



UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ DEGLI STUDI DI FIRENZE
DIPARTIMENTO DI INGEGNERIA INFORMATICA

Apartament

Link Github:
<https://github.com/Pennelli02/SweProject>

Autori:

Pennelli Lorenzo Maria

Leuter Lorenzo

Docente corso:

Vicario Enrico

Indice

1	Introduzione	3
1.1	Descrizione del progetto	3
1.2	Ambiente di sviluppo, Architettura del progetto e pratiche usate	3
2	Progettazione	4
2.1	Use Case Diagram	4
2.2	Use Case Template	5
2.3	Mockups	8
2.4	Class Diagram	12
2.5	Dettagli di Progetto	13
2.5.1	Singleton	14
2.5.2	Mapper	14
2.5.3	Builder Telescoping Constructor	15
2.5.4	DAO	15
2.6	ER Diagram e Relational Model	15
2.6.1	Dettagli implementativi del database	17
3	Implementazione	18
3.1	Business Logic	18
3.1.1	UserController	18
3.1.2	ProfileController	18
3.1.3	ResearchController	19
3.1.4	AdminController	19
3.2	Domain Model	19
3.2.1	RegisterUser	19
3.2.2	Booking	19
3.2.3	Accommodation	19
3.2.4	Review	20
3.2.5	SearchParametersBuilder e SearchParameters	20
3.3	DAO	20
3.3.1	DatabaseConnection	20
3.3.2	UserDAO	20
3.3.3	BookingDAO	20
3.3.4	PreferenceDAO	21
3.3.5	ReviewDAO	21
3.3.6	AccommodationDAO	21
3.4	Interfaccia CLI	21
4	Test	22
4.1	Domain Model Test	22
4.2	Business Logic Test	22
4.3	DAO	24
4.4	Risultati dei test	25

1 Introduzione

1.1 Descrizione del progetto

L'obiettivo è creare un'applicazione java che permetta di gestire le prenotazioni di alloggi, in particolare verranno trattati B&B, appartamenti e hotel. 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, Architettura del progetto e pratiche usate

L'applicazione è stata sviluppata nel linguaggio Java e il database è stato implementato con PostgreSQL. La connessione tra il progetto e il database è realizzata tramite JDBC. L'architettura del progetto è rappresentata dalla figura sottostante:

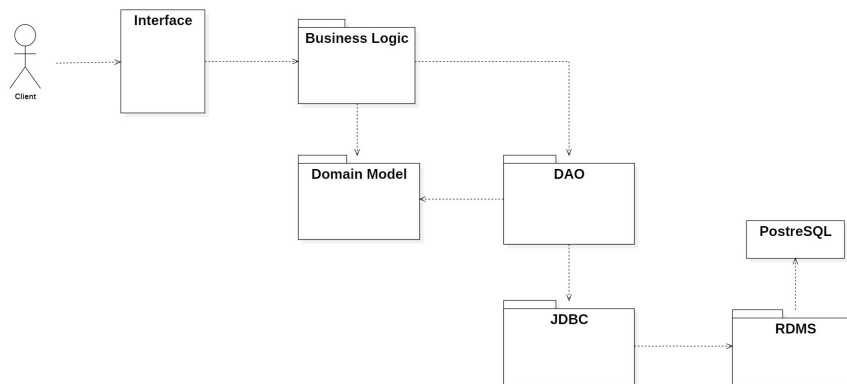


Figura 1: Diagramma dell'architettura del progetto

L'architettura del progetto è articolata in tre componenti:

- **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).

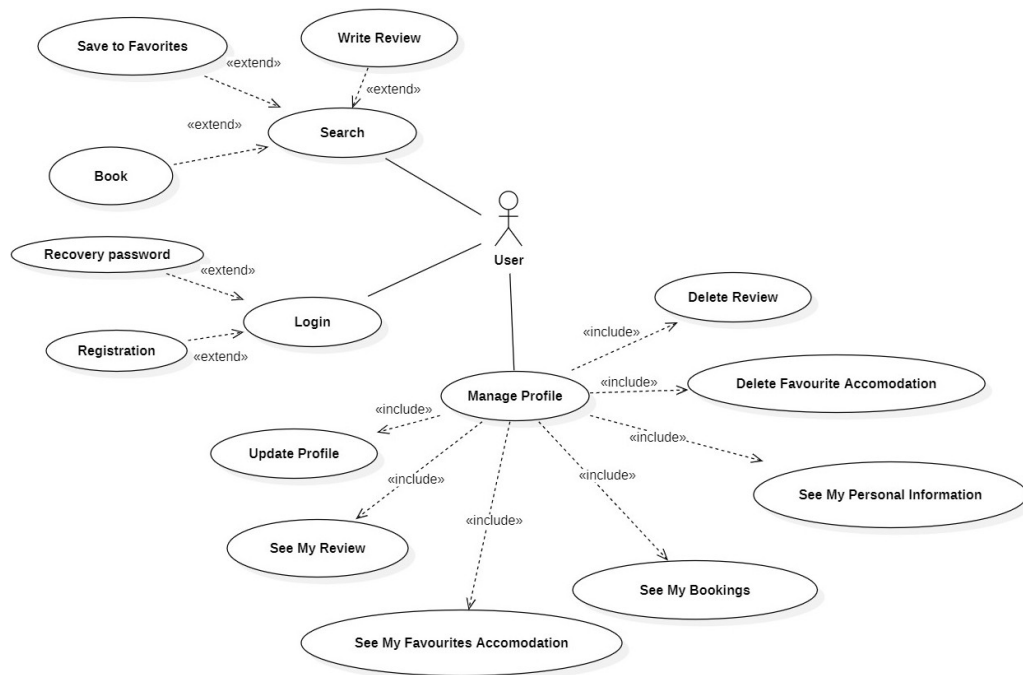
Sono state utilizzate le seguenti piattaforme e software:

