Bookig App

Andrea Gepponi

Maggio - Giugno 2024

Index

1 Introduzione

Elaborato per il superamento dell'esame di Ingegneria del Software, appartenente al modulo Basi di Dati / Ingegneria del Software del corso di Laurea Triennale in Ingegneria Informatica dell'Università degli Studi di Firenze. Il progetto è stato sviluppato da Andrea Gepponi, matricola 7047622, durante il periode Maggio - Giugno 2024 (a.a. 2023/2024).

Il codice sorgente è disponibile al seguente link: https://github.com/AndreaGepponi/BookingApp

1.1 Obbiettivo e descrizione progetto

Il progetto consiste in un programma che permette di mettere a disposizione delle strutture alberghiere e prenotare stanze.
Comprende un sistema a più utenti che possono aggiungere strutture effettuare prenotazioni e gestire il proprio profilo.
Gli utenti si suddividono in Manager e Customer.
Il Manager aggiunge strutture disponibili mentre il Customer effettua prenotazioni e lascia recenzioni.
Per la gestione e il salvataggio dei dati è stato creato e connesso un database.

1.2 Struttura e pratiche utilizzate

Il software 'e stato sviluppato in Java, mentre per la gestione e il salvataggio dei dati 'e stato connesso un database PostgreSQL ed 'e stata utilizzata la libreria JDBC (Java DataBase Connectivity).

Per mantenere una separazione delle responsabilit'a, la struttura del progetto 'e stata divisa in tre parti principali: Business Logic, Domain Model e ORM. Questi tre packages si occupano in modo distinto della logica di business, della rappresentazione dei dati e dell'accesso ai dati (Figura 1):

- **Business Logic**: contiene le classi che implementano la logica di business del sistema.
- **Domain Model**: contiene le classi che rappresentano le entità del sistema.
- **ORM**: contiene le classi che implementano l'Object-Relational Mapping. In questo modo è possibile rendere i dati persistenti e recuperarli dal database.

Per utilizzare il software 'e stata creata un'interfaccia da riga di comando (CLI) che permette di interagire con il sistema in modo semplice e intuitivo.
Gli Use Case Diagram e i Class Diagram seguono lo standard UML (Unified Modeling Language) e sono stati realizzati con il software StarUML. Infine, per la parte di testing 'e stato utilizzato JUnit. Le piattaforme e i software utilizzati sono stati:

- IntelliJ IDEA: IDE per lo sviluppo in Java
- StarUML: software per la creazione di diagrammi UML
- Draw.io: altro software per la creazione di diagrammi
- PgAdmin: software per la gestione di database PostgreSQL
- GitHub: piattaforma per la condivisione del codice sorgente
- Lunacy: software per la creazione di mock-up
- tree.nathanfriend.io: sito per la creazione di diagrammi di directory

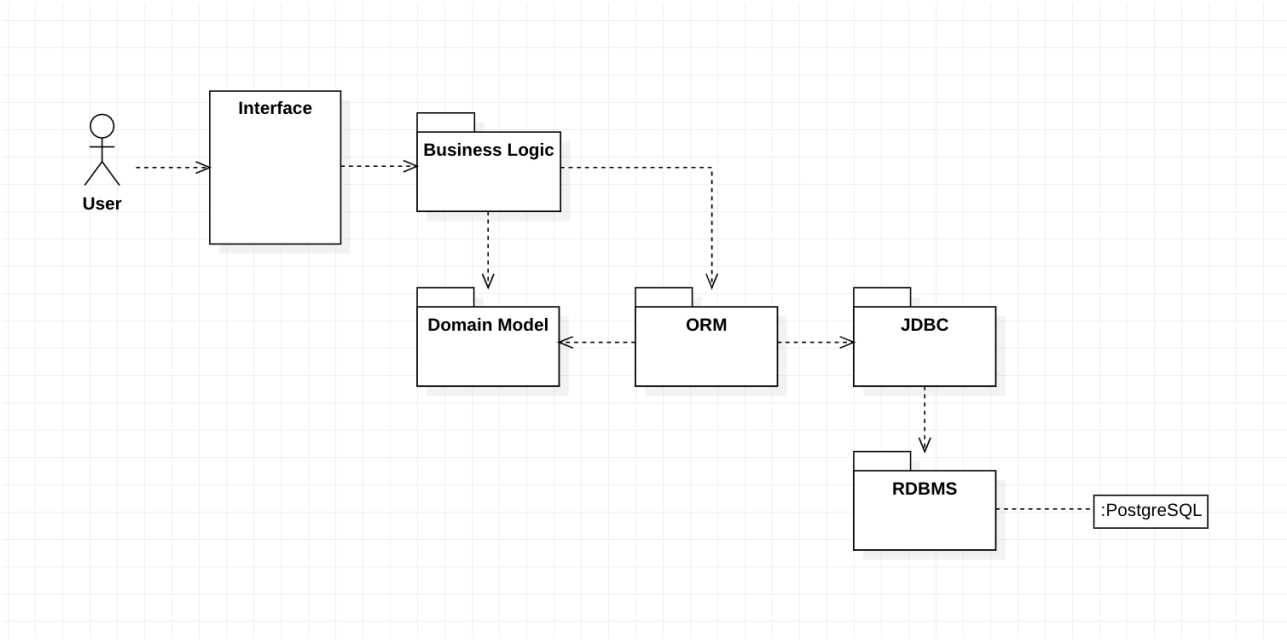

Figure 1: Package Dependency Diagram

2 Progettazione

2.1 Use case Diagram

Il funzionamento del sistema è mostrato nello use case diagram, sia lato Customer che lato Manager.

