



Library Management Application

Elaborato di Ingegneria del Software

Docente: Enrico Vicario

Studenti: Luca Lascialfari, Marco Siani, Tommaso Puzzo

Contenuti

1	Introduzione	1
1.1	Obiettivi del progetto	1
1.2	Tecnologie e architettura utilizzate	1
2	Progettazione	2
2.1	Attori coinvolti e relativi casi d'uso	2
2.2	Use Case Template	3
2.2.1	Use Case condivisi	3
2.2.2	Utente	5
2.2.3	Amministratore	7
2.3	Diagramma delle classi	10
2.4	Aspetti rilevanti del progetto	14
2.4.1	Model View Controller	14
2.4.2	Singleton	14
2.4.3	DAO	15
2.5	Page Navigation Diagram	16
2.6	Entity Relationship Diagram	17
3	Implementazione delle classi	18
3.1	DomainModel	18
3.1.1	Element	19
3.1.2	Book	19
3.1.3	DigitalMedia	19
3.1.4	Periodic Publication	20
3.1.5	Genre	20
3.1.6	User	20
3.2	Controller	21
3.2.1	AddItemViewController	21
3.2.2	BaseViewController	21
3.2.3	BorrowedItemsViewController	23
3.2.4	ElementCheckViewController	23
3.2.5	ElementDetailsViewController	23
3.2.6	HomeViewController	24
3.2.7	LoginViewController	24
3.2.8	MenuBarViewController	24
3.2.9	SignupViewController	24
3.3	Orm	26
3.3.1	ConnectionManager	26
3.3.2	ElementDAO	26
3.3.3	BookDAO	26
3.3.4	BorrowsDAO	27
3.3.5	DigitalMediaDAO	27
3.3.6	GenreDAO	27
3.3.7	PeriodicPublicationDAO	27
3.3.8	UserDAO	28

3.4	Service	28
3.4.1	LibraryAdminService	28
3.4.2	LibraryUserService	28
3.4.3	MainService	28
3.4.4	UserService	28
3.5	View	29
3.5.1	SignUp-view	30
3.5.2	AddItem-view	30
3.5.3	ElementDetails-view	31
3.5.4	Login-view	31
3.5.5	Home-view	32
3.5.6	Borrowed-view	32
3.6	HelloApplication	33
3.7	Struttura del database	33
4	Testing	35
4.1	BookDAOTest	36
4.2	BorrowsDAOTest	36
4.3	ConnectionManagerTest	37
4.4	DigitalMediaDAOTest	37
4.5	ElementDAOTest	38
4.6	GenreDAOTest	39
4.7	PeriodicPublicationDAOTest	40
4.8	UserDAOTest	40
4.9	MainServiceTest	41
4.10	UserServiceTest	41
4.11	LibraryUserServiceTest	41
4.12	LibraryAdminServiceTest	41
4.13	Risultati Maven	42

1 Introduzione

1.1 Obiettivi del progetto

Questo progetto implementa un sistema di gestione di una biblioteca che permette:

- agli utenti di prendere in prestito e restituire libri, media digitali e periodici.
- agli amministratori di aggiungere, modificare e rimuovere ognuno di questi elementi dal catalogo della biblioteca.

Per **elemento** si intende un qualsiasi prodotto presente nel catalogo della biblioteca, indipendentemente dalla sua tipologia effettiva.

1.2 Tecnologie e architettura utilizzate

L'applicativo è sviluppato in **Java 23** utilizzando l'IDE **IntelliJ** della suite di JetBrains. Per garantire la persistenza dei dati, è stato adottato un database da remoto gestito tramite **PostgreSQL**¹. La connessione al database è gestita attraverso le librerie **JDBC**. L'interfaccia grafica è stata sviluppata utilizzando le librerie open-source **JavaFX v. 17.0.6**², le quali permettono la creazione di applicazioni GUI in Java. Per la realizzazione delle pagine di navigazione è stato impiegato **Scene Builder**, che fornisce un'interfaccia grafica utile a tale scopo. I diagrammi di progettazione, inclusi i diagrammi delle classi e quelli relativi ai casi d'uso, sono stati creati tramite **StarUML**. Il processo di testing è stato realizzato con l'utilizzo del framework **JUnit 5**³, supportato dall'uso della libreria **Mockito v. 5.7.0**⁴ per la simulazione delle dipendenze e dall'adozione di **Maven v. 4.0.0**⁵ per l'automazione del processo di build.

Tra le architetture utilizzate abbiamo invece:

- **MVC**: Struttura l'applicazione separando i dati o **Model**, l'interfaccia utente, definita anche come **View**, e la logica di controllo, implementata tramite i **Controller**.
- **Service Layer**: Introduce un livello intermedio tra il Controller e il Database.
- **DAO**: Fornisce un'interfaccia per l'accesso ai dati, incapsulando le query SQL e l'interazione con il database.

¹[PostgreSQL](#)

²[JavaFX](#)

³[JUnit](#)

⁴[Mockito](#)

⁵[Maven](#)

2 Progettazione

2.1 Attori coinvolti e relativi casi d'uso

L'applicativo individua 2 diversi attori:

- Amministratore,
- Utente.

Un ulteriore attore è il sistema stesso, il quale però agisce solo in risposta alle azioni degli altri.

Ognuno degli attori è caratterizzato da diverse funzionalità, alcune delle quali in comune, come si può notare nei seguenti diagrammi dei casi d'uso.

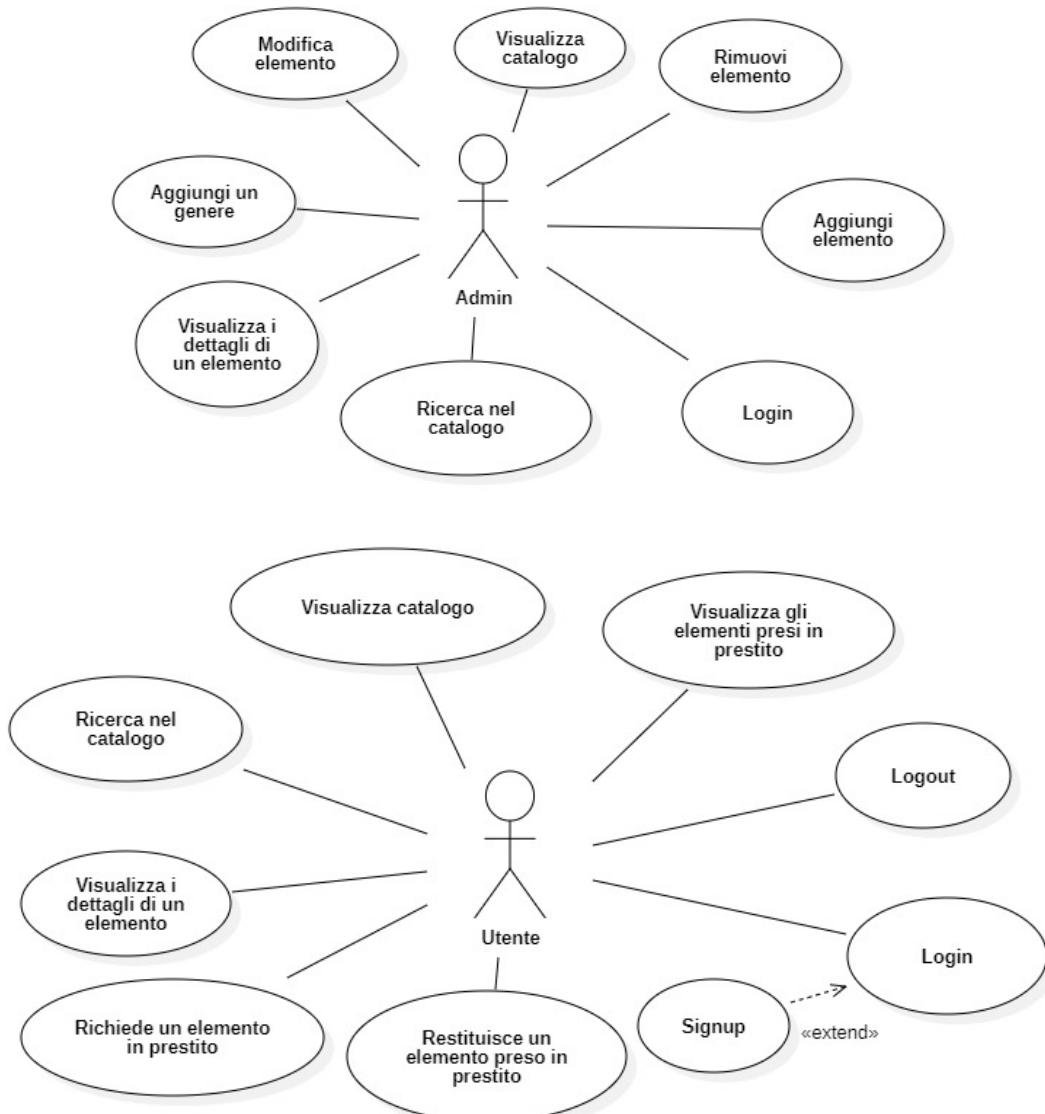


Figure 1: Diagramma dei casi d'uso

2.2 Use Case Template

2.2.1 Use Case condivisi

In questa sezione sono riportati gli **Use Case** relativi a molteplici attori.

Use Case	1. Login
Level	Function
Description	Utente usa e-mail e password per fare il login
Actors	Utente, Amministratore
Pre-conditions	Disporre di un account Essere connessi al database
Post-conditions	Utente esegue l'accesso correttamente o viene segnalato errore al login
Normal flow	1. Utente inserisce e-mail e password 2. Utente preme il tasto Login 3. Login avviene con successo
Alternative flows	3A. Se identificativo non nel database, compare scritta "E-mail non presente" 3B. Se identificativo esiste ma password sbagliata, compare scritta "Password errata" 3C. Se non c'è connessione al database, compare scritta "Impossibile connettersi con il database"

Table 1: Use Case Login

Use Case	2. Visualizza dettagli elemento
Level	Function
Description	Utente visualizza informazioni dettagliate di uno specifico elemento selezionato
Actors	Utente, Amministratore
Pre-conditions	1. Essere connessi al database 2. Aver effettuato il login
Post-conditions	Utente visualizza informazioni dettagliate sull'elemento
Normal flow	1. Utente clicca su un elemento dalla pagina Home 2. Le informazioni sull'elemento vengono fornite
Variations	1. Un Utente può accedere ai dettagli di un elemento anche dalla pagina Prestiti
Alternative flows	1A. Se non c'è connessione al database, compare la scritta "Impossibile connettersi al database."

