

Relazione su Applicativo Java dedicato alla gestione di un'Attività Ristorativa

Poggesi Alessio, Gaetani Lovatelli Michelangelo

November 8, 2024

Contents

| | | |
|----------|---|-----------|
| 1 | Introduzione | 3 |
| 2 | Progettazione | 3 |
| 2.1 | Use case diagram | 3 |
| 2.2 | Use case templates | 4 |
| 2.3 | Class diagram | 12 |
| 3 | Implementazione | 12 |
| 3.1 | Domain Model | 12 |
| 3.1.1 | Customer | 13 |
| 3.1.2 | Reservation | 13 |
| 3.1.3 | Order | 13 |
| 3.1.4 | Table | 14 |
| 3.1.5 | Food | 14 |
| 3.2 | Business Logic | 14 |
| 3.2.1 | CustomerController | 14 |
| 3.2.2 | AdminController | 16 |
| 3.2.3 | BrigadeController | 16 |
| 3.2.4 | OrderSubject e OrderObserver | 17 |
| 3.3 | Orm | 17 |
| 3.3.1 | DatabaseConnect | 17 |
| 3.3.2 | CustomerDAO | 18 |
| 3.3.3 | ReservationDAO | 19 |
| 3.3.4 | OrderDAO | 20 |
| 3.3.5 | MenuDAO | 21 |
| 3.3.6 | TableDAO | 21 |
| 3.4 | Struttura dei Package e File del Progetto | 21 |
| 3.5 | Database | 21 |
| 4 | Testing | 22 |
| 4.1 | Testing Controllers | 23 |
| 4.1.1 | AdminControllerTest | 23 |
| 4.1.2 | CustomerControllerTest | 24 |
| 4.1.3 | BrigadeControllerTest | 24 |

| | | |
|----------|------------------------------|-----------|
| 4.2 | Testing DAOs | 25 |
| 4.2.1 | CustomerDAOTest | 25 |
| 4.2.2 | ReservationDAOTest | 25 |
| 4.2.3 | OrderDAOTest | 25 |
| 4.2.4 | MenuDAOTest | 25 |
| 4.2.5 | TableDAOTest | 26 |
| 5 | Sviluppo tramite IA | 26 |

1 Introduzione

Il programma rappresenta un sistema di gestione per attività ristorativa, sviluppato in Java, mira ad automatizzarne le principali mansioni. Il software offre delle soluzioni sia per il cliente che per l'organizzazione interna del ristorante.

- **Customer:** il cliente ha la possibilità di creare un proprio profilo attraverso il quale potrà effettuare una prenotazione e visualizzare o modificare quelle passate o attive. Per giunta può consultare il menù del ristorante e nell'eventualità eliminare il proprio profilo.
- **Admin:** l'amministratore ha un ruolo centrale nella gestione del ristorante. Tale ruolo potrà visualizzare tutte le prenotazioni, creare l'ordine richiesto dal cliente e farne un resoconto. Inoltre avrà la possibilità di modificare il menù del ristorante.
- **Brigade:** la brigata potrà consultare la lista degli ordini inoltrati dall'admin, e segnalarne la loro ultimazione.

2 Progettazione

2.1 Use case diagram

L'applicativo ha 3 diversi attori: il Customer (cliente), l'Admin (amministratore) e Brigade (brigata di cucina), essi interagiscono indirettamente per automatizzare i processi del ristorante.