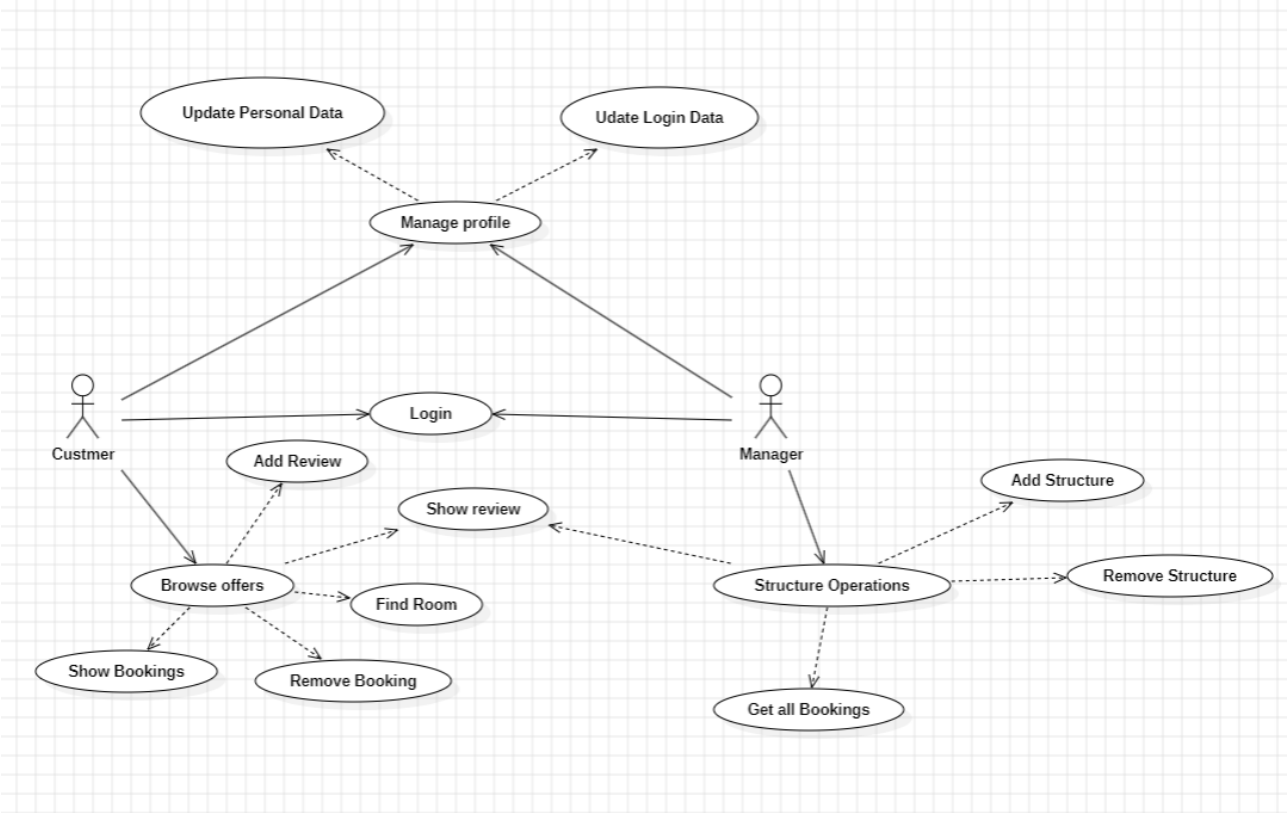

Figure 2: Use Case Diagram

2.2 Use Case Template

Di seguito sono riportati i template di alcuni dei casi d'uso implementati, in ognuno possiamo trovare: una breve descrizione, il livello del caso d'uso, gli attori coinvolti, le precondizioni, le post-condizioni, il flusso base e, eventualmente, i flussi alternativi.

Use Case #1	Accesso al sistema (Sign in)
Brief Description	L'utente accede al sistema tramite le proprie credenziali
Level	User Goal
Actors	Customer, Manager
Pre-conditions	L'utente deve essere nella pagina iniziale di accesso.
Basic Flow	1. L'utente inserisce le proprie credenziali (username e password) 2. L'utente preme il pulsante di accesso 3. Il sistema verifica le credenziali 4. Il sistema autentica l'utente
Alternative Flows	• 3a. Se le credenziali fornite non sono corrette, il sistema mostra un messaggio di errore all'utente
Post-conditions	L'utente è autenticato nel sistema e ha accesso alle funzionalità riservate

Table 1: Use Case Template 1 (Sign in)

Use Case #2	Registrazione nel sistema (Sign up)
Brief Description	L'utente si registra nel sistema creando un nuovo account
Level	User Goal
Actors	Customer, Manager
Pre-conditions	L'utente deve essere nella pagina iniziale di registrazione.
Basic Flow	1. L'utente fornisce i dettagli richiesti per la registrazione 2. L'utente conferma la registrazione 3. Il sistema verifica i dati forniti 4. Il sistema crea un nuovo account per l'utente
Alternative Flows	• 3a. Se l'utente fornisce dati non validi o se l'email o lo username sono già utilizzati da un altro account, il sistema mostra un messaggiodi errore
Post-conditions	L'utente 'e registrato nel sistema e pu'o accedere utilizzando le credenziali create durante la registrazione

Table 2: Use Case Template 2 (Sign up)

Use Case #3	Ricerca e prenotazione di una stanza
Brief Description	L'utente ricerca una stanza da prenotare e ne sceglie una.
Level	Function
Actors	Customer
Pre-conditions	L'utente deve essere nella pagina di Browsing.
Basic Flow	1. L'utente fornisce i dettagli della struttura 2. Il sistema fornisce una lista di stanze che soddisfano le condizioni dell'utente 3. L'utente decide se confermare la prenotazione o meno
Alternative Flows	• 2a. Se non ci sono stanze disponibili viene generato un messaggio d'errore
Post-conditions	L'utente ha a prenotato una stanza

Table 3: Use Case Template 3 (Browsing)

Use Case #4	Cancellazione di una prenotazione
Brief Description	L'utente cancella una sua prenotazione.
Level	User goal
Actors	Customer
Pre-conditions	L'utente deve essere nella pagina di Browsing.
Basic Flow	1. L'utente fornisce il nome della struttura e il periodo della prenotazione.
Post-conditions	L'utente ha rimosso una prenotazione.

Table 4: Use Case Template 4 (Deleting)

Use Case #5	Aggiunta di una struttura
Brief Description	L'utente aggiunge una struttura al database.
Level	User goal
Actors	Manager
Pre-conditions	L'utente deve essere nella pagina di Action.
Basic Flow	1. L'utente fornisce i dati della struttura da inserire.
Post-conditions	L'utente ha aggiunto una struttura.

Table 5: Use Case Template 5 (Adding Structure)

2.3 Mockups

Ecco di seguito alcuni possibili Mock-ups relativi alle interfacce grafiche del sistema.