- **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, tra cui il 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

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:



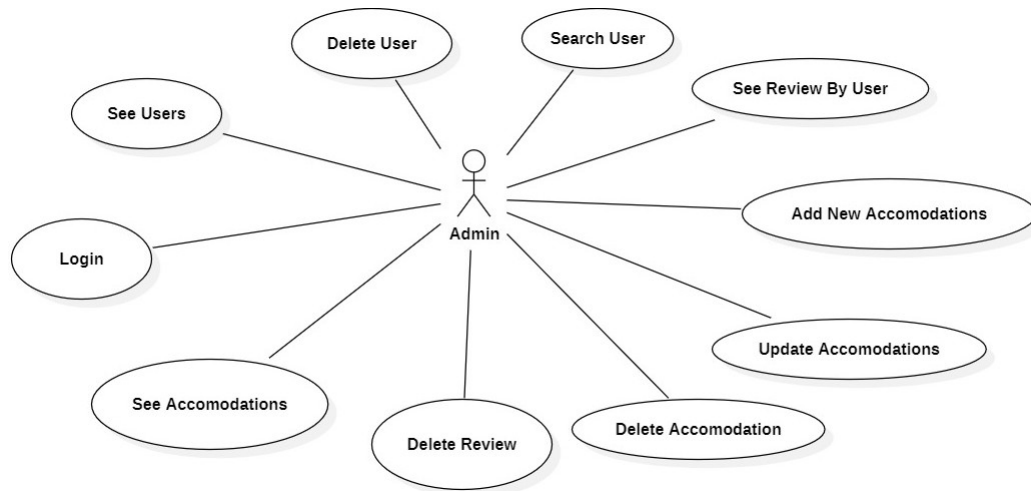


Figura 2: Use Case Diagram dello User e dell'Admin

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
Descrizione	L'utente accede al sistema inserendo le sue credenziali.
Livello	Function
Attori	Utente, Admin
Flusso Base	<ol style="list-style-type: none"> 1. L'utente inserisce le sue credenziali (email e password) (MK#1 e MK#3). 2. L'utente preme il pulsante di Login. 3. Il sistema verifica le credenziali. 4. Il sistema autentica l'utente.
Flusso Alternativo	<ol style="list-style-type: none"> 3a. Se le credenziali sono errate, il sistema invia un messaggio di errore. 3b. Se si verifica un problema all'interno del database durante la ricerca dell'utente, il sistema invia un messaggio di errore. 4a. Se l'accesso viene effettuato dall'utente, sarà indirizzato alla sua pagina personale.
Post-condizioni	L'utente è autenticato dal sistema e ha accesso alle sue funzionalità.

Tabella 1: Template che descrive il caso d'uso del login

Use Case 2	Search
Descrizione	L'utente cerca l'alloggio di suo interesse.
Livello	User Goal
Attori	User
Flusso Base	<ol style="list-style-type: none"> 1. L'utente inserisce le informazioni per effettuare la sua ricerca (MK#6). 2. L'utente preme il pulsante per effettuare la ricerca. 3. Il sistema usa le informazioni per ricercare gli alloggi. 4. Il sistema restituisce all'utente una lista di alloggi (MK#7).
Flusso Alternativo	<ol style="list-style-type: none"> 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. 3b. Se si verifica un problema all'interno del database durante la ricerca dell'alloggio, il sistema invia un messaggio di errore.
Pre-condizioni	L'utente deve aver fatto il login.
Post-condizioni	L'utente riceverà una lista di alloggi da consultare.

Tabella 2: Template che descrive il caso d'uso dello Research

Use Case 3	Book
Descrizione	L'utente prenota un alloggio.
Livello	User Goal
Attori	User
Flusso Base	<ol style="list-style-type: none"> 1. L'utente preme il pulsante per effettuare la prenotazione dell'alloggio (MK#2). 2. Il sistema riceve la richiesta di prenotazione e verifica la disponibilità dell'alloggio. 3. Il sistema restituisce all'utente un messaggio di conferma.
Flusso Alternativo	<ol style="list-style-type: none"> 2a. Se la disponibilità è zero, il sistema restituisce un messaggio di errore. 2b. Se si verifica un problema durante il salvataggio della prenotazione nel database, il sistema invia un messaggio di errore.
Pre-condizioni	L'utente deve aver fatto il login e deve aver effettuato la ricerca.
Post-condizioni	La prenotazione effettuata verrà aggiunta a quelle già effettuate dell'utente.

