

UNIVERSITÀ DEGLI STUDI DI FIRENZE

## Indice

<b>1 Introduzione</b>	<b>2</b>
1.1 Descrizione del Progetto .....	2
1.2 Struttura del Progetto .....	2

# 1 Introduzione

## 1.1 Descrizione del progetto

L'obiettivo del progetto è creare un'applicazione java che permetta di gestire le prenotazioni di alloggi (in questo progetto verranno trattati alcuni tipi di alloggi: Appartamenti, Hotel, B&B). Gli utenti avranno la possibilità di effettuare ricerche e prenotazioni degli alloggi disponibili, selezionare il numero di persone che soggiorneranno, la data di inizio e di fine del soggiorno, e molti altri filtri che verranno spiegate successivamente. Inoltre, l'utente potrà anche cancellare le prenotazioni se necessario, poter inserire tra i preferiti gli alloggi e lasciare delle recensioni degli alloggi da loro prenotati in precedenza. È inoltre presente un Admin che può cancellare definitivamente gli utenti o recensioni, aggiungere nuovi alloggi, modificare gli alloggi già presenti oppure rimuoverli.

## 1.2 Ambiente di sviluppo, Struttura del progetto e pratiche usate

Il progetto è stato sviluppato nel linguaggio Java e il database è stato implementato con PostgreSQL. La connessione tra il progetto e il database è realizzata tramite JDBC. La struttura del progetto è rappresentata dalla figura sottostante:

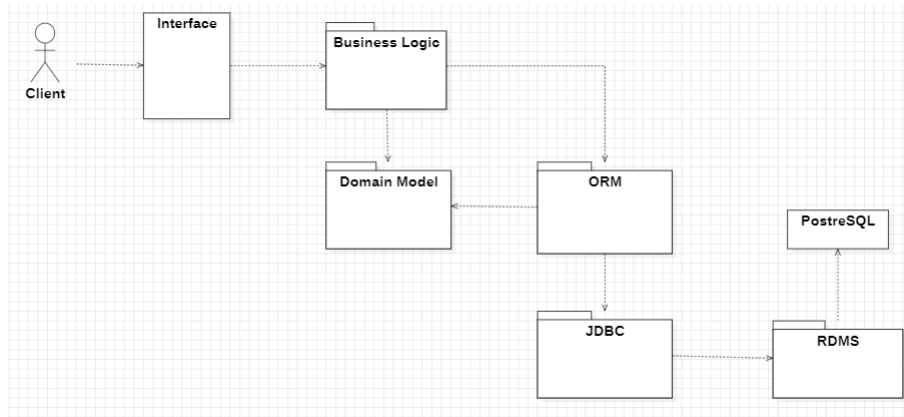


Figura 1: Diagramma della struttura del progetto

La struttura del progetto è stata divisa in tre parti:

- **Business Logic**: contiene le classi che implementano la logica di business nel sistema.
- **Domain Model**: contiene le classi che rappresentano le entità del sistema.

- **DAO:** contiene le classi che permettono di gestire la comunicazione tra l'applicazione e il database andando a separare la logica per l'interazione con i dati dal resto dell'applicazione.

Infine per l'uso del software e l'interazione con il sistema è stata creata un'interfaccia a linea di comando (CLI).

