

# Gestore prenotazione aule per Scuola di Musica di Fiesole

Alessandro Marinai, Eleonora Ristori

Marzo 2022

## Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Finalità del progetto . . . . .	3
1.2	Tecnologie e standard utilizzati . . . . .	3
<b>2</b>	<b>Progettazione</b>	<b>4</b>
2.1	Attori coinvolti e relativi casi d'uso . . . . .	4
2.1.1	User generico . . . . .	5
2.1.2	Studente . . . . .	6
2.1.3	Docente . . . . .	7
2.1.4	Admin . . . . .	8
2.2	Class diagram . . . . .	12
2.3	Design Patterns . . . . .	13
2.3.1	Model View Controller . . . . .	13
2.3.2	Singleton . . . . .	14
2.4	Page Navigation Diagram . . . . .	14
2.5	Entity Relationship Diagram . . . . .	15
<b>3</b>	<b>Implementazione delle classi</b>	<b>16</b>
3.1	Model . . . . .	16
3.1.1	User . . . . .	17
3.1.2	Instrument . . . . .	17
3.1.3	RoomGraph . . . . .	17
3.1.4	Tag . . . . .	17
3.1.5	Booking . . . . .	17
3.1.6	GraphManager . . . . .	17
3.1.7	UserManager . . . . .	17
3.2	Controller . . . . .	18
3.2.1	LoginController . . . . .	18
3.2.2	UserController, StudentController e TeacherController . .	18
3.3	DAO . . . . .	19
3.3.1	ConnectionManager . . . . .	20

3.3.2	ModelDAO . . . . .	20
3.3.3	UserDAO . . . . .	21
3.4	Altre classi . . . . .	21
3.4.1	JavaMailUtil . . . . .	21
3.4.2	InterfaceController . . . . .	22
3.5	Viste . . . . .	22
3.6	Struttura del database . . . . .	25
<b>4</b>	<b>Testing</b>	<b>26</b>
4.1	DAO . . . . .	26
4.1.1	ModelDAOTest . . . . .	27
4.1.2	UserDAOTest . . . . .	27
4.2	Model . . . . .	27
4.2.1	BookingTest . . . . .	27
4.2.2	GraphManagerTest . . . . .	27
4.2.3	RoomGraphTest . . . . .	28
4.2.4	UserManagerTest . . . . .	28
4.3	Controller . . . . .	28
4.3.1	LoginControllerTest . . . . .	28
4.3.2	StudentControllerTest . . . . .	29
4.3.3	TeacherControllerTest . . . . .	29
4.3.4	AdminControllerTest . . . . .	29
4.4	Esiti . . . . .	30

# 1 Introduzione

## 1.1 Finalità del progetto

L'applicativo ha come obiettivo quello di digitalizzare il sistema di prenotazione delle aule studio presso la Scuola di Musica di Fiesole. Ad oggi, infatti, tale gestione è attribuita ad un registro cartaceo giornaliero in cui gli studenti nel giorno stesso in cui necessitano di un'aula possono verificarne l'effettiva disponibilità ed effettuare la prenotazione. Questo rende il sistema particolarmente inefficiente costringendo gli studenti a recarsi presso i locali del conservatorio ed a consultare tale registro rischiando di non avere la possibilità di trovare un luogo in cui studiare e senza avere la possibilità di prenotarsi per i giorni successivi.

L'applicativo ha quindi come scopo quello di implementare il servizio di prenotazione delle aule consentendo agli utenti di poter fissare degli spazi adatti al loro studio anche in più date e alla scuola di tenere traccia di chi ha occupato le diverse aule. Il sistema distingue due categorie principali di utenti:

- Gli studenti tra cui si distinguono diverse tipologie quali allievi di pianoforte o percussioni che necessitano di aule attrezzate, oppure quelli con mobilità limitata causata dal peso dello strumento, come ad esempio arpa o contrabbasso.
- I docenti che devono poter prenotare aule per lezioni aggiuntive in vista di concerti od audizioni.

Si considera infine un ultimo attore, un amministratore, con il compito di definire le aule e gli slot disponibili per le prenotazioni escludendo quelli già impiegati per le lezioni e di provvedere alla eventuale cancellazione di prenotazioni in caso di problemi sopraggiunti nella scuola.

## 1.2 Tecnologie e standard utilizzati

L'applicativo è sviluppato in Java. Per garantire una permanenza in memoria è stato creato un database locale gestito tramite PostgreSQL. Il progetto resta comunque aperto all'estensione ad un database connesso alla rete.

La connessione al suddetto database sfrutta invece le librerie di JDBC (Java DataBase Connectivity).

L'interfaccia grafica è realizzata tramite le librerie open source di JavaFX. Durante la progettazione è stato utilizzato il tool SceneBuilder, fornito da JavaFX per permettere un design pratico e celere dell'interfaccia.

Per i diagrammi di classi viene utilizzato lo standard UML.

Il testing è realizzato sfruttando le librerie di JUnit.

## 2 Progettazione

### 2.1 Attori coinvolti e relativi casi d'uso

L'applicativo individua 3 diversi tipi di attori: l'admin, lo studente ed il docente, ciascuno caratterizzato da diverse funzionalità. Alcune funzioni tuttavia sono comuni ai diversi tipi di utenti. Tutti possono effettuare il log in da una interfaccia comune di accesso che provvede a verificare l'identità dell'utente, segnalando un errore se l'utente non è già registrato, e a reindirizzarlo sull'interfaccia dedicata all'espletazione degli altri casi d'uso. L'opzione di sign up, invece, è consentita solo ad utenti di tipo studente o docente, in quanto si prevede l'esistenza di un unico amministratore del sistema, già presente al rilascio del programma. Ogni nuovo utente deve quindi definire uno username ed una password oltre ad identificarsi come studente o docente ed indicare lo strumento suonato.