Tabella 3: Template che descrive il caso d'uso del Book

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

Tabella 4: Template che descrive il caso d'uso del Registration

Use Case 5	Delete User
Descrizione	L'Admin elimina un utente dal database.
Livello	User Goal
Attori	Admin
Flusso Base	<ol style="list-style-type: none"> 1. L'admin inserisce i parametri che caratterizzano l'utente (es: username o email). 2. L'admin preme il pulsante per effettuare l'eliminazione dell'utente. 3. Il sistema verifica le informazioni fornite. 4. Il sistema elimina l'utente dal database.
Flusso Alternativo	<ol style="list-style-type: none"> 3a. Se l'admin inserisce dati non validi, il sistema invia un messaggio di errore. 3b. Se si verifica un problema durante l'eliminazione dell'utente dal database, il sistema invierà un messaggio di errore.
Pre-condizioni	L'admin deve aver effettuato il login.
Post-condizioni	L'utente è stato eliminato con successo e non può più accedere all'applicazione a meno che non venga effettuata una nuova registrazione.

Tabella 5: Template che descrive il caso d'uso del Delete User

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

Tabella 6: Template che descrive il caso d'uso dell'Add Accomodation

2.3 Mockups

Sono riportati alcuni mockup, realizzati con **Lunacy**, relativi ad una possibile interfaccia grafica per l'applicazione:

The mockup shows a login interface for an administrator. At the top left is a back arrow icon. 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 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 3: Mockup per il login effettuato da un admin - MK#1

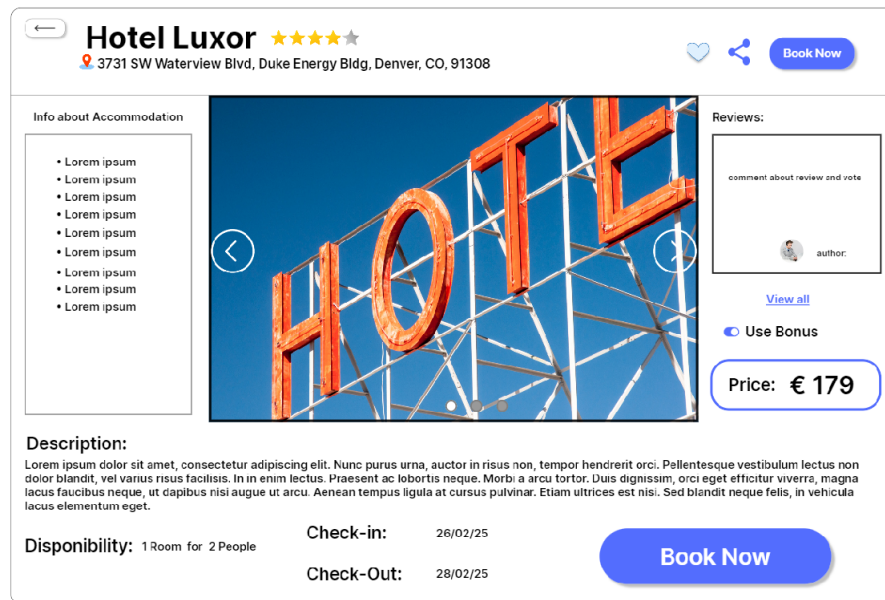


Figura 4: Mockup per mostrare in dettaglio un alloggio con la possibilità di prenotarlo - MK#2

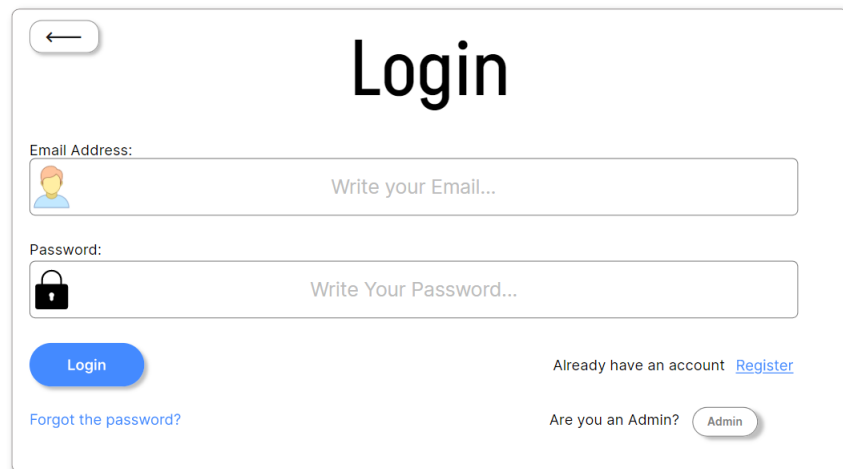



Figura 5: Mockup per il login effettuato da un utente - MK#3

←

Profile

Logout



Name:

Federica

Surname:

Rossi

Email:

FedericaRossi@gmail.com

Password:

Password

Username:


Fede02R

Update


Fidelity points: 120/500

My reviews:


view all →



Delete




Delete




Delete

My Savings:


view all →



Unsave




Unsave



Unsave

My Bookings

view all



Details


Delete

Check-in: 30/08/25

Figura 6: 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 7: 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

Budget:

Insert your max price...

