UNIVERSITÀ DI FIRENZE

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

MUSIC IN TOWN

Autori:

Paolo Sbarzagli e Cristian Sician

Corso:

Ingegneria del Software

Professore:

Enrico Vicario

APRILE 2023

Indice

1 Introduzione

1.1 Finalità del progetto

L'applicativo ha come obbiettivo quello di implementare un efficiente sistema di organizzazione e promozione degli eventi a livello cittadino. Ad oggi, infatti, l'organizzazione di un evento non sfrutta tutte le potenzialità che una rete di connessione tra organizzatori, musicisti e spettatori potrebbe fornire. L'attuale sistema risulta infatti inefficiente soprattutto per i musicisti, che anzichè disporre di un hub che gli permetta di candidarsi e promuovere la propria musica live, devono ricorrere ad agenzie di booking che dispongano dei contatti necessari per interagire con gli organizzatori di eventi. In questo modo dunque, si ha una grande limitazione al potenziale del singolo musicista, che dipende da numerosi intermediari per le proprie esibizioni live.
Un'altra limitazione riguarda gli organizzatori, che anzichè scegliere tra un vasto numero di esecutori, riescono a raggiungere solo quelli accettati dalle agenzie di booking.
Infine si ha un'ulteriore problematica che riguarda il potenziale pubblico di un evento, in quanto spesso non c'è un luogo unico in cui trovare tutti gli avvenimenti in programma in una determinata area.

Il gestionale permette quindi di automatizzare il processo di gestione degli eventi, dalla loro creazione alla selezione di un musicista tra la lista di quelli registrati.

Il sistema è pensato principalmente per due tipologie di eventi:

- I **Public Events**, organizzati su suolo pubblico da un organizzatore, previa autorizzazione comunale.
- I **Private Events**, organizzati all'interno di un locale privato gestito da un proprietario.

Ognuno di questi eventi può essere **Open** se consente ai musicisti di candidarsi per suonare all'evento stesso, **Closed** se il musicista è stato scelto al momento della creazione dall'organizzatore e dunque si ha solo una visualizzazione dell'evento.

Nello specifico, il sistema distingue cinque categorie principali di utenti:

- I **Planners**, figura destinata alla creazione degli eventi sia sul suolo pubblico che su quello privato. Possono generare eventi e selezionare i musicisti che si esibiranno.
- I **Musicians**, che si registrano inserendo le proprie caratteristiche (genere, numero di membri, nome d'arte ecc.). Possono candidarsi agli eventi aperti che sono stati organizzati in attesa di un riscontro da parte dell'organizzatore.
- Gli **Owners**, che gestiscono uno dei locali registrati nel gestionale. Possono organizzare eventi presso il proprio locale oppure accettare quelli proposti da un planner esterno.
- La **Municipality**, che si occupa semplicemente dell'approvazione di una richiesta di evento su suolo pubblico da parte di un planner.
- Il **Basic User**, che può visualizzare tutti gli eventi in programma e filtrarli per data, oltre che una lista dettagliata dei locali.

1.2 Tecnologie e standard utilizzati

Il software è realizzato in Java tramite l'IDE IntelliJ. Il modello dati è rappresentato dal package domainModel, mentre la logica dell'applicativo risiede nel package businessLogic.
La memorizzazione dei dati è stata affidata ad un database (come accordato per integrare la parte di laboratorio di informatica), realizzato tramite PostgreSQL e le librerie di JDBC. L'accesso al database è gestito dal package DAO che si interfaccia tra il codice ed i dati.
Per i diagrammi delle classi, realizzati con StarUML, è stato utilizzato lo standard UML, mentre il testing è stato realizzato sfruttando le librerie di JUnit. Anche il diagramma ER è stato realizzato sfruttando StarUML.
Inoltre, durante il versionamento su GitHub, abbiamo seguito le regole di nomenclatura consigliate da conventionalcommits.org.

2 Progettazione

2.1 Diagramma dei casi d'uso

Come specificato in precedenza, nel sistema sono coinvolti cinque attori, ciascuno caratterizzato da diverse funzionalità. La Fig. 1 rappresenta il diagramma dei casi d'uso degli attori dell'applicativo in modo schematico e semplificato, raggruppando le operazioni di visualizzazione (eventi e locali) comuni a tutti gli utenti.
Altra funzionalità comune a due tipologie di utente (Owner e Planner) è quella che permette di creare eventi privati e di selezionare le Band che si esibiranno presso gli stessi.
Le rimanenti operazioni sono invece specifiche dell'utente, come ad esempio la possibilità di effetuare candidature (Musician), quella di approvare eventi privati (Owner) e quella di approvare eventi pubblici (Municipality).

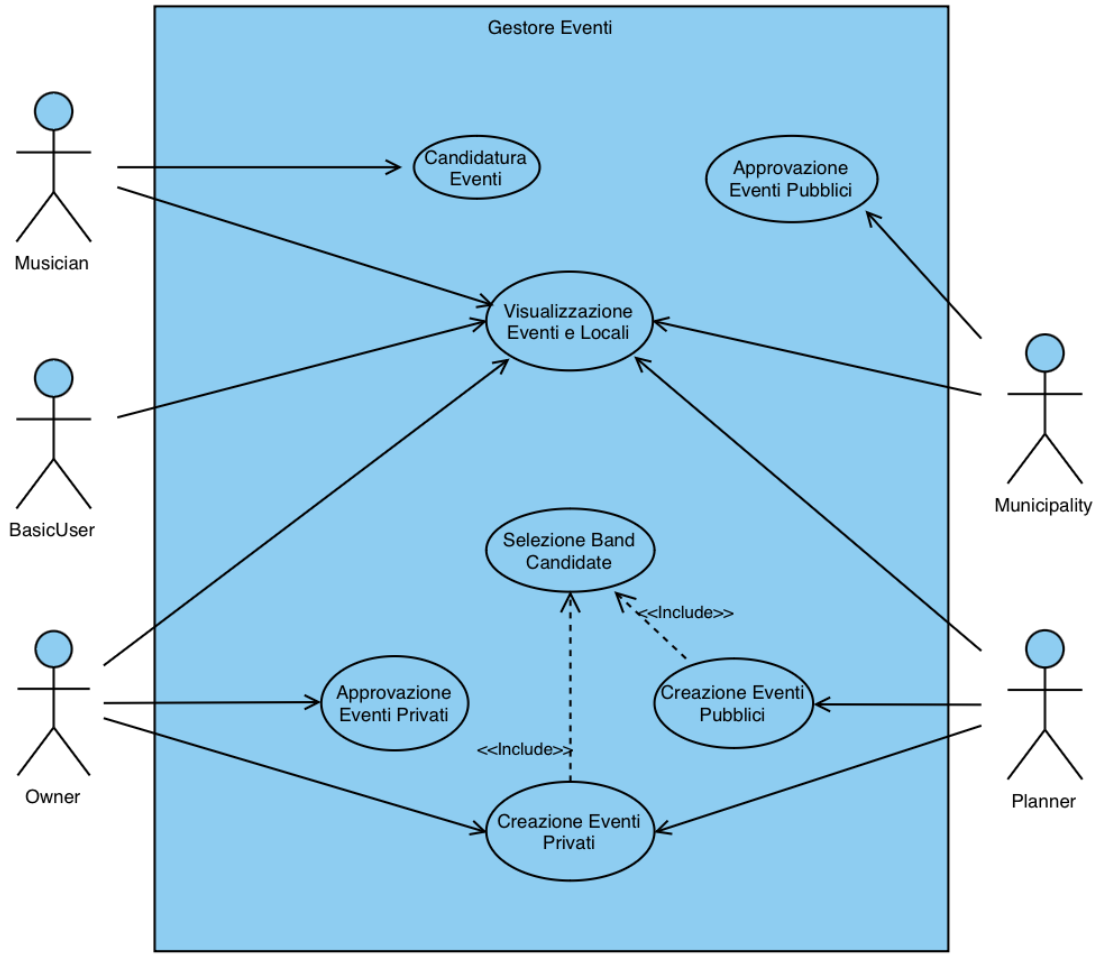

Figura 1: Diagramma dei casi d'uso

2.1.1 Use Case Template

In seguito vengono riportati i template dei singoli casi d'uso, in modo specifico e dettagliato. Vengono analizzati: gli attori coinvolti, il percorso di funzionamento, eventuali percorsi alternativi ed eventuali pre-condizioni o post-condizioni.