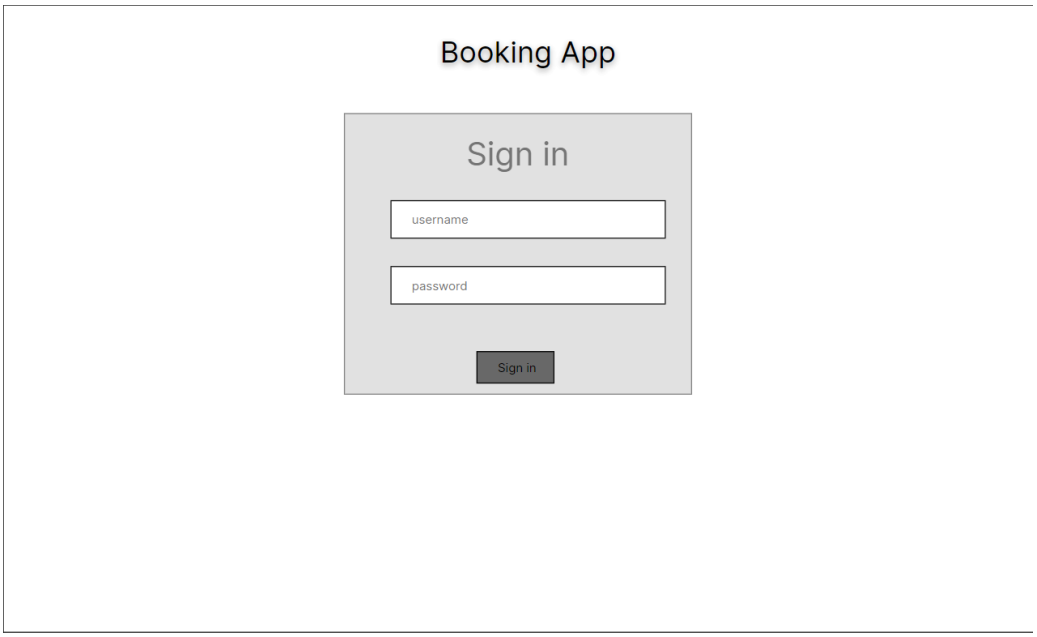

Figure 3: Sign in mockup

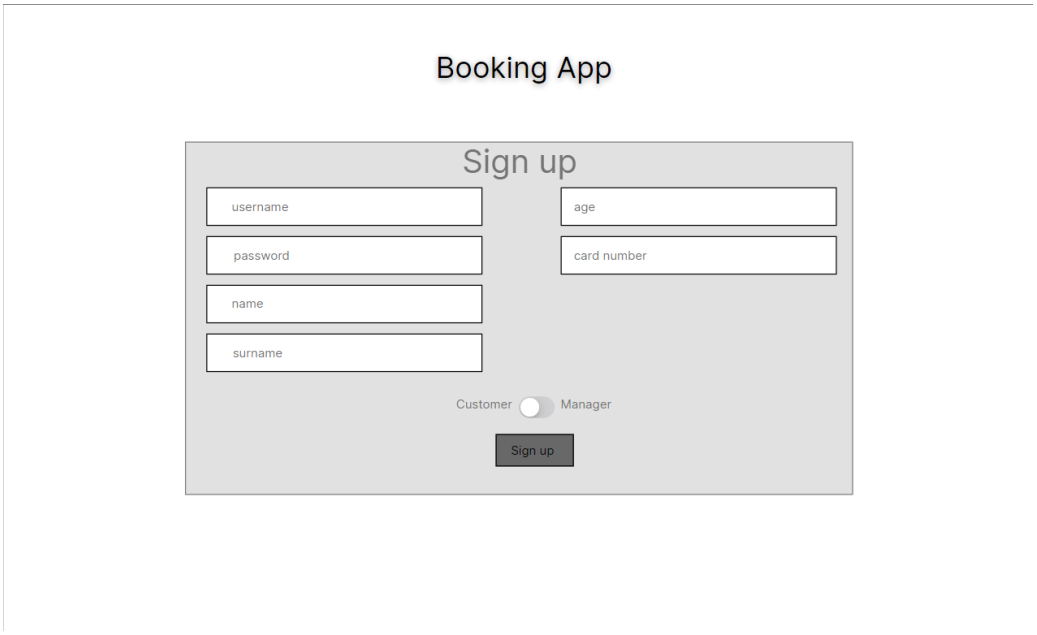

Figure 4: Sign up mockup

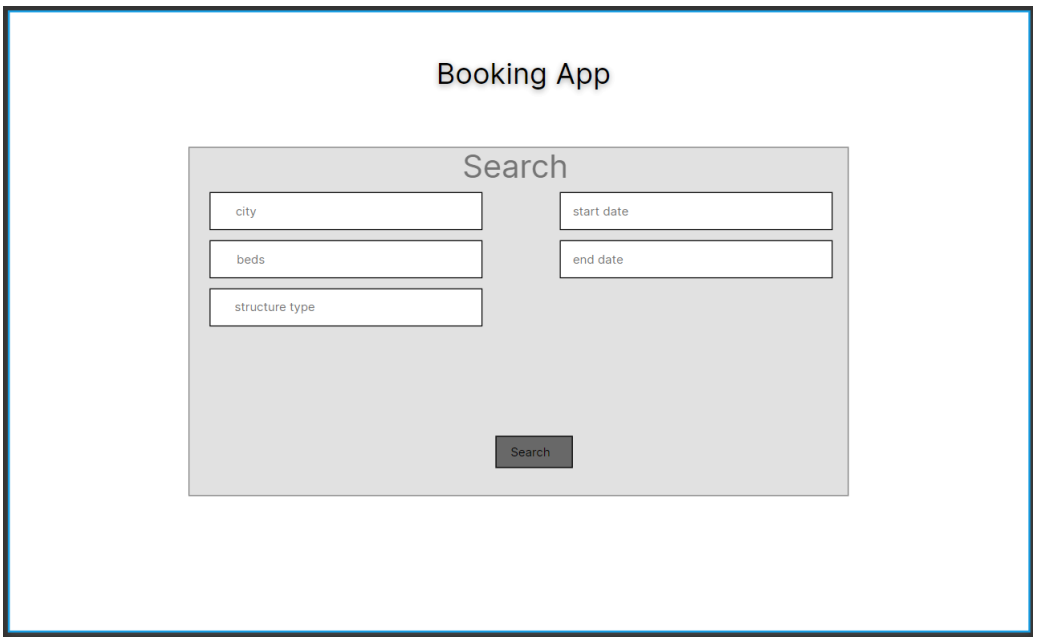

Figure 5: Search mockup

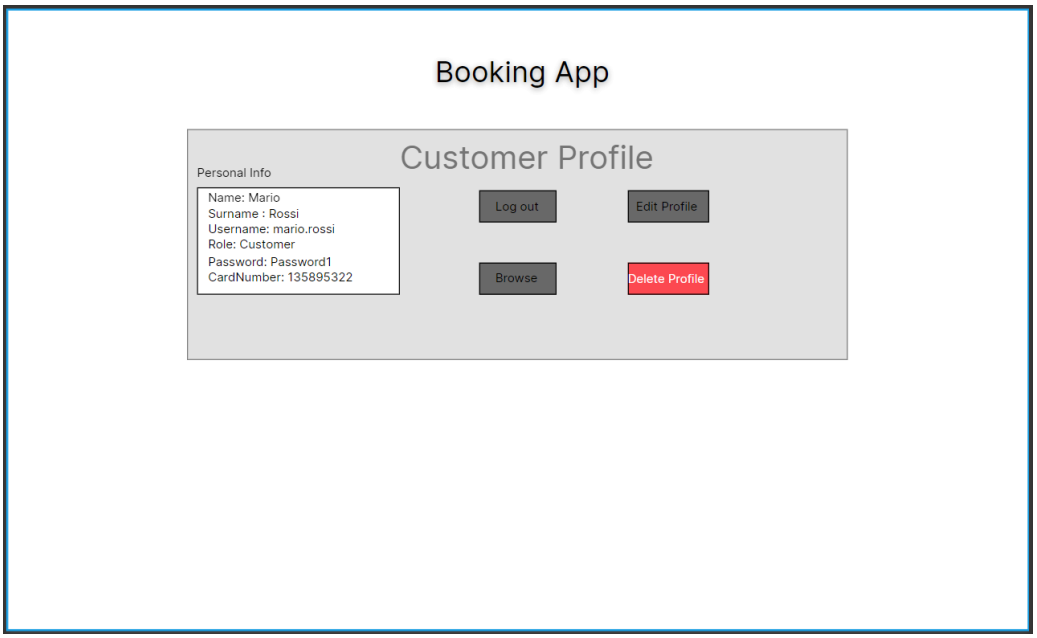

Figure 6: Profile mockup

2.4 Class Diagram

Il softwere si suddivide in tre packages, di seguito il diagramma delle classi di ciascuno di essi:

- Business Logic:contiene le classi che implementano la logica di business del sistema, ovvero i seguenti controller:quello che gestisce l'accesso e la registrazione dei nuovi utenti (LoginController),quello che permette ai Customer di cercare e prenotare stanze, di aggiungere e leggere recenzioni(CustomerProfileController),quello che permette ai Manager di aggiungere e rimuovere strutture e leggere recenzioni (ManagerProfileController).
- Domain Model:contiene le classi che rappresentano le entità del sistema, ovvero:Customer, Manager, CreditCard, Review, Period, Booking,Room, Structure.Contiene anche BookingMapper che implementa il mapper per le prenotazioni.
- ORM:contiene le classi che implementano l'Object-Relational Mapping, quindi contiene una classe per ogni entit'a del sistema:CustomerDAO, ManagerDAO,CreditCardDAO,ReviewDAO, StructureDAO, RoomDAO, BookingDAO. Contiene anche la classe ConnectionManager che si occupa di gestire la connessione al database.