🔵

Min Star Ratings

⚪

Specific Star Rating

★

★

★

★


★

☐ Refundable
 ☐ Free Wifi
 ☐ Smoking Area
 ☐ Parking Area
 ☐ Coffe Machine
 ☐ Room Service
 ☐ Cleaning Service
 ☐ Spa
 ☐ For Kids
 ☐ Good for Animal

Figura 8: 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

Figura 9: Mockup che mostra i risultati di una ricerca - MK#7

11

2.4 Class Diagram

L'architettura è divisa 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**).

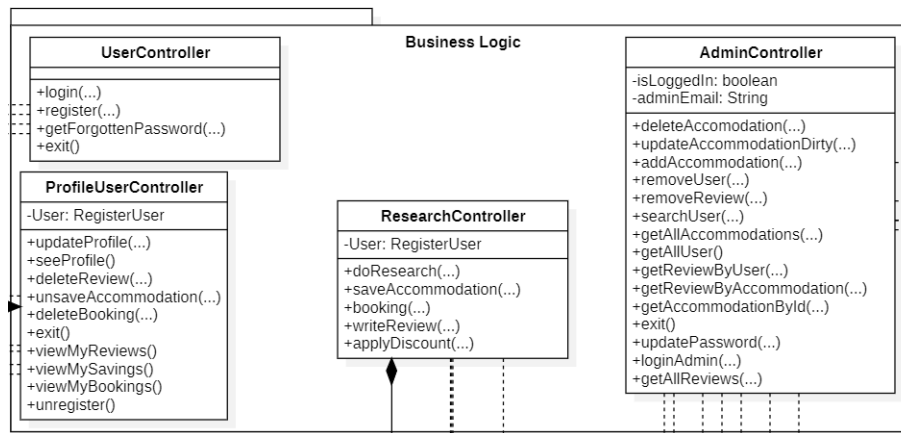


Figura 10: Class Diagram - Business Logic

- **Domain Model**: consiste nell'insieme di classi che rappresentano i concetti con cui interagisce l'applicazione: **RegisterUser**, **Review**, **Booking**, **Accommodation**, **SearchParameters**, **SearchParametersBuilder**.

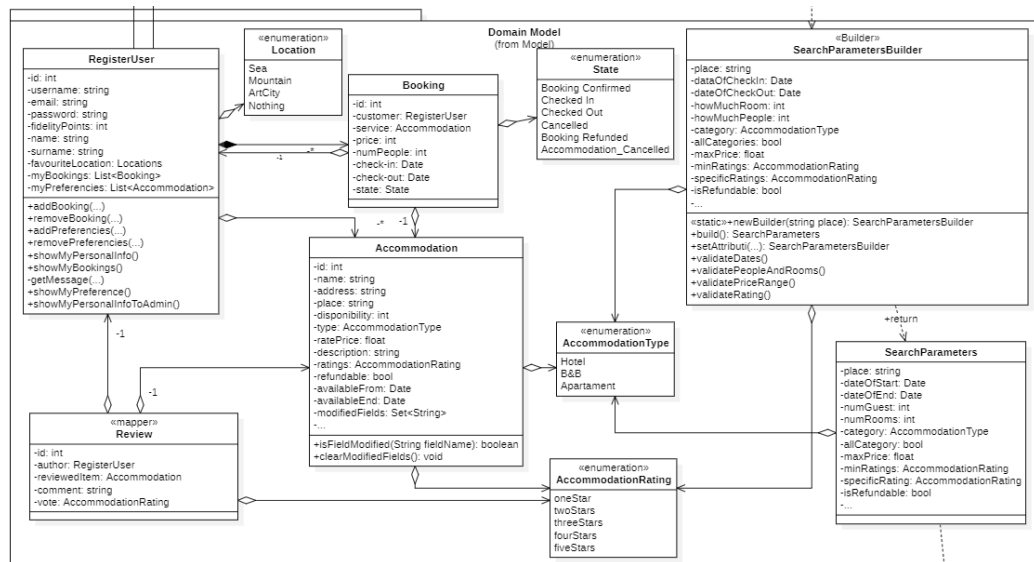


Figura 11: Class Diagram - Domain Model

- **DAO:** è il package che si occupa di gestire la connessione con il database: **UserDAO**, **BookingDAO**, **PreferenceDAO**, **ReviewDAO**, **AccommodationDAO**, **DatabaseConnection**.