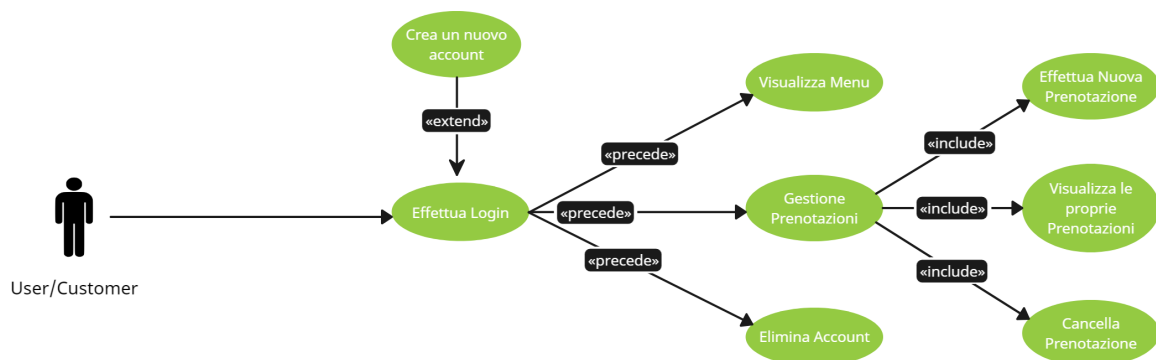


Figure 1: Use Case - Customer

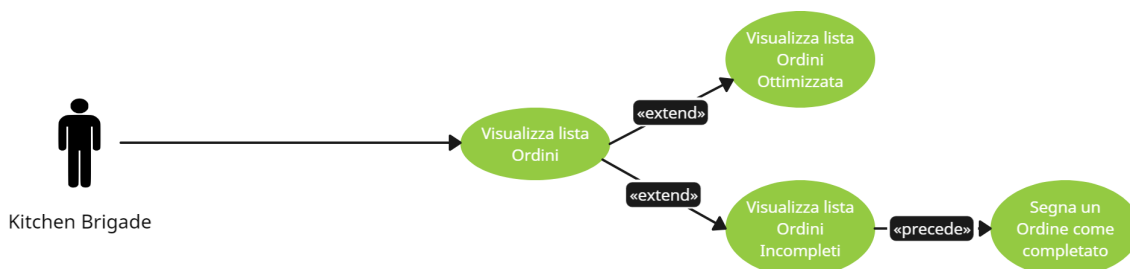


Figure 2: Use Case - Brigade

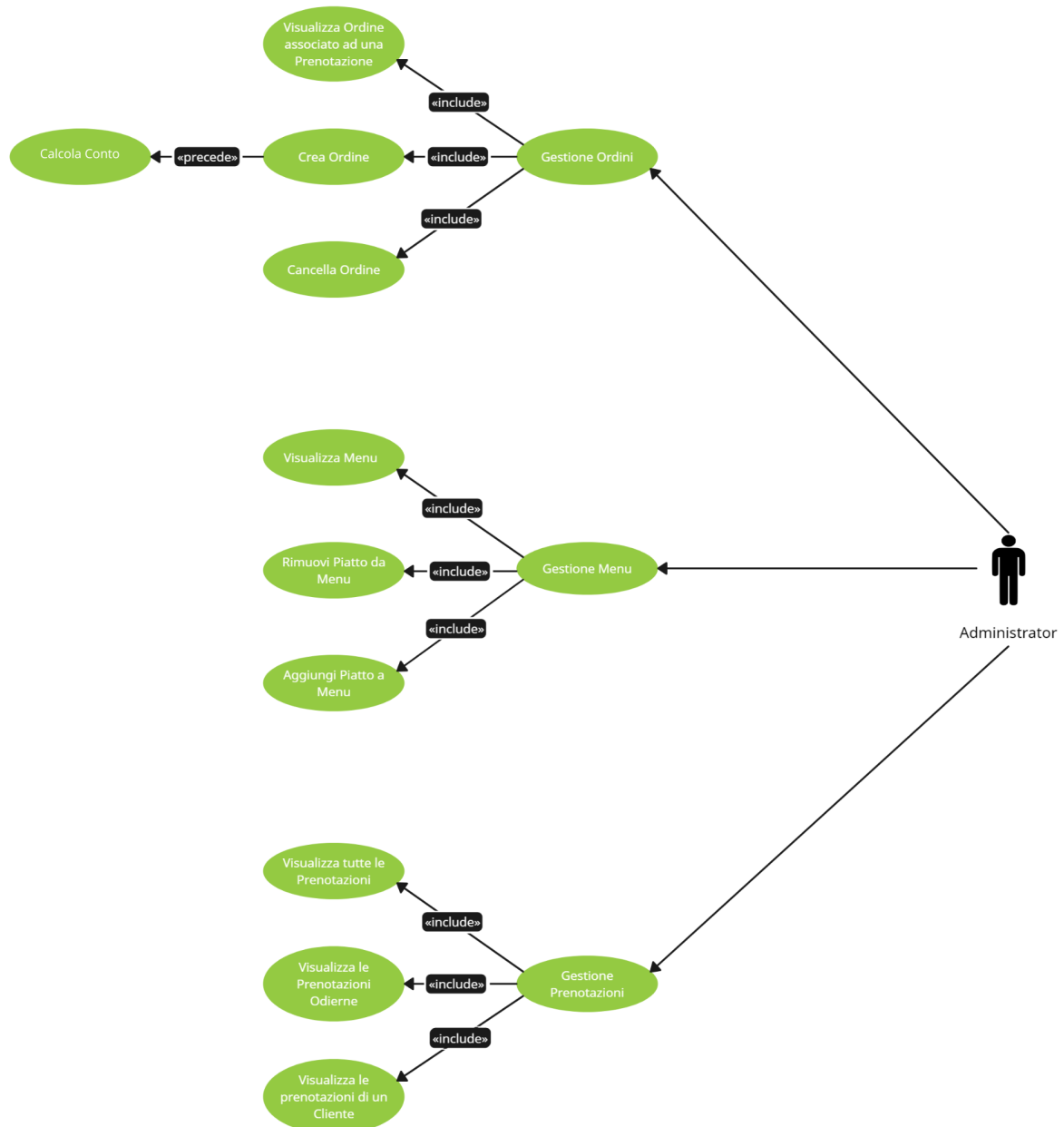


Figure 3: Use Case - Admin

2.2 Use case templates

Di seguito sono riportati i principali casi d'uso del software.

| Use Case 1 | Creare un nuovo account |
|----------------|--|
| Description | Il cliente crea un nuovo account personale con le sue credenziali. |
| Level | User goal |
| Main Actors | Customer |
| Pre-condition | Il cliente non deve possedere già un account. |
| Post-condition | Il cliente possiede un account. |

| | |
|--------------------|--|
| Basic Course | <ol style="list-style-type: none"> 1. L'utente accede all'interfaccia dedicata al cliente premendo "1" 2. L'utente seleziona la voce "Registrati" 3. L'utente inserisce i dati richiesti tra cui "cognome", "nome" e "numero di telefono" |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto, o nel caso il numero di telefono inserito sia già stato utilizzato, il processo di creazione dell'account viene annullato. |

Table 1: Use Case Template 1 permette al cliente di creare un nuovo account.

| | |
|--------------------|---|
| Use Case 2 | Eliminazione account |
| Description | Il cliente elimina il proprio account personale. |
| Level | User goal |
| Main Actors | Customer |
| Pre-condition | Il cliente deve possedere un proprio account personale. |
| Post-condition | Il cliente non possiede più il proprio account. |
| Basic Course | <ol style="list-style-type: none"> 1. L'utente accede all'interfaccia dedicata al cliente premendo "1" 2. L'utente seleziona la voce "Effettua il login" e compila i campi richiesti 3. L'utente seleziona la voce "Elimina Account" 4. L'utente digita "1" per confermare l'operazione |
| Alternative Course | L'utente non possiede un account e quindi non può effettuare il login. L'utente inserisce altri input e non completa l'operazione. |

Table 2: Use Case Template 2 permette al cliente di eliminare il proprio account.

| | |
|-------------------|---|
| Use Case 3 | Effettuare il login |
| Description | Il cliente effettua il login al proprio account personale |
| Level | User goal |
| Main Actors | Customer |
| Pre-condition | Il cliente deve possedere un proprio account personale. |
| Post-condition | Il cliente accede al proprio account e all'interfaccia dedicata |

| | |
|--------------------|--|
| Basic Course | <ol style="list-style-type: none"> 1. L'utente accede all'interfaccia dedicata al cliente premendo "1" 2. L'utente seleziona la voce "Effettua il login" 3. L'utente inserisce i campi relativi al cognome e numero di telefono |
| Alternative Course | Se l'utente non possiede un proprio account o inserisce dei dati non validi, non accede al proprio account e viene riportato all'interfaccia precedente |

Table 3: Use Case Template 3 permette al cliente di accedere al proprio account.

| | |
|--------------------|--|
| Use Case 4 | Effettuare una prenotazione |
| Description | L'utente effettua una prenotazione inserendo data e ora, numero di persone ed eventuali richieste speciali |
| Level | User goal |
| Main Actors | Customer |
| Pre-condition | Il cliente deve possedere un profilo o registrarne uno nuovo |
| Post-condition | Il cliente possiede una prenotazione |
| Basic Course | <ol style="list-style-type: none"> 1. Il cliente accede al proprio profilo o si registra 2. Il cliente interagisce con l'interfaccia scegliendo la voce "Effettua una nuova prenotazione". 3. Il cliente inserisce i dati richiesti per completare la prenotazione 4. Se tutti i valori sono corretti, il sistema verifica la disponibilità del ristorante e accoglie la prenotazione. |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto, sono invalidi o la disponibilità in quelle date è limitata il sistema storna la prenotazione e lo segnala al cliente. |

Table 4: Use Case Template 4 consiste nel permettere ai clienti di effettuare prenotazioni nel ristorante.

| | |
|-------------------|--|
| Use Case 5 | Eliminare una prenotazione |
| Description | Il cliente elimina una prenotazione precedentemente effettuata |
| Level | User goal |
| Main Actors | Customer |

| | |
|--------------------|--|
| Pre-condition | Il cliente deve possedere un proprio account personale e una prenotazione. |
| Post-condition | Il cliente non possiede più una specifica prenotazione. |
| Basic Course | <ol style="list-style-type: none"> 1. L'utente accede all'interfaccia dedicata al cliente premendo "1" 2. L'utente seleziona la voce "Effettua il login" e compila i campi richiesti 3. L'utente seleziona la voce "Cancella una prenotazione" 4. L'utente consulta le proprie prenotazioni e digita l'ID della prenotazione che vuole eliminare 5. L'utente digita "1" per confermare l'operazione |
| Alternative Course | Se il cliente non possiede una prenotazione o inserisce input non validi, l'operazione di eliminazione viene interrotta. |