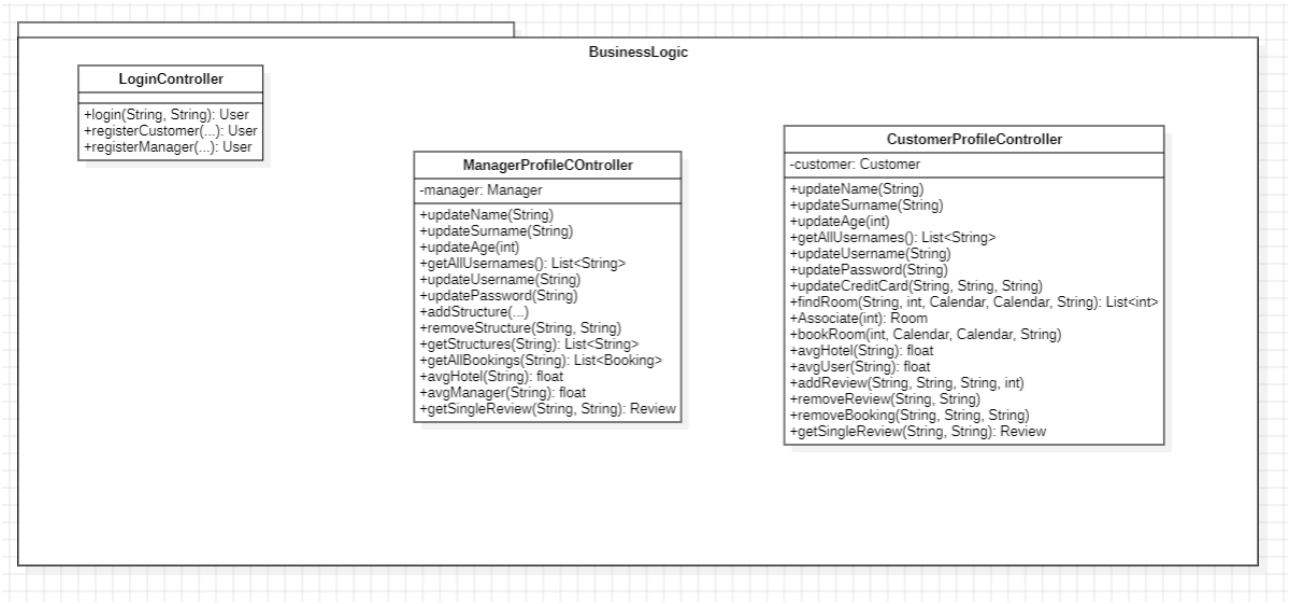

Figure 7: Business Logic Diagram

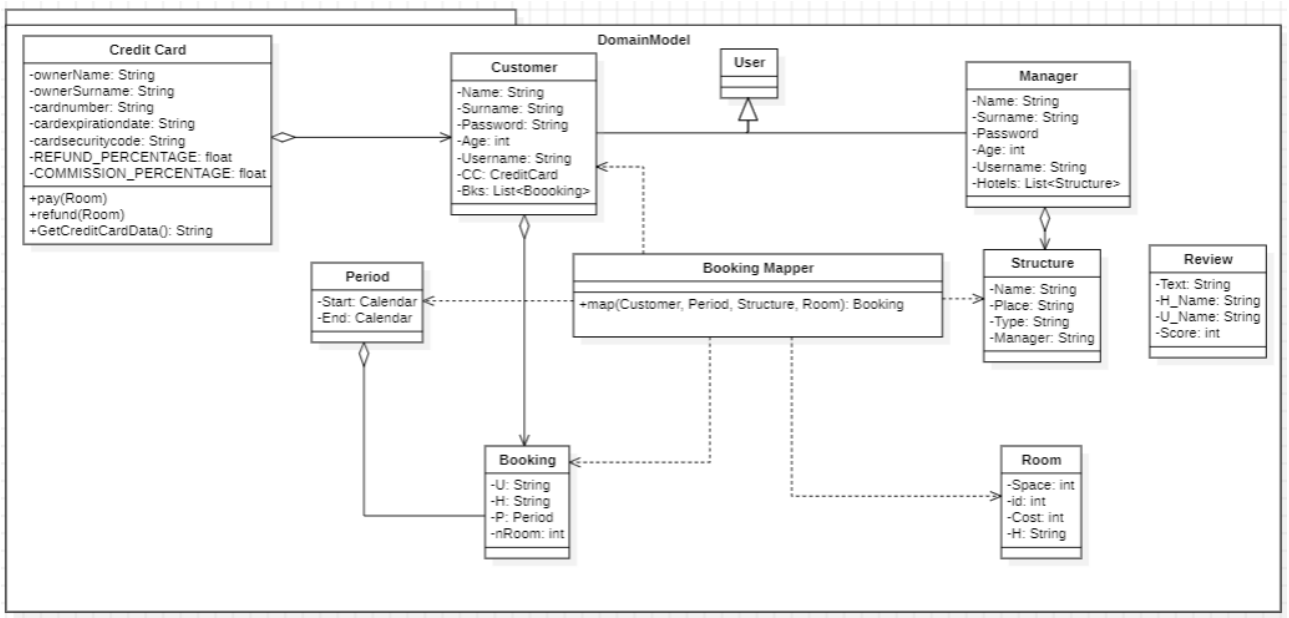

Figure 8: Domain Model Diagram

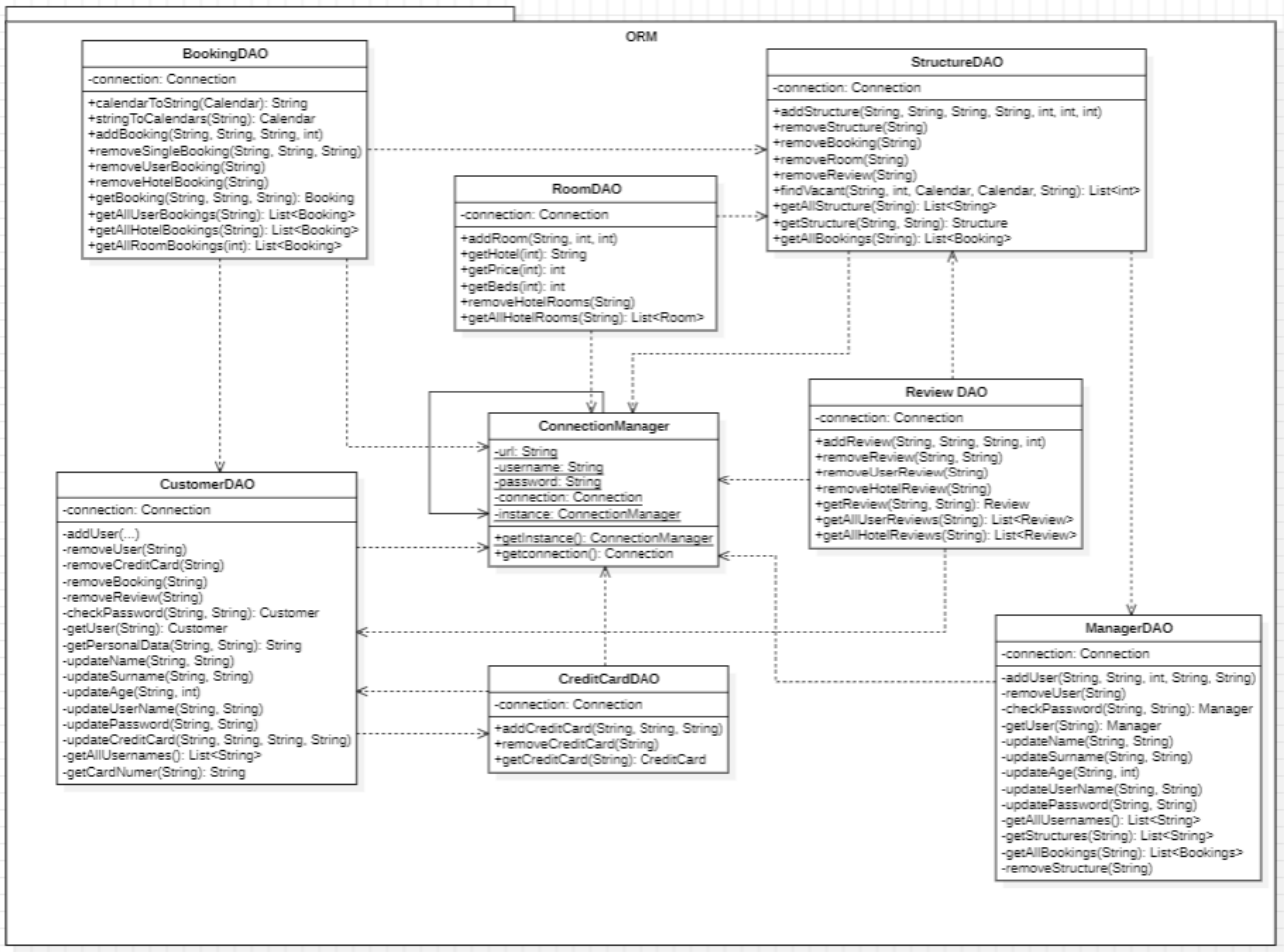

Figure 9: ORM Diagram

2.5 ER Diagram

Per la progettazione del database è stato realizzato un ER diagram.

- User:rappresenta sia l'entità Customer che Manager, distinte dall'attributo "role".

- CreditCard: rappresenta l'entità CreditCard.
- Structure: rappresenta l'entità Structure.
- Room: rappresenta l'entità Room.
- Booking: risolve la relazione Booking tra Structure, Booking e Room.