Table 2: Use Case Dettagli elemento

2.2 Use Case Template

Use Case	3. Logout
Level	Function
Description	Utente effettua il logout
Actors	Utente, Amministratore
Pre-conditions	Disporre di un account Avere eseguito il login
Post-conditions	Utente viene disconnesso correttamente
Normal flow	1. Utente preme il tasto Logout 2. Logout avviene con successo

Table 3: Use Case Logout

Use Case	5. Ricerca nel catalogo
Level	User goal
Description	Utente utilizza filtri per ottenere una lista di elementi con le caratteristiche desiderate
Actors	Utente, Amministratore
Pre-conditions	1. Disporre di un account 2. Essere connessi al database
Post-conditions	Vengono mostrati uno o più elementi cercati o viene segnalato errore al login
Normal flow	1. Utente inserisce parametri di ricerca 2. Vengono visualizzati a schermo gli elementi che corrispondono ai parametri inseriti

Table 4: Use Case Ricerca nel catalogo

Use Case	6. Visualizza catalogo
Level	Function
Description	Utente visualizza una tabella contenente gli elementi presenti nel catalogo
Actors	Utente, Amministratore
Pre-conditions	1. Disporre di un account 2. Essere connessi al database 3. Aver effettuato il login
Post-conditions	Utente visualizza una serie di elementi
Normal flow	1. Utente accede e visualizza tutti gli elementi presenti nel catalogo

Table 5: Use Case Visualizza catalogo

2.2.2 Utente

L'Utente è l'attore che si connetterà al sistema con lo scopo di visualizzare il catalogo, prenotare uno o più elementi e di restituirli. Qui di seguito saranno presentati solo gli **Use Case** che prevedono l'**Utente** come unico attore:

Use Case	4. Signup
Level	User Goal
Description	Utente crea un account personale
Actors	Utente
Pre-conditions	1. Disporre di una e-mail 2. Essere connessi al database
Post-conditions	Utente crea un account correttamente o viene segnalato errore al signup
Normal flow	1. Utente inserisce e-mail, password, nome e cognome 2. Utente preme il tasto Signup 3. Signup avviene con successo
Variations	1. Utente può inserire anche un numero di telefono
Alternative flows	3A. Se il testo inserito nel campo e-mail non corrisponde a un indirizzo e-mail possibile, compare scritta "E-mail non valida" 3B. Se la password contiene meno di 8 caratteri o non ha almeno una lettera minuscola, una maiuscola, una cifra e un carattere speciale (!, #, \$, %, &, =, ?, @), compare la scritta "Password non valida" 3C. Se il nome non è stato inserito, compare la scritta "Nome non inserito" 3D. Se il cognome non è stato inserito, compare la scritta "Cognome non inserito" 3E. Se è stato inserito un numero di telefono non valido, compare la scritta "Numero di telefono non valido" 3F. Se e-mail nel database, compare scritta "E-mail già in uso" 3G. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database."

Table 6: Use Case Signup

2.2 Use Case Template

Use Case	7. Richiesta di un elemento in prestito
Level	User Goal
Description	Utente prende un elemento in prestito
Actors	Utente
Pre-conditions	1. Disporre di un account 2. Essere connessi al database 3. Aver effettuato il login
Post-conditions	L'elemento viene dato in prestito all'Utente o viene segnalato un errore
Normal flow	1. Utente visualizza un elemento 2. Utente preme il tasto per il prestito 3. Utente riceve l'elemento in prestito
Alternative flows	3A. Se l'elemento è già stato preso in prestito dall'Utente, compare la scritta "Elemento già preso in prestito" 3B. Se l'elemento non è disponibile, compare la scritta "Elemento già preso in prestito" 3C. Se non c'è connessione al database, compare la scritta "Impossibile connettersi al database."

Table 7: Use Case Richiesta di un elemento in prestito

Use Case	8. Restituzione di un elemento preso in prestito
Level	User goal
Description	Utente restituisce uno o più elementi che ha preso in prestito
Actors	Utente
Pre-conditions	Disporre di un account Essere connessi al database Aver preso in prestito almeno un elemento
Post-conditions	Elemento viene reso nuovamente disponibile per il prestito o viene segnalato errore
Normal flow	1. Utente visualizza i dettagli di un elemento preso in prestito 2. Utente preme il tasto "Restituisci" 3. L'elemento viene restituito correttamente
Alternative flows	3A. Se l'elemento selezionato non è stato preso in prestito dall'Utente, compare la scritta "Elemento non preso in prestito." 3B. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database."

Table 8: Use Case Restituzione di un elemento preso in prestito

2.2 Use Case Template

Use Case	9. Visualizza elementi presi in prestito
Level	User Goal
Description	Visualizzazione degli elementi attualmente presi in prestito
Actors	Utente
Pre-conditions	1. Essere connessi al database 2. Avere un account 3. Aver effettuato il login
Normal flow	1. Utente clicca su apposito menu 2. Viene visualizzata una lista con gli elementi attualmente presi in prestito dall'utente
Alternative flows	1A. Se non c'è attualmente alcun prestito per l'Utente, compare la scritta "Nessun elemento preso in prestito." 1A. Se non c'è connessione al database, compare la scritta "Impossibile connettersi al database."

Table 9: Use Case Visualizza elementi presi in prestito

2.2.3 Amministratore

L'amministratore è quell'attore che si occupa di gestire il sistema e il suo database. Tra i suoi compiti troviamo quelli di aggiungere, rimuovere e modificare i vari elementi e anche quello di aggiungere o rimuovere account amministratori.

Qui saranno presentati tutti gli **Use Case** che prevedono l'amministratore come unico attore in gioco:

Use Case	11. Rimuovere elemento
Level	User goal
Description	Amministratore elimina un elemento dal database
Actors	Amministratore
Pre-conditions	Avere effettuato il login Essere connessi al database Il database deve contenere almeno un elemento
Post-conditions	Elemento viene eliminato dal database o viene segnalato un errore
Normal flow	1. Amministratore visualizza i dettagli dell'elemento che intende eliminare 2. Amministratore preme il tasto "Rimuovi" 3. La rimozione dell'elemento avviene con successo
Alternative flows	3A. Se l'elemento è al momento in prestito, compare la scritta "Impossibile rimuovere un elemento ancora in prestito." 3B. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database."

Table 10: Use Case Rimuovere elemento

2.2 Use Case Template

Use Case	10. Aggiungere elemento al database
Level	User goal
Description	Amministratore aggiunge un elemento nel database
Actors	Amministratore
Pre-conditions	Avere effettuato il login Essere connessi al database
Post-conditions	Elemento viene aggiunto nel database correttamente o viene segnalato un errore
Normal flow	<ol style="list-style-type: none"> 1. Amministratore clicca sul bottone "Aggiungi elemento" 2. Amministratore compila il form con le informazioni necessarie 3. Amministratore clicca su "Save" 4. L'elemento viene aggiunto al database
Alternative flows	<p>3A. Se l'Amministratore clicca su "Cancel", viene annullata l'operazione</p> <p>4A. Se i campi del form non sono stati compilati correttamente, compare la scritta "Informazioni insufficienti o non valide."</p> <p>4B. Se l'elemento che deve essere inserito è un libro ed esiste già nel catalogo un libro con lo stesso codice ISBN, compare la scritta "ISBN già presente nel database."</p> <p>4C. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database."</p>

Table 11: Use Case Aggiungere elemento al database

Use Case	13. Aggiunta di un genere
Level	User Goal
Description	L'amministratore della biblioteca può aggiungere uno o più generi al database.
Actors	Amministratore
Pre conditions	Il sistema è connesso al database. Il amministratore ha effettuato l'accesso.
Normal Flows	<ol style="list-style-type: none"> 1.L'amministratore seleziona l'opzione per aggiungere un nuovo genere. 2. L'amministratore inserisce i nomi dei generi. 3. I generi vengono salvati correttamente nel database, o viene segnalato errore.
Alternative Flows	<p>3A.Se un genere con quel nome è già presente, non viene aggiunto e compare la scritta di errore "Genere già presente nel database".</p> <p>3B. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database".</p>

Table 12: Caso d'Uso 13: Aggiunta di un genere

2.2 Use Case Template

Use Case	12. Modifica un elemento nel database
Level	User goal
Description	Amministratore modifica un elemento selezionato
Actors	Amministratore
Pre-conditions	Avere effettuato il login Essere connessi al database Il database deve contenere almeno un elemento
Post-conditions	Amministratore modifica correttamente l'elemento o viene segnalato un errore
Normal flow	<ol style="list-style-type: none"> 1. Amministratore visualizza i dettagli dell'elemento che intende modificare 2. Amministratore clicca su "Modifica" 3. Pagina della modifica viene caricata 4. L'Amministratore effettua le modifiche necessarie 5. Viene premuto il tasto "Save" 6. Le modifiche vengono aggiunte al database
Alternative flows	<p>5A. Se l'Amministratore clicca su "Cancel", viene annullata l'operazione</p> <p>6A. Se i campi del form non sono stati compilati correttamente, compare la scritta "Informazioni insufficienti o non valide."</p> <p>6B. Se viene modificato l'ISBN di un libro ed esiste già nel catalogo un libro con lo stesso codice ISBN, compare la scritta "ISBN già presente nel database."</p> <p>6C. Se non c'è connessione al database, compare scritta "Impossibile connettersi al database."</p>

Table 13: Use Case Modifica un elemento nel database

2.3 Diagramma delle classi

2.3 Diagramma delle classi

Il diagramma delle classi è il seguente:

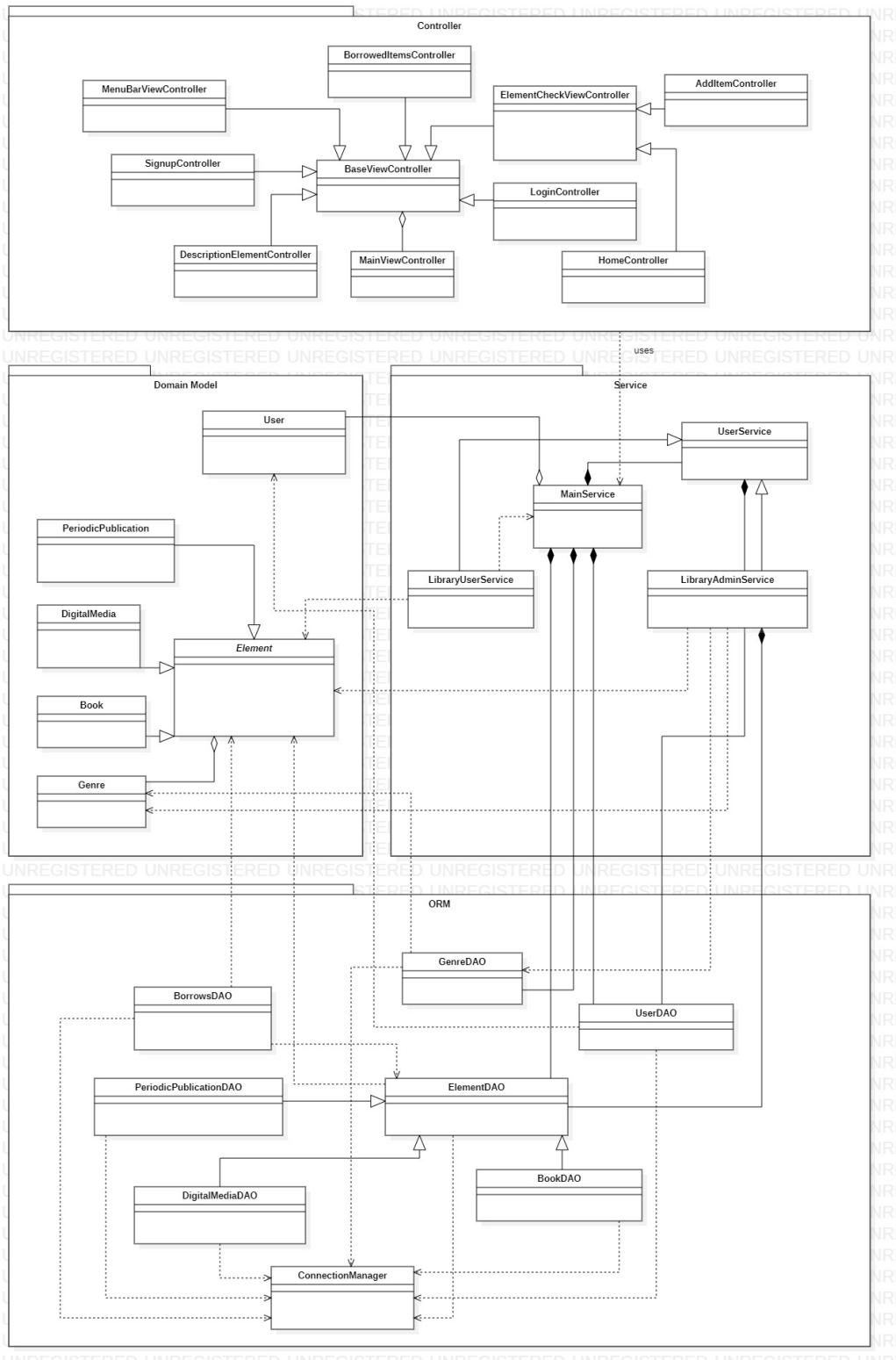


Figure 2: Diagramma delle classi

2.3 Diagramma delle classi

Entrando più nel dettaglio i diagrammi UML delle singole componenti sono:

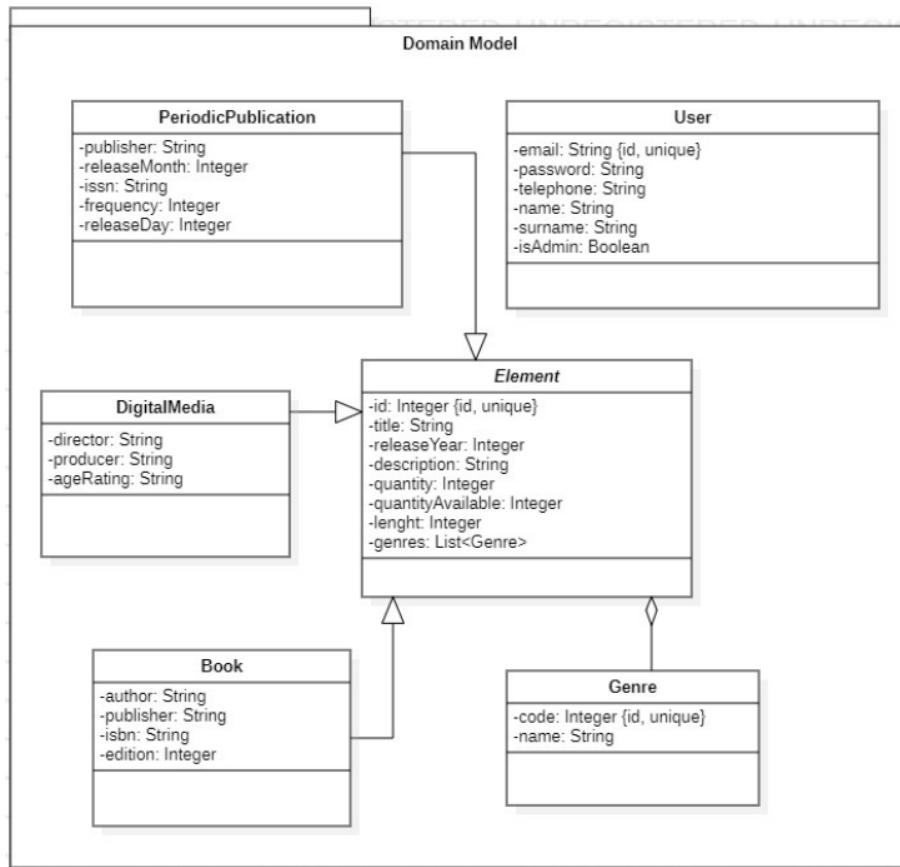


Figure 3: Nel diagramma UML del Domain Model possiamo trovare le classi: User, Element, Genre e classi come PeriodicPublication, DigitalMedia e Book che estendono Element

2.3 Diagramma delle classi

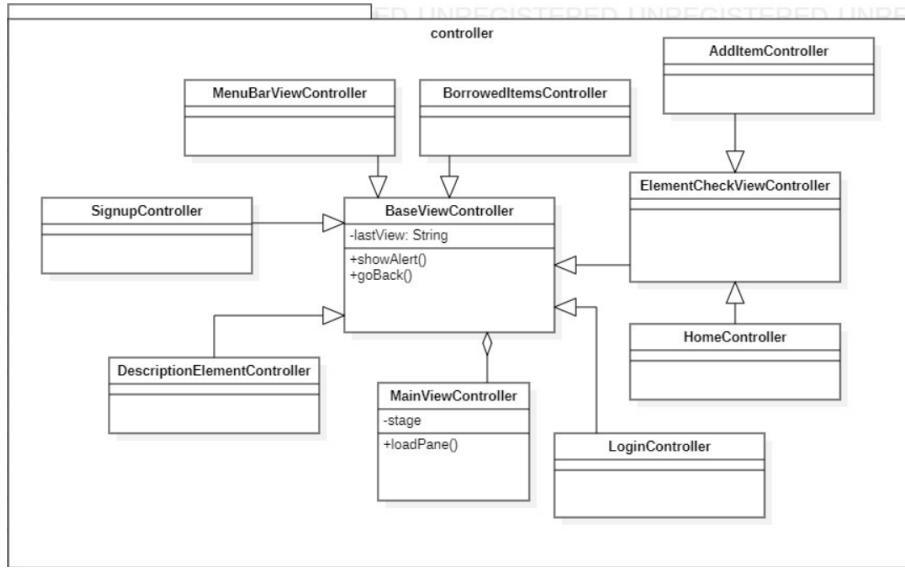


Figure 4: Nel diagramma UML del controller invece possiamo trovare le classi: **MenuBarViewController**, **BorrowedItemsController**, **SignupController**, **DescriptionElementController**, **LoginController**, **ElementCheckViewController** che ereditano tutte dalla classe **BaseViewController**. Poi ci possiamo trovare **BaseViewController**, **MainViewController** e **HomeController** e **AddItemController** che ereditano da **ElementCheckViewController**

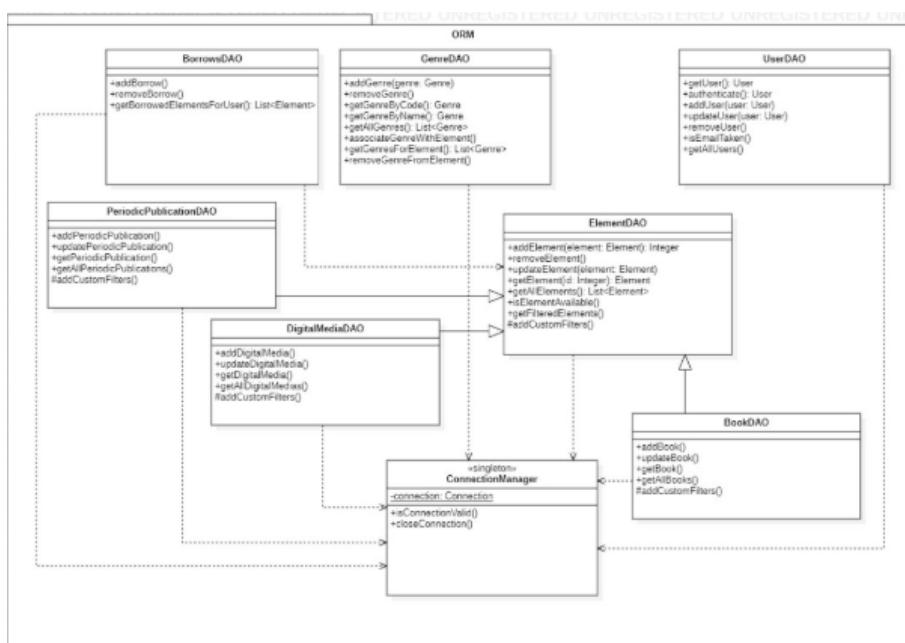


Figure 5: Nel Diagramma UML di ORM, possiamo vedere tutte le classi che si interfacciano con il database. Queste classi sono: **BorrowsDAO**, **PeriodicPublicationDAO**, **GenreDAO**, **UserDAO**, **ElementDAO**, **DigitalMediaDAO**, **ConnectionMenager** e **BookDAO**

2.3 Diagramma delle classi

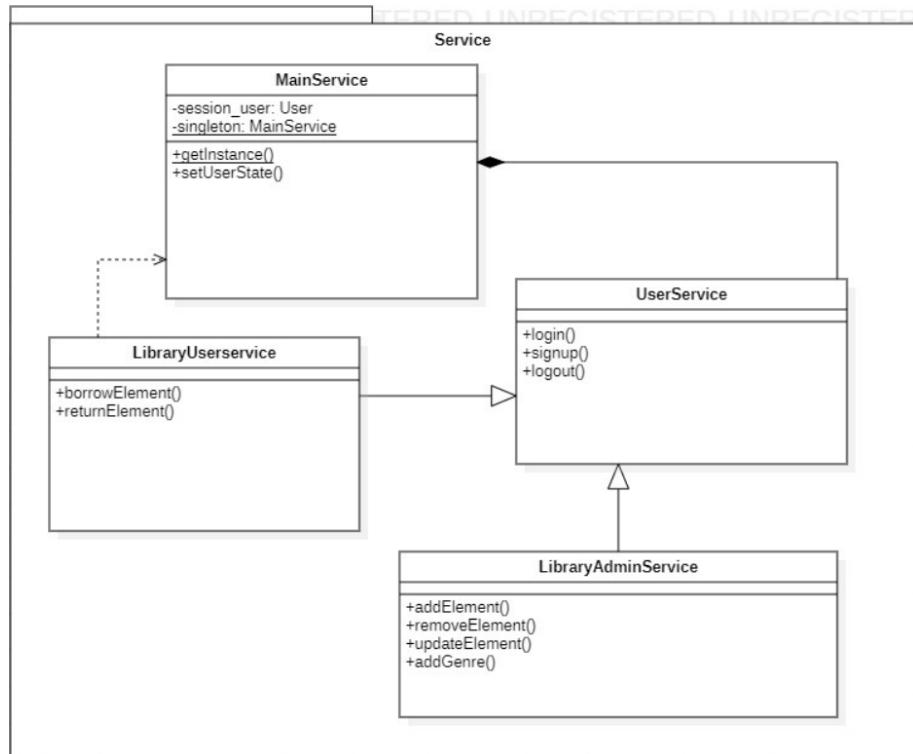


Figure 6: Nel diagramma UML di service, invece ci troviamo: MainService, UserService, LibraryUserService e LibraryAdminService. LibraryUserService e LibraryAdminService estendono UserService

2.4 Aspetti rilevanti del progetto

Nella progettazione di questo applicativo sono stati utilizzati alcuni **Design Pattern**.