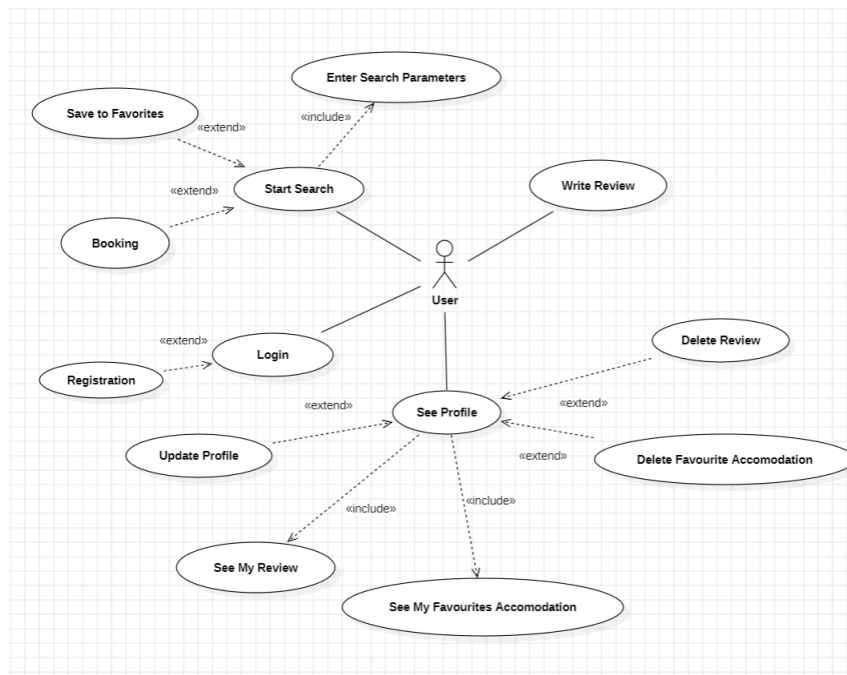
Sono state utilizzate le seguenti piattaforme e software:

- **IntelliJ IDEA:** IDE per lo sviluppo in Java.
- **StarUML:** software per la creazione di diagrammi UML.
- **Draw.io:** software per la realizzazione di altri diagrammi (es: modello ER).
- **PgAdmin:** software per la gestione del database PostgreSQL.
- **GitHub:** piattaforma contenente il codice sorgente.
- **Lunacy:** software per la realizzazione dei mockup.

## 2 Progettazione

### 2.1 Use Case Diagram

In questo progetto sono presenti 2 tipi di utenti: lo User e l'Admin. Nel diagramma sottostante vengono rappresentati i casi d'uso per i due tipi di utenti:



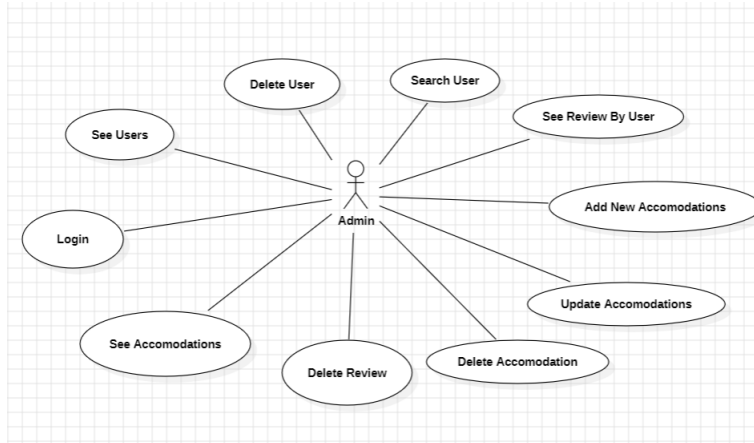


Figura 2: Use Case Diagram dello User e dell'Admin

## 2.2 Use Case Template

Sono di seguito alcuni template dei casi d'uso. In alcuni di essi sono presenti riferimenti a mockup presenti successivamente (sezione 2.3):

Use Case 1	Login
Description	L'Utente accede al sistema inserendo le sue credenziali.
Level	User Goal
Actors	User, Admin
Basic-Flow	<ol style="list-style-type: none"> <li>1. L'Utente inserisce le sue credenziali (email e password)(MK#1 e MK#3).</li> <li>2. L'utente preme il pulsante di Login.</li> <li>3. Il sistema verifica le credenziali.</li> <li>4. Il sistema autentifica l'utente.</li> </ol>
Alternative-Flow	<ol style="list-style-type: none"> <li>3a. Se le credenziali sono incorrette, il sistema invia un messaggio di errore.</li> <li>3b. Se si verifica un problema all'interno del database durante la ricerca dell'utente, il sistema invierà un messaggio di errore.</li> <li>4a. Se l'accesso viene effettuato dall'utente, allora verrà indirizzato nella sua pagina apposita.</li> </ol>
Post-conditions	L'utente è autenticato dal sistema e ha accesso alle sue funzionalità.