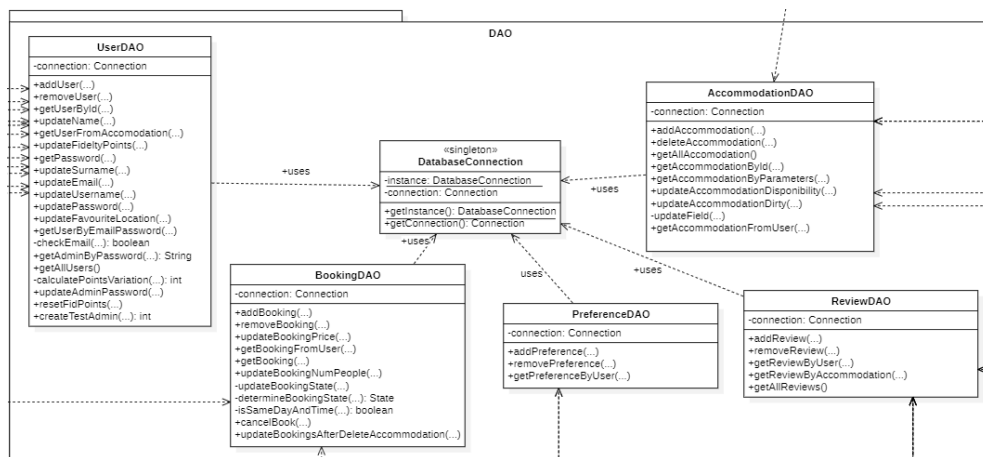


Figura 12: Class Diagram - DAO

2.5 Dettagli di Progetto

Nell'architettura sono stati usati diversi design pattern:

2.5.1 Singleton

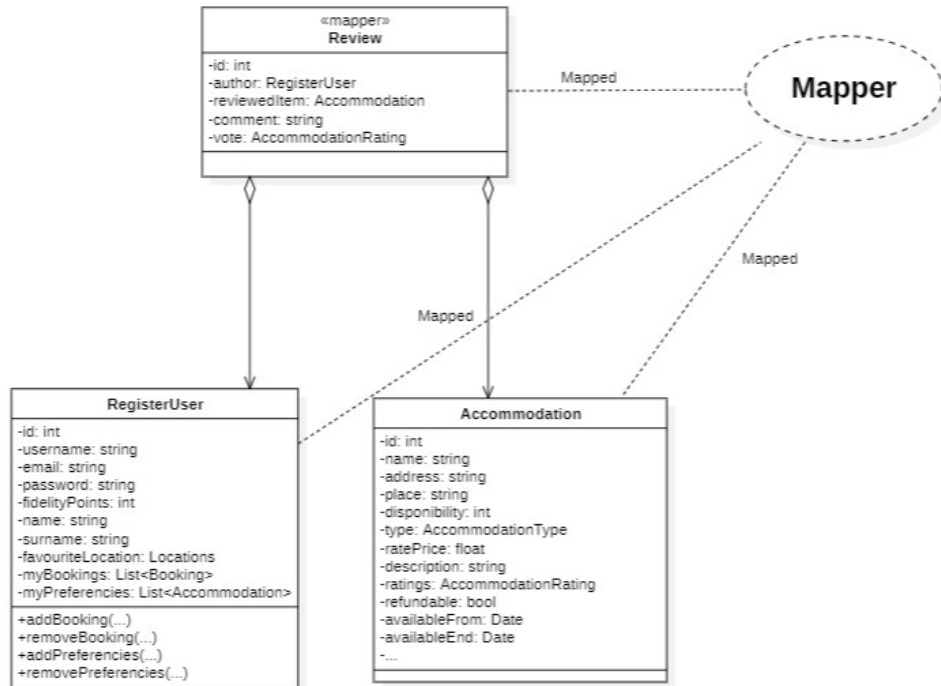
Lo scopo del Singleton è stato utilizzato per garantire che la connessione al database venisse effettuata una singola volta e per evitare conflitti tra connessioni.

```
public class DatabaseConnection {  
    // Istanza singleton  
    private static DatabaseConnection instance;  
  
    // Connessione al database  
    private Connection connection;  
  
    // Costruttore privato per prevenire l'istanziamento diretto  
    private DatabaseConnection() {}  
  
    // Metodo per ottenere l'istanza singleton  
    public static synchronized DatabaseConnection getInstance() {  
        if (instance == null) {  
            instance = new DatabaseConnection();  
        }  
        return instance;  
    }  
  
    // Metodo per ottenere la connessione al database  
    public Connection getConnection() {  
        return connection;  
    }  
}
```

Snippet 1: Implementazione Singleton

2.5.2 Mapper

Lo scopo del Mapper è quello di creare la relazione tra utenti, recensioni e alloggi.