USE CASE #1	**CREAZIONE EVENTO**
Description	Lo User (Planner o Owner) crea un evento
Level	User goal
Actors	Planner, Owner
Basic Course	1. Lo User accede al menù degli eventi. 2. Seleziona l'opzione "Crea un nuovo evento". 3. Inserisce tutte le info richieste relative all'evento. 4. Sceglie se l'evento sarà aperto a sottoscrizioni da parte dei Musician. 5. Inserisce il luogo privato o pubblico in cui l'evento si svolgerà. 6. Vengono controllati i valori inseriti. 7. Se i valori PlaceID e Date sono corretti l'evento viene generato.
Alternative Course 1	7. Se i valori PlaceID e Date sono errati l'evento viene rifiutato.
Alternative Course 2	8. Se l'evento è pubblico viene posto in attesa di una conferma da parte della relativa Municipality.
Alternative Course 3	8. Se l'evento è privato e creato da un Planner viene posto in attesa della conferma dell'Owner del Private Place.
Pre-Conditions	Nessuna

Figura 2: Creazione di un Evento

USE CASE #2	**SOTTOSCRIZIONE AD UN EVENTO**
Description	Il Musician si candida per suonare ad un evento in programma.
Level	User goal
Actors	Musician
Basic Course	1. il Musician accede al menù degli eventi. 2. Seleziona l'opzione "Candidati per un evento". 3. Inserisce l'ID dell'evento. 4. Viene controllato se l'ID esiste e se l'evento è Open. 5. La candidatura viene aggiunta alla lista delle candidature per l'evento.
Alternative Course 1	5. Se l'ID è errato non viene effettuata la candidatura.
Alternative Course 2	5. Se l'evento non è open non viene effettuata la candidatura.
Pre-Conditions	L'evento deve essere di categoria Open, ovvero aperto alla sottoscrizione dei musicisti

Figura 3: Candidatura ad un Evento aperto

USE CASE #3	VISUALIZZAZIONE COMPLETA EVENTI
Description	Lo User (di qualsiasi tipologia) visualizza tutti gli eventi in programma con le relative info collegate a ciascuno.
Level	User goal
Actors	Qualsiasi tipo di User
Basic Course	1. Lo User accede al menù degli eventi 2. Seleziona l'opzione "Visualizza tutti gli eventi". 3. Viene stampata a schermo la lista di tutti gli eventi e le relative info.
Alternative Course	Nessuno
Pre-Conditions	Nessuna

Figura 4: Visualizzazione di tutti gli Eventi in programma

USE CASE #4	VISUALIZZAZIONE FILTRATA EVENTI
Description	Lo User (di qualsiasi tipologia) visualizza tutti gli eventi in programma fino ad una specifica data, con le relative info collegate a ciascuno.
Level	User goal
Actors	Qualsiasi tipo di User
Basic Course	1. Lo User accede al menù degli eventi 2. Seleziona l'opzione "Filtra gli eventi". 3. Lo User inserisce la data fino alla quale vuole visualizzare gli eventi. 4. Viene controllato se la data inserita è valida 5. Viene stampata a schermo la lista con tutti gli eventi e le relative info.
Alternative Course 1	5. Se la data inserita è precedente a quella odierna viene generato un errore e si torna al punto 3.
Alternative Course 2	5. Se non sono presenti eventi fino a quella data viene visualizzato un messaggio di errore.
Pre-Conditions	Nessuna

Figura 5: Visualizzazione filtrata degli Eventi in programma

USE CASE #5	VISUALIZZAZIONE PLACES
Description	Lo User (di qualsiasi tipologia) visualizza tutti i club e luoghi pubblici con le relative info.
Level	User goal
Actors	Qualsiasi tipo di User
Basic Course	1. Lo User seleziona l'opzione "Visualizza posti". 2. Viene stampata a schermo la lista di tutti i posti registrati con le relative info.
Alternative Course	Nessuno
Pre-Conditions	Nessuna

Figura 6: Visualizzazione dei luoghi adibiti ad ospitare Eventi

USE CASE #6	ACCETTAZIONE/RIFIUTO EVENTO
Description	Lo User (Municipality o Owner) accetta o rifiuta un evento organizzato dal Planner presso uno dei propri Places.
Level	User goal
Actors	Municipality, Owner.
Basic Course	1. Lo User accede al menù degli eventi. 2. Seleziona l'opzione "Visualizza proposte evento". 3. Viene stampata a schermo la lista con tutte le info dell'evento proposto. 4. Accetta la realizzazione dell'evento. 5. il Planner viene notificato. 6. L'evento viene inserito nel database degli eventi attivi.
Alternative Course	4. Rifiuta la realizzazione dell'evento. 5. Il Planner viene notificato. 6. L'evento viene cancellato.
Pre-Conditions	Il Planner deve aver creato un evento.

Figura 7: Accettazione o rifiuto di un Evento Pubblico

USE CASE #7	SELEZIONE BAND PER EVENTO
Description	Lo User (Owner o Planner) Seleziona il Musician che suonerà nell'evento pianificato.
Level	User goal
Actors	Planner, Owner.
Basic Course	1. Lo User accede al menù degli eventi. 2. Seleziona l'opzione "Seleziona Musician per evento". 3. Viene stampata a schermo la lista con tutte le candidature e info dei Musician per l'evento organizzato. 4. Inserisce l'ID della band scelta. 5. il Musician relativo viene notificato.
Alternative Course	Nessuno
Pre-Conditions	Almeno un Musician deve aver effettuato la candidatura.

Figura 8: Selezione di una band per Evento aperto

USE CASE #8	VISUALIZZA SOTTOISCRIZIONI AD EVENTI
Description	Lo User (Musician) visualizza tutti gli eventi a cui ha effettuato una candidatura.
Level	User goal
Actors	Musician
Basic Course	1. Lo User accede al menù degli eventi. 2. Seleziona l'opzione "Visualizza candidature". 3. Viene stampata a schermo la lista con tutti gli eventi a cui lo User si è candidato.
Alternative Course	3. Se lo User non ha effettuato nessuna candidatura viene stampato un messaggio di errore.
Pre-Conditions	Nessuna

Figura 9: Visualizzazione delle candidature di un Musicista

2.2 Class Diagram

Di seguito (Fig. 10) è riportato il diagramma delle classi dell'applicativo e come queste sono legate e interagiscono tra loro.
Sono presenti i seguenti packages:

- BusinessLogic: figura 11 rappresenta la logica di controllo del sistema. Contiene tutte le classi che si occupano di manipolare e controllare i dati e i servizi del domainModel, come ad esempio l'EventController per controllare la creazione o modifica di eventi, o il MusicianController per gestire tutte le possibili azioni eseguite dall'utente musicista.

- DomainModel: figura 12 rappresenta il modello dei dati. Contiene tutte le classi tramite le quali sono stati rappresentati i dati e gli oggetti nell'applicativo. Questo package mantiene quindi le istanze concrete del gestionale sulle quali agiscono i controller.

- DAO: la figura 13 contiene tutte le classi necessarie all'interazione con il database, realizzate seguendo il pattern DAO (Data Access Object) specifico per la gestione dei database e descritto in modo accurato nella sezione dedicata 2.3.

- Interface: la figura 14 rappresenta le varie interfacce a disposizione degli utenti per la visualizzazione delle operazioni e delle liste di eventi.