Figura 3: Template che descrive il caso d'uso del login

Use Case 2	Start Search
Description	L'utente cerca l'alloggio di suo interesse.
Level	User Goal
Actors	User
Basic Flow	<ol style="list-style-type: none"> <li>1. L'utente inserisce le informazioni per effettuare la sua ricerca (MK#6).</li> <li>2. L'utente preme il pulsante per effettuare la ricerca.</li> <li>3. Il sistema usa le informazioni per ricercare gli alloggi.</li> <li>4. Il sistema restituisce all'utente una lista di alloggi (MK#7).</li> </ol>
Alternative Flow	<ol style="list-style-type: none"> <li>3a. Se l'utente non inserisce alcune informazioni necessarie alla ricerca (es: il luogo dove vuole andare, la data di check-in, la data di check-out), il sistema restituisce un messaggio di errore.</li> <li>3b. Se si verifica un problema all'interno del database durante la ricerca dell'alloggio, il sistema invierà un messaggio di errore.</li> </ol>
Pre-conditions	L'utente deve aver fatto il login.
Post-conditions	L'utente riceverà una lista di alloggi da consultare.

Figura 4: Template che descrive il caso d'uso dello Start Research

Use Case 3	Booking
Description	L'utente prenota un alloggio.
Level	User Goal
Actors	User
Basic-Flow	<ol style="list-style-type: none"> <li>1. L'Utente preme il pulsante per effettuare la prenotazione dell'alloggio(MK#2).</li> <li>2. Il sistema riceve la richiesta di prenotazione e verifica la disponibilità dell'alloggio.</li> <li>3. Il sistema restituisce all'utente un messaggio di conferma.</li> </ol>
Alternative-Flow	<ol style="list-style-type: none"> <li>2a. Se la disponibilità è zero allora il sistema restituisce un messaggio di errore.</li> <li>2b. Se si verifica un problema durante il salvataggio della prenotazione nel database, il sistema invierà un messaggio di errore.</li> </ol>
Pre-conditions	L'utente deve aver fatto il login e deve aver effettuato la ricerca.
Post-conditions	La prenotazione effettuata verrà aggiunta a quelle già effettuate dell'utente.

Figura 5: Template che descrive il caso d'uso del Booking

Use Case 4	Registration
Description	L'utente si registra all'interno del sistema.
Level	User Goal
Actors	User
Basic-Flow	<ol style="list-style-type: none"> <li>1. L'Utente inserisce i parametri per registrarsi(MK#5).</li> <li>2. L'utente preme il pulsante per effettuare la registrazione.</li> <li>3. Il sistema verifica le informazioni fornite.</li> <li>4. Il sistema crea un nuovo account per l'utente.</li> </ol>
Alternative-Flow	<ol style="list-style-type: none"> <li>3a. Se l'utente inserisce dati non validi (es: l'email/username già usati), il sistema invia un messaggio di errore all'utente.</li> <li>3b. Se si verifica un problema durante il salvataggio dell'utente nel database, il sistema invierà un messaggio di errore.</li> </ol>
Post-conditions	L'utente è registrato all'interno del sistema e può accedere tramite le sue credenziali.

Figura 6: Template che descrive il caso d'uso del Registration

Use Case 5	Delete User
Description	L'Admin elimina un utente dal database.
Level	User Goal
Actors	Admin
Basic-Flow	<ol style="list-style-type: none"> <li>1. L'admin inserisce i parametri che caratterizzano l'utente(es:username o email).</li> <li>2. L'admin preme il pulsante per effettuare l'eliminazione dell'utente.</li> <li>3. Il sistema verifica le informazioni fornite.</li> <li>4. Il sistema elimina l'utente dal database.</li> </ol>
Alternative-Flow	<ol style="list-style-type: none"> <li>3a. Se l'admin inserisce dati non validi, il sistema invia un messaggio di errore.</li> <li>3b. Se si verifica un problema durante l'eliminazione dell'utente dal database, il sistema invierà un messaggio di errore.</li> </ol>
Pre-conditions	L'admin deve aver effettuato il login.
Post-conditions	L'utente è stato eliminato con successo, con conseguenza che l'utente in questione non sarà in grado di effettuare l'accesso all'applicazione, a meno che non viene effettuata una nuova registrazione.