2.4.1 Model View Controller

Il Model View Controller è un pattern in grado di separare la logica di presentazione dei dati dalla logica di business.

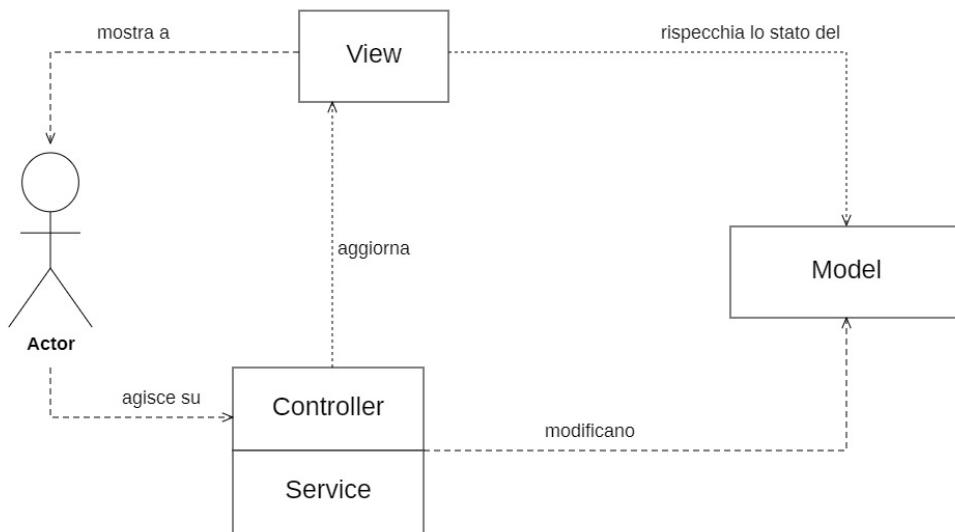


Figure 7: Diagramma UML del pattern MVC adattato al progetto

Nell'applicativo la parte del Model è rappresentata dall'insieme dei package **DomainModel** e **DAO**, con l'aggiunta del database stesso.

Il Controller è composto dai package **Service** e **Controller**.

Infine la View è costituita dall'insieme dei vari file in formato fxml.

2.4.2 Singleton

Nella gestione della connessione al database e nel mantenimento dello stato del programma è stato implementato il pattern Singleton. Questo garantisce l'esistenza di un unico **ConnectionManager** responsabile di gestire la connessione al database, e di un solo **MainController**, responsabile invece del mantenimento dello stato del sistema.

2.4.3 DAO

Il **DAO**, o Data Access Object, è un pattern utilizzato per separare la logica di accesso ai dati da quella di business in un'applicazione. Permette di isolare le operazioni di lettura e scrittura dei dati su un database da parte dell'applicazione che gestisce la logica aziendale.

Il pattern offre un'interfaccia che definisce le operazioni di base come creare, leggere, aggiornare ed eliminare, ovvero le operazioni **CRUD** sui dati.

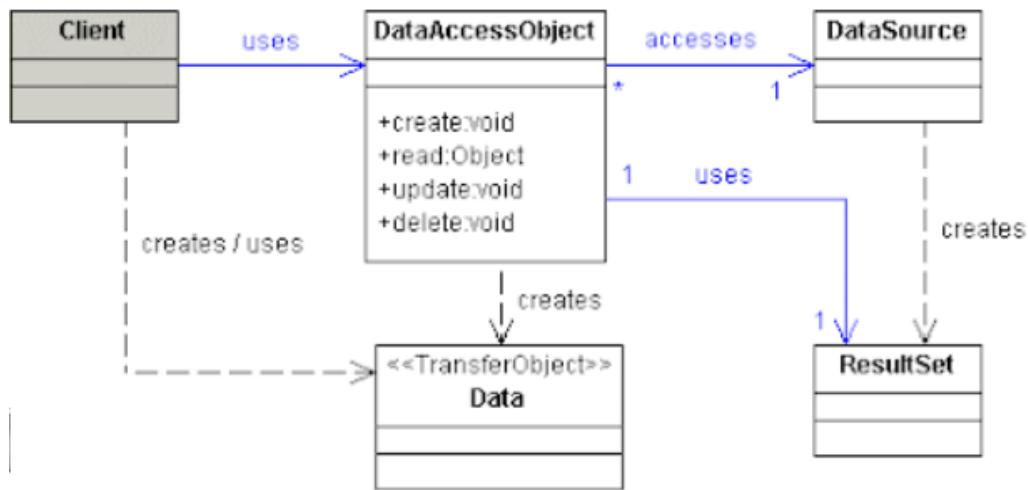


Figure 8: DAO

2.5 Page Navigation Diagram

L'interazione prevista che l'utente deve avere con l'interfaccia è la seguente:

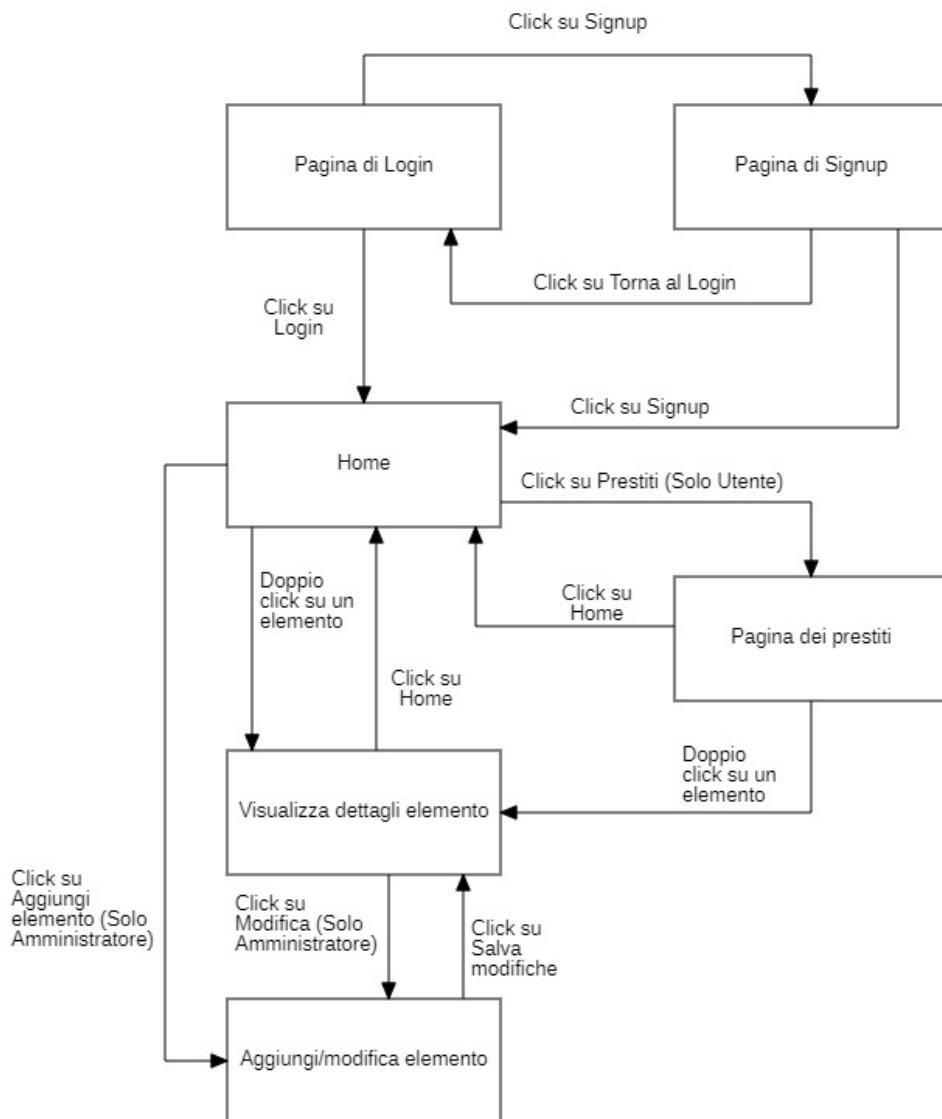


Figure 9: Diagramma di Navigazione

2.6 Entity Relationship Diagram

Qui di seguito è presentato il diagramma ER:

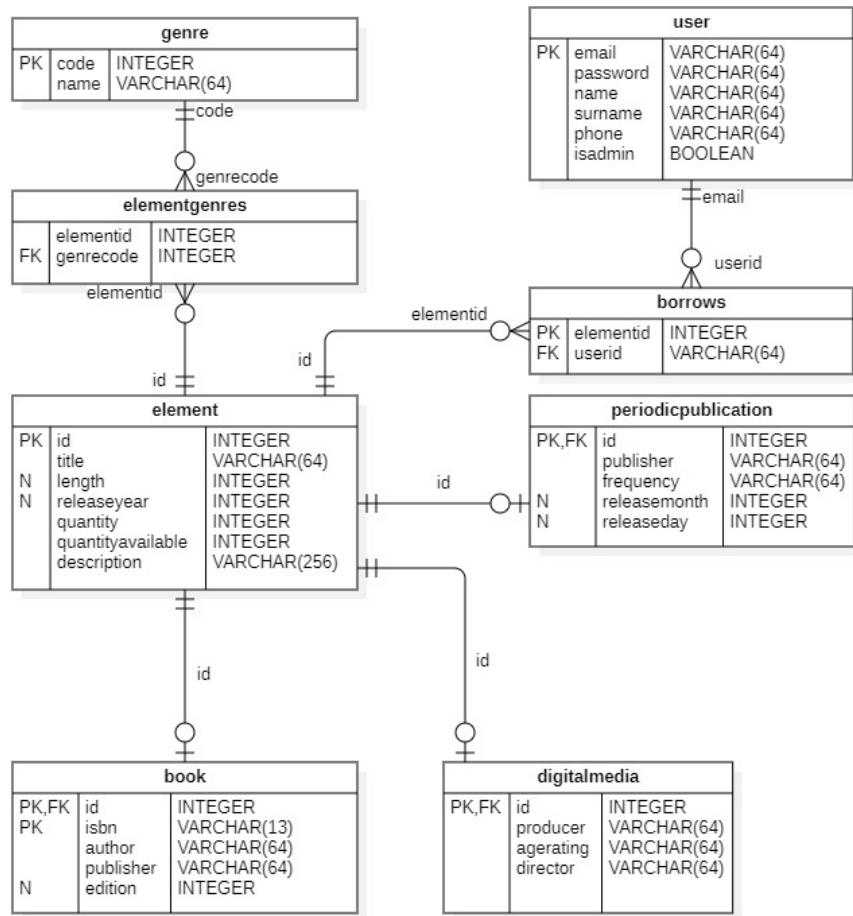


Figure 10: Diagramma ER

3 Implementazione delle classi

Il codice sorgente è articolato nel modo seguente:

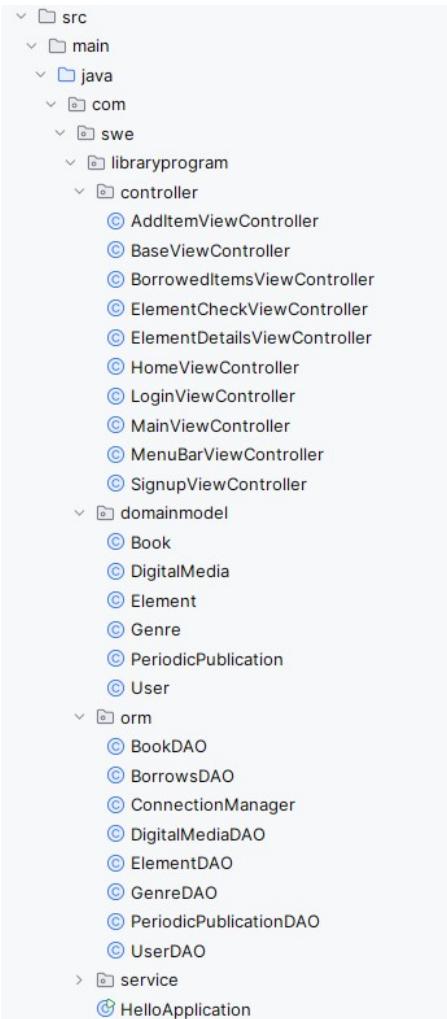


Figure 11: Stuttura del codice sorgente

3.1 DomainModel

Il DomainModel è il package che si occupa di definire un modello di composizione di classi su cui è possibile eseguire i casi d'uso espressi nello **Use Case Diagram**. Questo package si articola nelle seguenti classi, che descrivono le **entità del dominio** dell'applicazione:

3.1.1 Element

Questa classe si occupa di fornire attributi e metodi comuni tra i vari elementi presenti all'interno del sistema. Gli elementi condivisi tra le varie classi sono:

1. id
2. title
3. releaseYear
4. description
5. quantity
6. quantityAvailable
7. lenght
8. genres

3.1.2 Book

Questa classe mantiene i dati riguardanti i **libri** presenti all'interno del catalogo della biblioteca.

Tra questi dati, oltre a quelli ereditati dalla classe **Element** abbiamo:

- isbn
- author
- publisher
- edition

3.1.3 DigitalMedia

Questa classe si occupa di mantenere le informazioni relative ai **media digitali** presenti all'interno della biblioteca.

Tra questi dati, oltre a quelli ereditati dalla classe **Element** abbiamo:

- producer
- ageRating
- director

3.1.4 Periodic Publication

Questa classe è stata progettata per mantenere le informazioni relative alle pubblicazioni di riviste o altri elementi periodici. Oltre alle informazioni ereditate dalla classe **Element**, al suo interno troviamo:

- publisher
- frequency
- releaseMonth
- releaseDay

3.1.5 Genre

Questa classe viene utilizzata per mantenere informazioni sui vari generi che caratterizzano un elemento.

Al suo interno troviamo gli attributi **name** e **code**

3.1.6 User

Questa classe si occupa di mantenere le informazioni relative agli utenti all'interno del sistema. Tra queste informazioni è presente un attributo **Boolean** utilizzato per identificare se un utente è **Amministratore**, o se non lo è.

Al suo interno troviamo:

- email
- password
- name
- surname
- phone
- isAdmin

L'**email** è utilizzata per effettuare le operazioni di login e signup, oltre ad essere l'**identificativo** di un user.

3.2 Controller

In questo **package** si trovano tutte le classi che si occupano di fornire i servizi richiesti da un utente tramite la **GUI**.

3.2.1 AddItemViewController

Questa classe è il **controller** di una vista che si occupa di gestire l'interfaccia grafica per aggiungere un **item**, che può essere un libro, una rivista periodica o un media digitale all'interno del sistema della libreria. Questa classe estende la classe **ElementCheckViewController** e permette di fare questa operazione in maniera corretta tramite controlli di validità e permettendo di selezionare anche il tipo di item che stiamo inserendo, in modo tale da richiedere per ognuno di questi ultimi, solo le informazioni che lo riguardano.

3.2.2 BaseViewController

Questa classe è un controller che viene utilizzato per offrire metodi comuni a tutti i controller e permette inoltre di tornare alla vista precedente tramite attributo **lastView**. Molti controller estendono questa classe e fanno l'overloading dei suoi metodi.

```
package com.swe.libraryprogram.controller;

import ...

public class BaseViewController {

    MainViewController mainViewController;  22 usages
    String lastView = null;  2 usages

    @FXML
    protected void initialize() {
    }

    public void setMainViewController(MainViewController mainViewController) {
        this.mainViewController = mainViewController;
    }

    public void showAlert(String title, String message) {
        Alert alert = new Alert(Alert.AlertType.NONE);
        alert.setTitle(title);
        alert.setContentText(message);
        ButtonType okButton = new ButtonType("OK");
        alert.getButtonTypes().add(okButton);
        alert.showAndWait();
    }

    public void setLastView(String lastView) { this.lastView = lastView; }

    public void goBack() { mainViewController.loadBottomPane(lastView); }
}
```

Figure 12: Classe BaseViewController

3.2.3 BorrowedItemsViewController

Questa classe è un controller che gestisce la visualizzazione degli elementi presi in prestito da un utente della biblioteca.

Questa classe estende **BaseViewController** e interagisce con **MainService** e **LibraryUserService** per recuperare i dati degli elementi presi in prestito.

```
protected void initialize() {
    super.initialize();
    closeButton.setOnAction( ActionEvent event -> handleCloseButton());

    // Collega le colonne ai campi della classe Element
    titleColumn.setCellValueFactory(new PropertyValueFactory<>( s: "title"));
    borrowedElementsTable.setRowFactory( TableView<Element> tv -> {
        TableRow<Element> row = new TableRow<>();
        row.setOnMouseClicked( MouseEvent event -> {
            if (event.getClickCount() == 2 && !row.isEmpty()) {
                MainService.getInstance().setSelectedElementId(row.getItem().getId());
                mainViewController.loadBottomPane( fxmlView: "descriptionElement");
            }
        });
        return row;
    });
    loadBorrowedElementsData();
}

}
```

Figure 13: Metodo initialize() di BorrowedItemsController

3.2.4 ElementCheckViewController

Questo controller serve a gestire la **validazione** dell'inout nei campi di testo. Estende **BaseViewController** e contiene dei metodi per formattare correttamente il testo inserito dall'utente prima che esso venga accettato nel campo selezionato. Ognuna di queste operazioni viene fatta sulla base di alcuni filtri stabiliti durante la progettazione, atti ad accettare o meno il testo.

3.2.5 ElementDetailsViewController

Questa classe controller serve per la visualizzazione dei dettagli di uno degli elementi selezionato, presenti all'interno del catalogo della biblioteca.

3.2.6 HomeViewController

Questa classe controller gestisce la schermata principale dell'applicazione, mostrando il catalogo degli elementi, presenti all'interno del sistema, utilizzando una tabella. Inoltre questa classe fornisce anche la scelta di filtri, utili a ricercare un determinato elemento, o magari utili a cercare elementi simili a quelli già presi in prestito in passato.

3.2.7 LoginViewController

Questa classe controller gestisce la schermata di **login** dell'applicativo e permette quindi l'autenticazione e l'accesso, di un utente, al sistema. Estende il controller **BaseViewController**, per esempio per mostrare messaggi di errore che possono avvenire nell'autenticazione.

3.2.8 MenuBarController

Questa classe controller gestisce la barra dei menu dell'applicazione. A seconda del tipo di utente loggato, quindi utente normale o amministratore, questo controller mostra più o meno opzioni. Un altro suo compito è quello di gestire le azioni associate ai menu, come la navigazione tra le viste, la gestione dei generi e il logout.

3.2.9 SignupViewController

Questa classe controller si occupa di gestire la schermata di registrazione del sistema, permettendo la creazione di un nuovo utente, validandone gli input inseriti e facendolo accedere in automatico, nel caso in cui l'operazione andasse a buon fine.

3.2 Controller

```
protected void initialize() {
    super.initialize();
    homeMenuItem.setOnAction( ActionEvent event -> goToHome());
    addItemMenuItem.setOnAction( ActionEvent event -> goToAddItem());
    addGenresMenuItem.setOnAction( ActionEvent event -> {
        if (MainService.getInstance().getUser().isAdmin()) {

            Stage popupStage = new Stage();
            popupStage.initModality(Modality.WINDOW_MODAL);
            popupStage.initOwner(mainViewController.getStage());
            popupStage.setResizable(false);
            popupStage.setWidth(600);
            popupStage.setHeight(200);
            popupStage.setTitle("Aggiungi Generi");

            Label instructionLabel = new Label( s: "Inserisci nomi dei generi da inserire separati da \",\"");
            TextField inputField = new TextField();
            Button submitButton = new Button( s: "Aggiungi");

            submitButton.setOnAction( ActionEvent e -> {
                String inputText = inputField.getText();
                ArrayList<String> genresToAddList = new ArrayList<>();
                if (inputText != null && !inputText.trim().isEmpty()) {
                    genresToAddList = Arrays.stream(inputText.split( regex: ","))
                        .map(String::trim)
                        .map(String::toLowerCase)
                        .collect(Collectors.toCollection(ArrayList::new));
                }
                for (String newGenreName : genresToAddList) {
                    if (!((LibraryAdminService) MainService.getInstance().getUserService()).addGenre(newGenreName)) {
                        showAlert( title: "Errore", message: "Impossibile inserire il genere " + newGenreName + ". Forse è già presente?");
                    }
                }
                popupStage.close();
            });
        }
    });
}

HBox buttonBox = new HBox(submitButton);
buttonBox.setAlignment(Pos.BOTTOM_RIGHT);
buttonBox.setStyle("-fx-padding: 10; -fx-spacing: 10;");

VBox layout = new VBox( v: 10, instructionLabel, inputField, buttonBox);
layout.setStyle("-fx-padding: 20; -fx-alignment: top-center;");

Scene scene = new Scene(layout, v: 300, v1: 150);
popupStage.setScene(scene);
popupStage.showAndWait();

}
};
```

Figure 14: Metodo `initialize()` di MenuBarViewController

3.3 Orm

Questo package si occupa di gestire le comunicazioni con il database sia in lettura che in scrittura.

3.3.1 ConnectionManager

La classe si occupa della gestione della connessione al database PostgreSQL per l'applicazione e garantisce che essa sia **unica** attraverso il design pattern **singleton**.

La classe contiene: La classe fornisce quindi un'interfaccia sicura e centralizzata

Metodo	Descrizione
<code>getInstance()</code>	Restituisce l'unica istanza della classe, in quanto Singleton.
<code>getConnection()</code>	Restituisce una connessione attiva al database, creandola se necessario.
<code>setDbUser(String dbUser)</code>	Imposta dinamicamente il nome utente del database.
<code>setDbPass(String dbPass)</code>	Imposta dinamicamente la password del database.
<code>isConnectionValid()</code>	Verifica se la connessione al database è ancora valida.
<code>closeConnection()</code>	Chiude la connessione al database se è attiva.

Table 14: Metodi principali della classe `ConnectionManager`

per la gestione della connessione al database, seguendo il pattern Singleton e implementando metodi per configurare, verificare e chiudere la connessione in modo efficiente.