Table 5: Use Case Template 5 permette al cliente di eliminare una prenotazione.

| | |
|--------------------|--|
| Use Case 6 | Effettuare un ordine |
| Description | L'admin crea un ordine in base alle richieste del cliente e lo associa alla sua prenotazione. |
| Level | User goal |
| Main Actors | Admin |
| Pre-condition | Si deve far riferimento ad una prenotazione esistente. |
| Post-condition | Viene generato un ordine. |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin accede all'interfaccia dedicata e sceglie la voce "Crea un ordine" 2. L'admin inserisce l'ID relativo alla prenotazione del cliente 3. L'admin inserisce uno alla volta l'ID dei piatti richiesti dal cliente e invia 0 per confermare l'ordine |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto o sono invalidi viene segnalato all'admin. |

Table 6: Use Case Template 6 consiste nel creare un ordine composto dai piatti richiesti dal cliente.

| | |
|-------------------|---|
| Use Case 7 | Aggiungi un piatto al menù |
| Description | L'admin aggiunge un nuovo piatto al menù del ristorante |
| Level | User goal |

| | |
|--------------------|--|
| Main Actors | Admin |
| Pre-condition | Accedere all'interfaccia |
| Post-condition | Il menù possiede un nuovo piatto |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Aggiungi piatto al menù" 3. L'admin compila i campi richiesti tra cui "nome del piatto", "descrizione" e "costo" |
| Alternative Course | Se gli input inseriti sono invalidi il piatto non viene aggiunto. |

Table 7: Use Case Template 7 permette all'admin di inserire un nuovo piatto al menù.

| | |
|--------------------|--|
| Use Case 8 | Rimuovi un piatto dal menù |
| Description | L'admin rimuove un piatto dal menù del ristorante |
| Level | User goal |
| Main Actors | Admin |
| Pre-condition | Il menù deve possedere almeno un piatto |
| Post-condition | Il piatto non è più presente nel menù |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Rimuovi un piatto dal menù" 3. L'admin consulta la lista dei piatti presenti nel menù e digita l'ID del piatto che vuole rimuovere |
| Alternative Course | Se gli input inseriti sono invalidi o non corretti l'operazione viene annullata. |

Table 8: Use Case Template 8 permette all'admin di rimuovere un piatto dal menù.

| | |
|-------------------|--|
| Use Case 9 | Visualizza menù |
| Description | L'admin visualizza il menù del ristorante |
| Level | User goal |
| Main Actors | Admin |
| Pre-condition | Deve esistere un menù |
| Post-condition | Viene mostrato il menù del ristorante |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Visualizza il menù" |

| | |
|--------------------|--|
| Alternative Course | Se gli input inseriti sono diversi, non sarà possibile visualizzare il menù. |
|--------------------|--|

Table 9: Use Case Template 9 permette all'admin di consultare il menù.

| | |
|--------------------|---|
| Use Case 10 | Calcola il conto |
| Description | L'admin calcola il conto di una prenotazione |
| Level | User goal |
| Main Actors | Admin |
| Pre-condition | Si deve far riferimento ad una prenotazione esistente |
| Post-condition | Viene calcolato il conto |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Calcola il conto di una prenotazione" 3. L'admin inserisce l'ID della prenotazione di cui vuol calcolare il conto |
| Alternative Course | Se gli input inseriti sono invalidi l'operazione viene annullata. |

Table 10: Use Case Template 10 permette all'admin di calcolare il conto di una prenotazione.

| | |
|--------------------|--|
| Use Case 11 | Visualizza tutte le prenotazioni |
| Description | L'admin visualizza tutte le prenotazioni effettuate dai clienti |
| Level | User goal |
| Main Actors | Admin |
| Pre-condition | Accedere all'interfaccia |
| Post-condition | Vengono mostrate tutte le prenotazioni |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Visualizza tutte le prenotazioni" |
| Alternative Course | Se gli input inseriti sono diversi, non sarà possibile visualizzare le prenotazioni |

Table 11: Use Case Template 11 permette all'admin di consultare tutte le prenotazioni.

| | |
|--------------------|---|
| Use Case 12 | Visualizza tutte le prenotazioni giornaliere |
| Description | L'admin visualizza tutte le prenotazioni effettuate dai clienti per un determinato giorno |
| Level | User goal |
| Main Actors | Admin |

| | |
|--------------------|---|
| Pre-condition | Accedere all'interfaccia |
| Post-condition | Vengono mostrate tutte le prenotazioni effettuate per una data |
| Basic Course | <ol style="list-style-type: none"> 1. L'admin digita "3" e accede all'interfaccia dedicata 2. L'admin sceglie la voce "Visualizza tutte le prenotazioni di oggi" 3. L'admin inserisce una data |
| Alternative Course | Se gli input inseriti sono diversi o invalidi, l'operazione viene interrotta |

Table 12: Use Case Template 12 permette all'admin di consultare tutte le prenotazioni per una data.

| | |
|--------------------|---|
| Use Case 13 | Completare un ordine |
| Description | La brigata segnala il completamento di un ordine |
| Level | User goal |
| Main Actors | Brigade |
| Pre-condition | Deve esserci un ordine esistente. |
| Post-condition | L'ordine viene completato e segnalato all'admin. |
| Basic Course | <ol style="list-style-type: none"> 1. La brigata dall'interfaccia dedicata scegli la voce "Segna un ordine come completo" 2. La brigata inserisce l'ID dell'ordine completato |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto o sono invalidi viene segnalato. |

Table 13: Use Case Template 13 consiste nel segnare un ordine come completo.

| | |
|--------------------|--|
| Use Case 14 | Visualizzare gli ordini |
| Description | La brigata visualizza tutti gli ordini effettuati |
| Level | User goal |
| Main Actors | Brigade |
| Pre-condition | Deve esserci un ordine esistente. |
| Post-condition | Viene mostrata la lista degli ordini |
| Basic Course | <ol style="list-style-type: none"> 1. La brigata digita "2" e accede all'interfaccia dedicata 2. La brigata sceglie la voce "Visualizza la lista degli ordini" |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto o sono invalidi, l'operazione viene interrotta. |

Table 14: Use Case Template 14 permette di visualizzare la lista degli ordini.

| | |
|--------------------|---|
| Use Case 15 | Visualizzare gli ordini incompleti |
| Description | La brigata visualizza tutti gli ordini incompleti |
| Level | User goal |
| Main Actors | Brigade |
| Pre-condition | Deve esserci un ordine esistente incompleto. |
| Post-condition | Viene mostrata la lista degli ordini incompleti |
| Basic Course | <ol style="list-style-type: none"> 1. La brigata digita "2" e accede all'interfaccia dedicata 2. La brigata sceglie la voce "Visualizza la lista degli ordini incompleti" |
| Alternative Course | Se i valori inseriti non hanno il formato richiesto o sono invalidi, l'operazione viene interrotta. |

Table 15: Use Case Template 15 permette di visualizzare la lista degli ordini incompleti.