Figura 7: Template che descrive il caso d'uso del Delete User

Use Case 6	Add Accommodation
Description	L'admin aggiunge un nuovo alloggio al database.
Level	User Goal
Actors	Admin
Basic Flow	<ol style="list-style-type: none"> <li>1. L'admin inserisce i parametri che caratterizzano l'alloggio.</li> <li>2. L'admin preme il pulsante per effettuare l'aggiunta dell'alloggio.</li> <li>3. Il sistema verifica le informazioni fornite.</li> <li>4. Il sistema registra l'alloggio appena inserito.</li> </ol>
Alternative Flow	3b. Se si verifica un problema durante la registrazione dell'alloggio nel database, il sistema invierà un messaggio di errore.
Pre-conditions	L'admin deve aver fatto il login.
Post-conditions	L'alloggio è stato registrato con successo e sarà disponibile per le successive ricerche degli utenti.

Figura 8: Template che descrive il caso d'uso dell'Add Accomodation

## 2.3 Mockups

Nella sezione in questione sono riportati alcuni mockup relativi ad una possibile interfaccia grafica per l'applicazione:

The mockup shows a mobile-style login interface. At the top left is a back arrow button. The main title is 'Login Admin'. Below it is a text input field containing the word 'ADMIN'. Underneath is a password field with a lock icon and the placeholder text 'Insert Password...'. A prominent blue button labeled 'Enter' is positioned below the password field. At the bottom right, there is a link that says 'Are not an admin? Login'.

Figura 9: Mockup per il login effettuato da un admin - MK#1



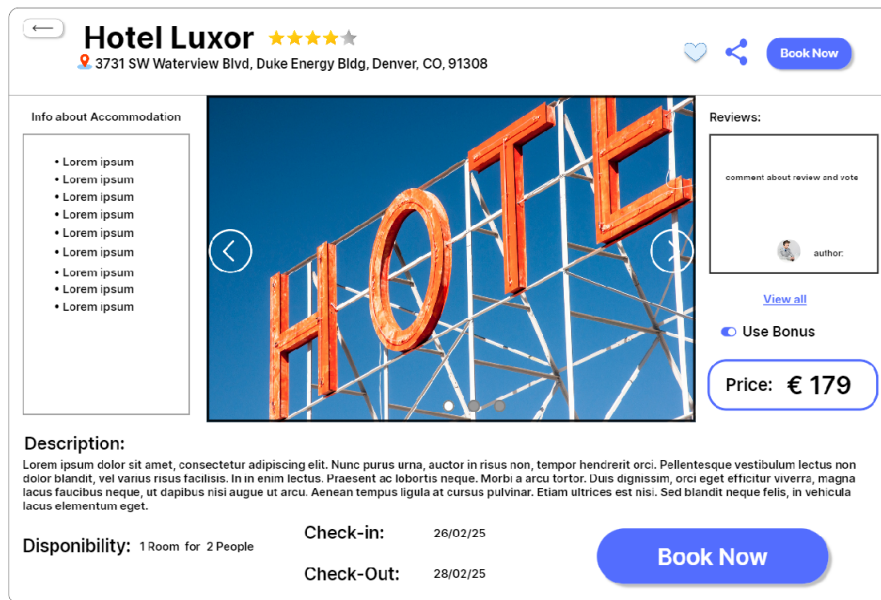


Figura 10: Mockup per mostrare in dettaglio un alloggio con la possibilità di prenotarlo - MK#2