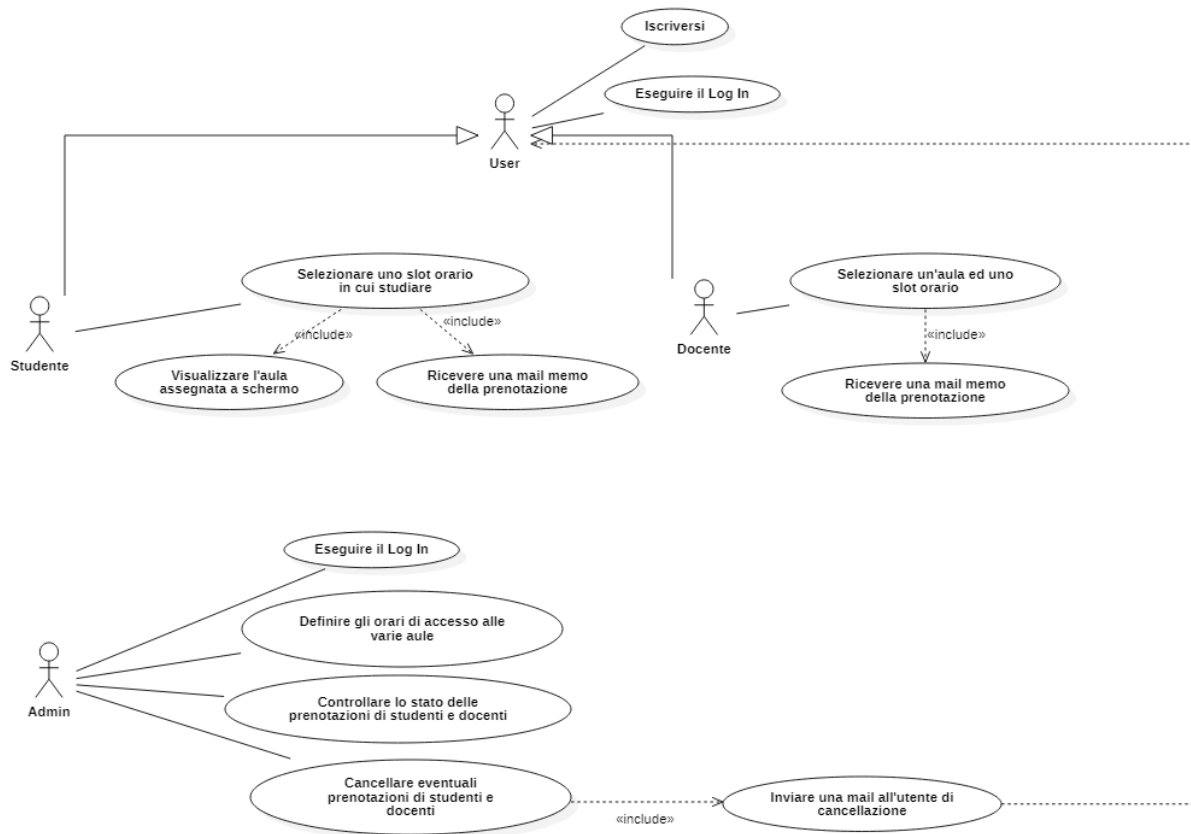


Figura 1: Casi d'uso relativi ai diversi attori dell'applicativo

### 2.1.1 User generico

Alcuni use case non coinvolgono un singolo attore ma più attori. Si tratta di use case di tipo High Summary dato che permettono la gestione dell'identificazione degli utenti e non consistono in un vero obiettivo per chi usa l'applicazione.

UC	Log in
Description	L'utente esegue il log in utilizzando le sue credenziali
Level	Function
Main actors	Studente, Docente e Admin
Main course	<ol style="list-style-type: none"><li>1. L'utente inserisce username e password</li><li>2. Preme il pulsante "Log in"</li><li>3. Il log in avviene correttamente e il sistema mostra la pagina successiva</li></ol>
Alternative course	<ol style="list-style-type: none"><li>3. A. Se lo username inserito non è presente nel sistema, il sistema mostra una scritta che indica che lo username inserito non esiste</li><li>4. A. L'utente premendo il pulsante "Sign up" può aprire la schermata per registrarsi</li><li>3. B. Se lo username esiste ma la password è errata, il sistema mostra una scritta che indica che la password inserita è sbagliata</li><li>3. C. Il sistema rileva un problema di connessione al Database e lo notifica all'utente</li></ol>

Figura 2: Use Case Log in (Interfaccia di riferimento Figura: 20)

UC	Sign up
Description	L'utente esegue il sign up inserendo le credenziali desiderate
Level	Function
Main actors	Studente, Docente
Main course	<ol style="list-style-type: none"> <li>1. L'utente inserisce username e password desiderati</li> <li>2. L'utente indica il ruolo che ha all'interno della scuola (Docente/Studente)</li> <li>3. L'utente seleziona il suo strumento da una lista</li> <li>4. L'utente preme il pulsante "Sign up"</li> <li>5. La registrazione avviene con successo, il sistema registra il nuovo utente e mostra una scritta che notifica il successo dell'operazione di registrazione</li> <li>6. L'utente può tornare alla schermata di Log in attraverso il pulsante "Torna al Log in"</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>5. A. Se l'username inserito è già registrato, il sistema non registra l'utente e mostra una scritta che indica che il nome utente è già in uso</li> <li>5. B. Il sistema rileva un problema di connessione al Database e lo notifica all'utente</li> </ol>

Figura 3: Use Case Sign up (Interfaccia di riferimento Figura: 21)

### 2.1.2 Studente

Una volta effettuato il log in, allo studente è consentito solamente di selezionare il giorno e la fascia oraria in cui vuole prenotarsi. La scelta dell'aula è lasciata all'algoritmo che terrà conto dei requisiti dello studente (se necessita di aule attrezzate o insonorizzate) e dell'ottimizzazione dell'inquinamento acustico per quanto possibile. L'utente riceverà quindi una notifica a schermo in cui si indica l'aula a lui designata e una mail di conferma prenotazione.

UC	Prenotazione Studente
Description	L'utente di tipo Studente prenota un'aula
Level	User goal
Main actors	Studente
Main course	<ol style="list-style-type: none"> <li>1. L'utente seleziona una data</li> <li>2. L'utente seleziona uno slot orario compreso tra le 9:00 e le 19:00</li> <li>3. L'utente preme il pulsante "Prenota"</li> <li>4. Il sistema prenota un'aula disponibile per quella data e fascia oraria e mostra una scritta che indica il successo dell'operazione e il nome dell'aula prenotata</li> <li>5. Il sistema invia una mail di conferma della prenotazione all'utente contenente data, ora e aula prenotata</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>5. A. Se nello slot orario selezionato non c'è nessuna aula adatta alla prenotazione dell'utente viene notificato dal sistema</li> </ol>
Pre-Conditions	L'utente ha eseguito correttamente il Log in e ha delle credenziali di tipo Studente

Figura 4: Use Case Prenotazione Studente (Interfaccia di riferimento Figura: 22)

### 2.1.3 Docente

Ai docenti, invece, è consentita la scelta dell'aula da prenotare. L'algoritmo esclude dalla scelta tutte le aule che non soddisfano le esigenze del docente (ad esempio l'assenza del pianoforte per i docenti dello strumento) e quelle già occupate da altre prenotazioni.

UC	Prenotazione Docente
Description	L'utente di tipo Docente prenota un'aula
Level	User goal
Main actors	Docente
Main course	<ol style="list-style-type: none"> <li>1. L'utente seleziona una data</li> <li>2. L'utente seleziona uno slot orario compreso tra le 9:00 e le 19:00</li> <li>3. L'utente preme il pulsante "Mostra aule disponibili"</li> <li>4. Il sistema mostra la lista delle aule prenotabili e compatibili con le caratteristiche dell'utente</li> <li>5. L'utente seleziona un'aula dalla lista</li> <li>6. L'utente preme il tasto "Prenota aula selezionata"</li> <li>7. Il sistema prenota l'aula nello slot orario desiderato</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>4. A. Se nello slot orario selezionato non c'è nessuna aula adatta alla prenotazione dell'utente il sistema non mostra alcuna aula</li> <li>4. B. Il sistema rileva un problema di connessione al Database e lo notifica all'utente</li> <li>7. A. Il sistema rileva un problema di connessione al Database e lo notifica all'utente</li> </ol>
Pre-Conditions	L'utente ha eseguito correttamente il Log in e ha delle credenziali di tipo Docente

Figura 5: Use Case Prenotazione Docente (Interfaccia di riferimento Figura: 23)

#### 2.1.4 Admin

L'admin può limitare l'accesso alle aule in diversi orari. Può anche controllare lo stato delle diverse prenotazioni indicando la data ed eventualmente cancellarle. L'applicativo provvederà in automatico ad inviare una mail di segnalazione agli utenti la cui prenotazione è stata cancellata.