2.3 Class diagram



Figure 4: Grafico uml

3 Implementazione

3.1 Domain Model

Il Domain Model è il package che contiene le entità del dominio rappresentate nel software. Di seguito una descrizione più specifica delle classi in cui si articola.

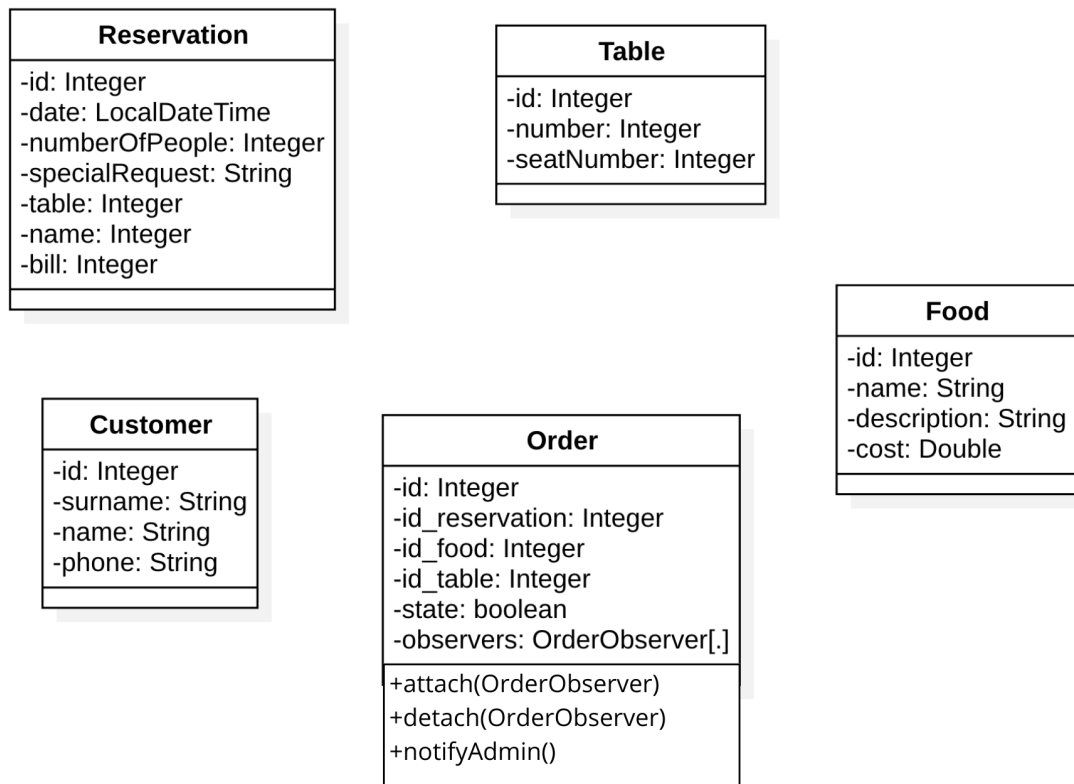


Figure 5: Grafico domain model

3.1.1 Customer

La classe Customer rappresenta un cliente, viene caratterizzato tramite gli attributi "id", "surname", "name" e "phone", che corrispettivamente equivalgono ad un codice univoco identificativo, al cognome, al nome e al numero di telefono del cliente.

3.1.2 Reservation

La classe Reservation rappresenta una prenotazione e possiede gli attributi: "id", "date", "numberOfPeople", "specialRequest", "table", "name", "bill". Vengono impiegati come codice univoco della prenotazione, data e ora, numero di ospiti e un eventuale richiesta speciale, mentre l'attributo "table" e "name" fanno riferimento al tavolo assegnato e al codice identificativo del cliente che ha effettuato la prenotazione, infine "bill" rappresenta il conto totale.

3.1.3 Order

La classe Order rappresenta un ordine. I suoi attributi sono: "id", "id_reservation", "id_food", "id_table", "state" ed "observers", quest'ultimo viene impiegato per implementare il pattern Observer, infatti ogni ordine possiede una lista di osservatori che vengono notificati al proprio cambiamento di stato. Gli altri attributi invece fanno riferimento ai codici univoci della prenotazione, del piatto richiesto e del tavolo. Più precisamente l'ordine è composto dal singolo piatto richiesto ed esso viene legato al tavolo

e alla prenotazione a cui è associato il cliente, quindi per ogni prenotazione esisteranno più ordini con piatti diversi.

3.1.4 Table

La classe Table rappresenta un tavolo del ristorante, caratterizzato dagli attributi "id", codice univoco, "number", il numero del tavolo, e "seatNumbers", il numero di posti che può accogliere il tavolo.

3.1.5 Food

La classe Food rappresenta un piatto del menù, e viene distinto tramite gli attributi "id", "name", nome del piatto, "cost", costo del piatto, e "description", una descrizione del piatto.

3.2 Business Logic

Il package Business Logic contiene le classi controller, una per ogni attore, che implementano gli use case sfruttando le classi del domain model ed i DAO dell'orm. Di seguito una breve descrizione delle classi e dei metodi di cui è composto il package, con approfondimenti sulle funzioni più importanti.