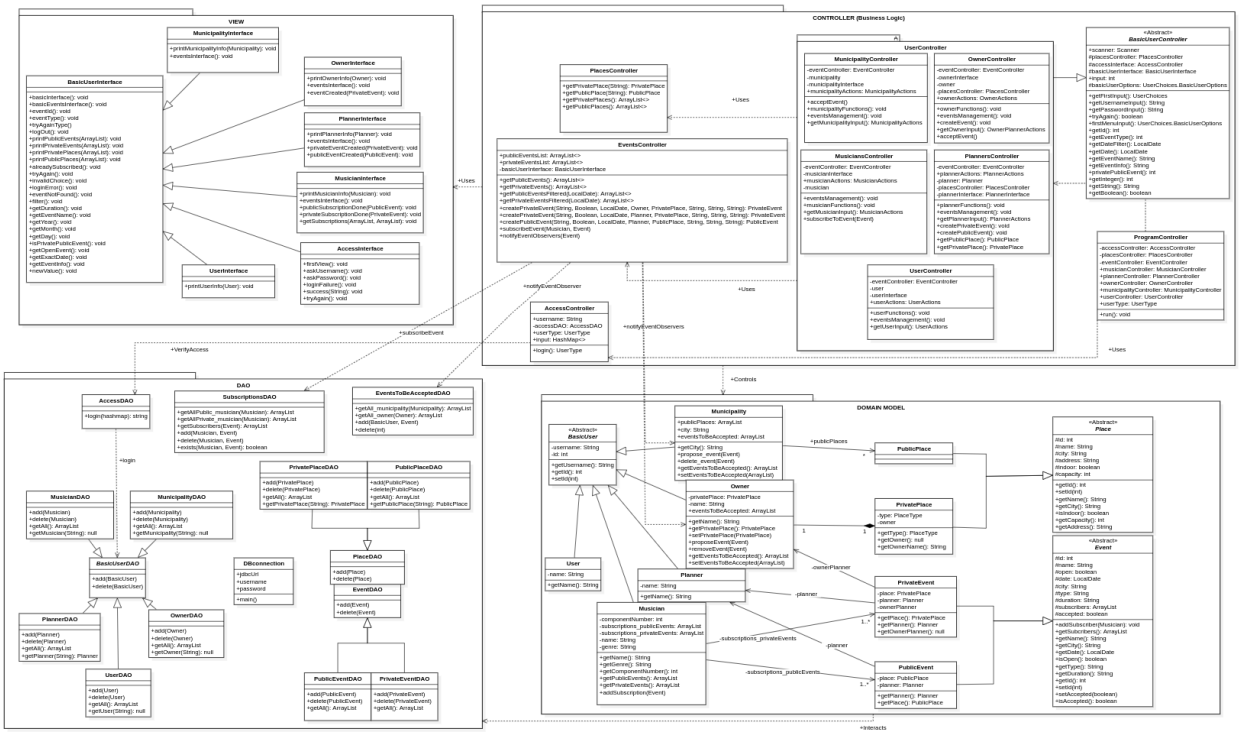

Figura 10: Class diagram

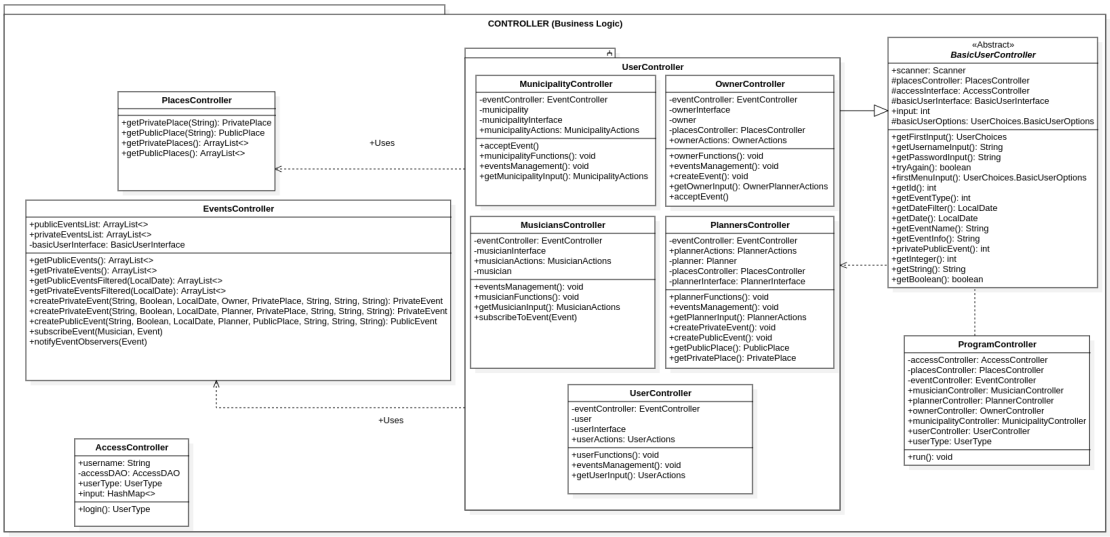

Figura 11: Business Logic diagram

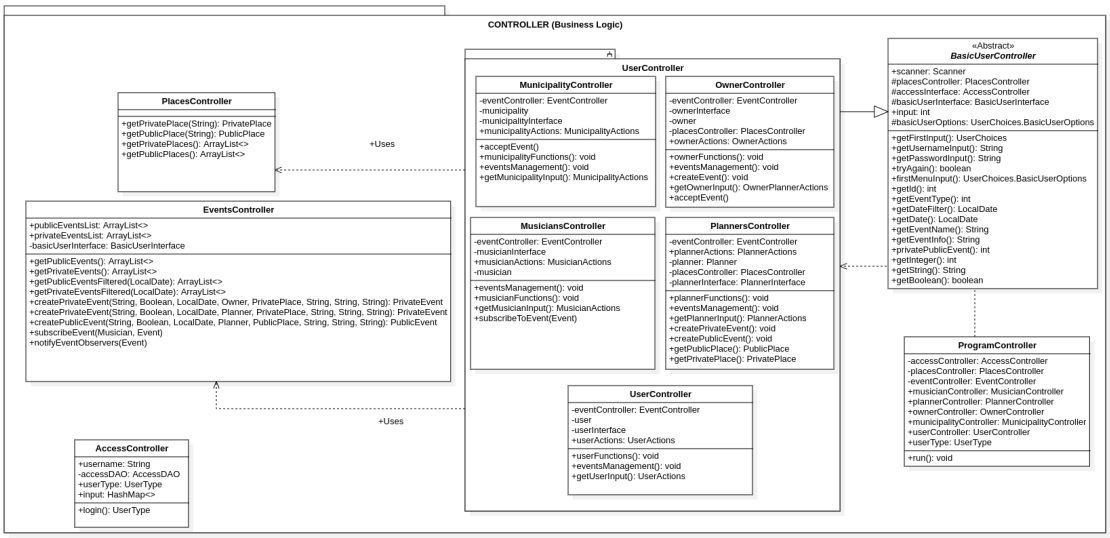

Figura 12: Domain Model diagram

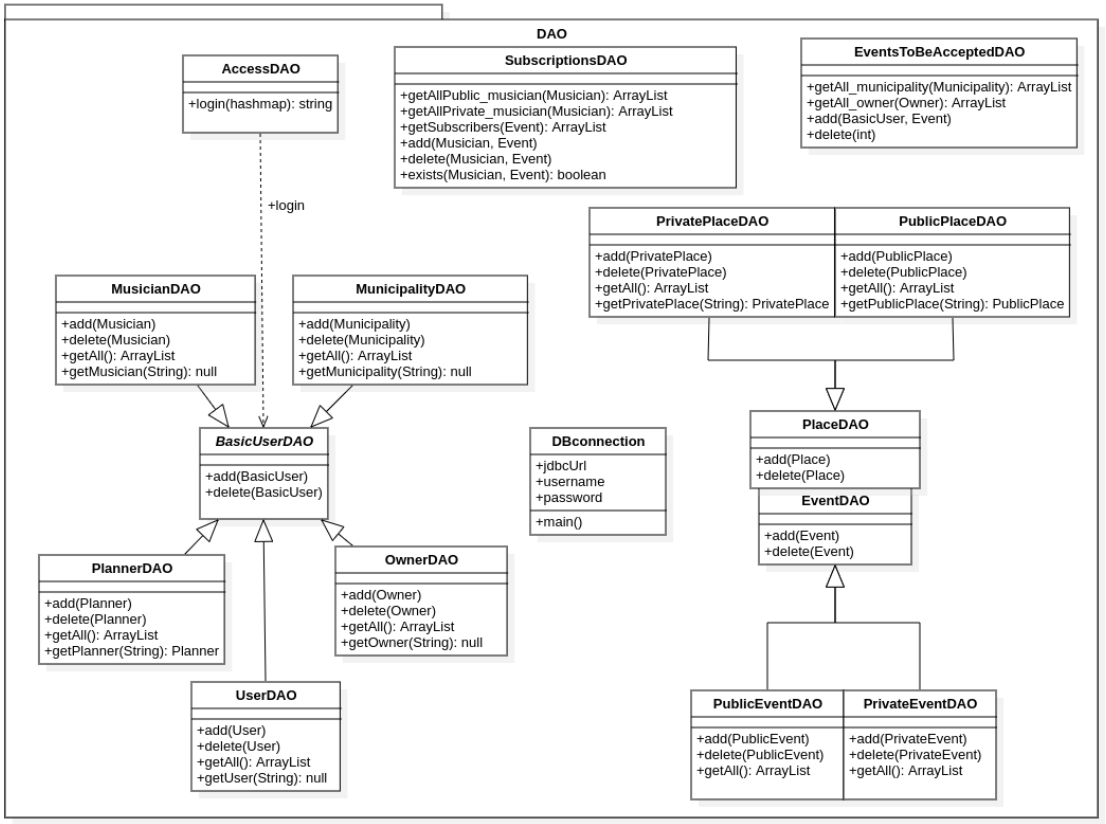

Figura 13: DAO diagram

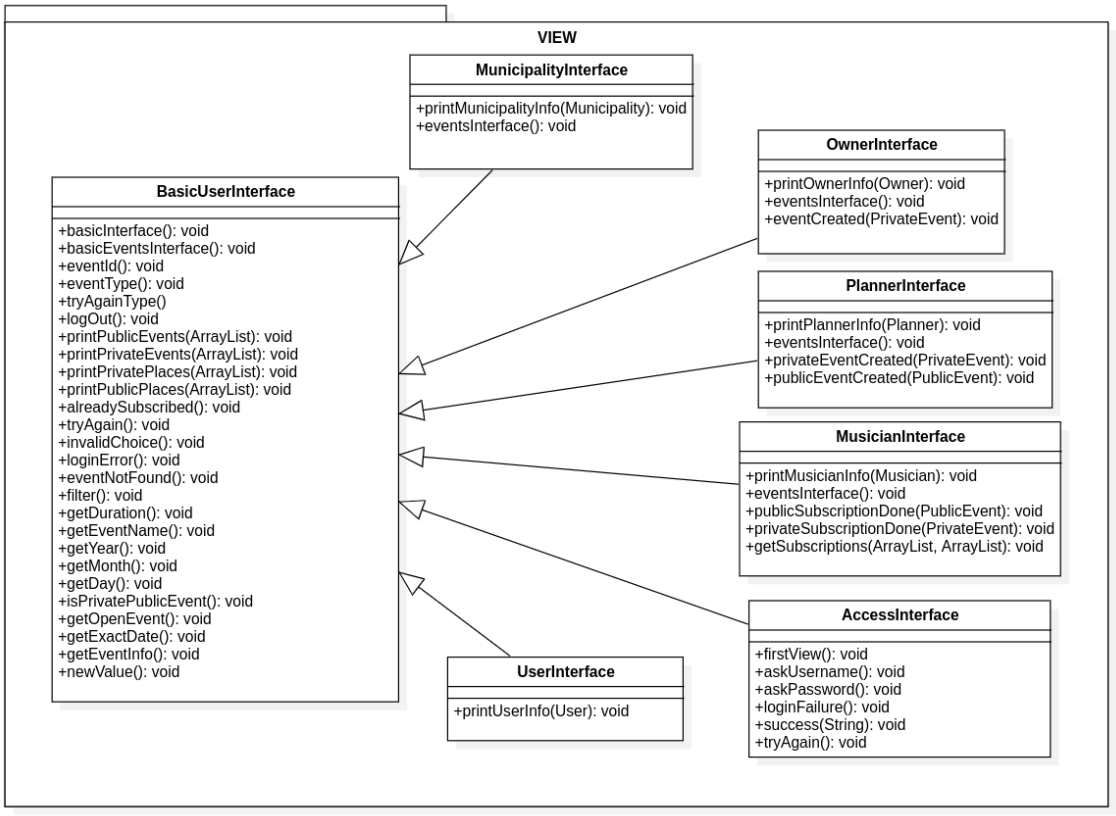

Figura 14: Interface diagram

2.3 Elementi notevoli della progettazione

2.3.1 DAO

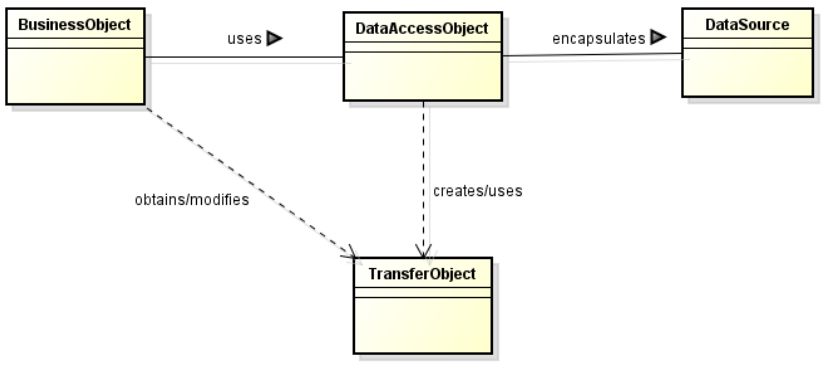

Figura 15: DAO pattern

DAO (Data Access Object) è un design pattern architetturale utilizzato per separare la logica di accesso ai dati dalla logica di business all'interno di un'applicazione. Lo scopo principale del pattern DAO è quello di fornire un'interfaccia astratta tra il livello di business e il livello di accesso ai dati, in modo che le operazioni di accesso ai dati possano essere eseguite indipendentemente dal tipo di archiviazione sottostante o dalla tecnologia utilizzata per accedere agli stessi.
Nel nostro caso è presente una classe generica per l'accesso al gestionale (AccessDAO) e una classe specifica per ogni membro (MusicianDAO, OwnerDAO ecc.) e per i locali ed eventi. In questo modo viene suddivisa la responsabilità di acesso ai dati tra le varie classi, ciascuna delle quali si occuperà di formire o modificare i dati richiesti relativi al singolo utente.

2.3.2 Model View Controller

Il design pattern MVC è un pattern architetturale che riguarda la gestione della logica complessiva del progetto. Si compone di tre parti ben distinte, ognuna delle quali ricopre uno specifico ruolo all'interno dell'architettura dell'applicativo.

- Il **Model** rappresenta i dati dell'applicazione, ovvero gli oggetti concreti sui quali vengono effettuate le operazioni. Il modello gestisce i dati, accetta le richieste di aggiornamento dai controller e notifica le modifiche agli osservatori (come la vista) affinché possano essere visualizzate dagli utenti.
- La **View** Si occupa della presentazione dei dati all'utente, mostrando l'interfaccia ed interpretando i dati del modello per renderizzarli in modo appropriato. La vista riceve notifiche dal controller riguardo ai cambiamenti dei dati e si aggiorna di conseguenza per rifletterli.
- Il **Controller** Rappresenta la parte più corposa, quella che gestisce le interazioni degli utenti e le richieste di input, traducendole in azioni da eseguire sul modello o sulla vista. Riceve gli input dall'utente, interagisce con il modello per eseguire le operazioni richieste e aggiorna la vista di conseguenza.

