente

Ultimiamo il nostro documento di progettazione con la descrizione e l’illustrazione dei mock-up delle schermate principali della nostra interfaccia utente.
Ogni immagine sarà preceduta da una rappresentazione testuale e procederemo iterativamente, circa un ordine logico da noi designato.
Esordiamo con la prima schermata e vediamone i dettagli:

- **Titolo:** "Biblioteca Universitaria", identifica lo scopo dell’applicazione.
  Sfondo: Sulla destra è presente una pila di libri, un elemento grafico immediatamente riconoscibile che rafforza il tema della biblioteca e aggiunge un tocco visivo.
  Area Centrale: Il cuore dell’interazione è costituito dai tre pulsanti principali, disposti in maniera verticale.

- **Titolo:** “Gestione Libri”, serve ad identificare la sezione.
  Pulsanti: Tre pulsanti orizzontali che rappresentano le funzioni di gestione dei libri.

- **Titolo:** “Modifica Libro”, serve ad identificare la sezione.
  Campi di input: Sono presenti i 5 campi per l’identificazione e la descrizione di un libro, affiancati da aree di testo dove inserire i dati.
  Pulsante d’azione: “Conferma”, serve a salvare le modifiche.

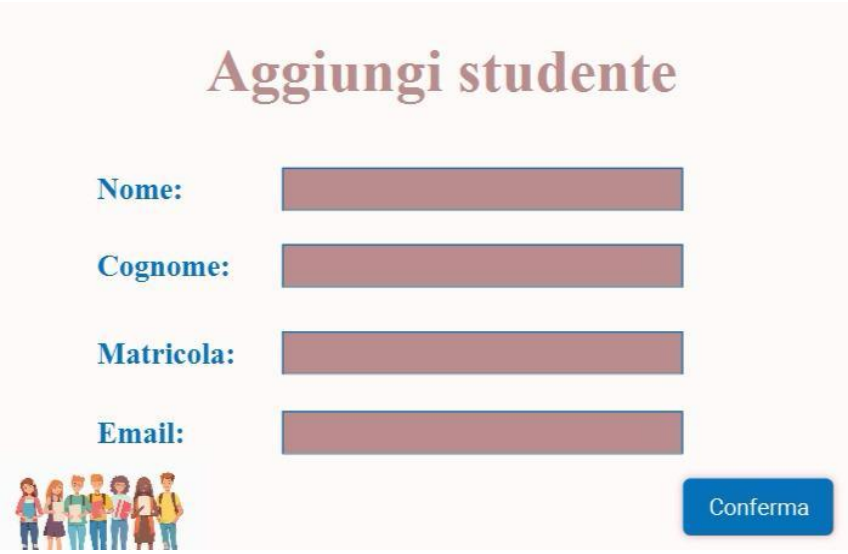


- **Titolo:** “Ricerca Libro”, serve ad identificare la sezione.

Pulsanti: Due pulsanti orizzontali che rappresentano le funzioni successive alla ricerca, ovvero una modifica o una rimozione del libro.

- **Titolo**: “Ricerca Prestito”, serve ad identificare la sezione.
  Campi di input: Sono presenti due campi di input per l’inserimento di due parametri: l’identificativo dello studente (cognome o matricola) e l’identificativo del libro (titolo oppure ISBN).
  Sfondo: L’icona della lente d’ingrandimento, posta accanto alla barra di ricerca, suggerisce chiaramente la funzione legata all’input testuale.
  Pulsante d’azione: “Conferma”, serve a restituire dei risultati.

- **Titolo**: “Aggiungi studente”, serve ad identificare la sezione.
  Campi di input: Sono presenti i 4 campi per l’identificazione di un nuovo studente, affiancati da aree di testo dove inserire i dati.
  Pulsante d’azione: “Conferma”, serve a salvare i dati dello studente.
  Sfondo: In basso a sinistra, è presente un’immagine decorativa raffigurante degli studenti, che aggiunge un tocco visivo e rafforza il tema dell’interfaccia.



- **Titolo**: “Registro studenti”, serve ad identificare la sezione.

Tabella: Contiene il registro delle informazioni degli studenti, ordinato per cognome e nome.

Sfondo: Attualmente la tabella mostra il messaggio "Nessun contenuto nella tabella", indicando che non ci sono dati registrati.

- **Titolo**: “Ricerca Prestito”, serve ad identificare la sezione.

Pulsante d’azione: “Chiudi Prestito”, serve a registrare la restituzione di un libro preso in prestito.