- Review: risolve la relazione Review tra User e Structure.

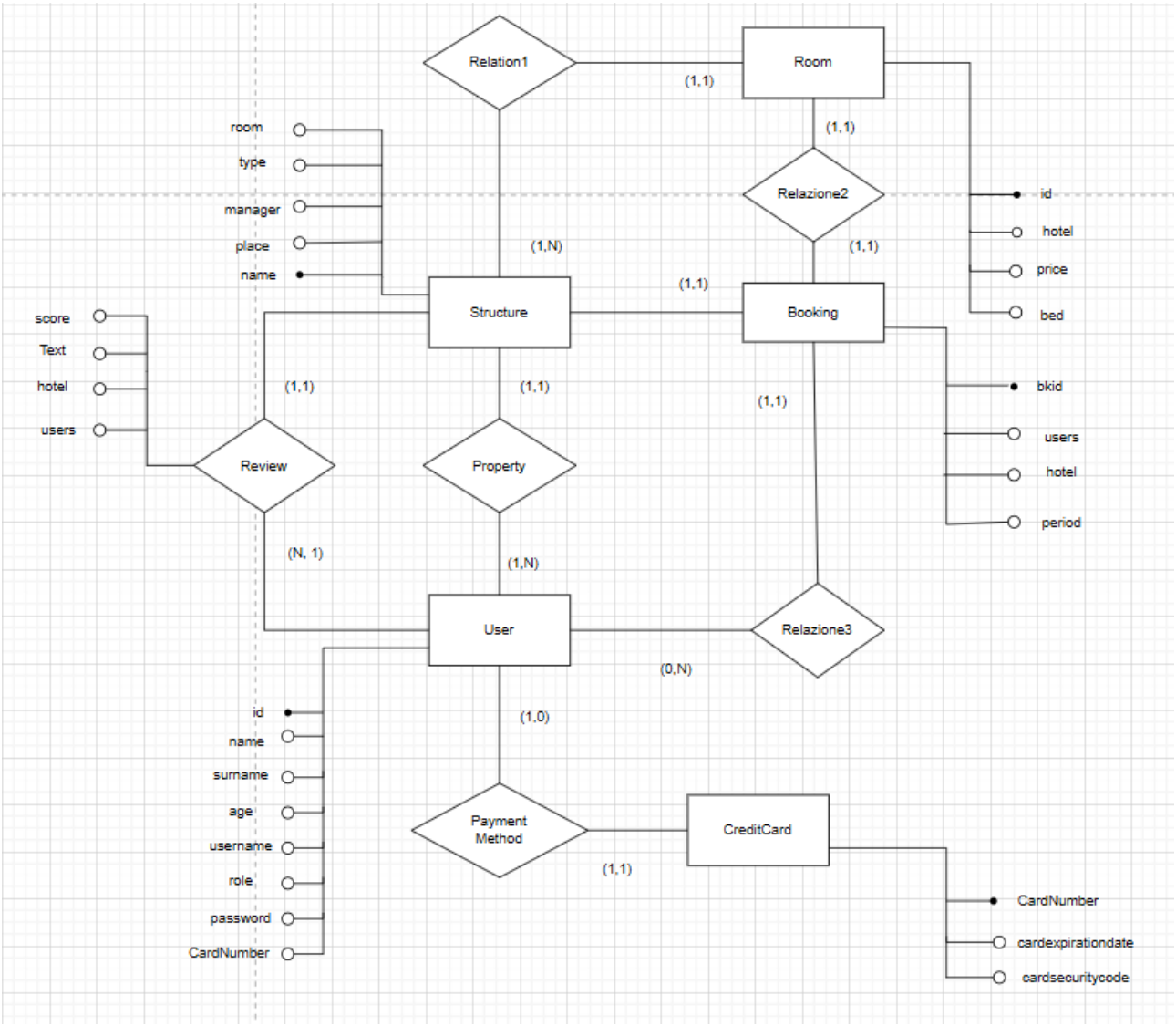

Figure 10: ER Diagram

2.6 Navigation Diagram

Il seguente diagramma rappresenta quelle che sono le pagine principali del sistema e le possibili azioni che l'utente può compiere. Sono anche rappresentati i modi con cui si pu'o navigare tra le varie pagine.
Alcune pagine sono:

- la pagina iniziale: dove l'utente pu'o registrarsi o effettuare il login.
- User Page: da dove si può accedere al profilo o alle azioni.
- Profile Editor: da dove possono essere modificate le informazioni del profilo.