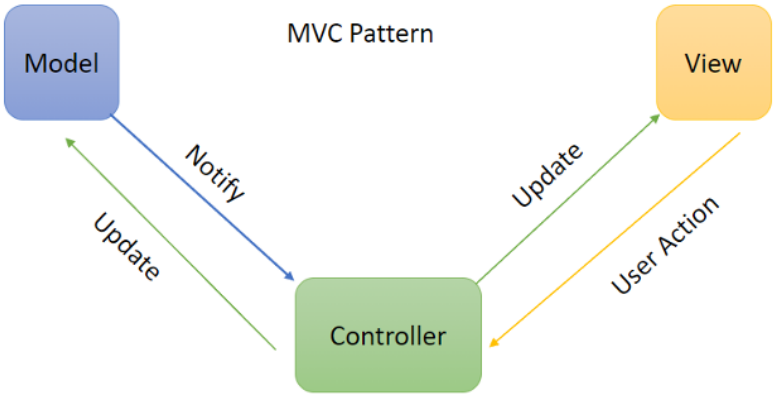

Figura 16: MVC

2.4 Entity-Relationship Diagram

L'Entity Relationship Diagram in figura 17 descrive la struttura e le correlazioni delle tabelle del database dell'applicativo.
Come si evince dal diagramma gli eventi sono il fulcro dell'applicativo ed intrattengono una serie di relazioni uno-a-molti con cisacun membro del programma. Infatti: ogni Musician si sottoiscrive a molti eventi, ogni Planner/Owner crea molti eventi, ogni Municipality accetta/rifiuta molti eventi e ciascun user visualizza molti eventi.
Per quanto riguarda invece i Places questi intrattengono relazioni con gli attori che li possiedono: la Municipality ha una relazione di uno-a-molti, poichè ciascun comune può possedere più aree pubbliche adibite ad aree eventi, mentre l'Owner ha una relazione uno-ad-uno in quanto nell'applicativo è previsto un unico locale per ogni Owner.
Di seguito è presente la figura che rappresenta il modello:

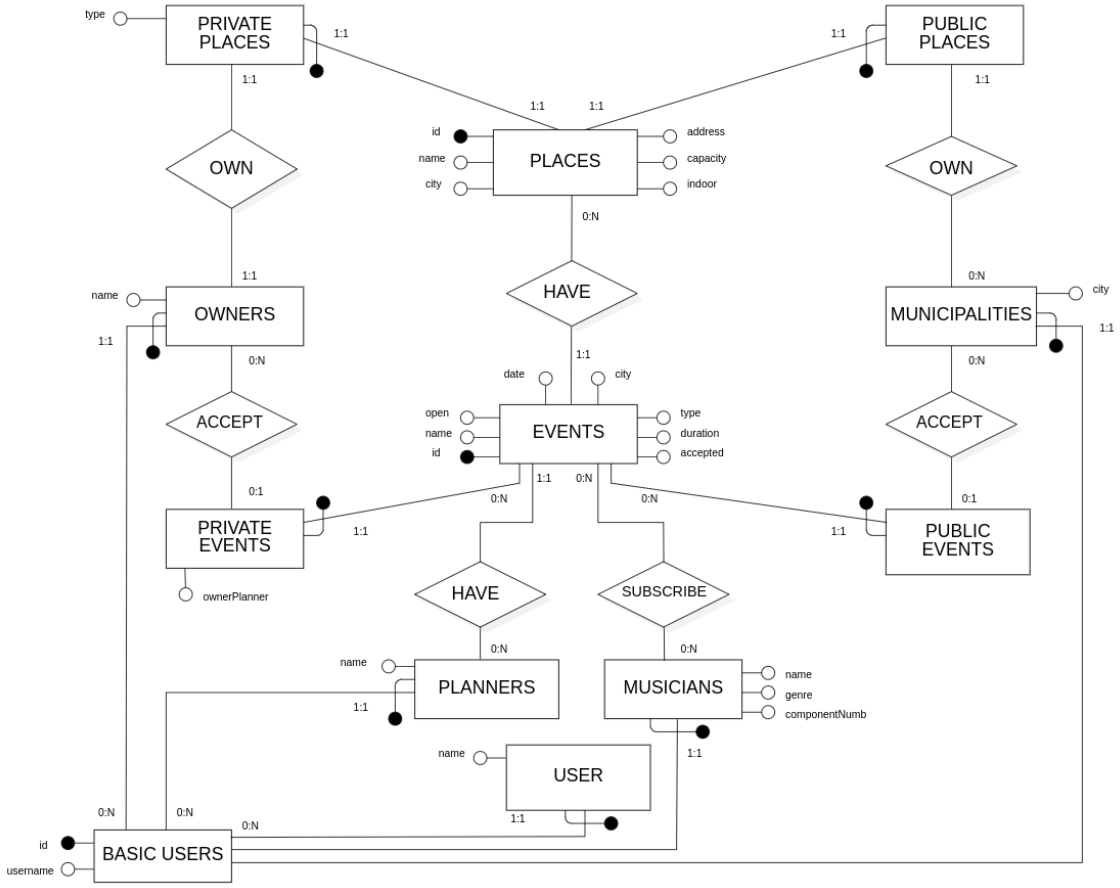

Figura 17: Entity-Relationship Diagram

3 Implementazione delle classi

Come già enunciato in precedenza, il progetto prevede la divisione in vari packages: il modello dati nel package DomainModel, la logica di controllo nel package BusinessLogic, l'interfaccia/visualizzazione nella view ed infine la gestione dell'accesso al database nel package DAO.
I quattro packages appena enunciati sono contenuti nella cartella `src/main`.
Il codice sorgente è articolato nel seguente modo:

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:

3.1.1 Musician, Owner, Planner, Municipality e User

Come è possibile vedere dal diagramma UML, le classi che rappresentano tutti gli utenti dell'applicazione estendono la classe astratta BasicUser, che contiene le informazioni di base che definiscono il profilo di una persona (username ed id) ed i metodi per ottenere tali informazioni (getter e setter). Spetta poi alle implementazioni concrete aggiungere attributi e metodi specifici per ogni tipologia di utente, in modo da rendere chiara la divisione logica tra gli attori. Nello specifico:

- **Musician**: possiede vari attributi aggiuntivi, tra cui le informazioni relative al profilo musicale (componentNumber, genre, name), quelle relative alle sottoiscrizioni effettuate (publicEvents e privateEvents) e tutti i metodi per ottenere tali informazioni.
- **Owner**: aggiunge alla classe base l'attributo del nome e quello relativo al locale che possiede e gestisce (privatePlace).
- **Planner**: estende la classe base con il nome e gli attributi riguardanti gli eventi organizzati ed i conseguenti metodi che ritornano questi valori.

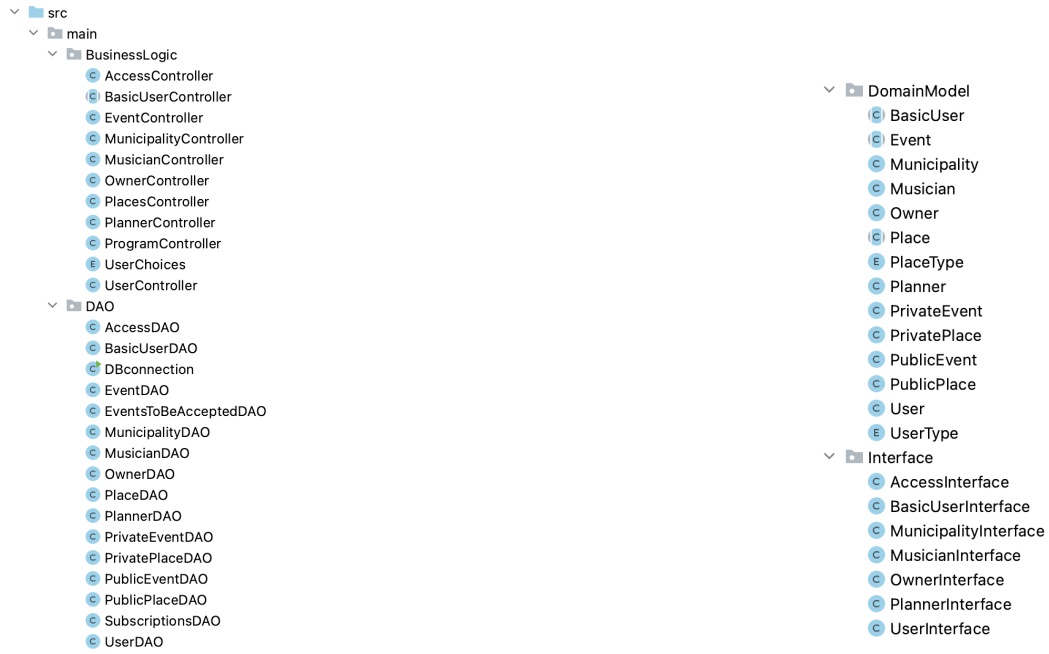

Figura 18: Composizione cartelle e files dell'applicativo

- **Municipality**: aggiunge alla classe base la città rappresentata ed i luoghi pubblici gestiti nei quali è possibile richiedere la realizzazione di un evento.
- **User**: non possiede attributi aggiuntivi, in quanto rappresenta il profilo base necessario alla sola visualizzazione degli eventi presenti sul territorio, dunque non necessita di ulteriori specificazioni.