UC	Admin rende aula non prenotabile
Description	L'utente di tipo Admin rende un'aula non prenotabile in una fascia oraria
Level	User goal
Main actors	Admin
Main course	<ol style="list-style-type: none"> <li>1. L'utente seleziona una data</li> <li>2. L'utente seleziona uno slot orario compreso tra le 9:00 e le 19:00</li> <li>3. L'utente preme il pulsante "Mostra prenotazioni"</li> <li>4. Il sistema mostra una lista delle aule disponibili e una lista delle aule non disponibili</li> <li>5. L'utente seleziona un'aula dalla lista delle aule disponibili</li> <li>6. L'utente preme il tasto "Rendi aula non prenotabile"</li> <li>7. Il sistema rimuove l'aula selezionata dalla lista delle aule prenotabili e la inserisce nelle aule prenotate segnalando che è prenotata dall'admin cioè che nessun altro può prenotarla</li> <li>8. Il sistema memorizza una prenotazione a nome admin</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>3. A. Se l'utente non ha selezionato nessuna data il sistema indica all'utente di inserirla attraverso una scritta</li> <li>6. A. Se l'utente non ha selezionato nessun'aula dalla lista il sistema indica all'utente di selezionarne una</li> <li>7. A. Il sistema rileva un problema di connessione al Database e lo notifica all'utente senza modificare le liste</li> </ol>
Pre-Conditions	L'utente ha eseguito correttamente il Log in ed è l'admin

Figura 6: Use Case Aula non prenotabile (Interfaccia di riferimento Figura: 24)

UC	Admin cancella prenotazione utente
Description	L'utente di tipo Admin cancella una prenotazione utente perché l'aula non è più disponibile
Level	User goal
Main actors	Admin
Main course	<ol style="list-style-type: none"> <li>1. L'utente seleziona una data</li> <li>2. L'utente seleziona uno slot orario compreso tra le 9:00 e le 19:00</li> <li>3. L'utente preme il pulsante "Mostra prenotazioni"</li> <li>4. Il sistema mostra una lista delle aule disponibili e una lista delle aule prenotate</li> <li>5. L'utente seleziona un'aula dalla lista delle aule prenotate</li> <li>6. L'utente preme il tasto "Cancella prenotazione utente"</li> <li>7. Il sistema rimuove la prenotazione dell'utente e ne mette una a nome admin</li> <li>8. Il sistema invia una mail all'utente soggetto della cancellazione</li> <li>9. Il sistema cambia i dati della prenotazione impostando il nome utente ad "admin"</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>4. A. Se l'utente non ha selezionato nessuna data il sistema indica all'utente di inserirla attraverso una scritta</li> <li>7. A. Se l'utente non ha selezionato nessun'aula dalla lista il sistema indica all'utente di selezionarne una</li> <li>8. A. Il sistema rileva un problema di connessione al Database e lo notifica all'utente senza modificare le liste</li> </ol>
Pre-Conditions	L'utente ha eseguito correttamente il Log in ed è l'admin

Figura 7: Use Case Cancella prenotazione

UC	Admin rende aula nuovamente disponibile
Description	L'utente di tipo Admin cancella una prenotazione utente perché l'aula non è più disponibile
Level	User goal
Main actors	Admin
Main course	<ol style="list-style-type: none"> <li>1. L'utente seleziona una data</li> <li>2. L'utente seleziona uno slot orario compreso tra le 9:00 e le 19:00</li> <li>3. L'utente preme il pulsante "Mostra prenotazioni"</li> <li>4. Il sistema mostra una lista delle aule disponibili e una lista delle aule prenotate</li> <li>5. L'utente seleziona un'aula dalla lista delle aule prenotate</li> <li>6. L'utente preme il tasto "Rendi aula nuovamente disponibile"</li> <li>7. Se l'aula selezionata risulta prenotata dallo stesso admin il sistema rimuove la prenotazione dell'admin e reinserisce l'aula nella lista delle aule disponibili</li> </ol>
Alternative course	<ol style="list-style-type: none"> <li>5. A. Se l'utente non ha selezionato nessuna data il sistema indica all'utente di inserirla attraverso una scritta</li> <li>7. A. Se l'utente non ha selezionato nessun'aula dalla lista il sistema indica all'utente di selezionarne una</li> <li>7. B. Se l'utente ha selezionato un'aula il cui user non risulta essere l'admin il sistema indica all'utente di selezionare un'aula prenotata dall'admin</li> <li>7. C. Il sistema rileva un problema di connessione al Database e lo notifica all'utente senza modificare le liste</li> </ol>
Pre-Conditions	L'utente ha eseguito correttamente il Log in ed è l'admin

Figura 8: Use Case Rendi aula disponibile

## 2.2 Class diagram

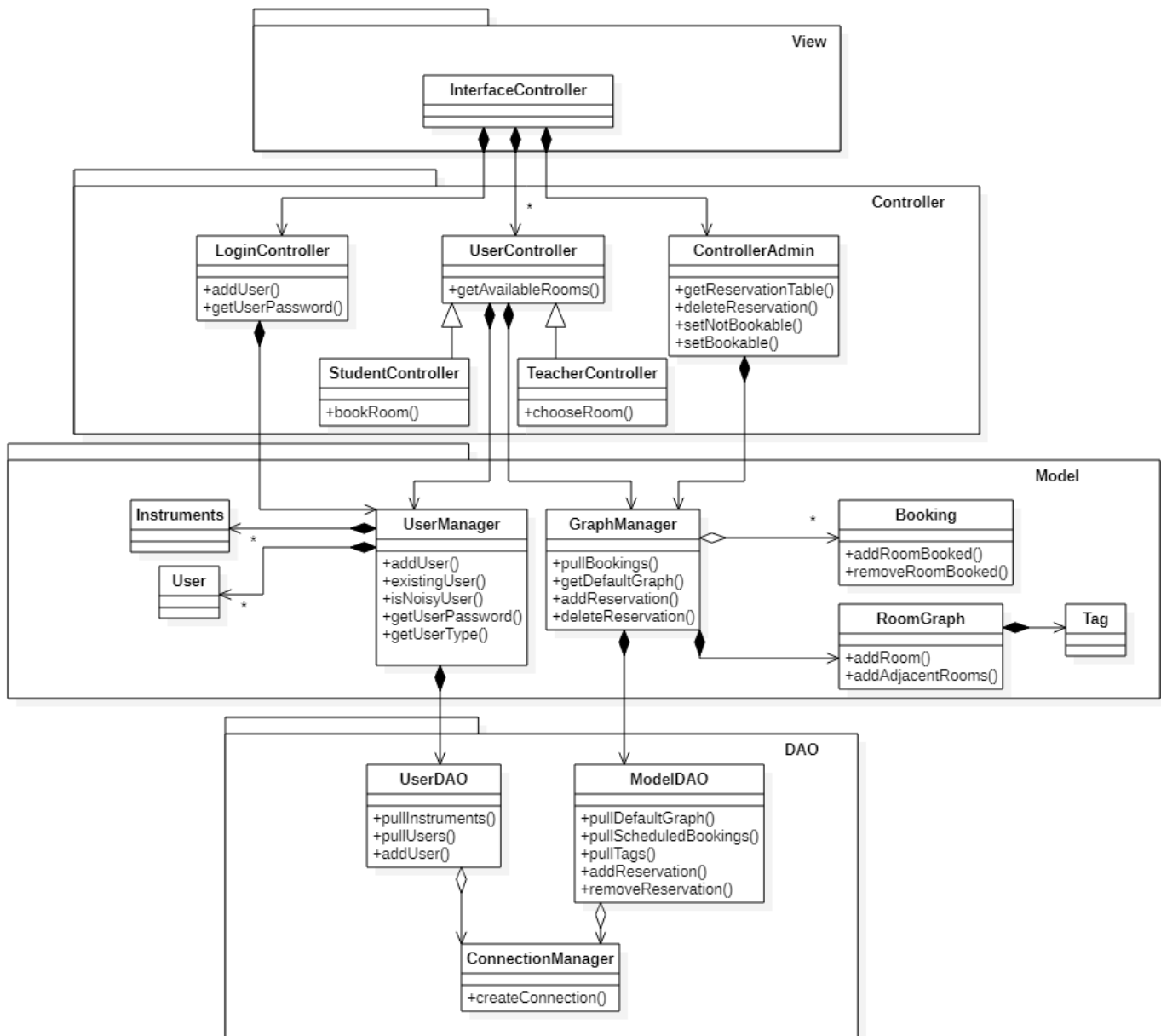


Figura 9: Diagramma delle classi dell'applicativo

La struttura del programma segue lo schema dell'MVC con l'aggiunta di un package DAO che permette l'interazione con un database. Ciascun package ha

un ruolo diverso all'interno del programma:

- La View si occupa di gestire l'interfaccia grafica con cui interagisce l'utente.
- I Controller richiedono servizi al modello raccogliendo le informazioni fornite dall'utente attraverso la view.
- Il modello gestisce i servizi forniti all'utente mantenendo lo stato del sistema coerente e pronto per fornire altri servizi.
- Il DAO controlla le interazioni con il database

Nella suddivisione in package del codice non è stato possibile inserire la classe `InterfaceController` in una sottocartella `view` per permettere ai metodi della libreria `JavaFX` di trovare correttamente il controller dell'interfaccia.

## 2.3 Design Patterns

### 2.3.1 Model View Controller

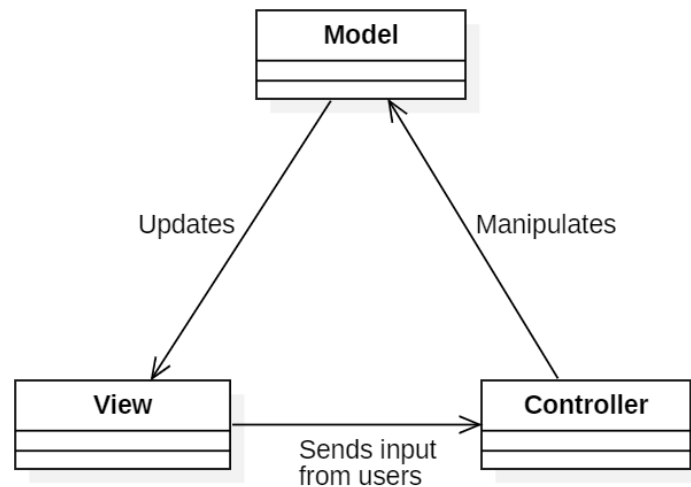


Figura 10: Diagramma UML di un generico pattern MVC

Il Model View Controller è un pattern in grado di separare la logica di presentazione dei dati dalla logica di business. Nel nostro caso il model può essere visto come l'insieme dei package `Model` e `DAO` con l'aggiunta del database stesso. La parte del codice del controller è quella inclusa nell'omonimo package ed è costituita da 5 classi, di cui 2 estendono la classe `UserController` che racchiude delle funzionalità comuni allo `StudentController` ed al `TeacherController`.

La view è costituita unicamente dalla classe `InterfaceController` che gestisce tutte le interazioni dell'utente con l'interfaccia. La scelta di usare una singola classe è stata dettata dalle esigenze delle librerie di JavaFX che prevedono l'esistenza di un unico controller dell'interfaccia.

### 2.3.2 Singleton

Nella gestione della connessione al database è stato implementato il pattern Singleton. Questo garantisce l'esistenza di un unico `ConnectionManager` responsabile di gestire la connessione al database sia che la richiesta provenga dallo `UserDAO` che dal `ModelDAO`.

## 2.4 Page Navigation Diagram

Ecco come è previsto che l'utente interagisca con l'interfaccia:

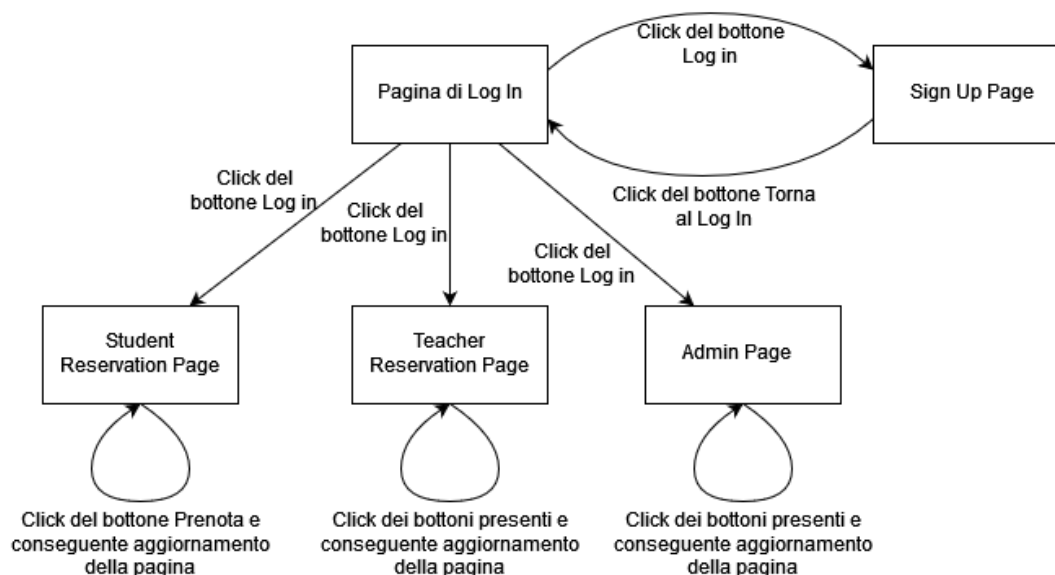


Figura 11: Page Navigation Diagram

## 2.5 Entity Relationship Diagram

L'Entity Relationship Diagram in figura descrive la struttura e le correlazioni delle tabelle del database dell'applicativo.

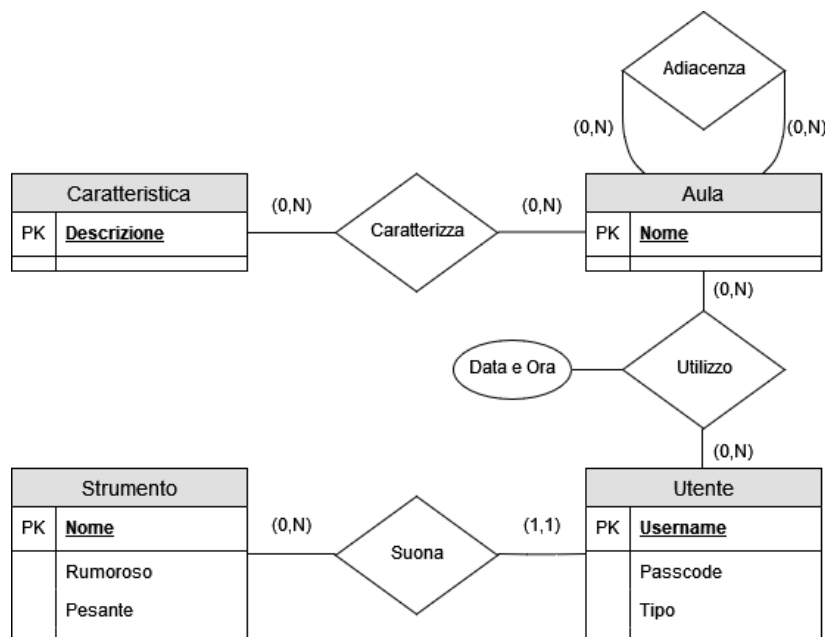


Figura 12: Entity Relationship Diagram

Sulla base dello schema E-R in figura 12, è stato scelto di tradurre la relazione "Caratterizza" in una tabella separata da quelle delle entità coinvolte per ridurre il numero di tuple da memorizzare. Analogamente, la relazione "Utilizzo" è stata tradotta introducendo nel database una nuova tabella mentre la relazione "Suona", presentando una partecipazione obbligatoria e con cardinalità massima pari a 1, è stata accorpata nella tabella Utente. La struttura effettiva il database è documentato nella sezione 3.6.