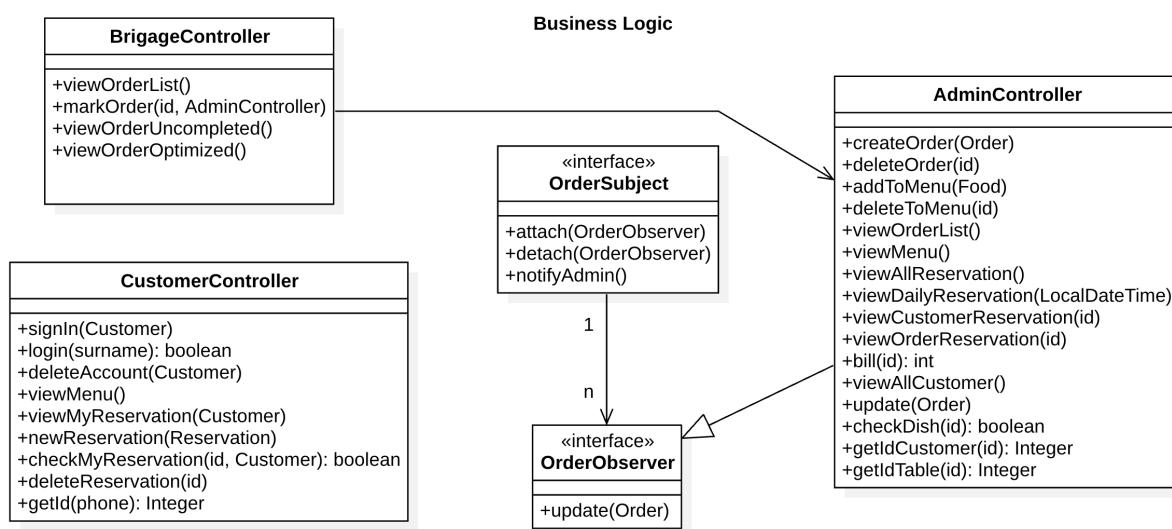


Figure 6: Grafico business logic

3.2.1 CustomerController

La classe CustomerController gestisce diverse operazioni relative ai clienti del ristorante. Il metodo "signIn()" permette ai clienti di registrarsi nel sistema, verificando che il numero di telefono non sia già utilizzato e aggiungendo il cliente al database. Il metodo "login()" consente ai clienti di effettuare il login, verificando la presenza dell'utente nel database. "deleteAccount()" permette al cliente di eliminare il proprio account. "viewMenu()" visualizza il menù del ristorante con tutti i piatti disponibili, mostrando id, nome, prezzo e descrizione del piatto. "viewMyReservation()" consente ai clienti di visualizzare le loro prenotazioni. "newReservation()" permette di effettuare una nuova

prenotazione, verificando la disponibilità dei tavoli. "checkMyReservation()" verifica se una determinata prenotazione appartiene al cliente. "deleteReservation()" elimina una prenotazione. "getId()" restituisce l'ID del cliente basato sul numero di telefono.

Approfondiamo il funzionamento del metodo "newReservation()": la funzione gestisce l'allocazione delle prenotazioni verificando la disponibilità dei posti a sedere nel ristorante. Il funzionamento del metodo si basa su due passaggi principali:

- **Calcolo dei Posti Occupati:** Il metodo utilizza **ReservationDAO** per ottenere la lista dei tavoli già prenotati per la data specificata nella prenotazione. La somma dei posti occupati da queste prenotazioni viene calcolata e memorizzata nella variabile **n_busy**;
- **Verifica della Disponibilità e Allocazione dei Tavoli:** Se la differenza tra i posti totali disponibili (ottenuti da **TableDAO**) e quelli già occupati (**n_busy**) è sufficiente per ospitare gli ospiti della nuova prenotazione, il metodo procede con l'allocazione del tavolo. Viene creata una lista di tavoli liberi, escludendo quelli già prenotati per la stessa data. Viene selezionato un tavolo disponibile per la prenotazione e la prenotazione viene inserita nel database tramite **ReservationDAO**;

In caso contrario, se non ci sono abbastanza posti disponibili, la prenotazione non viene effettuata e viene visualizzato un messaggio di errore. Di seguito il codice della funzione:

```
1 public void newReservation(Reservation reservation) throws
   ClassNotFoundException, SQLException {
2     ReservationDAO reservationDAO = new ReservationDAO();
3     TableDAO tableDAO = new TableDAO();
4
5     int n_busy = 0;
6     for (int i = 0; i < reservationDAO.getTableDate(reservation.getDate()).
       size(); i++) {
7         n_busy += reservationDAO.getTableDate(reservation.getDate()).get(i)
           .getSeatNumbers();
8     }
9
10    if ((tableDAO.getTotalSeat() - n_busy) >= reservation.getNumberOfPerson
       ()) {
11        ArrayList<Table> freeTable = new ArrayList<>();
12        for (int i = 0; i < tableDAO.getAllTable().size(); i++) {
13            if (!reservationDAO.getTableDate(reservation.getDate()).
              contains(tableDAO.getAllTable().get(i))) {
14                freeTable.add(tableDAO.getAllTable().get(i));
15            }
16        }
17        reservation.setTable(freeTable.get(0).getId());
18        reservationDAO.insertReservation(reservation);
19        System.out.println("Booking made!");
20    } else {
21        System.out.println("Booking not available for these dates...");
22    }
23 }
```

Listing 1: Metodo newReservation

3.2.2 AdminController

La classe AdminController implementa l'interfaccia OrderObserver e gestisce diverse operazioni amministrative. Il metodo "createOrder()" permette di creare un nuovo ordine, impostando il suo stato iniziale a false e associandolo all'osservatore. "deleteOrder()" elimina un ordine esistente. "addToMenu()" aggiunge un nuovo piatto al menù, mentre "deleteToMenu()" lo rimuove. "viewOrderList()" visualizza tutti gli ordini, e "viewMenu()" mostra il menù del ristorante. "viewAllReservation()", "viewDailyReservation()" e "viewCustomerReservation()" visualizzano rispettivamente tutte le prenotazioni, le prenotazioni giornaliere e quelle di un cliente specifico. "viewOrderReservation()" mostra gli ordini associati a una prenotazione. Il metodo "bill()" calcola il conto totale di una prenotazione. "viewAllCustomer()" elenca tutti i clienti. "update()" notifica quando un ordine è pronto. "checkDish()" verifica la presenza di un piatto nel menù, mentre "getIdCustomer()" e "getIdTable()" restituiscono rispettivamente l'ID del cliente e del tavolo associati a una prenotazione.

Di seguito vengono riportati alcuni layout dei seguenti metodi:

- "viewMenu()": Il metodo visualizza l'intero menù del ristorante, mostrando id, nome, prezzo e descrizione di ciascun piatto disponibile.

| ID | Piatto | Prezzo | Descrizione |
|-----|--------------------------|--------|-------------------------------------|
| 1 | Bruschetta al Pomodoro | 6.00 | Pane tostato con pomodoro, aglio... |
| 2 | Panzerotti Pugliesi | 9.50 | Calzoni fritti ripieni di... |
| 3 | Minestrone alla Milanese | 7.00 | Zuppa di verdure con fagioli... |
| ... | | | |

- "viewAllReservation()": La funzione mostra tutte le prenotazioni effettuate, includendo id prenotazione, id cliente, data, numero di ospiti e richieste speciali.

| ID prenotazione | ID cliente | Data | Ospiti | Richiesta |
|-----------------|------------|------------------|--------|-------------|
| 61 | 10 | 2024-07-26 20:30 | 5 | fuori |
| 84 | 28 | 2024-04-10 20:30 | 4 | vegetariano |
| 85 | 30 | 2024-04-12 20:30 | 5 | dentro |
| 87 | 12 | 2024-04-18 20:30 | 7 | nulla |
| ... | | | | |