3.1.2 Event

Rappresenta la classe astratta degli eventi, che mantiene tutti gli attributi comuni alle due tipologie di eventi concreti che possono essere realizzati. Contiene dunque le informazioni generiche dell'evento (date, city, type, duration, name ecc.), l'attributo "open" che distingue la tipologia dell'evento (chiuso o aperto) e le informazioni relative alla lista di candidati (subscriptions) o eventualmente l'id ed il nome del musicista scelto. Le due classi derivate sono:

- **PrivateEvent**: estende la classe base inserendo l'attributo riguardande il luogo privato presso cui si svolgerà l'evento (privatePlace) ed una reference all'organizzatore dell'evento, che può essere sia l'Owner stesso (ownerPlanner) che un qualisasi Planner.
- **PublicEvent**: estende la classe base inserendo l'attributo riguardante il luogo pubblico presso il quale si svolgerà l'evento (publicPlace) ed una reference all'organizzatore, che in questo caso sarà necessariemente un Planner.

Entrambe le classi forniscono i getter necessari all'ottenimento delle informazioni specificate.

3.1.3 Place

Rappresenta la classe astratta dei luoghi nei quali è possibile organizzare un evento e mantiene tutte le caratteristiche comuni alla due tipologie di luoghi concreti. Contiene quindi le informazioni base (city, id, address, indoor ecc.) ed i metodi per recuperare queste informazioni. Le due classi derivate sono:

- **PrivatePlace**: estende la classe base inserendo l'attributo della tipologia del locale (type). Contiene inoltre una reference all'oggetto del proprietario (Owner), necessario al fine di una chiara visualizzazione a schermo della lista completa dei locali e delle operazioni di notifica dell'Owner stesso.
- **PublicEvent**: non estende in alcun modo la classe base, in quanto tutte le informazioni necessarie al suo corretto funzionamento sono già contenute in questa.

Entrambe le classi forniscono i getter necessari all'ottenimento delle informazioni specificate.
I places rappresentano dunque i luoghi fisici necessari alla realizzazione di un evento, infatti ogni volta che un Planner oppure un Owner crea un Evento, l'applicativo richiederà di indicare un locale o area pubblica presso il quale l'evento verrà svolto.

3.2 BusinessLogic

Business Logic è il package che si occupa della gestione dei dati e dunque controlla il flusso dell'applicazione ogni qual volta sia necessario interagire con l'utente oppure svolgere un'operazione. Dispone dei metodi per creare, modificare e eliminare gli elementi del Domain Model. In altre parole la BusinessLogic rappresenta il Controller del design patten MVC.

3.2.1 AccessController

Questa classe è formata dai metodi necessari all'accesso di un qualsiasi tipo di user all'interno dell'applicativo. La presenza di questa classe consente di evitare la creazione di ogni entità presente nel DomainModel, andando a controllare attraverso il DAO la tipologia di User e la sua effettiva sottoiscrizione al programma attraverso la richiesta di Username e Password. Se il Login va a buon fine il controllo del flusso viene passato al controller relativo allo specifico utente.

3.2.2 BasicUserController

È una classe astratta che fornisce i metodi e gli attributi comuni a tutti i controller degli utenti. Tra questi troviamo ad esempio gli attributi ed i metodi designati alla ricezione degli input da parte dell'utente, i metodi che consentono la visualizzazione degli eventi (che avviene con l'ausilio della Interface), ed altri metodi ed attributi vari per la gestione degli errori.
Questa classe viene estesa da ogni controller user-oriented presente all'interno della BusinessLogic.

3.2.3 EventController

Questa classe fornisce metodi ed attributi relativi alla modifica e creazione di eventi sia pubblici che privati.
Si occupa infatti di recuperare le subscription dei musicisti e di farli sottoiscrivere ad un evento. Si occupa inoltre di recuperare la lista degli eventi da stampare a schermo e della creazione di un evento pubblico o privato.
Infine permette di notificare Owner e Municipality nel caso in cui un Planner richieda di effettuare un evento presso i luoghi da loro gestiti.

3.2.4 PlacesController

Questa classe fornisce metodi ed attributi relativi alla gestione dei luoghi sia pubblici che privati presenti sull'applicativo.
È una classe molto semplice in quanto implementa semplicemente i metodi che ritornano una lista di tutti i Places registrati oppure uno specifico Places che viene cercato attraverso il proprio Name.

3.2.5 ProgramController

Questa classe fornisce metodi ed attributi per la gestione del flusso dell'intero applicativo. Attraverso il metodo run chiamato nel file main, viene infatti attivato l'access controller che permette il login dell'utente. Successivamente, all'interno dello stesso metodo viene scelto, attraverso l'utilizzo di uno switch, l'utente a cui passare il controllo del programma. Questo passaggio avviene per mezzo della creazione dello specifico controller e la chiamata al metodo che gestisce le funzioni dello stesso.
Viene mostrata di seguito una parte del codice del sopraenunciato metodo.

```
if(userType != null) {
    switch (userType) {
        case MUSICIAN:
            this.musicianController = new MusicianController(accessController.username, eventController, placesController);
            this.musicianController.musicianFunctions();
            break;
        case PLANNER:
            this.plannerController = new PlannerController(accessController.username, eventController, placesController);
            this.plannerController.plannerFunctions();
            break;
        case MUNICIPALITY:
            this.municipalityController = new MunicipalityController(accessController.username, eventController, placesController);
            this.municipalityController.municipalityFunctions();
            break;
        case USER:
            this.userController = new UserController(accessController.username, eventController, placesController);
            this.userController.userFunctions();
            break;
        case OWNER:
            this.ownerController = new OwnerController(accessController.username, eventController, placesController);
            this.ownerController.ownerFunctions();

    }
}
```

Figura 19: ProgramController

3.2.6 User Oriented Controller

Di seguito verrà enunciata una lista contenente tutti i controller legati alla gestione delle azioni di ogni singlo utente dell'applicazione. Infatti nonostante queste classi siano divise, poichè contengono tipologie di azioni differenti, sono accomunate da due metodi chiamati **"Tipologia Utente"+Functions** ed **eventsManagement**. Entrambi i metodi presentano una funzione switch per il controllo delle scelte utente, la prima relativamente al menù principale, la seconda relativamente al menù di visualizzazione e gestione degli eventi.
Oltre a questi metodi, le classi relative ai controller user oriented implementano quelli relativi agli specifici input di ogni utente. Questi prendono infatti l'input dell'utente e lo confrontano con le scelte possibili per quello specifico utente. Di seguito è presente una lista con le specifiche di ogni controller.

- **MunicipalityController**: Classe che fornisce i metodi per l'esecuzione delle azioni disponibili agli utenti di tipo Municipality. Contiene un metodo per accettare/rifiutare gli eventi proposti dai planner sul suolo pubblico messo a disposizione.
- **MusicianController**: Classe che fornisce i metodi e gli attributi collegati alle azioni eseguibili da un utente di tipo Musician. All'interno di questa, ha una grande rilevanza la funzione di eventsManagement in quanto consente al musicista la possibilità di candidarsi ad un evento aperto, attraverso l'utilizzo dell'EventController e l'aggiunta dell'evento alla lista delle sottoscrizioni presente nella classe Musician del DomainModel.
- **OwnerController**: Classe che fornisce i metodi per l'esecuzione delle azioni disponibili agli utenti di tipo Owner. Implementa vari metodi tra cui quello di selectCandidate e selectMusician, utili per la scelta (tra le varie subscriptions) del musicista che si esibirà ad un evento organizzato in precedenza dall'Owner stesso. Implementa anche il metodo createEvent, che attraverso l'utilizzo dell'eventController permette la creazione di un evento.
- **PlannerController**: Classe che fornisce i metodi per l'esecuzione delle azioni disponibili agli utenti di tipo Planner. Anche in questo caso, come in quello dell'ownerController, vengono implementati i metodi selectCandidate e selectMusician per la selezione di un musicista tra quelli candidati ed il metodo createEvent, che in questo caso è specifico a seconda che l'evento creato sia privato o pubblico.
- **UserController**: Classe che fornisce i metodi per l'esecuzione delle azioni disponibili agli utenti di tipo User. In questo caso, oltre alle specifiche funzioni elencate all'inizio della sezione non vi sono aggiunte.

```java
public void eventsManagement() throws SQLException {
    boolean quitMenu = false;
    while(!quitMenu) {
        musicianInterface.eventsInterface();
        musicianActions = getMusicianInput();
        switch (musicianActions) {
            case SeeAllEvents:
                musicianInterface.printPrivateEvents(eventController.getPrivateEvents());
                musicianInterface.printPublicEvents(eventController.getPublicEvents());
                break;
            case SeeEventsSubscriptions:
                musicianInterface.getSubscriptions(musician.getPublicEvents(), musician.getPrivateEvents());
                break;
            case SubscribeEvent:
                subscribeToEvent();
                break;
            case FilterEvents:
                LocalDate date = getDateFilter();
                musicianInterface.printPublicEvents(eventController.getPublicEventsFiltered(date));
                musicianInterface.printPrivateEvents(eventController.getPrivateEventsFiltered(date));
                break;
            case Exit:
                quitMenu = true;
                break;
        }
    }
}
```