### 3 Implementazione delle classi

Il codice sorgente del progetto è articolato nel seguente modo:

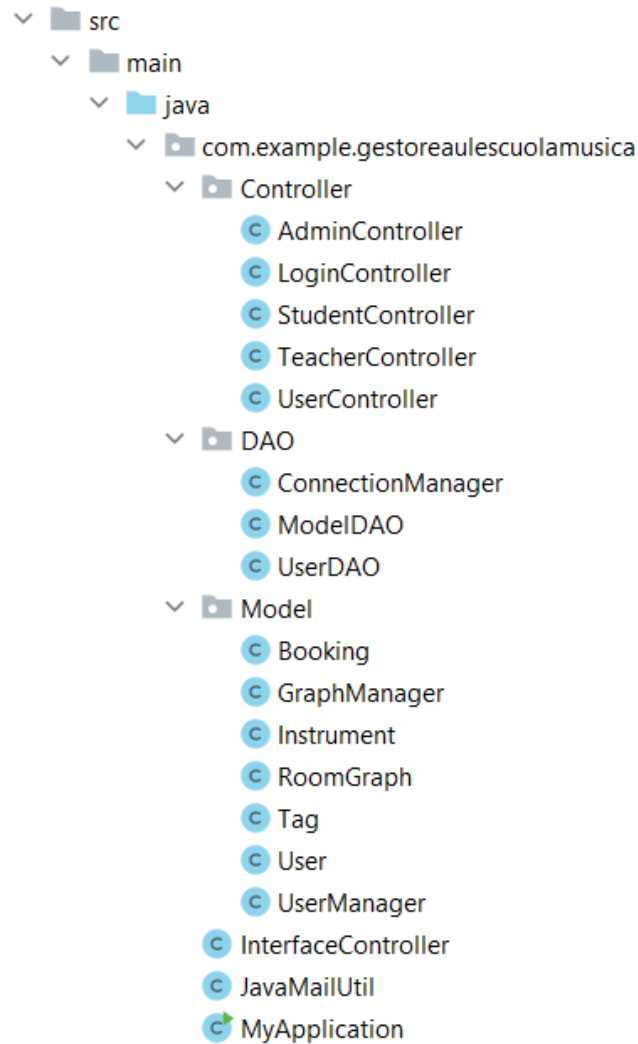


Figura 13: Struttura delle classi e dei package del progetto descritto

#### 3.1 Model

Il model 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:



### **3.1.1 User**

Classe che rappresenta l'utente dell'applicazione e che si occupa di rappresentare e gestire tutti quegli aspetti comuni ai vari tipi di utenti (Studenti e Docenti). Ciascun utente è dotato di uno username (indirizzo e-mail) identificativo, una password, lo strumento ed il tipo (se studente o docente).

### **3.1.2 Instrument**

La classe Instrument mantiene i dati relativi agli strumenti musicali di cui sono offerti corsi presso la Scuola di Musica. Ciascuno strumento è identificato dal nome e presenta informazioni circa la sua rumorosità e portabilità, informazioni necessarie all'algoritmo per stabilire l'aula più adatta ad un certo utente.

### **3.1.3 RoomGraph**

Tale grafo tiene traccia della struttura dell'edificio della Scuola di Musica, servendosi di una mappa in cui ad ogni aula associa una lista di aule adiacenti. Tale classe consente di aggiungere nuove aule ed aule adiacenti ad una stanza data.

### **3.1.4 Tag**

La classe Tag associa ad ogni aula una caratteristica che la contraddistingue come in una sorta di mapper, così da trasformare una relazione molti a molti (un'aula può avere più caratteristiche e più aule possono godere di una stessa caratteristica) in una più semplice relazione uno ad uno.

### **3.1.5 Booking**

La classe Booking tiene traccia delle prenotazioni eseguite dagli utenti, servendosi di una mappa che ad ogni aula prenotata associa l'username dell'occupante.

### **3.1.6 GraphManager**

Il GraphManager è il vero gestore dell'intero applicativo. Mantenendo riferimenti alle strutture quali il grafo/pianta della scuola, i tag che contraddistinguono le varie aule, si interfaccia direttamente con il DAO per prelevare, aggiungere o cancellare le prenotazioni effettuate dagli utenti.

### **3.1.7 UserManager**

Il suo ruolo è quello di gestione degli utenti. Mantiene una lista aggiornata degli utenti registrati all'applicativo consentendo al momento del Log in di controllare la correttezza delle credenziali inserite. Inoltre dialoga con il DAO per consentire l'inserimento nel database di un nuovo utente che ha eseguito il Sign up.

## 3.2 Controller

È il package responsabile di fornire i servizi richiesti dall'utente tramite la GUI. Il programma prevede diversi tipi di Controller, sulla base dei diversi tipi di utente che si interfacciano con esso.

### 3.2.1 LoginController

Si occupa della gestione delle operazioni di Log in e Sign up. In fase di Log in, quando l'utente inserisce le credenziali di accesso, controlla se esiste lo username inserito e se la password associata è quella corretta. Si occupa inoltre di determinare il tipo di utente che ha eseguito l'accesso. In fase di Sign up, invece, fornisce i dati allo userManager che provvederà all'inserimento dell'utente nel database e nel modello.

### 3.2.2 UserController, StudentController e TeacherController

Sia lo StudentController che il TeacherController ereditano metodi ed attributi dalla classe base UserController. Quest'ultima mantiene i riferimenti al GraphManager e allo UserManager del modello che utilizza per fornire i servizi richiesti. Dato che la prenotazione di un'aula avviene in modo diverso per studenti e docenti, lo UserController consente solo di determinare le aule disponibili in un certo orario tenendo conto delle esigenze del richiedente (ad esempio se è uno studente di pianoforte), interrogando il graphManager. Questa classe fornisce inoltre un metodo che permette di cancellare le prenotazioni.

Lo studentController fornisce il metodo bookRoom() che provvede in maniera automatica a fornire un'aula all'utente sulla base di un algoritmo che esegue l'assegnazione tenendo conto della trasportabilità dello strumento e minimizzando l'inquinamento acustico. Questo è realizzato attribuendo un grado di handicap ad ogni sala: per ogni aula vicina già occupata da uno strumento rumoroso si aumenta l'handicap; ciò avviene anche se la stanza è ad un piano sopraelevato e lo strumento è pesante.

```

public String bookRoom(String studentName, LocalDateTime time) throws SQLException, MessagingException {
    User student = userManager.getUser(studentName);
    Instrument instrument = student.getInstrument();
    ArrayList<String> rooms = getAvailableRooms(studentName, time);

    if(!rooms.isEmpty()){
        ArrayList<Integer> handicaps = new ArrayList<>();
        for (String room : rooms) {
            int handicap = 0;
            ArrayList<String> adjacentRooms = defaultGraph.getAdjacentRooms(room);
            for (String adjacentRoom : adjacentRooms) {
                if (userManager.isNoisyUser(bookings.getRoomsBooked().get(adjacentRoom))) {
                    handicap = handicap + 3;
                }
            }
            if (instrument.isHeavy() && isUpperFloorRoom(room)) {
                handicap = handicap + 2;
            }
            handicaps.add(handicap);
        }
        int min = handicaps.get(0);
        for (Integer handicap : handicaps) {
            if (min > handicap)
                min = handicap;
        }
        String tmp = rooms.get(handicaps.indexOf(min));
        graphManager.addReservation(time, tmp, student.getUsername());
        JavaMailUtil.sendReservationConfirm(studentName, time, tmp);
        return tmp;
    }
    return null;
}