3.2.3 BrigadeController

La classe BrigadeController gestisce le operazioni della brigata di cucina. Il metodo "viewOrderList()" visualizza tutti gli ordini, mostrando ID, nome del piatto, tavolo e stato. "markOrder()" permette di segnare un ordine come completato, notificando l'admin. "viewOrderUncompleted()" visualizza solo gli ordini non completati. "viewOrderOptimized()" mostra prima gli ordini completati e poi quelli non completati, ottimizzando la visualizzazione. Questi metodi permettono alla brigata di gestire efficacemente gli ordini del ristorante. Di seguito viene riportato il layout della funzione "viewOrderList()" che ci permette di concretizzare la concezione dell'ordine:

| ID | Piatto | Tavolo | Stato |
|----|--------|--------|-------|
|----|--------|--------|-------|

| | | | |
|-----|-------------------------|---|-------|
| 10 | Zeppole di San Giuseppe | 1 | true |
| 11 | Panna Cotta | 3 | false |
| 12 | Tiramisu | 1 | true |
| 13 | Frittura di Pesce | 4 | false |
| 15 | Saltimbocca alla Romana | 2 | true |
| ... | | | |

3.2.4 OrderSubject e OrderObserver

Le classi OrderSubject e OrderObserver sono delle interfacce debite alla realizzazione del pattern Observer. L'interfaccia OrderObserver è implementata dalle classi che necessitano di essere notificate quando lo stato di un ordine cambia. Essa contiene il metodo "update()", che viene chiamato per aggiornare l'osservatore con le informazioni dell'ordine modificato. L'interfaccia OrderSubject definisce i metodi per gestire gli osservatori: "attach()" per aggiungere un osservatore, "detach()" per rimuoverne uno e "notifyAdmin()" per notificare tutti gli osservatori registrati. Questa interfaccia è implementata dalle classi che desiderano notificare gli osservatori quando il loro stato cambia. Nel nostro caso solo l'amministratore terrà sotto controllo lo stato di ciascun ordine e appena quest'ultimo cambierà esso verrà notificato.

3.3 Orm

Il package Object-Relational Mapping contiene le classi DAO che offrono all'applicazione servizi inerenti alla persistenza dei dati. Di seguito sono riportate le classi appartenenti al package.

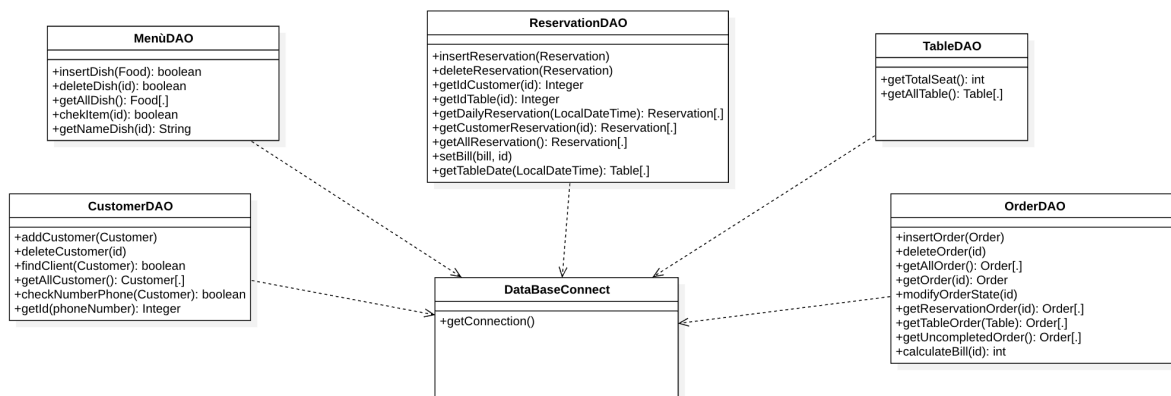


Figure 7: Grafico object-relational mapping

3.3.1 DatabaseConnect

La classe DatabaseConnect si occupa di gestire la connessione al database per l'applicazione. Utilizza le credenziali di accesso e l'URL del database PostgreSQL per stabilire una connessione. Il metodo "getConnection()" è statico e restituisce un oggetto di tipo Connection, garantendo l'unicità della connessione attraverso l'uso del pattern Singleton. Questo metodo lancia eccezioni per gestire eventuali problemi di connessione o di caricamento del driver del database.

3.3.2 CustomerDAO

La classe CustomerDAO gestisce le operazioni CRUD (Create, Read, Update, Delete) per i dati dei clienti nel database. Il metodo "addCustomer()" aggiunge un nuovo cliente utilizzando una query SQL di inserimento. "deleteCustomer()" elimina un cliente, comprese le sue prenotazioni e ordini associati, mediante query SQL di eliminazione. "findClient()" verifica l'esistenza di un cliente basato su cognome e numero di telefono all'interno del database. "getAllCustomer()" recupera tutti i clienti e li memorizza in una lista di oggetti Customer. "checkNumberPhone()" controlla se un numero di telefono è già presente nel database. "getId()" restituisce l'ID del cliente basato sul numero di telefono. "getCustomer()" recupera un cliente specifico in base al suo ID.

Di seguito è riportato il codice delle funzioni principali:

- "addCustomer()": il metodo permette di aggiungere un utente al database insieme al proprio nome, cognome e numero di telefono;

```
1 public void addCustomer(Customer customer) throws ClassNotFoundException,
  SQLException {
2     try {
3         Connection connection = DatabaseConnect.getConnection();
4         String query = "INSERT INTO Cliente (cognome, nome, telefono) VALUES
          (?, ?, ?)";
5         PreparedStatement statement = connection.prepareStatement(query);
6         statement.setString(1, customer.getSurname());
7         statement.setString(2, customer.getName());
8         statement.setString(3, customer.getPhone());
9         statement.executeUpdate();
10        statement.close();
11    } catch (SQLException | ClassNotFoundException e) {
12        e.printStackTrace();
13    }
14 }
```

Listing 2: Metodo addCustomer

- "deleteCustomer()": il metodo elimina definitivamente un utente dal database, la logica di cancellazione è strutturata in modo tale da rispettare le dipendenze tra le tabelle del database, implementando manualmente un'eliminazione a cascata;

```
1 public void deleteCustomer(Integer id) throws ClassNotFoundException,
  SQLException {
2     String deleteOrdersQuery = "DELETE FROM Ordini WHERE id_prenotazione
          IN (SELECT id_prenotazione FROM Prenotazione WHERE cliente_id = ?)
          ";
3     String deleteReservationQuery = "DELETE FROM Prenotazione WHERE
          cliente_id = ?";
4     String deleteClient = "DELETE FROM Cliente WHERE cliente_id = ?";
5     try (Connection connection = DatabaseConnect.getConnection();
6         PreparedStatement deleteOrdersStatement = connection.
          prepareStatement(deleteOrdersQuery);
7         PreparedStatement deleteReservationStatement = connection.
          prepareStatement(deleteReservationQuery);
8         PreparedStatement deleteClientStatement = connection.
          prepareStatement(deleteClient)) {
```

```

9
10     deleteOrdersStatement.setInt(1, id);
11         deleteOrdersStatement.executeUpdate();
12
13     deleteReservationStatement.setInt(1, id);
14     deleteReservationStatement.executeUpdate();
15
16     deleteClientStatement.setInt(1, id);
17     deleteClientStatement.executeUpdate();
18 } catch (SQLException | ClassNotFoundException e ) {
19     e.printStackTrace();
20 }
21 }