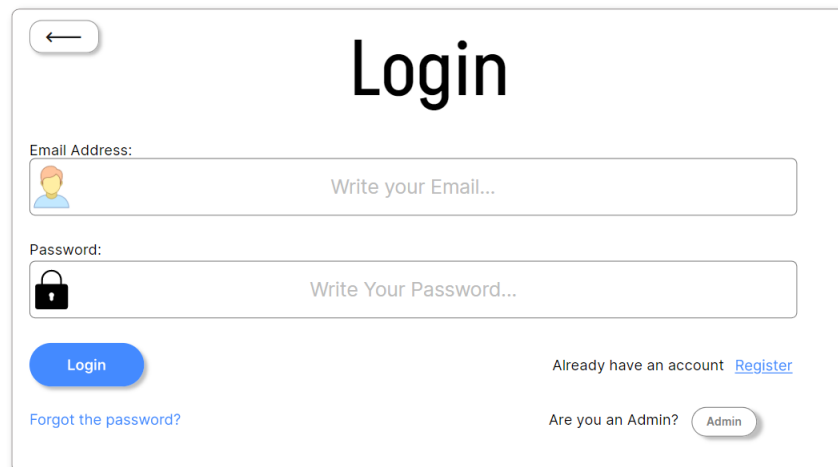



Figura 11: Mockup per il login effettuato da un utente - MK#3

←

Profile

Logout



Name:

Federica

✎

Surname:

Rossi

✎

Email:

FedericaRossi@gmail.com

✎

Password:

Password

✎

Username:

Fede02R


✎


Update


Fidelity points: 120/500

My reviews:

view all →










★ My Savings:

view all →








My Bookings

view all



Details


Delete

Check-in: 30/08/25

Figura 12: Mockup per mostrare il profilo di un utente - MK#4

←

Create an Account



Choose an Avatar

Name:

Name...

Surname:

Surname...

Email:

Email...

Password:

Password...

Username:

Username...

Favourite Location

Locations

▼

1 Sea

2 Mountain

3 Art City

4 Nothing

Create

Delete

Have already an Account?


Login

Figura 13: Mockup per creare un account - MK#5

10

←

Site Name



Username

?

Find your best Accommodation

🏠

Where do you go?...

📅

Data Check-In

Data Check-Out

👤

1 Adult

🛏️

1 Room

Search

⬢

Accurate Research

▼

Category

▼

1 B&B

2 Hotel

3 Apartment

4 Nothing

Refundable

Free Wifi

Smoking Area

Parking Area

Coffe Machine

Room Service

Cleaning Service

Spa

For Kids

Good for Animal

Budget:

Insert your max price...

⬢

Min Star Ratings

⬢

Secific Star Rating

★

★

★


★

★

Figura 14: Mockup per effettuare la ricerca attraverso i filtri - MK#6

←

Site Name



Username

?

Find your best Accommodation

🏠

Where do you go?...

📅

Data Check-In

Data Check-Out

👤

1 Adult

🛏️

1 Room

Search

⬢


Accurate Research

▼

Results:

📌

Save



Hotel Luxor

987 Cedar Tree Way, Raleigh, North Carolina

Description:

lorem ipsum lorem ipsum lorem ipsum....

★

★

★

★

★

13 Reviews

Details

11

Figura 15: Mockup che mostra i risultati di una ricerca - MK#7

## 2.4 Class Diagram

Come spiegato precedentemente, Il progetto è diviso in 3 package:

- **Business Logic**: è il package che contiene i controller, che sono 4: quello che gestisce l'accesso, la registrazione e l'eliminazione degli utenti (**UserController**), quello che gestisce il profilo dell'utente (**ProfileUserController**), quello che gestisce la logica dell'applicazione (ricerca, prenotazione, recensione) (**ResearchController**) e quello che gestisce le azioni che può effettuare l'Admin (**AdminController**).
- **Domain Model**: consiste nell'insieme di classi che rappresentano i concetti con cui interagisce l'applicazione: **RegisterUser**, **Review**, **Booking**, **Accommodation**, **SearchParameters**, **ReviewMapper**, **SearchParametersBuilder**.
- **DAO**: è il package che si occupa di gestire la connessione con il database: **UserDAO**, **BookingDAO**, **PreferenceDAO**, **ReviewDAO**, **AccommodationDAO**, **DatabaseConnection**.



database venisse effettuata una singola volta e per evitare conflitti tra connessioni.

### 2.5.2 Mapper

Lo scopo del Mapper è di semplificare il codice e renderlo più modulare, separando la logica di relazione degli oggetti dal resto dell'applicazione. Nel nostro progetto è stata implementata per creare la relazione tra utenti, recensioni e alloggi.