```

Figura 14: Metodo relativo alla prenotazione di un'aula per lo studente

Il teacherController, invece, consente al docente di scegliere l'aula tra quelle disponibili e compatibili con le sue esigenze. La lista delle aule viene calcolata tramite il metodo fornito dalla superclasse.

```

public void chooseRoom(String room, LocalDateTime time, String teacher) throws SQLException, MessagingException {
    graphManager.addReservation(time, room, teacher);
    JavaMailUtil.sendReservationConfirm(teacher, time, room);
}

```

Figura 15: Metodo relativo alla prenotazione di un'aula per il docente

### 3.3 DAO

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

### 3.3.1 ConnectionManager

Si occupa di fornire la connessione al database al DAO che la richiede garantendo che essa sia unica attraverso il design pattern Singleton.

```
public class ConnectionManager {
    private static String url;
    private static String user;
    private static String password;
    private static ConnectionManager instance = null;
    private static Connection connection;

    private ConnectionManager(){
        url = "jdbc:postgresql://localhost:5432/ProgettoSWE";
        user = "webuser";
        password = "password";
    }

    public static ConnectionManager getConnectionManager() throws SQLException {
        if(ConnectionManager.instance == null){
            ConnectionManager.instance = new ConnectionManager();
            connection = ConnectionManager.createConnection();
        }
        return ConnectionManager.instance;
    }

    private static Connection createConnection() throws SQLException {
        return DriverManager.getConnection(url, user, password);
    }

    public Connection getConnection() { return connection; }
}
```

Figura 16: Metodo relativo alla prenotazione di un'aula per lo studente

### 3.3.2 ModelDAO

Si interfaccia con il database nella gestione dell'occupazione delle aule. Il modelDAO fornisce i metodi per prelevare i dati dal database come il grafo che descrive l'adiacenza delle aule, le prenotazioni ed i tag con le caratteristiche delle stanze. Tali metodi vengono invocati appena il programma viene aperto. Consente inoltre di aggiungere e rimuovere le prenotazioni dal database.

```

public HashMap<LocalDateTime, Booking> pullScheduledBookings() throws SQLException {
    connection = connectionManager.getConnection();
    Statement statement = connection.createStatement();
    String sql = "select * from utilizzo order by data_ora";
    ResultSet rs = statement.executeQuery(sql);
    HashMap<LocalDateTime, Booking> scheduledRoomBooked = new HashMap<>();
    while (rs.next()) {
        String room = rs.getString( columnLabel: "aula");
        String user = rs.getString( columnLabel: "utente");
        LocalDateTime time = rs.getTimestamp( columnLabel: "data_ora").toLocalDateTime();
        if(!scheduledRoomBooked.containsKey(time)){
            Booking booking = new Booking();
            booking.addRoomBooked(room, user);
            scheduledRoomBooked.put(time, booking);
        } else {
            scheduledRoomBooked.get(time).addRoomBooked(room, user);
        }
    }
    statement.close();
    return scheduledRoomBooked;
}

```

Figura 17: Metodo relativo alla prenotazione di un'aula per lo studente

### 3.3.3 UserDAO

Si interfaccia con il database nella gestione degli utenti. Consente di prelevare i dati degli utenti e degli strumenti di cui esistono corsi presso la scuola e di aggiungere nuovi utenti che hanno eseguito la registrazione.

```

public void addUser(String username, String password, String instrument, int type) throws SQLException {
    connection = connectionManager.getConnection();
    Statement statement = connection.createStatement();
    String sql = "insert into utente values('"+ username + "','"+ password + "','"+ instrument + "','"+ type + "')";
    statement.executeUpdate(sql);
    statement.close();
}

```

Figura 18: Metodo relativo alla prenotazione di un'aula per lo studente

## 3.4 Altre classi

### 3.4.1 JavaMailUtil

Tale classe si occupa di inviare mail di conferma e di cancellazione delle prenotazioni agli utenti. Tale classe si serve dei servizi offerti dalla libreria ja-

vax.mail per consentire l'invio di mail dall'indirizzo di posta elettronica dedicato noreply.scuoladimusica@gmail.com.

### 3.4.2 InterfaceController

Gestisce tutte le pagine dell'interfaccia interfacciandosi con i tre controller: LoginController, StudentController e TeacherController.

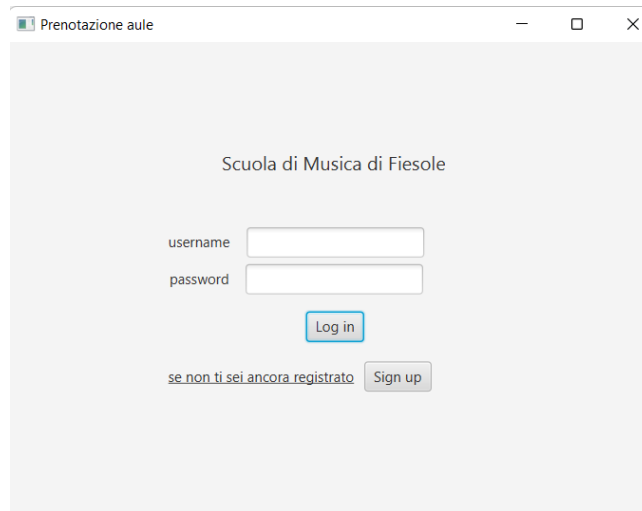
## 3.5 Viste



Figura 19: Viste del progetto descritto

Le viste sono realizzate utilizzando JavaFX che, grazie all'applicazione SceneBuilder consente di realizzare interfacce in modo rapido e semplice. Si riportano in dettaglio gli scopi di ciascun file .fxml:

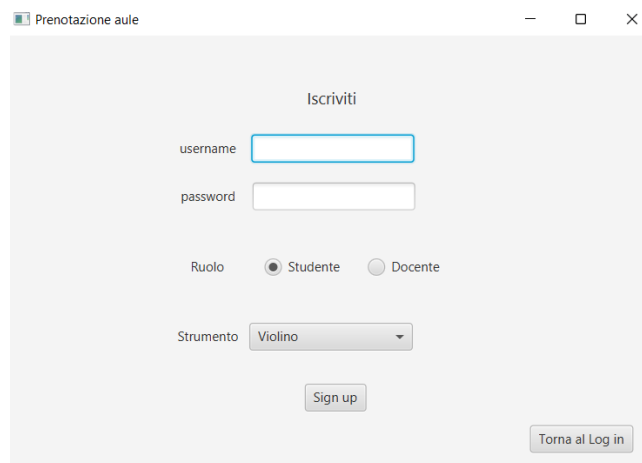
- hello-view.fxml è l'interfaccia che compare all'avviamento del programma e consente di effettuare il Log in o di spostarsi nella pagina di Sign up.



The screenshot shows a window titled "Prenotazione aule". Inside, the text "Scuola di Musica di Fiesole" is centered. Below it, there are two input fields labeled "username" and "password". A blue "Log in" button is positioned below the password field. At the bottom, there is a link "se non ti sei ancora registrato" and a "Sign up" button.

Figura 20: Interfaccia di Log In

- sign-up-view.fxml consente di eseguire il sign up.