3.3.2 ElementDAO

Questa classe si occupa di fornire tutti i metodi necessari per la gestione di un elemento all'interno del sistema. Tali metodi si occupano, oltre che degli elementi stessi, anche dei dati complessi associati all'elemento su cui si opera, come ad esempio l'associazione di un genere ad un nuovo elemento.

Da essa ereditano le classi:

- BookManager
- DigitalMediaManager
- PeriodicPublicationManager

3.3.3 BookDAO

La classe BookManager gestisce le operazioni relative ai libri nel database, estendendo **ElementManager**. Si occupa di aggiungere, aggiornare, recuperare ed effettuare ricerche sui libri.

3.3.4 BorrowsDAO

La classe `BorrowsManager` gestisce i prestiti di elementi nella biblioteca, interagendo con la tabella `borrows` del database dalla quale possiamo ricavare informazioni che collegano i vari utenti agli elementi che questi hanno preso in prestito.

3.3.5 DigitalMediaDAO

Questa classe si occupa della gestione di oggetti di tipo **DigitalMedia**. Essa estende la classe `ElementManager` e interagisce con il database tramite query SQL per eseguire operazioni di inserimento, aggiornamento, recupero e filtraggio di media digitali.

3.3.6 GenreDAO

Questa classe fornisce metodi per aggiungere, rimuovere e recuperare i generi e per associare generi agli elementi nel sistema. Utilizza il `ConnectionManager` per ottenere le connessioni al database e le query SQL per manipolare i dati.

Tra i suoi metodi notiamo, oltre ai metodi di aggiunta e rimozione dal database:

Metodo	Descrizione
<code>getAllGenres()</code>	Recupera tutti i generi dal database. La query <code>SELECT</code> eseguita senza filtri restituirà tutti i generi presenti nella tabella <code>genres</code> .
<code>associateGenreWithElement(Integer elementId, Integer genreCode)</code>	Associa un genere a un elemento. La query <code>INSERT</code> inserisce una riga nella tabella di associazione <code>elementgenres</code> utilizzando l'ID dell'elemento e il codice del genere. Restituisce <code>true</code> se l'associazione ha successo. Viene invocato dai metodi <code>addElement</code> e <code>updateElement</code> di <code>ElementDAO</code> .
<code>getGenresForElement(Integer elementId)</code>	Recupera tutti i generi associati a un elemento. La query <code>SELECT</code> unisce le tabelle <code>genres</code> ed <code>elementgenres</code> per ottenere i generi per l'elemento specificato.
<code>removeGenreFromElement(Integer elementId, Integer genreCode)</code>	Rimuove l'associazione tra un elemento e un genere. La query <code>DELETE</code> elimina la riga corrispondente nella tabella <code>elementgenres</code> , che collega l'elemento al genere. Restituisce <code>true</code> se l'operazione ha successo. Viene invocato dai metodi <code>updateElement</code> e <code>removeElement</code> di <code>ElementDAO</code> .

Table 15: Metodi della classe `GenreManager` e descrizione

3.3.7 PeriodicPublicationDAO

La classe `PeriodicPublicationManager` è una sottoclasse di `ElementManager` ed è responsabile della gestione delle pubblicazioni periodiche.

3.3.8 UserDAO

Questa classe si occupa della gestione degli utenti all'interno del database e offre la possibilità di: creare, aggiungere, rimuovere e aggiornare gli utenti.

Inoltre, questa classe si occupa anche di controllare l'unicità degli attributi utili all'identificazione dell'utente, insieme a un controllo che permetta di vedere se l'attributo inserito esiste o meno all'interno del sistema.

3.4 Service

In un progetto i **service** permettono di separare la **logica di business** e la gestione dei dati dai **controller** che si occupano dell'interfaccia grafica. Sono classi riutilizzabili, ovvero possono essere usate da più controller senza duplicare il codice.

Comunicano con database e altre risorse.

3.4.1 LibraryAdminService

Questa classe service estende **UserService** e si occupa di gestire le operazioni amministrative dell'applicativo, come ad esempio l'aggiunta, la rimozione e l'aggiornamento di elementi all'interno della biblioteca.

Utilizza i **DAO** per interagire con il database e controlla prima la correttezza dei dati prima di eseguire operazioni, gestendo anche errori ed eccezioni.

3.4.2 LibraryUserService

Questo service estende **UserService** e si occupa di gestire le operazioni che possono essere effettuate da un utente normale della biblioteca, come la richiesta di prestito e la restituzione.

3.4.3 MainService

Questo service è alla base dell'applicazione. In questa classe viene implementato il design pattern **Singleton** e comunicare con il DAO.

Il suo compito è quello di:

1. Gestire l'accesso ai dati
2. Tenere traccia dello stato dell'utente
3. Fornisce i servizi per il resto dell'applicativo

3.4.4 UserService

Questo è il service principale per la gestione degli utenti.

Si occupa di gestire l'autenticazione, la registrazione e il logout al sistema. Inoltre permette di cercare, filtrare e visualizzare gli elementi nel sistema.

3.5 View

Nel progetto sono presenti varie viste, realizzate utilizzando JavaFX che, grazie all'applicazione SceneBuilder, consente di realizzare interfacce in modo rapido e semplice. Di seguito verranno mostrate e brevemente spiegate alcune viste:

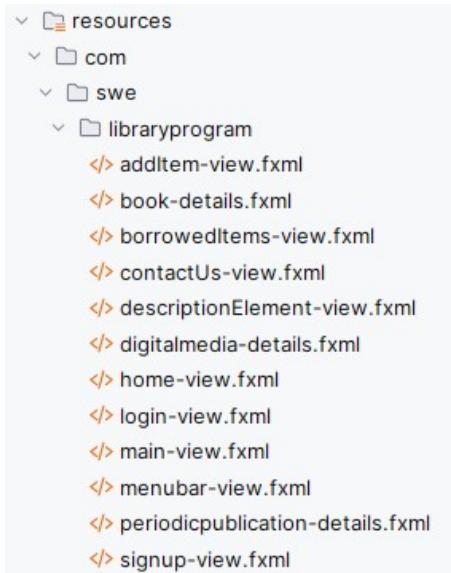


Figure 15: Viste dell'applicativo

3.5 View

3.5.1 SignUp-view

The screenshot shows a window titled "Library Management System" with a sub-titile "Informazioni". The form contains fields for user registration:

- E-mail * (highlighted with a blue border)
- Conferma E-mail *
- Password *
- Conferma password *
- Nome *
- Cognome *
- Telefono

Below the fields, there is a note: "La password deve:" followed by a list of requirements:

- essere lunga tra gli 8 e i 20 caratteri
- contenere almeno una lettera minuscola
- contenere almeno una lettera maiuscola
- contenere almeno un numero
- contenere almeno un carattere speciale

At the bottom right are two buttons: "Back to Login" and "SignUp".

Figure 16: Screenshot della pagina di Signup

3.5.2 AddItem-view

The screenshot shows a window titled "Library Management System" with a sub-titile "Navigazione Aiuto". The form is for adding or modifying item details:

Tipologia*: Libro	Autore:
Titolo*:	ISBN*:
Quantità: Default: 1	Edizione:
Disponibili: Lasciare vuoto se uguale a Quantità	Età minima: T, 6+, 10+, etc.
Casa editrice:	Producer:
Lunghezza:	Cadenzza: Giornaliera, Bimestrale...
Anno: 2025	Giorno: ▾
Generi:	Mese: ▾
Descrizione:	

At the bottom are two buttons: "Save" (highlighted with an orange border) and "Cancel".

Figure 17: Screenshot della pagina usata per aggiungere nuovi elementi al catalogo, visibile solo agli Amministratori. Questa pagina viene anche utilizzata per la modifica degli elementi

3.5 View

3.5.3 ElementDetails-view

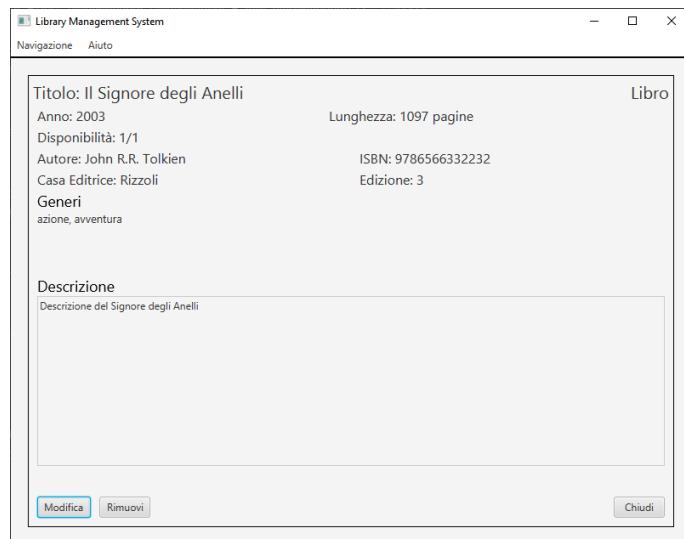


Figure 18: Screenshot della pagina che mostra i dettagli dell'elemento selezionato

3.5.4 Login-view

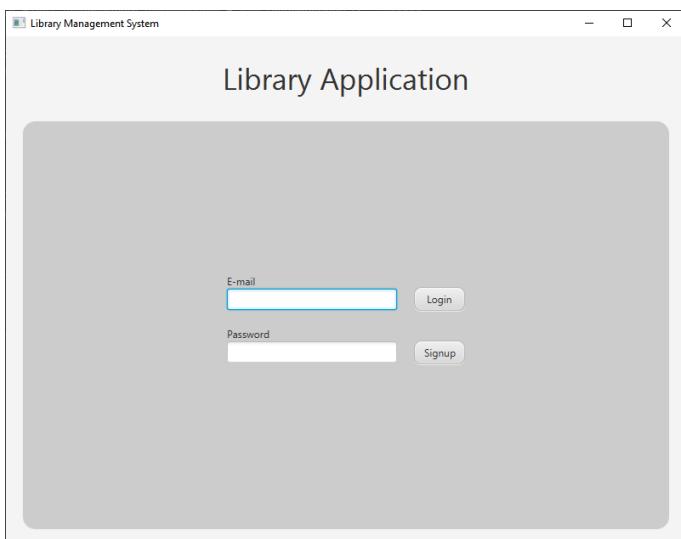


Figure 19: Screenshot della pagina di Login

3.5 View

3.5.5 Home-view

The screenshot shows the 'Library Management System' home page. At the top, there is a navigation bar with 'Library Management System', 'Navigazione', and 'Aiuto'. Below the navigation bar is a table with columns: Titolo, Anno, Disponibili, Generi, and Lunghezza. The table contains three rows of book information:

Titolo	Anno	Disponibili	Generi	Lunghezza
L' Era Glaciale 3: L' Alba dei Dinosauri	2012	1		93
Corriere della Sera	2025	3	azione	
Il Signore degli Anelli	2003	1	azione, av...	1097

To the right of the table are several search fields:

- Titolo:
- Generi:
- Anno: È disponibile
- Lunghezza: pagine/minuti

Figure 20: Screenshot della pagina Home

3.5.6 Borrowed-view

The screenshot shows the 'Library Management System' borrowed page. At the top, there is a navigation bar with 'Library Management System', 'Navigazione', and 'Aiuto'. Below the navigation bar is a table with a single column labeled 'Titolo'. The table contains three rows of book titles:

Titolo
Corriere della Sera
Il Signore degli Anelli
L' Era Glaciale 3: L' Alba dei Dinosauri

At the bottom right of the table is a 'Chiudi' button.