### 2.5.3 Builder

Il Builder separa la costruzione del suo oggetto dalla sua rappresentazione, rendendo la creazione della classe più flessibile, leggibile e facile da mantenere. Nel nostro progetto viene usato per creare la classe parametri di ricerca essendo che presenta tanti attributi, spesso opzionali, e consente una gestione più efficiente.

### 2.5.4 DAO

Il DAO (Data Access Object) è un design pattern che si occupa di separare le classi che si interfacciano al database dall'applicazione. Questo viene fatto per poter meglio implementare il principio di singola responsabilità e aumentare la manutenibilità del codice.

## 2.6 ER Diagram e Relational Model

Per il database e la sua gestione, è stato realizzato un ER Diagram (Figura 17), e il Relational Model derivato (Figura 18). Ci sono state delle scelte progettuali precise, in particolare quella riguardante l'entità **Alloggio**, dove per differenziare i tipi di alloggio è stato usato un'attributo al posto di una generalizzazione, dovuto al fatto che nel progetto si faranno uso di informazioni che sono a comune tra i vari alloggi, favorendo accessi più veloci ma a discapito di un notevole spreco di memoria e la presenza di valori nulli.

Alla fine sono state definite le seguenti tabelle (Figura 19):

- **User**: rappresenta l'entità **User**.
- **Booking**: rappresenta l'entità **Booking**.
- **Accommodation**: rappresenta l'entità **Accommodation**.
- **Review**: rappresenta la relazione **Review** che avviene tra l'entità **User** e l'entità **Accommodation**.
- **Favourites**: rappresenta la relazione **Like** che avviene tra l'entità **User** e l'entità **Accommodation**.