Snippet 2: UML del Mapper

2.5.3 Builder Telescoping Constructor

Il Builder Telescoping Constructor viene usato per creare la classe parametri di ricerca che presenta tanti attributi, spesso opzionali, e consente una gestione più efficiente. L'unico scopo della classe è quello creare un oggetto di tipo ParametriRicerca e ci riesce grazie ai metodi di setting, che ritornando la classe stessa, consentono di avere una chiara creazione dell'oggetto, attraverso un metodo statico e evitando l'overloading dei costruttori tradizionale (**build**).

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 aumenta 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 13), e il Relational Model derivato (Figura 14), entrambi realizzati con **Draw.io**. 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:

- **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**.

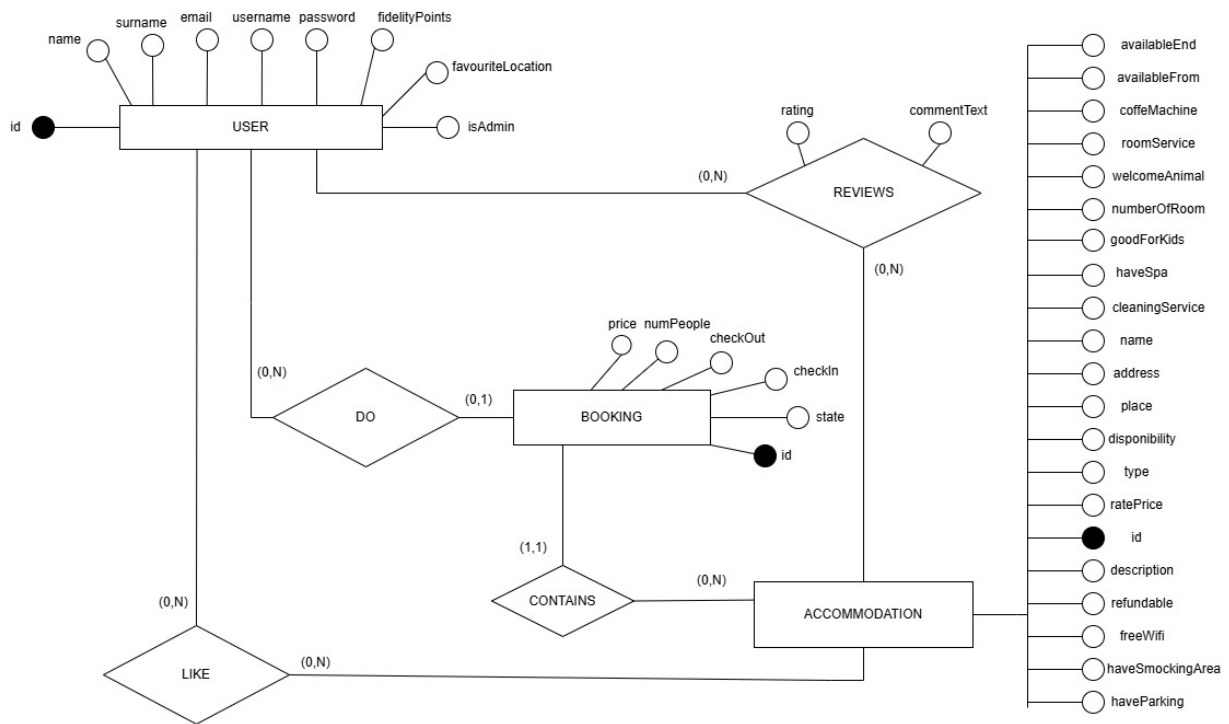


Figura 13: ER Diagram

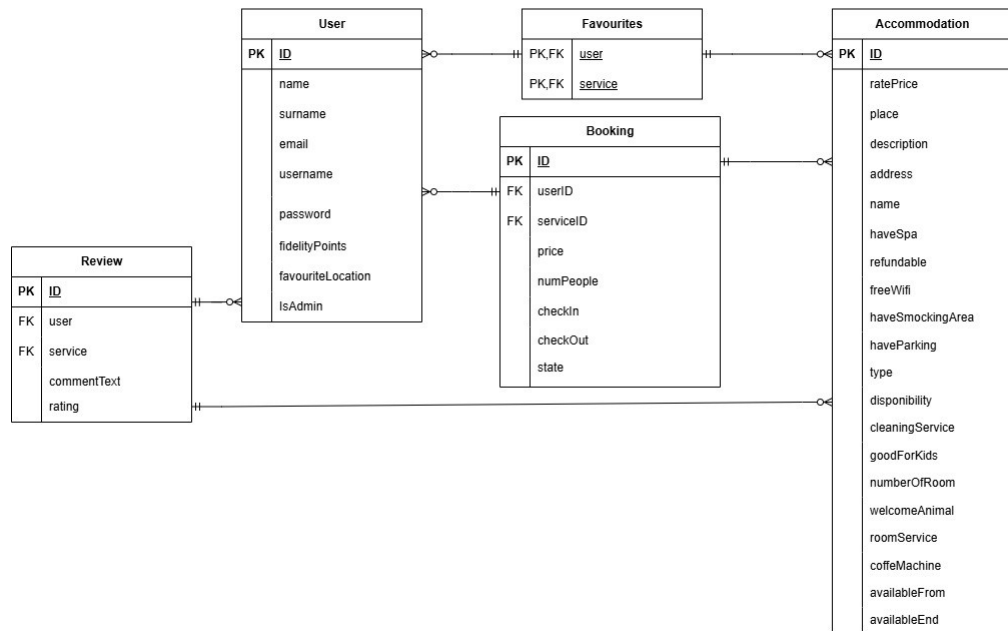


Figura 14: Tables of the database

2.6.1 Dettagli implementativi del database

Per gestire l'update del rating dell'alloggio all'interno del database è stata optata come soluzione l'utilizzo dei trigger. Il rating dell'alloggio viene aggiornato ogni volta che viene aggiunta o eliminata una sua recensione aggiornandone il valore con la media dei rating delle sue recensioni.

```

● CREATE OR REPLACE FUNCTION update_accommodation_rating()
  RETURNS TRIGGER AS $$
  DECLARE
    avg_rating INTEGER;
● BEGIN
    SELECT ROUND(AVG(rating))::INTEGER
    INTO avg_rating
    FROM Reviews
    WHERE accommodationId = NEW.accommodationId;

    UPDATE Accommodation
    SET rating = avg_rating
    WHERE id = NEW.accommodationId;

    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

● -- Trigger dopo l'inserimento di una recensione
  CREATE TRIGGER trg_update_rating_after_insert
  AFTER INSERT ON Reviews
  FOR EACH ROW
  EXECUTE FUNCTION update_accommodation_rating();

● -- Trigger dopo la cancellazione di una recensione
  CREATE TRIGGER trg_update_rating_after_delete
  AFTER DELETE ON Reviews
  FOR EACH ROW
  EXECUTE FUNCTION update_accommodation_rating();

```

Figura 15: trigger implementati per l'aggiornamento del rating per gli alloggi.

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, il prezzo, il check-in, il check-out e lo stato della prenotazione.

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 periodo di disponibilità, 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. Consente di stabilire una correlazione tra l'utente e l'alloggio in maniera discreta, senza che tale connessione sia direttamente percepita dagli interessati.

3.2.5 SearchParametersBuilder e SearchParameters

Queste classi implementano il design pattern Builder Telescopic Constructor 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**. 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**.

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'ultimo metodo viene utilizzato 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.