Figure 21: Screenshot della pagina dei Prestiti, visibile solo ad un Utente

3.6 HelloApplication

Questa è la classe Main del sistema. Si tratta di una classe derivata di **Application**, il cui scopo è quello di gestire la configurazione della finestra e il caricamento della vista iniziale, oltre che a inizializzare i vari singleton.

La funzione main si occupa di avviare il programma.

3.7 Struttura del database

Per la gestione dei dati sia degli utenti, degli elementi, delle prese in prestito e delle restituzioni si utilizza un database relazionale gestito attraverso **PostgreSQL**. In figura si mostra la struttura del database.

```
CREATE TABLE users (
    email VARCHAR PRIMARY KEY,
    password VARCHAR,
    name VARCHAR,
    surname VARCHAR,
    phone VARCHAR,
    isadmin BOOLEAN
);

CREATE TABLE elements (
    id SERIAL PRIMARY KEY,
    title VARCHAR,
    releaseyear INT,
    description TEXT,
    quantity INT,
    quantityavailable INT,
    length INT
);

CREATE TABLE books (
    id INT PRIMARY KEY,
    isbn VARCHAR,
    author VARCHAR,
    publisher VARCHAR,
    edition VARCHAR,
    FOREIGN KEY (id) REFERENCES elements(id)
);
```

```
CREATE TABLE digitalmedias (
    id INT PRIMARY KEY,
    producer VARCHAR,
    agerating VARCHAR,
    director VARCHAR,
    FOREIGN KEY (id) REFERENCES elements(id)
);

CREATE TABLE periodicpublication (
    id INT PRIMARY KEY,
    publisher VARCHAR,
    frequency VARCHAR,
    releasemonth INT,
    releaseday INT,
    FOREIGN KEY (id) REFERENCES elements(id)
);

CREATE TABLE borrows (
    elementid INT,
    userid VARCHAR,
    PRIMARY KEY (elementid, userid),
    FOREIGN KEY (elementid) REFERENCES elements(id),
    FOREIGN KEY (userid) REFERENCES users(email)
);

CREATE TABLE genres (
    code VARCHAR PRIMARY KEY,
    name VARCHAR
);

CREATE TABLE elementgenres (
    elementid INT,
    genrecodes VARCHAR,
    PRIMARY KEY (elementid, genrecodes),
    FOREIGN KEY (elementid) REFERENCES elements(id),
    FOREIGN KEY (genrecodes) REFERENCES genres(code)
);
```

Figure 22: Struttura del database

4 Testing

In questa sezione verranno elencati i test che sono stati effettuati. Per fare i test sono stati utilizzati **Maven**, **JUnit5** e **Mockito-JUnit**.

I test sono di tipo strutturale e sono atti a verificare la corretta interazione con il database e sul suo utilizzo.

La struttura dei test è la seguente:

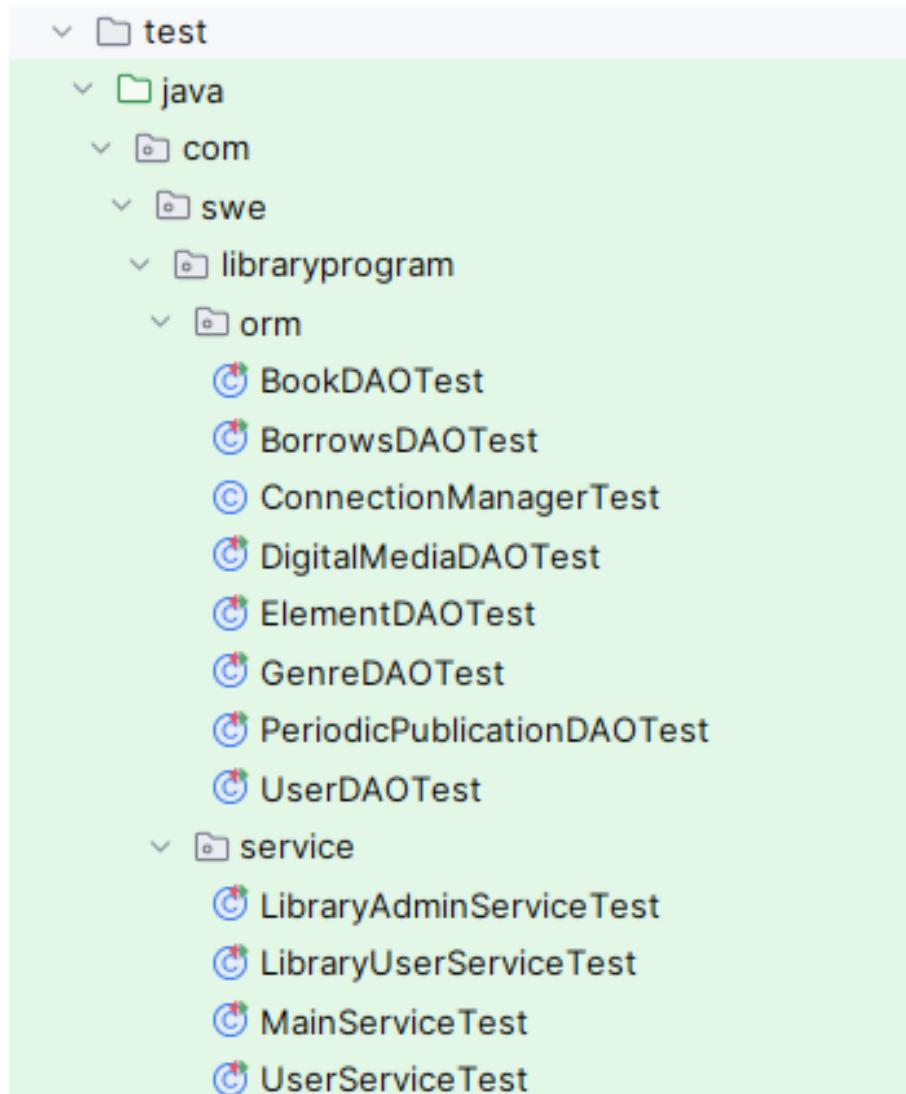


Figure 23: Struttura dei test

4.1 BookDAOTest

La classe **BookDAOTest** è una suite di test che verifica il corretto funzionamento della classe BookDAO, che si occupa della gestione dei libri nel database. I test utilizzano Mocking con Mockito per simulare il comportamento delle dipendenze, evitando di interagire direttamente con il database in alcuni casi.

Nome Test	Scopo	Aspettativa	Possibili Fallimenti
addBookTest	Verifica l'inserimento di un libro nel database	Deve restituire un ID valido dopo l'inserimento	Il libro non viene aggiunto e l'ID è null
updateBookTest	Verifica l'aggiornamento di un libro esistente	La funzione deve restituire <code>true</code> se l'aggiornamento ha successo	Il libro non viene aggiornato correttamente
getBookByIsbnTest	Verifica il recupero di un libro tramite ISBN	Il libro trovato deve avere lo stesso ISBN richiesto	Nessun libro trovato oppure ISBN errato

Table 16: Riepilogo dei test eseguiti su BookDAO.

Questa classe testa correttamente le operazioni CRUD sulla gestione dei libri, verificando che l'aggiunta, l'aggiornamento e il recupero funzionino come previsto. I test utilizzano mocking per isolare il database e altre classi, garantendo che i risultati siano affidabili senza dipendere da una connessione attiva al database.

4.2 BorrowsDAOTest

Questa classe di test **BorrowsDAOTest** verifica il corretto funzionamento della classe BorrowsDAO, che gestisce il prestito di elementi.

Nome Test	Scopo	Aspettativa	Possibili Fallimenti
addBorrowTest	Verifica l'aggiunta di un prestito	Deve restituire <code>true</code> dopo aver aggiunto il prestito	Il prestito non viene registrato e il metodo restituisce <code>false</code>
removeBorrowTest	Verifica la rimozione di un prestito	Deve restituire <code>true</code> dopo aver rimosso il prestito	Il prestito non viene rimosso dal database
getBorrowedElementsForUserTest	Verifica il recupero dei prestiti per un utente	Deve restituire un elenco di prestiti non null	Il metodo restituisce null, indicando un problema

Table 17: Tabella riassuntiva dei test su BorrowsDAO

I test effettuati sulla classe **BorrowsDAO** hanno verificato le funzionalità principali relative alla gestione dei prestiti. In particolare:

4.3 ConnectionManagerTest

- **Aggiunta di un prestito:** Il test `addBorrowTest` ha confermato che il metodo registra correttamente un nuovo prestito nel database, restituendo un valore positivo in caso di successo.
- **Rimozione di un prestito:** Il test `removeBorrowTest` ha dimostrato che è possibile eliminare un prestito esistente, garantendo che l'elemento venga effettivamente rimosso dal sistema.
- **Recupero dei prestiti di un utente:** Il test `getBorrowedElementsForUserTest` ha garantito che l'applicazione è in grado di restituire un elenco valido di elementi attualmente presi in prestito da un utente specifico.

I test hanno fornito risultati coerenti con le aspettative, garantendo che il sistema gestisca correttamente l'aggiunta, la rimozione e il recupero dei prestiti.

4.3 ConnectionManagerTest

Questa classe, `ConnectionManagerTest`, gestisce la connessione a un database PostgreSQL per i test.

- **Connessione al database:** Il metodo `getConnection()` viene chiamato per assicurarsi che la connessione al database per i test venga creata correttamente.
- **Verifica della validità:** `isValid()` garantisce che la connessione sia utilizzabile prima di procedere con altre operazioni.
- **Chiusura della connessione:** `closeConnection()` assicura che la connessione venga chiusa correttamente dopo l'uso.

Per questi test dobbiamo tenere di conto che:

- **Pattern Singleton:** Usato per mantenere un'unica connessione.
- **Gestione delle eccezioni:** Gli errori di connessione vengono catturati e stampati.
- **Metodi `setDbUser()` e `setDbPass()`:** Permettono di cambiare le credenziali.

4.4 DigitalMediaDAOTest

Questa classe testa le operazioni di gestione dei media digitali effettuati da **DigitalMediaDAO**, verificando che i metodi funzionino correttamente con il database.

I test condotti sulla classe `DigitalMediaDAO` confermano il corretto funzionamento delle operazioni principali sulla gestione dei media digitali. In particolare:

- Il metodo di inserimento di un media digitale, `addDigitalMediaTest`, ha sempre restituito un identificativo valido, dimostrando la corretta interazione con il database.
- Il metodo di aggiornamento `updateDigitalMediaTest` ha sempre restituito `true`, segnalando che le modifiche ai dati vengono applicate correttamente.