The screenshot shows the same window titled "Prenotazione aule", but with the text "Iscriviti" centered. Below it, there are "username" and "password" input fields. Under these, there is a "Ruolo" section with two radio buttons: "Studente" (which is selected) and "Docente". Below that is an "Strumento" dropdown menu currently showing "Violino". At the bottom center is a "Sign up" button, and at the bottom right is a "Torna al Log in" button.

Figura 21: Interfaccia di Sign up

- student-reservation-view.fxml è l'interfaccia che si presenta allo studente una volta effettuato il log in e gli consente di eseguire la prenotazione di un'aula definendo data e ora.

Figura 22: Interfaccia di prenotazione per gli Studenti

- teacher-reservation-view.fxml è l'interfaccia che si presenta al docente una volta effettuato il log in. Il docente può qui indicare una data ed un orario e il sistema restituisce una lista di aule disponibili tra le quali può scegliere.

Figura 23: Interfaccia di prenotazione per i Docenti

- admin-view.fxml è l'interfaccia dell'admin. Da qui indicando una data ed un orario l'amministratore può vedere le aule disponibili e quelle prenotate, con indicazione dell'utente occupante, e può rendere delle aule non prenotabili per esigenze della scuola, talvolta cancellando anche prenota-



zioni di utenti (che saranno notificati della cancellazione tramite mail), oppure renderle nuovamente prenotabili.

Figura 24: Interfaccia di gestione aule dell'admin

### 3.6 Struttura del database

Per la gestione dei dati sia degli utenti che delle aule e prenotazioni si utilizza un database relazionale gestito attraverso PostgreSQL. In figura si mostra la struttura del database.

```
table caratteristica(PK(descrizione))
table Aula(PK(nome))
table strumento(PK(nome), rumoroso,pesante)
table utente(PK(username), passcode, strumento,tipo) FK(strumento) ref strumento
table tag(PK(Aula varchar(50), caratteristica)) FK(Aula) ref aula, FK(caratteristica) ref caratteristica
table aule_adiacenti(PK(Aula, Adiacente)) FK(Aula) ref aula FK(Adiacente) ref aula
table utilizzo(PK(Aula, Data_ora),utente) FK(Aula) ref aula, FK(utente) ref utente
```

Figura 25: Struttura del database

## 4 Testing

È stato realizzato unit-testing di ogni componente dell'applicativo. I test si basano sul concetto di white-box perspective: si ha accesso alla logica interna del componente così da poter confrontare il risultato con quello atteso. È stato inoltre realizzato integration testing, per controllare la corretta integrazione tra le diverse classi. Le classi test sono state suddivise in package che ricalcano la suddivisione utilizzata per le classi a cui si riferiscono i test. Infine è stato effettuato un E2E test interagendo solo con l'interfaccia esterna dell'applicazione. Questo è stato realizzato manualmente (senza l'uso di un simulatore di interazione dell'utente visto che il numero di operazioni consentite a ciascuno non è così esteso) creando un utente di ogni tipo e verificando che ciascuna funzione a lui consentita si comporti nella maniera attesa: si controlla che avvenga il corretto aggiornamento del database, che l'invio delle mail di conferma e cancellazione delle prenotazioni vada a buon fine etc...

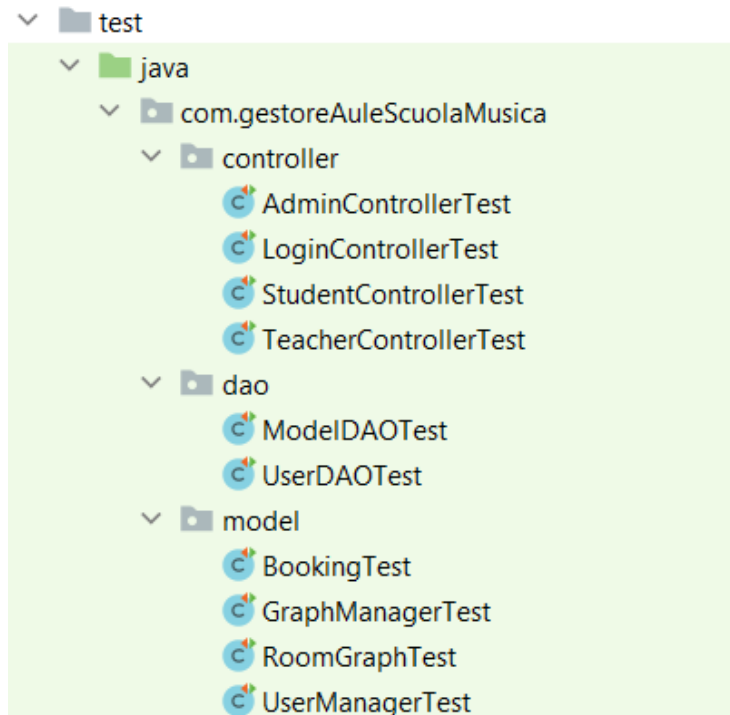


Figura 26: Struttura delle classi e dei package relativi al test del progetto

### 4.1 DAO

I test di questa sezione sono di tipo strutturale volti a verificare la corretta interazione con il database.

#### 4.1.1 ModelDAOTest

I test di questa classe hanno bisogno di un set up che instanzi una connessione al database e di un tear down che disconnetta dal database e che si preoccupi di lasciare il database esattamente come era stato trovato prima dell'inizio dei test.

Il primo metodo testato è `addReservation()` che permette di aggiungere una prenotazione nella tabella utilizzo del database. Si invoca quindi il metodo `addReservation()` e poi viene instaurata una connessione autonoma al database per controllare che l'inserimento sia avvenuto con successo e che ogni attributo della tupla aggiunta corrisponda a quelli realmente inseriti.

Per il testing del secondo metodo `deleteReservation()` la classe `ModelDAOTest` instaura una connessione al database e inserisce una nuova tupla alla tabella utilizzo. Poi viene chiamato il metodo `deleteReservation()` e, attraverso le Assertions, viene controllato che la tupla inserita inizialmente sia stata rimossa.

#### 4.1.2 UserDAOTest

L'unico metodo testato per questa classe è il metodo `addUser()`. Il metodo `addUserTest()`, dopo aver istanziato lo `userDAO`, esegue il metodo `addUser()`. In seguito instaura una connessione al database, esegue una query per trovare la tupla inserita e controlla che tutti i valori siano corrispondenti a quelli inseriti nella funzione `addUser()`. Infine, per lasciare il database nello stato in cui era prima del testing esegue un delete sulla tupla inserita inizialmente.

### 4.2 Model

#### 4.2.1 BookingTest

Il test strutturale consente di controllare il corretto funzionamento dei due metodi che alterano la mappa interna alla classe che mantiene l'insieme delle aule prenotate ed degli utenti che le occupano. Il metodo `addRoomBookedTest()` controlla che l'inserimento di una nuova prenotazione vada a buon fine, mentre `removeRoomBookedTest()` ne verifica la corretta cancellazione dalla struttura dati.

#### 4.2.2 GraphManagerTest

In questa classe si verifica il corretto funzionamento della classe `graphManager`, fulcro dell'intero applicativo nella gestione delle prenotazioni sia dal punto di vista strutturale che di integrazione con il `modelDAO` con cui dialoga nello scambio di dati. I metodi testati sono quelli necessari all'inserimento ed alla cancellazione di prenotazioni. Per consentire la creazione di nuove prenotazioni, nella fase di set up si provvede alla creazione di un utente di test "user\_test" che viene inserito provvisoriamente nel database. In `addReservationTest()` si invoca la funzione `addReservation()` della classe oggetto del test e si verifica sia accedendo direttamente al database con una query SQL sia nelle strutture

temporanee della classe stessa che la prenotazione sia stata inserita. In `deleteReservationTest()`, allo stesso modo, se ne controlla la cancellazione. Infine nel `tearDown()` si provvede alla cancellazione dell'utente fittizio in modo tale da lasciare la struttura del database inalterata rispetto all'inizio dei test.