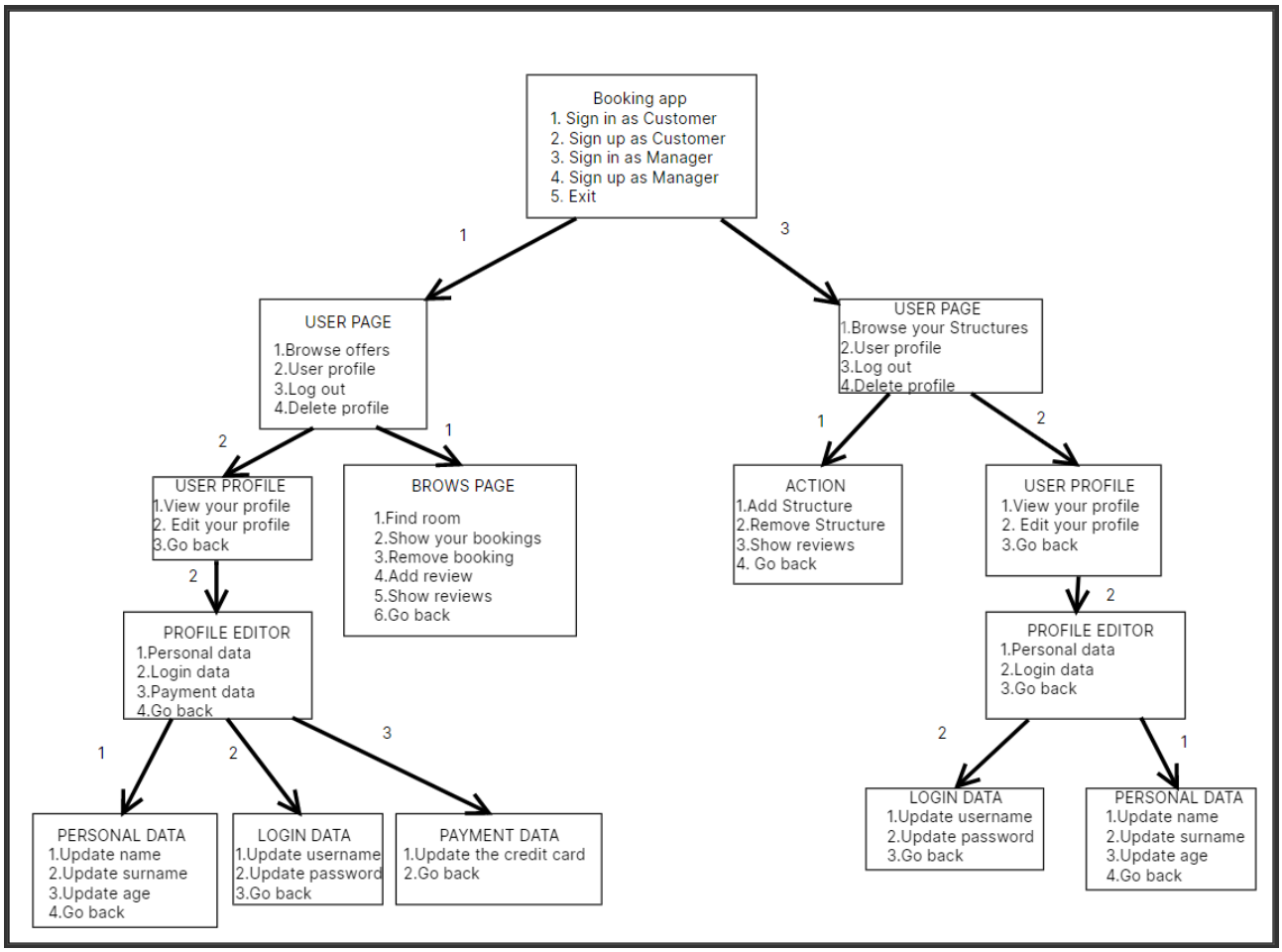

Figure 11: Navigation Diagram

2.7 Struttura della directory

Di seguito la struttura della directory che contiene:

- src: contiene il codice sorgente.
- test: contiene il codice di test.

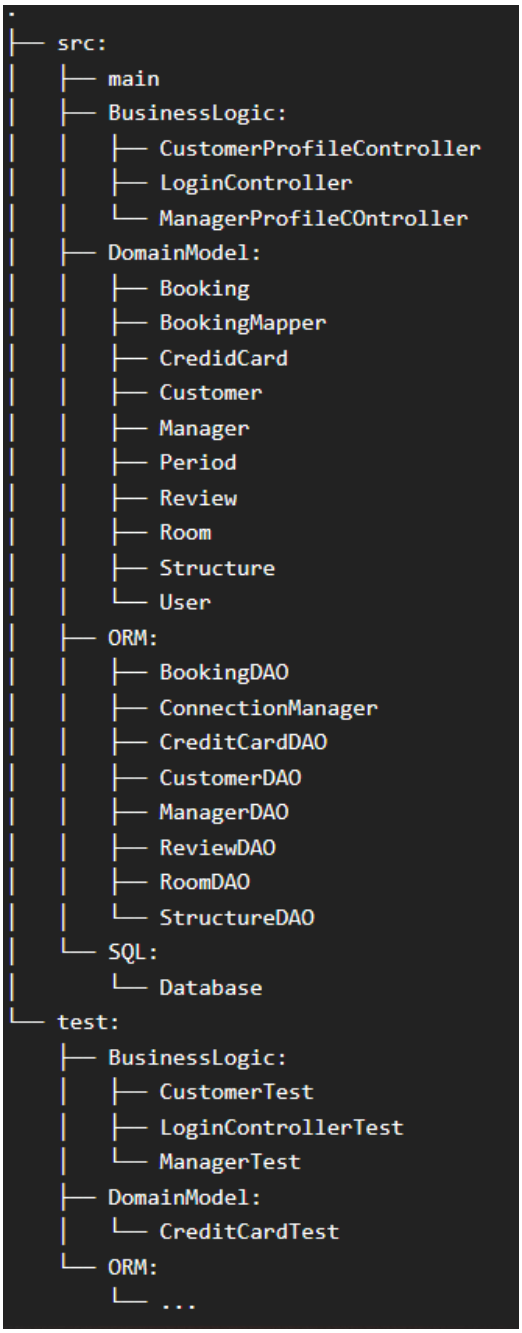

Figure 12: directory structure

3 Implementazione

3.1 Domain Model

Nel package Domain Model sono state implementate le classi che rappresentano le entit‘a del sistema.

3.1.1 User

La classe User è una classe astratta che permette di semplificare l'operazione di login degli utenti.

3.1.2 Customer

La classe Customer rapprresenta un utente che si è registrato nel sistema con l'intento di effettuare prenotazioni. I campi sono name, surname, Password, Age, Username, CC(CreditCard), e Bks(lista di prenotazioni).

3.1.3 Manager

La classe Manager rappresenta un utente che si è registrato nel sistema con l'intento di mettere disposizione strutture per le prenotazioni. I campi sono Name, Surname, Password, Age, Username e Hotels(lista di strutture).

3.1.4 CreditCard

La classe CreditCard rappresenta una carta di credito associata ad un Customer, utilizzata per i pagamenti.

3.1.5 Structure

La classe Structure rappresenta una struttura messa a disposizione da un Manager. I campi sono Name, Place, Type e Manager.

3.1.6 Room

La classe Room rappresenta una stanza di una struttura che è possibile prenotare. I campi sono Space(numero di posti letto), id, Cost e H(nome della struttura a cui appartengono).

3.1.7 Review

La classe Review rappresenta una recenzione lasciata da un utente ad una struttura. I campi sono Text, HName, UName e Score.

3.1.8 Period

La classe Period rappresenta l'intervallo di tempo per il quale un Customer intende prenotare una stanza. I campi sono Start e End.

3.1.9 Booking e BookingMapper

Booking rappresenta una prenotazione effettuata da un Customer su una determinata stanza di una struttura per un determinato periodo. I campi sono U(username del Customer), H(nome della struttura), P(periodo) e nRoom(id della stanza prenoata). La classe BookingMapper è stata creata per mappare i dati della prenotazione.

3.2 BusinessLogic

Nel package BusinessLogic sono state implementate le classi che gestiscono la logica di business del sistema.