Figura 20: Events Management menu

```java
public void plannerFunctions() throws SQLException {
    while (!Objects.equals(basicUserOptions, UserChoices.BasicUser.Exit)) {
        plannerInterface.basicInterface();
        basicUserOptions = firstMenuInput();
        if(planner == null) {
            plannerInterface.loginError();
            basicUserOptions = UserChoices.BasicUser.Exit;
        }
        switch (basicUserOptions) {
            case SeeEventsMenu:
                eventsManagement();
                break;
            case Exit:
                plannerInterface.logOut();
                break;
            case SeeInfo:
                plannerInterface.printPlannerInfo(planner);
                break;
            case SeeAllPlaces:
                plannerInterface.printPrivatePlaces(placesController.getPrivatePlaces());
                plannerInterface.printPublicPlaces(placesController.getPublicPlaces());
                break;
        }
    }
}
```

Figura 21: Planner Functions

3.2.7 UserChoices

All'interno della BusinessLogic è presente anche questo file che rappresenta una enumerazione di tutte le possibili scelte effettuate da ogni singolo utente. Infatti il file è suddiviso in una pima parte, comune atutti gli utenti nella quale è possibile scegliere tra le opzioni LOGIN ed EXIT, ed una seconda parte che implementa tutte le enumerazioni con le azioni dei singoli utenti. Unica eccezione, in cui le scelte sono infatti condivise, è rappresentata da OwnerPlannerActions, enumerazione che contiene le possibili azioni relative all'Owner ed al Planner, in quanto queste sono esattamente le stesse fatta eccezione per la approvazione di un evento nel proprio locale.

3.3 DAO

Questo package si occupa di gestire le connessioni e le operazioni sul database, implementando il pattern DAO, tramite l'utilizzo di un database PostgreSQL e l'utilizzo della libreria JDBC.

3.3.1 AccessDAO

Rappresenta l'istanza utilizzata per l'accesso all'applicazione da parte degli utenti. Questa classe infatti è orientata alla verifica delle credenziali di accesso di ogni tipologia di utente ed all'identificazione del ruolo dell'utente all'interno del gestionale.
Nel nostro caso sono stati implementati solamente 5 utenti, uno per tipologia, in modo da verificare il funzionamento dell'applicativo per ogni possibilità di utilizzo. Tuttavia, con una opportuna e semplice implementazione è possibile estendere il suo funzionamento ad una molteplicità di utenti.

3.3.2 Object Oriented DAO

Con Object Oriented DAO si intende quella parte di classi relative al DAO che riguardano l'implementazione dell'EventDAO e del PlaceDAO. Queste classi rappresentano la comunicazione tra l'applicativo ed il database per la generazione ed aggiornamento degli Event e dei Place registrati all'interno del database.
Questa sezione è formata da due classi che mantengono i metodi per Event Place privati e ulteriori due classi che mantengono invece i metodi di Event e Place pubblici. La differenziazione tra pubblico e privato, enunciata in precedenza, è data dalla necessità di rappresentare attraverso attributi differenti le due tipologie di eventi e luoghi.

- **PrivateEventDAO e PublicEventDAO**: Questa classe contiene i metodi per aggiungere (add), eliminare (delete) e ritornare (getAll) gli eventi privati. Ogni meotodo implementa una connessione con il database e svolge le necessarie operazioni utili a completare le azioni enunciate. Nonostante i metodi portino a termine le medesime operazioni per entrambe le tipologie di evento è l'implementazione che differisce per via degli attributi registrati in ognuna delle due.

- **PrivatePlaceDAO e PrivateEventDAO**: Anche in questo caso le classi implementano le azioni di aggiunta (add), rimozione (delete) e ritorno (getAll) relative ai Places, aggiungono inoltre la funzione di ritorno di uno specifico Place (getPublicPlace e getPrivatePlace), che attrverso un parametro dato dalla stringa del nome recupera la relativa entità dal database. Come nel precedente caso è l'implementazione data degli attributi che differisce, mentre il risultato è il medesimo per le due tipologie di metodi.

3.3.3 User Oriented DAO

Con User Oriented DAO si intende quella parte di classi relative al DAO che riguardano l'implementazione delle connessioni tra il codice e la varie tabelle degli utenti presenti nel database.
Queste classi sono **UserDAO**, **MusicianDAO**, **MunicipalityDAO**, **OwnerDAO** e **PlannerDAO**. Ciascuna implementa i metodi di aggiunta (add), rimozione (delete) ritorno (getAll) e ritorno di uno specifico elemento.
Questi metodi vengono utilizzati varie volte all'interno del codice del gestionale, in quanto risultano cruciali per il funzionamento di ogni funzionalità. Ad esempio, se volessimo stampare a schermo la lista dei musicisti che si sono candidati per un particolare evento, oppure la lista dei Places con i relativi proprietari e molto altro ancora.

3.4 Struttura del Database

Per la gestione dei dati tramite database è stata scelta la realizzazione di un database tramite PostgreSQL il cui schema, riportato nel file `MusicInTownDB.sql`, è riassunto in figura 22.

1. table basicUsers(PK id, username)
    - table municipalities(PK id FK(basicUsers.id), city)
    - table musicians(PK id FK(basicUsers.id), name, genre, componentNumb)
    - table owners(PK id FK(basicUsers.id), place FK(places.name), name)
    - table planners(PK id FK(basicUsers.id), name)
    - table users(PK id FK(basicUsers.id), name)
2. table events(PK id, name, open, date, city, type, duration, accepted)
    - table privateEvents(PK id FK(events.id), place, planner, ownerPlanner)
    - table publicEvents(PK id FK(events.id), place, planner)
3. table place(PK id, name, city, address, capacity, indoor)
    - table privatePlace(PK id FK(places.id), owner FK(owners.name), type)
    - table publicPlace(PK id FK(places.id))
4. table eventsToBeAccepted(id_controller FK(basicUsers.id), id_event FK(events.id))
5. table subscriptions(id_subscriber FK(musicians.id), id_event FK(events.id))

Figura 22: Schema del database

4 Test

I test sono stati realizzati con l'obbiettivo di testare tutte le funzionalità dell'applicativo, ponendo particolare attenzione all'esecuzione corretta di tutti i metodi e al riscontro di errori in presenza di dati non idonei all'esecuzione dei metodi.
Di seguito si riporta una panoramica generale dei vari test effettuati, contenuti nella cartella `src/test`, e gli esiti positivi di tutti i test effettuati.