3.4 Interfaccia CLI

Per l'applicazione è stata realizzata un'interfaccia a linea di comando, implementata nel **Main.java**. L'utente può navigare per l'applicazione compiendo le funzionalità del programma inserendo i vari comandi indicati dal sistema. La gestione delle scelte effettuata dall'utente è ottenuta attraverso i costrutti *do-while* e *switch-case*, mentre le varie funzionalità sono implementate in metodi specifici delle classi contenute nella **Business Logic**.

```

public static void userMenu(RegisterUser registerUser) throws SQLException, ClassNotFoundException {
    Scanner scanner = new Scanner(System.in);
    ArrayList<Accommodation> accommodations;
    ProfileUserController puc= new ProfileUserController(registerUser);
    int choice;
    do{
        System.out.println("MENU USER: " +
            "\n1. Manage Profile" +
            "\n2. Research: do an apartment search " +
            "\n3. Log out ");

        choice = scanner.nextInt();

        switch(choice) {
            case 1:{
                profileMenu(registerUser,puc);
                break;
            }
        }
    }
}

```

Figura 16: implementazione del menu dell'utente.

4 Test

Sono stati realizzati i test per verificare il corretto funzionamento del sistema, focalizzandosi principalmente sulle funzionalità della **Business Logic** e del **Dao**. Sono stati realizzati utilizzando la libreria **JUnit**.

4.1 Domain Model Test

Sono stati implementati i test più rilevanti per il corretto funzionamento della logica di sistema e dei design pattern. I test implementati sono:

- **AccommodationTest.**
- **SearchParametersBuilderTest.**

4.2 Business Logic Test

Per i controller sono stati effettuati dei test su tutte le loro funzioni, prestando attenzione che seguono le direttive dello use case. Ne vengono riportati alcuni:

```

@Test
void login() throws SQLException, ClassNotFoundException {
    //testiamo il login di una persona registrata nel database
    user=userController.register(testEmail, testPassword, testUsername, testName, testSurname, testLocation);
    RegisterUser loginUser=userController.login(testEmail, testPassword);
    assertNotNull(loginUser);
    assertEquals(testUsername,loginUser.getUsername());
    assertEquals(testPassword,loginUser.getPassword());
    assertEquals(testEmail,loginUser.getEmail());
    assertEquals(testName,loginUser.getName());
    assertEquals(testSurname,loginUser.getSurname());
    assertEquals(testLocation, loginUser.getFavouriteLocations());

    //testiamo il login nel caso di una persona non registrata
    String testEmail2="test2@gmail.com";
    String testPassword2="Test1234!";
    loginUser=userController.login(testEmail2, testPassword2);
    assertNull(loginUser);

    //testiamo il login nel caso uno metta la password errata, ma l'email giusta
    loginUser=userController.login(testEmail, testPassword2);
    assertEquals( expected: -1, loginUser.getId());
    assertEquals(testEmail, loginUser.getEmail());
}