3.2.1 LoginController

Questa classe controlla la gestione dell'accesso e della registrazione degli utenti tramite le funzioni login(), registerCustomer() e registerManager(): la prima riceve in ingresso l'username, la password e il ruolo (Customer / Manager) dell'utente e controlla la correttezza delle cradenziali nel database, le altre due ricevono in ingresso tutti i dati dell'utente e li inseriscono nel database.

```java
public class BookingMapper {

    1 usage
    public Booking map(String u_name, Calendar s, Calendar e, String h, int id){
        Booking booking = new Booking();

        booking.setH(h);
        booking.setnRoom(id);
        booking.setP(s, e);
        booking.setU(u_name);

        return booking;
    }
}
```

Figure 13: Snippet di BookingMapper

3.2.2 CustomerProfileController

Questa classe si occupa di tutte le azioni che possono essere effettuate da un Customer, permette di aggiornare i dati personali tramite le funzioni updateName(), updateSurname(), updateAge(), updateUsername(), updatePassword(), updateCreditCard(). Permette di ricercare stanze disponibili con la funzione findRoom() che richiede la città, la data di inizio e fine di prenotazione, il tipo di struttura e il numero di posti letto. Permette di prenotare una stanza tramite le funzioni associate()(che associa ad ogni id la stanza corrispondente) e bookRoom() che aggiunge una prenotazione. Permette infine di aggiungere e rimuovere recenzioni tramite addReview() e removeReview(), di cercare una singola recenzione tramite getSinglereview() e di vedere il punteggio medio di una determinata struttura e il punteggio medio delle recenzioni lasciate da un singolo utente tramite avgHotel() e avgUser().

3.2.3 ManagerProfileController

Questa classe gestisce tutte le azioni che possono essere effettuate da un Manager,permette di aggiornare i dati personali tramite le funzioni updateName(), updateSurname(), updateAge(), updateUsername(), updatePassword(). Permette di aggiungere e rimuovere strutture tramite addStructure e removeStructure. Permette di cercare singole recenzioni e di vedere il punteggio medio di una struttura tramite getSinglereview() e avgHotel. Permette infine di visualizzare tutte le prenotazioni per per una struttura tramite getAllBookings.

3.3 Object-Relational Mapping

Nel package ORM sono state implementate le classi che si occupano dell'Object-Relational Mapping, cioè delle operazioni di lettura e scrittura dati nel database.

3.3.1 ConnectionManager

La classe si occupa di gestire la connessione al database per le altre classi DAO tramite il metodo getConnection(). Contiene inoltre i dati di accesso al database, ovvero URL, username e password, che vengono utilizzati per stabilire la connessione. Per garantire che ci sia una sola istanza della classe in tutto il sistema e quindi per evitare conflitti tra connessioni, la classe 'e implementata come un singleton.

3.3.2 CustomerDAO e ManagerDAO

Queste due classi si occupano della gestione dei dati degli utenti, come le altre classi DAO permettono alle classi del package BusinessLogic di accedere ai dati salvati nel database. Entrambe le classi contengono metodi per l'aggiornamento di dati personali e per aggiungere o rimuovere utenti. In alcuni metodi vengono utilizzati anche altri DAO, per esempio quando viene rimosso un Customer viene rimossa anche la carta di credito associata insieme alle recenzioni e prenotazioni fatte da quel Customer.

```java
public static ConnectionManager getInstance() {

    if (instance == null) { instance = new ConnectionManager(); }

    return instance;

}

7 usages
public Connection getConnection() throws SQLException, ClassNotFoundException {

    Class.forName( className: "org.postgresql.Driver");

    if (connection == null)
        try {
            connection = DriverManager.getConnection(url, username, password);
        } catch (SQLException e) {
            System.err.println("Error: " + e.getMessage());
        }

    return connection;

}
```

Figure 14: Snippet di ConnectionManager

```java
public void removeUser(String username) throws SQLException, ClassNotFoundException     {

    String sql = String.format("DELETE FROM \"User\" WHERE username = '%s'", username);

    PreparedStatement preparedStatement = null;

    // remove CreditCard method (CASCADE DELETE)
        removeCreditCard(username);
    // remove booking (CASCADE DELETE)
        removeBooking(username);
    // remove review (CASCADE DELETE)
        removeReview(username);

    try {
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.executeUpdate();
        System.out.println("User removed successfully.");
    } catch (SQLException e) {
        System.err.println("Error: " + e.getMessage());
    } finally {
        if (preparedStatement != null) { preparedStatement.close(); }
    }

}
```

Figure 15: Snippet di CustomerDAO

3.3.3 BookingDAO

La classe BookingDAO si occupa della gesione dei dati delle prenotazioni e permette alle classi della BusinessLogic di accedere ai dati delle prenotazioni salvati neldatabase. Contiene metodi come addBooking(), removeBooking(), getSingleBooking(), e metodi come calendarToString per convertire oggetti java.Calendar in stringhe da salvare nel database.

3.3.4 CreditCardDAO

Questa classe si occupa della gestione della carta di credito associata ad un Customer. Contiene i metodi addCreditCard(), removeCreditCard() e getCreditCard().

3.3.5 StructureDAO

La classe StructureDAO gestisce i dati relativi alle strutture e contiene metodi che permettono di cercare stanze libere findVacant() e ottenere tutte le prenotazioni su stanze di quella struttura getAllBooking(). Anche in questo caso vengono utilizzati altri DAO, quando una struttura viene rimossa vengono rimosse tutte le stanze, prenotazioni e recenzioni associate ad essa.

```
public void removeStructure(String name) throws SQLException, ClassNotFoundException{
    String sql = String.format("DELETE FROM \"Structure\" WHERE name = '%s'", name);

    PreparedStatement preparedStatement = null;

    // remove Booking (CASCADE DELETE)
    removeBooking(name);
    // remove Room (CASCADE DELETE)
    removeRoom(name);
    // remove Review (CASCADE DELETE)
    removeReview(name);

    try {
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.executeUpdate();
        System.out.println("Structure removed successfully.");
    } catch (SQLException e) {
        System.err.println("Error: " + e.getMessage());
    } finally {
        if (preparedStatement != null) { preparedStatement.close(); }
    }
}
```

Figure 16: Snippet di StructureDAO

3.3.6 RoomDAO

La classe RoomDAO si occupa dei dati relativi alle stanze. Contiene metodi come getPrice(), getHotel(), getBeds().

3.3.7 ReviewDAO

La classe ReviewDAO permette di ottenere una singola recenzione oppure ottenere recenzioni di un determinato Utente o di una determinata struttura tramite i metodi getReview(), getAllUserReview, getAllHotelReview(). Altri metodi permettono l'aggiunta di una recenzione o la rimozione di una o più recenzioni.

3.4 Interfaccia CLI

Per l'utilizzo del sistema è stata implementata un'interfaccia a linea di comando nel file main.java. L'utente pu'o navigare tra le varie pagine e compiere le funzionalità del programma inserendo i vari comandi indicati dal sistema. Nella pagina iniziale si ha scelta tra effettuare il login come Customer o come Manager, registrarsi come Customer o come Manager oppure chiudere il sistema. Ogni pagina è implementata tramite un ciclo do-while e uno switch-case per gestire le varie scelte dell'urente, mentre le varie funzionalità sono implementate con metodi specifici delle classi del package BusinessLogic. La navigazione tra le pagine avviene tramite chiamate agli stessi metodi della classe main.

```java
public static void handleLoginAction() throws SQLException, ClassNotFoundException, ParseException {

    Scanner scanner = new Scanner(System.in);
    LoginController loginController = new LoginController();

    String input;

    do {

        System.out.println(
                """
                \s
                 1. Sign in as Customer
                 2. Sign up as Customer
                 3. Sign in as Manager
                 4. Sign up as Manager
                 5. Exit
                \s"""
        );
```

Figure 17: Snippet main

4 Testing

Sono stati realizzati test per verificare il corretto funzionamento del sistema, in particolare sono stati effettuati dei test funzionali su alcuni metodi del package Business Logic e dei test strutturali su alcuni metodi del package Domain Model; per il package ORM non sono stati realizzati test specifici in quanto la correttezza dei metodi ‘e stata verificata indirettamente tramite i test sulla logica di business.