```

Listing 3: Metodo deleteCustomer

3.3.3 ReservationDAO

La classe ReservationDAO gestisce le operazioni relative alle prenotazioni nel ristorante. Il metodo "insertReservation()" aggiunge una nuova prenotazione e restituisce l'ID generato. "deleteReservation()" elimina una prenotazione, compresi gli ordini associati. "getIdCustomer()" e "getIdTable()" restituiscono rispettivamente l'ID del cliente e del tavolo associati a una prenotazione. "getDailyReservation()" recupera tutte le prenotazioni di un determinato giorno. "getCustomerReservation()" restituisce le prenotazioni di un cliente specifico. "getAllReservation()" recupera tutte le prenotazioni. "setBill()" aggiorna il conto di una prenotazione. "getTableDate()" restituisce i tavoli occupati in un certo giorno e ora.

Di seguito il codice della funzione "getAllReservation()" che mostra la logica e la sintassi impiegate per recuperare dal database le prenotazioni del ristorante:

```

1 public ArrayList<Reservation> getAllReservation() throws
  ClassNotFoundException, SQLException{
2     ArrayList<Reservation> allReservation= new ArrayList<>();
3     String query = "SELECT * FROM Prenotazione";
4     try (Connection connection = DatabaseConnect.getConnection();
5         PreparedStatement statement = connection.prepareStatement(query)) {
6         ResultSet rs = statement.executeQuery();
7         while(rs.next()) {
8             Integer id=rs.getInt("id_prenotazione");
9             LocalDateTime eventDate = rs.getTimestamp("data_evento").
              toLocalDateTime();
10            LocalDateTime time = rs.getTimestamp("ora_evento").
              toLocalDateTime();
11            Integer n = rs.getInt("n_personone");
12            String note=rs.getString("note");
13            Integer t=rs.getInt("tavolo");
14            Integer client_id=rs.getInt("cliente_id");
15            LocalDateTime dateTime = eventDate.withHour(time.getHour())
16                .withMinute(time.getMinute())
17                .withSecond(time.getSecond());
18            Reservation reservation = new Reservation(id, dateTime, n,
              note,t, client_id);
19            allReservation.add(reservation);
20        }

```

```

21         return allReservation;
22
23     } catch (SQLException | ClassNotFoundException e ) {
24         e.printStackTrace();
25         return null;
26     }
27 }

```

Listing 4: Metodo getAllReservation

3.3.4 OrderDAO

La classe OrderDAO gestisce le operazioni relative agli ordini nel ristorante. Il metodo "insertOrder()" aggiunge un nuovo ordine nel database e restituisce l'ID generato. "deleteOrder()" elimina un ordine esistente. "getAllOrder()" recupera tutti gli ordini e li memorizza in una lista di oggetti Order. "getOrder()" recupera un ordine specifico basato sul suo ID. "modifyOrderState()" aggiorna lo stato di un ordine a completato. "getReservationOrder()" restituisce gli ordini associati a una determinata prenotazione. "getTableOrder()" recupera gli ordini associati a un tavolo specifico. "getUncompletedOrder()" restituisce solo gli ordini non completati. "calculateBill()" calcola il totale del conto per una prenotazione sommando i prezzi dei piatti ordinati.

Approfondiamo il funzionamento del metodo "calculateBill()": la funzione esegue una query sul database che gli permette, tramite l'ID della prenotazione, di recuperare tutti gli ordini legati a quest'ultima. Raccolto l'insieme degli ordini ne somma il prezzo e restituisce il conto totale. Di seguito viene riportato il codice del metodo:

```

1 public int calculateBill(Integer id) throws ClassNotFoundException ,
   SQLException {
2     Integer bill = null;
3     String query = "SELECT o.id_prenotazione , SUM(m.prezzo) AS costo_totale
4                     \r\n" + "FROM Ordini o\r\n"
5                     + "JOIN Menu' m ON o.id_piatto = m.id_piatto\r\n" + "WHERE o.
6                     id_prenotazione = ? \r\n"
7                     + "GROUP BY o.id_prenotazione";
8     try (Connection connection = DatabaseConnect.getConnection();
9         PreparedStatement statement = connection.prepareStatement(query)) {
10        statement.setInt(1, id);
11        ResultSet rs = statement.executeQuery();
12        if (rs.next()) {
13            bill = rs.getInt("costo_totale");
14        }
15    } catch (SQLException | ClassNotFoundException e) {
16        e.printStackTrace();
17        return 0;
18    }
19    if (bill==null)
20        return 0;
21    return bill;
}

```

Listing 5: Metodo calculateBill

3.3.5 MenuDAO

La classe MenuDAO gestisce le operazioni legate al menù del ristorante. Il metodo "insertDish()" aggiunge un nuovo piatto al menù e restituisce l'ID generato. "deleteDish()" elimina un piatto esistente e i relativi ordini associati. "checkItem()" verifica se un piatto è presente nel menù. "getAllDish()" recupera tutti i piatti dal menù e li memorizza in una lista di oggetti Food. "getNameDish()" restituisce il nome di un piatto basato sul suo ID. "getDish()" recupera i dettagli di un piatto specifico.

3.3.6 TableDAO

La classe TableDAO gestisce le operazioni relative ai tavoli nel ristorante. Il metodo "getTotalSeat()" calcola e restituisce il numero totale di posti a sedere sommando i posti di tutti i tavoli nel database. "getAllTable()" recupera tutte le informazioni sui tavoli e le memorizza in una lista di oggetti Table.

3.4 Struttura dei Package e File del Progetto

Di seguito riportiamo la struttura del codice e il link alla repository del progetto:
[link github](#)

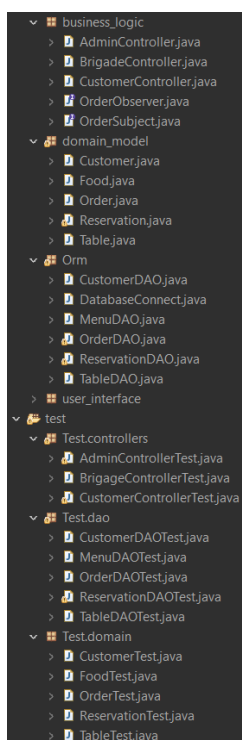


Figure 8: Struttura file progetto

3.5 Database

Per la gestione della persistenza dei dati abbiamo utilizzato un database PostgreSQL, un object relational database system open-source e gratuito.

Di seguito vengono riportati corrispettivamente il diagramma Entity-Relationship e lo schema logico del database.