Test	Scopo	Criterio di Successo
<code>addDigitalMediaTest()</code>	Verifica che un media digitale possa essere inserito nel database.	Il metodo restituisce un ID valido, non <code>null</code> .
<code>updateDigitalMediaTest()</code>	Controlla che i dati di un media digitale possano essere aggiornati.	Il metodo restituisce <code>true</code> .
<code>getDigitalMediaTest()</code>	Testa il recupero di un media digitale dal database.	Il metodo restituisce un oggetto valido, non <code>null</code> .

Table 18: Riassunto dei test effettuati sulla classe DigitalMediaDAO

- Il metodo di recupero dei media digitali `getDigitalMediaTest` ha restituito oggetti validi, confermando l'integrità dei dati salvati.

L'uso di **Mockito** per il mocking del `ConnectionManager` e del `MainService` ha permesso di isolare il codice e testare le funzionalità senza dipendere da un database reale. Inoltre, grazie all'uso del `rollback` alla fine dei test, i dati temporanei non vengono mantenuti nel database.

4.5 ElementDAOTest

I test condotti nella classe **ElementDAOTest** si occupano di verificare il corretto funzionamento di operazioni CRUD sugli elementi all'interno del database. Per testare la rimozione di un elemento si opera nel seguente modo:

- **Preparazione:** Viene inserito un elemento nel database con una query SQL. L'ID generato per l'elemento viene salvato per essere utilizzato nel test.
- **Test:** Viene chiamato il metodo `removeElement(generatedId)` per rimuovere l'elemento dal database. Successivamente, viene effettuata l'asserzione `assertTrue(removeElement(generatedId))`, che verifica che la rimozione sia stata eseguita correttamente.
- **Verifica:** Se il test passa, significa che l'elemento è stato correttamente rimosso dal database.

Invece per quanto riguarda il corretto recupero di un elemento:

- **Preparazione:** Viene inserito un nuovo elemento nel database, e l'ID generato per l'elemento viene salvato per essere utilizzato nel test.
- **Test:** Viene chiamato il metodo `getElement(generatedId)` per recuperare l'elemento dal database. Successivamente, viene effettuata l'asserzione `assertEquals(generatedId, element.getId())`, che verifica che l'ID dell'elemento recuperato corrisponda all'ID generato precedentemente.
- **Verifica:** Se l'asserzione passa, significa che il metodo `getElement()` funziona correttamente e recupera gli elementi dal database in modo corretto.

I test effettuati hanno consentito di verificare il corretto funzionamento delle operazioni di rimozione e recupero sugli "elementi" nel database. In particolare, i test hanno confermato che:

- Il metodo `removeElement()` rimuove correttamente un elemento dal database.
- Il metodo `getElement()` recupera correttamente un elemento dal database utilizzando l'ID.

4.6 GenreDAOTest

I test nella classe `GenreDAOTest` sono progettati per verificare il corretto funzionamento dei metodi relativi alla gestione dei generi associati agli elementi nel sistema. I test coprono vari scenari, tra cui l'associazione di generi agli elementi, l'aggiunta e la rimozione di generi, nonché il recupero dei generi per un dato elemento.

Test di Setup e Teardown:

- `@BeforeAll` e `@AfterAll`: Questi metodi vengono eseguiti prima e dopo tutti i test nella classe.
- `setUp()`: Crea un elemento e un genere nel database, i cui ID e codici vengono salvati per essere utilizzati nei test successivi.
- `tearDown()`: Rimuove i dati inseriti nel database durante i test, cancellando le associazioni e i dati generati.

Nel dettaglio dei test:

Test: associateGenreWithElementTest()

- **Scopo:** Verificare che un genere possa essere correttamente associato a un elemento.
- **Preparazione:** L'ID dell'elemento `generatedElementId` e il codice del genere `generatedGenreCode` sono già presenti nel database.
- **Esecuzione:** Viene chiamato il metodo `associateGenreWithElement(generatedElementId, generatedGenreCode)`.
- **Verifica:** L'asserzione `assertTrue(result)` verifica che l'associazione sia riuscita.

Test: addGenreTest()

- **Scopo:** Verificare che un nuovo genere possa essere aggiunto al database.
- **Preparazione:** Non è necessaria alcuna preparazione in quanto il test si concentra sull'aggiunta di un nuovo genere.
- **Esecuzione:** Viene chiamato il metodo `addGenre(new Genre("Insert Test"))`.
- **Verifica:** L'asserzione `assertTrue(result)` verifica che il genere sia stato aggiunto correttamente.

Test: getAllGenresTest()

- **Scopo:** Verificare che tutti i generi possano essere recuperati correttamente.
- **Preparazione:** I generi esistenti sono già nel database, incluso quello inserito nel test precedente.
- **Esecuzione:** Viene chiamato il metodo `getAllGenres()` per recuperare tutti i generi.
- **Verifica:** L'asserzione `assertNotNull(result)` verifica che la lista di generi non sia vuota.

Test: getGenresForElementTest()

- **Scopo:** Verificare che i generi associati a un elemento possano essere recuperati correttamente.
- **Preparazione:** Un'associazione tra l'elemento `generatedElementId` e il genere `generatedGenreCode` viene inserita nella tabella `elementgenres`.
- **Esecuzione:** Viene chiamato il metodo `getGenresForElement(generatedElementId)`.
- **Verifica:** L'asserzione `assertNotNull(result)` verifica che la lista di generi associati all'elemento non sia vuota.

Test: removeGenreFromElementTest()

- **Scopo:** Verificare che un genere possa essere rimosso da un elemento.
- **Preparazione:** Un'associazione tra l'elemento `generatedElementId` e il genere `generatedGenreCode` viene inserita nella tabella `elementgenres`.
- **Esecuzione:** Viene chiamato il metodo `removeGenreFromElement(generatedElementId, generatedGenreCode)`.
- **Verifica:** L'asserzione `assertTrue(result)` verifica che la rimozione sia riuscita.

4.7 PeriodicPublicationDAOTest

I test nella classe **PeriodicPublicationDAOTest** sono stati progettati con lo scopo di testare l'aggiunta di un elemento **PeriodicPublication** all'interno del database e di un suo eventuale aggiornamento.

4.8 UserDAOTest

I test nella classe **UserDAOTest** sono stati progettati per verificare il corretto funzionamento della classe **UserDAO**.

4.9 MainServiceTest

I test della classe **MainServiceTest** sono stati progettati per verificare il corretto funzionamento della classe **MainService**.

4.10 UserServiceTest

I test della classe **UserServiceTest** sono stati progettati per verificare il corretto funzionamento della classe **UserService**.

Questi test controllano la corretta implementazione degli Use Case 1, 4, 5 e 6, assicurando che vengano rispettati tutti i flussi alternativi.

4.11 LibraryUserServiceTest

I test della classe **LibraryUserServiceTest** sono stati progettati per verificare il corretto funzionamento della classe **LibraryUserService**.

Questi test controllano la corretta implementazione degli Use Case 7, 8 e 9, assicurando che vengano rispettati tutti i flussi alternativi.

4.12 LibraryAdminServiceTest

I test della classe **LibraryAdminServiceTest** sono stati progettati per verificare il corretto funzionamento della classe **LibraryAdminService**.

Questi test controllano la corretta implementazione degli Use Case 10, 11, 12 e 13, assicurando che vengano rispettati tutti i flussi alternativi.

4.13 Risultati Maven

I risultati ottenuti tramite i test effettuati con **Maven** sono i seguenti:

```
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.559 s
[INFO] Running com.swe.libraryprogram.orm.BorrowsDAOTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.726 s
[INFO] Running com.swe.libraryprogram.orm.DigitalMediaDAOTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.529 s
[INFO] Running com.swe.libraryprogram.orm.ElementDAOTest
Elemento con ID 2428 rimosso correttamente.

[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.677 s
[INFO] Running com.swe.libraryprogram.orm.GenreDAOTest
Generated Element ID: 2430
Cleanup successful: Deleted test data.

[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.579 s
[INFO] Running com.swe.libraryprogram.orm.PeriodicPublicationDAOTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.471 s
[INFO] Running com.swe.libraryprogram.orm.UserDAOTest
Utente inserito correttamente.

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.411 s
[INFO] Running com.swe.libraryprogram.service.LibraryAdminServiceTest
----- TST UC 10 -----
TST UC 10 FLOW 4A:
Testing per ciascun tipo di elemento...
Informazioni insufficienti o non valide.
Informazioni insufficienti o non valide.
Informazioni insufficienti o non valide.

TST UC 10 FLOW 4B:
ISBN già presente nel database
Informazioni insufficienti o non valide.

TST UC 10 FLOW 4C:
Impossibile connettersi al database
----- TST UC 12 -----
TST UC 12 FLOW 6A:
Testing per ciascun tipo di elemento...
Informazioni insufficienti o non valide.
Informazioni insufficienti o non valide.
Informazioni insufficienti o non valide.
```

```
TST UC 12 FLOW 6B:  
ISBN già presente nel database  
Informazioni insufficienti o non valide.  
TST UC 12 FLOW 6C:  
Impossibile connettersi al database  
----- TST UC 13 -----  
TST UC 13 FLOW 3A:  
Genere già presente nel database  
TST UC 13 FLOW 3B:  
Impossibile connettersi al database  
----- TST UC 11 -----  
TST UC 11 FLOW 3A:  
Impossibile rimuovere un elemento ancora in prestito  
TST UC 11 FLOW 3B:  
Impossibile connettersi al database  
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.369 s  
[INFO] Running com.swe.libraryprogram.service.LibraryUserServiceTest  
----- TST UC 9 -----  
TST UC 9 FLOW 2A:  
Nessun elemento preso in prestito.  
TST UC 9 FLOW 2B:  
Impossibile connettersi al database.  
----- TST UC 7 -----  
TST UC 7 FLOW 3A:  
Elemento già preso in prestito.  
TST UC 7 FLOW 3B:  
Elemento non disponibile per il prestito.  
TST UC 7 FLOW 3C:  
Impossibile connettersi al database.  
----- TST UC 8 -----  
TST UC 8 FLOW 3A:  
Elemento non preso in prestito.  
TST UC 8 FLOW 3B:  
Impossibile connettersi al database.
```

4.13 Risultati Maven

```
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.180 s
[INFO] Running com.swe.libraryprogram.service.MainServiceTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.021 s
[INFO] Running com.swe.libraryprogram.service.UserServiceTest
----- TST UC 1 -----
TST UC 1 FLOW 3A:
Identificativo non presente.
TST UC 1 FLOW 3B:
Password errata.
TST UC 1 FLOW 3C:
Impossibile connettersi con il database
----- TST UC 4 -----
TST UC 4 FLOW 3A:
E-mail non valida.
TST UC 4 FLOW 3B:
Password non valida.
TST UC 4 FLOW 3C:
Nome non inserito.
TST UC 4 FLOW 3D:
Cognome non inserito.
TST UC 4 FLOW 3E:
Numero di telefono non valido.
TST UC 4 FLOW 3F:
E-mail già in uso
TST UC 4 FLOW 3G:
Impossibile connettersi con il database
----- TST UC 5 -----
Testing di ricerca per ciascun campo...
----- TST UC 6 -----
TST UC 6 FLOW 3A:
Nessun elemento è stato recuperato.
TST UC 6 FLOW 3B:
Impossibile connettersi con il database
[INFO] Tests run: 16, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.203 s
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 61, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.903 s
```

Figure 24: Risultati dei test