```

Figura 17: test che verifica il login all'interno dell'applicazione.

```

@Test
void register() throws SQLException, ClassNotFoundException {
    user=userController.register(testEmail, testPassword, testUsername, testName, testSurname, testLocation);
    assertNotNull(user);
    assertEquals(testEmail, user.getEmail());
    assertEquals(testPassword, user.getPassword());
    assertEquals(testUsername, user.getUsername());
    assertEquals(testName, user.getName());
    assertEquals(testSurname, user.getSurname());
    assertEquals(testLocation, user.getFavouriteLocations());
}

```

Figura 18: test che verifica la registrazione all'interno dell'applicazione.

✓ AdminControllerTest	600 ms
✓ loginAdmin()	426 ms
✓ deleteAccommodation()	44 ms
✓ removeReview()	25 ms
✓ addAccommodation()	4 ms
✓ updateAccommodation()	9 ms
✓ searchUser()	6 ms
✓ getAllReview()	26 ms
✓ getReviewByUser()	11 ms
✓ getAllUser()	15 ms
✓ getReviewByAccommodation()	8 ms
✓ changePassword()	4 ms
✓ removeUser()	13 ms
✓ getAccommodationById()	3 ms
✓ getAllAccommodation()	6 ms

Figura 19: i test per AdminController.

✓ ProfileUserControllerTest	38 ms
✓ updateProfile()	7 ms
✓ unSaveAccommodation()	5 ms
✓ removeReview()	5 ms
✓ getReviewsByUser()	4 ms
✓ unRegister()	3 ms
✓ viewMyBookings()	2 ms
✓ removeBooking()	6 ms
✓ cancelABooking()	6 ms

Figura 20: i test per ProfileUserController.

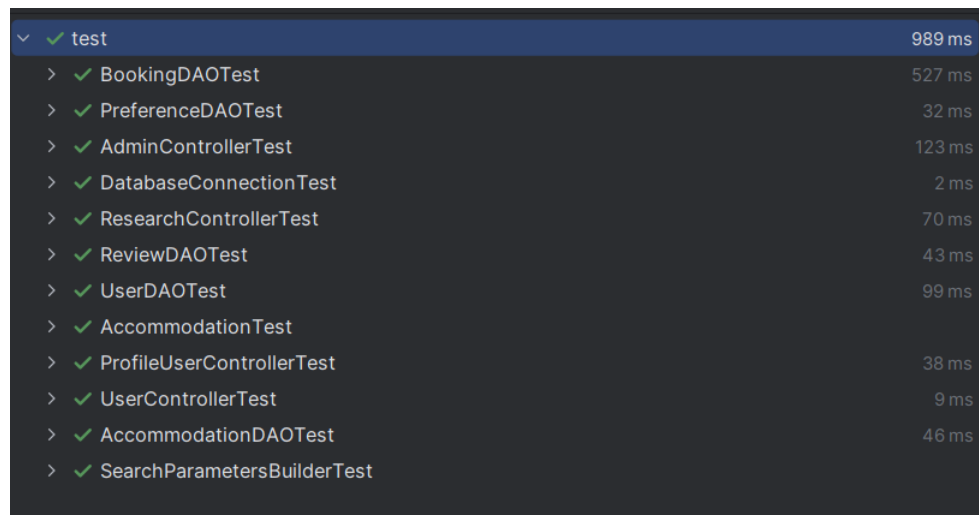
✓ ResearchControllerTest	72 ms
✓ doResearch()	23 ms
✓ booking()	16 ms
✓ applyDiscount()	9 ms
✓ saveAccommodation()	9 ms
✓ writeReview()	8 ms
✓ getReviews()	7 ms

Figura 21: i test per ResearchController.

4.3 DAO

Per ogni classe del DAO è stato effettuati dei test in modo da assicurarsi che i metodi si relazionino correttamente con il database. Questo controllo viene effettuato attraverso la cattura delle eccezioni in caso di errore.

4.4 Risultati dei test



✓ test	989 ms
> ✓ BookingDAOTest	527 ms
> ✓ PreferenceDAOTest	32 ms
> ✓ AdminControllerTest	123 ms
> ✓ DatabaseConnectionTest	2 ms
> ✓ ResearchControllerTest	70 ms
> ✓ ReviewDAOTest	43 ms
> ✓ UserDAOTest	99 ms
> ✓ AccommodationTest	
> ✓ ProfileUserControllerTest	38 ms
> ✓ UserControllerTest	9 ms
> ✓ AccommodationDAOTest	46 ms
> ✓ SearchParametersBuilderTest	

Figura 22: risultati dei test.