4.1 BusinessLogic Test

Sono implementati i seguenti test nelle seguenti classi:

- LoginControllerTest:
  loginTest(), registerCustomerTest(), registerManagerTest().
- CustomerTest:
  BookingOperationsTest(), findRoomTest(), getReviewsTest().
- ManagerTest:
  StructureOperationsTest().

I test della classe LoginControllerTest verificano il corretto funzionamento dei metodi login(), registerCustomer() e registerManager(), utilizzando utenti noti e nuovi.

```java
public void loginTest(){

    // Test case 1: this is a known user in the database
    String username = "mario.rossi";
    String password = "Password1";

    LoginController loginController = new LoginController();

    try {
        User user = loginController.login(username, password);
        assertNotNull(user);
    } catch (SQLException | ClassNotFoundException e) {
        System.err.println(e.getMessage());
    }

    // Test case 2: this is not a known user in the database
    String username2 = "sergio.bianchi";
    String password2 = "Password2";

    try {
        User user2 = loginController.login(username2, password2);
        assertNull(user2);
    } catch (SQLException | ClassNotFoundException e) {
        System.err.println(e.getMessage());
    }

}
```

Figure 18: Snippet LoginTest

I test della classe CustomerTest verificano il corretto funzionamento dei metodi bookRoom(), getBooking(), removeSingleBooking(), findRoom(), removesingleRoom(), addReview(), avgHotel(), avgUser removeReview(), utilizzando strutture note e nuove.

```java
public void findRoomTest() throws SQLException, ClassNotFoundException, ParseException{
    Calendar Start = Calendar.getInstance();
    Calendar End = Calendar.getInstance();
    Start.set( year: 2025,Calendar.JANUARY, date: 1);
    End.set( year: 2025,Calendar.JANUARY, date: 10);

    CustomerDAO customerDAO = new CustomerDAO();
    RoomDAO roomDAO = new RoomDAO();
    roomDAO.addRoom( Hn: "Hotel_Roma", c: 400, space: 10);
    Customer customer = customerDAO.getUser( username: "mario.rossi");
    CustomerProfileController customerProfileController = new CustomerProfileController(customer);

    try {
        ArrayList<Integer> rooms = customerProfileController.findRoom( City: "Roma", space: 10, Start, End, T: "Hotel");
        assertEquals(rooms.size(), actual: 1, delta: 0);
        roomDAO.removeSingleRoom(roomDAO.getlastid());
    } catch (SQLException | ClassNotFoundException E) {
        System.err.println(E.getMessage());
    }
}
```

Figure 19: Snippet findRoomTest

```java
public void BookingOperationsTest() throws SQLException, ClassNotFoundException , ParseException {
    Calendar Start = Calendar.getInstance();
    Calendar End = Calendar.getInstance();
    Start.set( year: 2025,Calendar.JANUARY, date: 1);
    End.set( year: 2025,Calendar.JANUARY, date: 10);

    CustomerDAO customerDAO = new CustomerDAO();
    Customer customer = customerDAO.getUser( username: "mario.rossi");
    CustomerProfileController customerProfileController = new CustomerProfileController(customer);

    BookingDAO bookingDAO = new BookingDAO();

    String s = bookingDAO.calendarToString(Start);
    String e = bookingDAO.calendarToString(End);
    String tot = s + "-" + e;


    try {
        customerProfileController.bookRoom( Id: 1, Start, End, user: "mario.rossi");
        Booking b = bookingDAO.getBooking( Un: "mario.rossi", Hn: "Hotel_Roma", tot);
        assertNotNull(b);
        bookingDAO.removeSingleBooking( Un: "mario.rossi", Hn: "Hotel_Roma", tot);
        Booking c = bookingDAO.getBooking( Un: "mario.rossi", Hn: "Hotel_Roma", tot);
        assertNull(c);
    } catch (SQLException | ClassNotFoundException E) {
        System.err.println(E.getMessage());
    }
}
```

Figure 20: Snippet BookingTest

Il test della classe ManagerTest verifica il corretto funzionamento di addStructure(), getStructure() e removeStructure, tramite una nuova struttura.

```java
public void StructureOperationsTest() throws SQLException, ClassNotFoundException{
    ManagerDAO managerDAO = new ManagerDAO();
    Manager manager = managerDAO.getUser( username: "giul");
    ManagerProfileController managerProfileController = new ManagerProfileController(manager);
    StructureDAO structureDAO = new StructureDAO();

    int [] s = {2, 2, 2};
    int [] r = {2, 3, 4};

    try {
       managerProfileController.addStructure( name: "Hotel_Name",  place: "Firenze",  Manager: "giul",  type: "Hotel", r, s,  c: 25);
       Structure hotel = structureDAO.getStructure( Mn: "giul", Hn: "Hotel_Name");
       assertNotNull(hotel);
       structureDAO.removeStructure( name: "Hotel_Name");
       Structure h = structureDAO.getStructure( Mn: "giul", Hn: "Hotel_Name");
       assertNull(h);

    } catch (SQLException | ClassNotFoundException E) {
        System.err.println(E.getMessage());
    }
}
```

Figure 21: Snippet StructureTest

4.2 DomainModel Test

Sono stati implementati i seguenti test nelle seguenti classi:

- CreditCardTest: payTest() e refundTest().

Questi test verificano il corretto funzionamento dei metodi di pagamento e rimborso della classe CreditCard. Attraverso la simulazione di un caso noto si verifica che i passaggi del pagamento e del rimborso avvengano correttamente.

```java
void refundTest(){

    Customer customer = new Customer( id: 1,  name: "Mario",  surname: "Rossi",  age: 30,  username: "mario.rossi",  password: "Password1",
            paymentMethod: "Credit Card",  cardNumberORuniqueCode: "135895322",  cardExpirationDateORaccountEmail: "2025-12-31",
            cardSecurityCodeORaccountPassword: "473");

    Room room = new Room( P: 4,  i: 1,  C: 50,  h: "Hotel_Name");

    System.out.println("CreditCardTest: refundTest()");

    customer.getCC().refund(room);

    System.out.println();

}
```

Figure 22: Snippet RefundTest

```
void payTest()  {

    Room room = new Room( P: 4,  i: 1,  C: 50,  h: "Hotel_Name");

    System.out.println("CreditCardTest: payTest()");

    String simulatedUserInput = "yes\n473\n";
    System.setIn(new ByteArrayInputStream(simulatedUserInput.getBytes()));

    ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    PrintStream originalOut = System.out;
    System.setOut(new PrintStream(outContent));


    // if the output stream is equal to this string,
    // it means that the payment was successful
    String expectedOutput = "\nPaying for room: " + room.getId() +
            "\n" + "Amount: " + "10.1" + "€ (including " +
            "1.0" + "% commission)" + "\n" +
            "\nConfirm the payment? (yes/no)" + "\n" +
            "Paying with credit card..." + "\n" +
            "Enter your credit card security code:" + "\n" +
            "Connecting to the bank..." + "\n" +
            "Payment successful!";
    System.setOut(originalOut);
    String output = outContent.toString();

    assertFalse(output.contains(expectedOutput));

    System.out.println();
```

Figure 23: Snippet PayTest