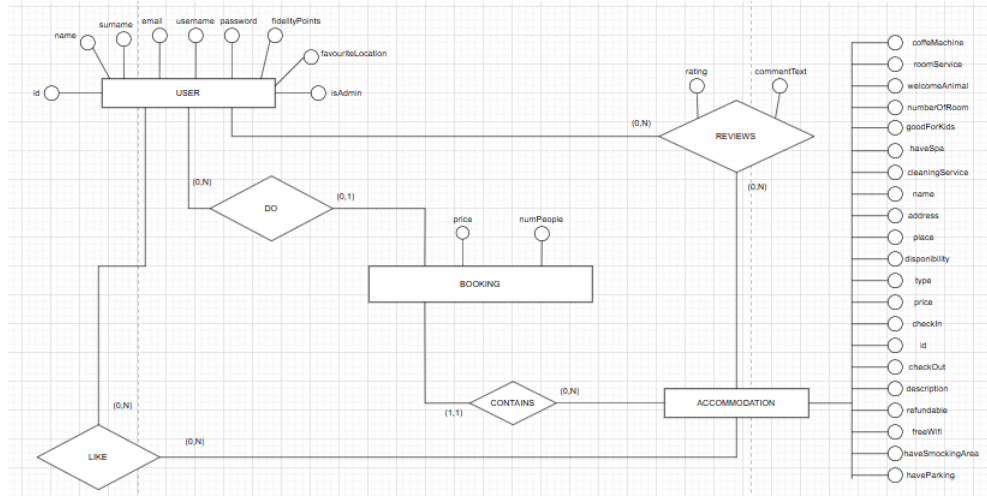


Figura 17: ER Diagram

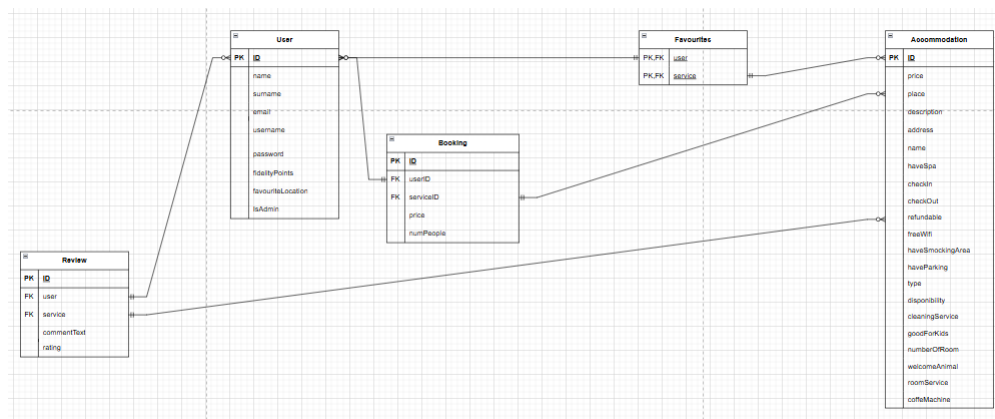


Figura 18: Tables of the database

**User** (userID, name, surname, email, username, password, fidelityPoints, favouriteLocation, isAdmin)

**Accommodation** (accommodationID, price, place, description, address, name, haveSpa, checkIn, checkOut, refundable, freeWifi, haveSmockingArea, haveParking, type, disponibility, cleaningService, goodForKids, numberOFRoom, welcomeAnimal, roomService, coffeMachine)

**Booking** (bookingID, userID, accommodationID, price, numPeople)

**Review** (reviewID, userID, accommodationID, commentText, rating)

**Favourites** (userID, accommodationID)

Figura 19: Relation Model

## 3 Implementazione

### 3.1 Business Logic

È il package che contiene i controller e che espone all'esterno le funzionalità dell'applicazione. È responsabilità dei controller gestire le dipendenze tra gli oggetti creati dai DAO.

#### 3.1.1 UserController

È la classe che si occupa di implementare le funzionalità di un utente generico per l'accesso all'applicazione. Infatti presenta i metodi di login, di registrazione, cancellare l'utente in uso e un eventuale recupera password.

#### 3.1.2 ProfileController

È la classe che si occupa di gestire le informazioni del profilo utente e dei servizi di cui ha usufruito (**deleteReview()**, **deleteBooking()**, **unsaveAccommodation()**, **viewMySavings()**, **viewMyReviews()**, **viewMyBookings()**).

#### 3.1.3 ResearchController

È la classe che si occupa di fornire i metodi che implementano la logica dell'applicazione. Infatti, permette la ricerca in base a dei parametri (**doResearch()**) e sugli alloggi ricercati permette, a un utente registrato, di effettuare una prenotazione (**booking()**), salvarlo tra i preferiti (**saveAccommodation()**) e scrivere una recensione su quell'alloggio (**writeReview()**). Per funzionare, oltre ai collegamenti ai relativi DAO, questa classe usa il **SearchParametersBuilder** che si trova nel **Domain Model**.

#### 3.1.4 AdminController

È la classe che si occupa di implementare le operazioni dell'admin, il quale può leggere, modificare, cancellare e aggiungere alloggi, mentre può solo leggere e cancellare utenti e recensioni.

### 3.2 Domain Model

È il package che rappresenta il modello dei dati e implementa le classi che raffigurano le entità del sistema.



### 3.2.1 RegisterUser

Contiene le informazioni relative all'utente registrato. Gli attributi della classe sono: id (utilizzato come identificativo), username, email (unica all'interno dell'applicazione), password, nome, cognome, punti fedeltà (che si aggiornano ad ogni acquisto e raggiunta una certa soglia, permette di avere degli sconti), località preferita che indica un genere di esperienza che preferisce, la lista delle prenotazioni effettuate e la lista dei suoi alloggi preferiti.

### 3.2.2 Booking

È la classe che rappresenta l'entità prenotazione, con tutte le informazioni relative ad essa. Possiede un id identificativo, l'acquirente, il servizio, per quante persone è la prenotazione e il prezzo.

### 3.2.3 Accommodation

È la classe che rappresenta l'entità alloggio e tiene conto dei suoi attributi. Molti dei suoi attributi non sono obbligatori, ma possono essere nulli perché dipendono dai servizi che offre l'alloggio. Ha un id (univoco), un nome, un indirizzo, un luogo, quante persone possono alloggiarci, tipo di alloggio (B&B, appartamento e hotel), il prezzo, il check-in, il check-out, la descrizione di cosa offre, e diversi parametri aggiuntivi che non sono obbligatori (visualizzabili nelle figure precedenti).