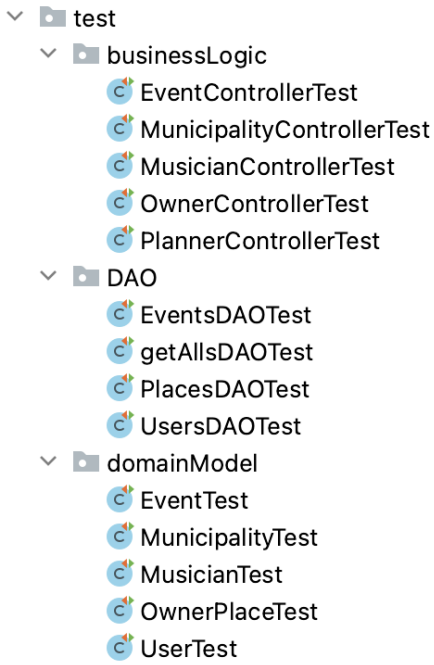

Figura 23: Panoramica della cartella dei Test

4.1 domainModel

Nella cartella `src/test/domainModel` sono situati i test per le classi che rappresentano le varie istanze degli utenti, degli eventi e dei luoghi. Questi riguardano la verifica della corretta creazione di ciascuna delle precedenti istanze, e il corretto funzionamento di metodi specifici ad ognuna. Figura 24

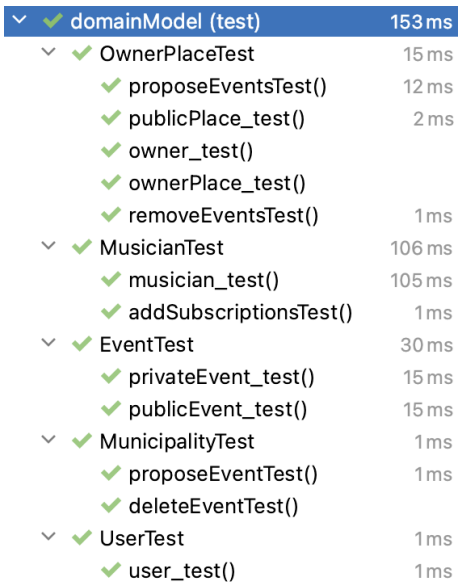

Figura 24: Panoramica della cartella dei Test

4.2 BusinessLogic

I test contenuti nella cartella `src/test/businessLogic` rappresentano dei test funzionali perché vanno a testare le operazioni di gestione dei vari elementi del domain model da parte della business logic. Figura 25
Si osservano i test degli Use Cases nr 2, 3, 7, 8

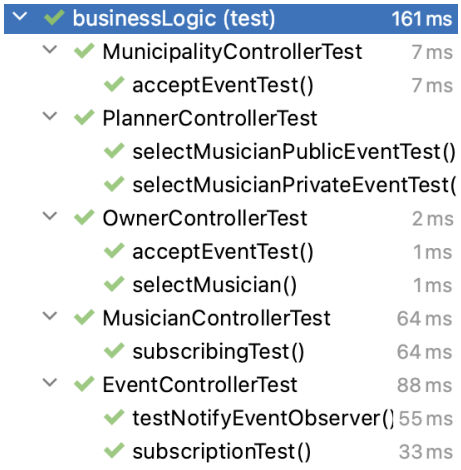

Figura 25: Panoramica della cartella dei Test

4.3 DAO

In questa sezione troviamo i test di tipo strutturale volti a verficare la corretta interazione con il database. In particolare essi sono divisi in quattro categorie: i test sui DAO degli Events, dei Places, di tutti gli Users e dei getAlls.
Come si può vedere nella figura 26 i test mirano alla corretta interazione con il database, attraverso la verifica diretta, nonché attraverso l'uso di query ad hoc. Un altro esempio è la figura 27, dove è possibile osservare la verifica dell'avvenuta sottoscrizione di un musicista a un evento, dalla prospettiva del Musician. In EventsDAOTest è possibile trovare un test simile, che adopera la prospettiva dell'Event. Figura 28

```java
@Test
public void addPublicEventTest() throws Exception {
    // Use DAO to add a Public Event
    PublicEventDAO.add(publicEvent);

    // Use a query to check the data inside the DB
    PreparedStatement getEvent = conn.prepareStatement( sql: "select * from publicevents_intero where publicevent_name =\'fantasia\'");
    ResultSet rs = getEvent.executeQuery();
    rs.next();

    // Verify that the data corresponds.
    Assertions.assertEquals( expected: "fantasia", rs.getString( columnLabel: "publicevent_name"));
    Assertions.assertEquals( expected: "aerosol", rs.getString( columnLabel: "publicevent_place"));
    Assertions.assertEquals( expected: "homo", rs.getString( columnLabel: "publicevent_planner"));
    Assertions.assertTrue(rs.getBoolean( columnLabel: "publicevent_open"));
    Assertions.assertEquals( expected: "2022-02-02", rs.getString( columnLabel: "publicevent_date"));
    Assertions.assertEquals( expected: "firenze", rs.getString( columnLabel: "publicevent_city"));
    Assertions.assertEquals( expected: "comico", rs.getString( columnLabel: "publicevent_type"));
    Assertions.assertEquals( expected: "1", rs.getString( columnLabel: "publicevent_duration"));
    Assertions.assertFalse(rs.getBoolean( columnLabel: "publicevent_accepted"));
}
```

Figura 26: test aggiunta al DB di un Musician

```java
@Test
public void subscribingTest() throws Exception{
    int subs_musician = m.getPublicEvents().size();
    ec.subscribeEvent(m, e);

    Assertions.assertEquals( expected: subs_musician+1, m.getPublicEvents().size());
}
```

Figura 27: test candidatura ad un evento

5 Conclusioni

Il gestionale, nonostante l'assenza di un'interfaccia utente ed il possibile ampliamento di alcune funzionalità, funziona correttamente ed è utilizzabile per una simulazione di ogni categoria di utente. Infatti eseguendo il run del programma su un qualsiasi IDE è possibile verificare che businessLogic, DomainModel, DAO e View collaborano correttamente e consentono un funzionamento lineare della piattaforma.
In conclusione possiamo affermare che, con qualche ulteriore implementazione l'applicativo potrebbe risultare uno strumento utile al fine della diffusione di eventi a livello locale, di cui beneficierebbero musicisti, organizzatori, proprietari di locali e fruitori di concerti.

5.1 Possibili aggiunte e miglioramenti

Il gestionale può essere ampliato in diversi modi. Più ci abbiamo lavorato sopra, più idee di nuove features e nuovi miglioramenti nascevano:

1. Aggiungere test per verificare il comportamento delle funzioni in caso di fallimento dell'operazione.
2. Aggiungere DAO update, senza i quali non si può propriamente definire CRUD.
3. Nel database è presente una tabella di municipalities seppur nel nostro progetto ci concentriamo su un unico comune. Con questa struttura, il progetto è facilmente ampliabile a una provincia o a una regione intera.
4. Aggiungere la possibilità di *aggiungere/rimuovere* un Place.
5. Aggiungere la possibilità ai planners di *cancellare* eventi da loro precedentemente proposti.

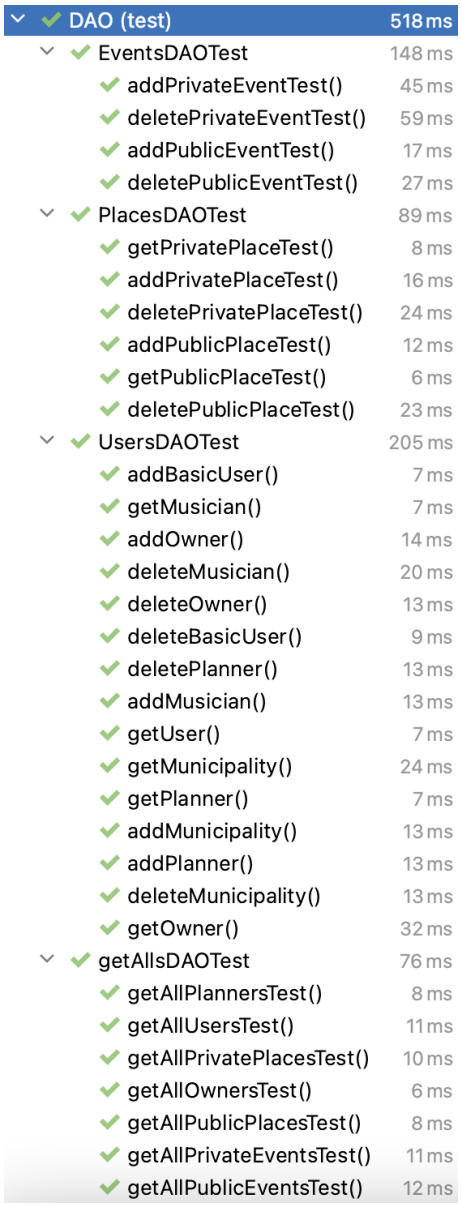

Figura 28: Panoramica della cartella dei Test