#### 4.2.3 RoomGraphTest

Anche in questo caso si testano strutturalmente i metodi responsabili dell'alterazione degli attributi della classe. I metodi `addRoomTest()` ed `addAdjacentRoomsTest()` provvedono rispettivamente alla verifica dell'inserimento nella struttura di una nuova aula e delle sue rispettive aule adiacenti.

#### 4.2.4 UserManagerTest

Il suo compito è di verificare la corretta gestione degli utenti da parte dello `UserManager` sia in termini delle strutture dati interne alla classe sia nell'interazione con lo `UserDAO`. Il primo metodo ad essere testato è `addUser()` che, grazie all'interazione con il DAO, consente di aggiungere un nuovo utente al database. Il test consiste nell'utilizzo del metodo suddetto per registrare l'utente e la verifica consiste nell'accesso diretto al database tramite query SQL. Il metodo `existingUserTest()` dopo aver inserito un utente verifica che esso sia riconosciuto come esistente, controllando che il booleano ritornato da `existigUser()` sia "true". Il metodo `isNoisyUserTest()`, invece, controlla la corretta individuazione di un utente rumoroso e ciò viene fatto inserendo un nuovo utente tramite il metodo `sopracitato` e poi verificando che il sistema individui correttamente questa sua caratteristica. Allo stesso modo lavorano i metodi `getUserPasswordTest()` e `getUserTypeTest()`.

### 4.3 Controller

I test di queste classi sono di carattere funzionale in quanto sono volti a valutare la correttezza dei metodi responsabili delle gestione degli use case. Tali sono sempre preceduti dal metodo `set up` che permette di instaurare una connessione e, nel caso fosse necessario, di creare dei nuovi oggetti per il testing. Il metodo `tearDown()` invece viene eseguito dopo i test e serve per far sì di non aver modificato nulla nè nel modello, nè nel database.

#### 4.3.1 LoginControllerTest

Il primo metodo testato è `addUser()`. Nel test viene quindi invocato tale metodo, poi viene interrogato il database per controllare che sia stato aggiunto l'utente sfruttando una connessione indipendente dallo `UserController`. Inoltre si controlla anche che il nuovo utente sia stato aggiunto correttamente nel modello, ovvero nella collezione di `User` mantenuta dallo `UserManager`. Gli altri 3 metodi che vengono testati sono `isExistingUser()`, `getUserPassword()` e `getUserType()`, ciascuno dei quali restituisce uno degli attributi dello `user`. Perciò il loro testing

si limita a invocare il metodo da testare e poi controllare la coerenza del valore restituito dallo stesso.

#### 4.3.2 StudentControllerTest

Il testing del metodo `getAvailableRooms()`, nonostante la sua verbosità, è molto semplice: prima invoca il metodo testato su 3 tipi di studente con esigenze diverse (studente di pianoforte, di percussioni e studente generico); poi controlla che gli elementi della lista restituita siano quelli attesi e che la dimensione della lista sia coerente con il risultato previsto.

Il metodo `bookRoomTest()` invoca il metodo `bookRoom()` e poi interroga il database e il modello verificando che la prenotazione sia stata registrata.

In modo simile il metodo `deleteReservationTest()` prima invoca `bookRoom()` e il metodo oggetto del test `deleteReservation()` sulla prenotazione appena inserita. Segue l'usuale controllo del database e del modello.

#### 4.3.3 TeacherControllerTest

Nel `TeacherController` si verifica la correttezza del metodo `chooseRoom()` che consente ad un insegnante di prenotare un'aula tra quelle disponibili. Nella fase di set up viene quindi creato un nuovo docente che sarà utilizzato per generare le prenotazioni di prova. Il test prevede quindi l'invocazione del metodo `chooseRoom()` e la successiva verifica nel database e nel modello che la prenotazione effettuata risulti presente.

#### 4.3.4 AdminControllerTest

Qui si testano i metodi relativi alle operazioni consentite all'admin del programma:

- `getAvailableRooms()` che gli consente di determinare le aule disponibili in una certa data e ora.
- `setNotBookable()` che permette di definire un'aula non prenotabile in un preciso slot orario
- `setBookable()` che annulla l'azione della precedente.
- `deleteReservation()` che cancella la prenotazione di un utente e rende l'aula non prenotabile in quello slot temporale.

Per il test di `getAvailableRooms()`, come già avveniva nel caso dello `StudentController`, il test viene eseguito controllando che la dimensione della lista delle aule disponibili coincida con quella attesa.

Nel caso di `setNotBookable()`, invece, si esegue la funzione e si controlla che l'aula scelta non risulti più tra quelle disponibili nella fascia oraria selezionata. Al termine del test si provvede a rendere di nuovo disponibile l'aula accedendo direttamente al database così da ripristinare la struttura iniziale.

Per `setBookable()` si procede allo stesso modo controllando il corretto annullamento della precedente operazione `setNotBookable()`. In questo caso, quindi, si controlla che l'aula risulti nuovamente tra le disponibili.

Il test di `deleteReservation()` è forse il più complesso, in quanto necessita della creazione di un utente fittizio che esegua una prenotazione che verrà poi cancellata. Una volta creati utente e prenotazione, viene invocato il metodo `deleteReservation()`. Si controlla quindi che nel database non figuri la prenotazione e che l'aula non risulti comunque disponibile. Infine si provvede alla cancellazione di tutte le informazioni inserite nel database per il test sfruttando il riferimento al `connectionManager` che fornisce accesso diretto al db.

## 4.4 Esiti

Di seguito sono forniti gli esiti dei 26 test effettuati.

✓	✓ BookingTest	19 ms
	✓ removeRoomBookedTest()	16 ms
	✓ addRoomBookedTest()	3 ms
✓	✓ StudentControllerTest	2 sec 357 ms
	✓ deleteReservationTest()	1 sec 674 ms
	✓ getAvailableRoomsTest()	6 ms
	✓ bookRoomTest()	677 ms
✓	✓ UserManagerTest	25 ms
	✓ getUserPasswordTest()	7 ms
	✓ addUserTest()	6 ms
	✓ existingUserTest()	5 ms
	✓ getUserTypeTest()	4 ms
	✓ isNoisyUserTest()	3 ms
✓	✓ TeacherControllerTest	763 ms
	✓ chooseRoomTest()	763 ms
✓	✓ AdminControllerTest	731 ms
	✓ deleteReservation()	722 ms
	✓ setBookable()	4 ms
	✓ setNotBookable()	3 ms
	✓ getAvailableRooms()	2 ms

✓	✓ RoomGraphTest	2 ms
	✓ addRoomTest()	1 ms
	✓ addAdjacentRoomsTest()	1 ms
✓	✓ ModelDAOTest	7 ms
	✓ removeReservationTest()	4 ms
	✓ addReservationTest()	3 ms
✓	✓ GraphManagerTest	9 ms
	✓ addReservationTest()	4 ms
	✓ deleteReservationTest()	5 ms
✓	✓ LoginControllerTest	13 ms
	✓ getUserPasswordTest()	3 ms
	✓ addUserTest()	3 ms
	✓ isAnExistingUserTest()	3 ms
	✓ getUserTypeTest()	4 ms
✓	✓ UserDAOTest	2 ms
	✓ addUserTest()	2 ms

Figura 27: Esiti dei test