### 3.2.4 Review

È la classe che rappresenta l'entità recensione. I campi sono id, autore, alloggio recensito, commento e il voto.

### 3.2.5 ReviewMapper

È una classe creata per mappare i dati della recensione, cioè per relazionare un utente registrato e un alloggio.

### 3.2.6 SearchParametersBuilder e SearchParameters

Queste classi implementano il design pattern Builder per la creazione dei parametri di ricerca. Il **SearchParametersBuilder** consente una creazione più facile da gestire e da estendere dei parametri di ricerca. Infatti il suo unico scopo è di creare la classe **SearchParameters** (devono possedere gli stessi attributi). Quest'ultima classe possiede solo gli attributi che poi serviranno alla ricerca all'interno del database. Gli attributi sono per la maggior parte uguali alla classe **Accommodation** (si possono visualizzare nell'UML visto precedentemente).

### 3.3 DAO

È il package che implementa l'omonimo design pattern descritto nella sezione 2.5.4. Le classi in questo package permettono alle classi presenti nella **Business Logic** di accedere ai vari dati di loro interesse.

#### 3.3.1 DatabaseConnection

È la classe che si occupa di gestire la connessione al database per le altre classi DAO tramite il metodo **getConnection()**. Classe implementata usando il design pattern *Singleton* per evitare conflitti tra connessioni.

#### 3.3.2 UserDAO

È la classe che si occupa della gestione dei dati degli utenti. Questa classe contiene molti metodi, offrendo la possibilità di aggiungere nuovi utenti e di rimuovere quelli già presenti nel database (rispettivamente **addUser()** e **removeUser()**), la possibilità di recuperare un utente tramite l'id identificativo (**getUserById()**) oppure tramite lo username (**getUserByUsername()**) o anche in altri modi. Infine la classe presenta i metodi che permettono di aggiornare i dati personali di un utente.

#### 3.3.3 BookingDAO

È la classe che si occupa della gestione dei dati che riguardano le prenotazioni effettuate dagli utenti. La classe mette a disposizione metodi che permettono di aggiungere o rimuovere delle prenotazioni (rispettivamente **addBooking()** e **removeBooking()**), di visualizzare le prenotazioni fatte da uno specifico utente (**getBookingFromUser()**) o vederle tutte (quest'ultima utilizzata da un admin per effettuare dei controlli e modifiche se fosse necessario).

#### 3.3.4 PreferenceDAO

È la classe che si occupa della gestione dei dati che riguardano le liste di alloggi preferiti dagli utenti, i quali posso essere visionati senza dover fare una nuova ricerca. Questa classe contiene i metodi che permettono di aggiungere un nuovo alloggio tra i preferiti (**addPreference()**), di rimuovere un alloggio tra i preferiti (**removePreference()**) e di visualizzare gli alloggi preferiti di uno specifico utente (**getPreferenceByUser()**).

#### 3.3.5 ReviewDAO

È la classe che si occupa della gestione delle recensioni scritte sugli alloggi da parte degli utenti. La classe contiene i metodi che permettono di aggiungere nuove recensioni (**addReview()**), di rimuovere le recensioni dall'applicazione (**removeReview()**), e di visualizzare le recensioni scritte da uno specifico utente (**getReviewByUser()**) o visualizzare le tutte le recensioni scritte su uno specifico alloggio (**getReviewByAccommodation()**).

### 3.3.6 AccommodationDAO

È la classe che si occupa della gestione dei dati degli alloggi. questa classe presenta molti metodi, permettendo di aggiungere nuovi alloggi (**addAccommodation()**), di rimuovere gli alloggi già presenti (**deleteAccommodation()**), di visualizzare tutti gli alloggi (**getAllAccommodation()**), di visualizzare uno nello specifico tramite il suo identificativo (**getAccommodationById()**, questo metodo è molto utile per la gestione degli alloggi da parte dell'admin) e di visualizzare gli alloggi che vengono ricercati tramite l'uso dei filtri (**getAccommodationByParameters()**). Infine sono presenti i metodi che permettono di aggiornare i dati di un alloggio.