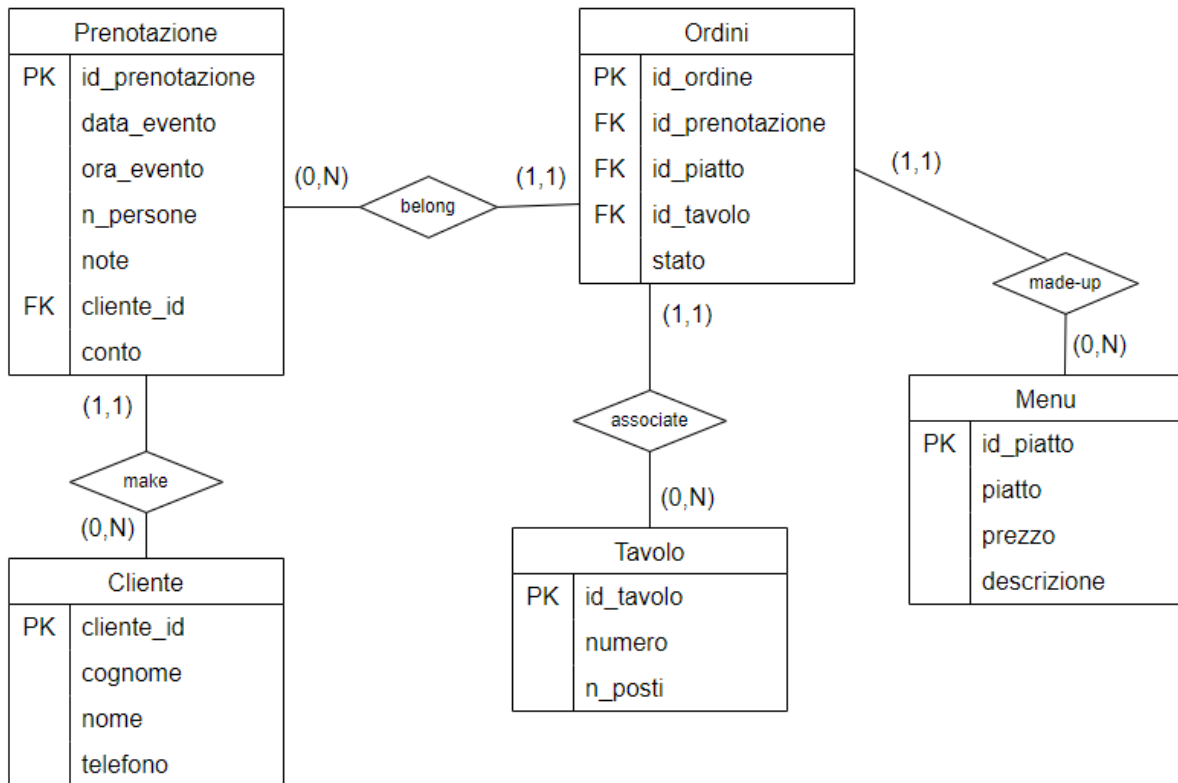


Figure 9: Diagramma Entity-Relationship del database

```

Cliente(PK(cliente_id), cognome, nome, telefono)

Menu(PK(id_piatto), piatto, prezzo, descrizione)

Prenotazione(PK(id_prenotazione), data_evento, ora_evento, n_persone,
             note, cliente_id, conto, FK(cliente_id) REF Cliente)

Ordini(PK(id_ordine), id_prenotazione, id_piatto, id_tavolo, stato,
       FK(id_prenotazione) REF Prenotazione, FK(id_piatto) REF Menu,
       FK(id_tavolo) REF Tavolo)

Tavolo(PK(id_tavolo), numero, n_posti)
  
```

Listing 6: Schema logico del db

4 Testing

Per il testing del codice è stato impiegato il framework JUnit, testando tutti i metodi e le classi del programma. Per garantire un ambiente di test coerente e pulito, sono stati utilizzati i metodi `setUp` e `tearDown` in ogni classe di test. Il metodo `setUp` viene eseguito prima di ogni test per inizializzare le risorse necessarie e preparare l'ambiente di test. Il metodo `tearDown`, invece, viene eseguito dopo ogni test per ripristinare lo stato iniziale e liberare le risorse utilizzate, assicurando che i test successivi non siano influenzati dalle modifiche apportate dai test precedenti. Questo approccio garantisce che anche se i test agiscono sul database, le operazioni di inizializzazione e pulizia manterranno l'integrità e

l'affidabilità dell'ambiente di test.

Di seguito riportiamo l'uso dei metodi "setUp()" e "tearDown()" per la classe di test CustomerDAOTest:

```
@Before
public void setUp() throws ClassNotFoundException, SQLException {
    customerDAO = new CustomerDAO();
    testCustomer = new Customer(null, "surname", "name", "1234567899");
    customerDAO.addCustomer(testCustomer);
    testCustomer.setId(customerDAO.getId(testCustomer.getPhone()));
}

...

@After
public void tearDown() {
    try {
        customerDAO.deleteCustomer(testCustomer.getId());
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

Listing 7: Metodi setUp e tearDown

4.1 Testing Controllers

4.1.1 AdminControllerTest

In questa classe sono presenti i test dei metodi della classe AdminController. Il test del metodo "createOrderTest()" verifica che un ordine venga creato correttamente e che l'ID dell'ordine non sia nullo. Il test di "deleteOrderTest()" verifica che un ordine possa essere eliminato e che non sia più presente nel database. Il test di "addToMenuTest()" verifica che un nuovo piatto venga aggiunto correttamente al menù. Il test di "deleteToMenuTest()" verifica che un piatto possa essere rimosso dal menù e che non sia più presente nel database. Il test di "billTest()" verifica che il totale del conto per una prenotazione venga calcolato correttamente. Il test di "checkDishTest()" verifica che un piatto esista nel menù. I test di "getIdCustomerTest()" e "getIdTableTest()" verificano rispettivamente che gli ID del cliente e del tavolo associati a una prenotazione vengano recuperati correttamente.

Di seguito è riportato il test della funzione "billTest()":

```
@Test
public void billTest() throws Exception {
    int bill = adminController.bill(order.getId_reservation());
    assertEquals(10, bill); // Assumendo un totale di test
    Order order1=new Order(null, reservation.getId(), food.getId(),
        reservation.getTable(), false);
    adminController.createOrder(order1);
    bill = adminController.bill(order1.getId_reservation());
    assertEquals(20, bill);
    adminController.deleteOrder(order1.getId());
    assertEquals(10, adminController.bill(order1.getId_reservation()));
}
```

```
}
```

Listing 8: Metodo `billTest`

4.1.2 CustomerControllerTest

In questa classe sono presenti i test dei metodi della classe `CustomerController`. Il test del metodo `signInTest()` verifica che un cliente possa registrarsi correttamente e che il suo ID venga recuperato correttamente dal database. Il test di `loginTest()` verifica che un cliente registrato possa effettuare il login e che un cliente non registrato non possa farlo. Il test di `deleteAccountTest()` verifica che un account cliente possa essere eliminato correttamente. Il test di `checkMyReservationTest()` verifica che una prenotazione effettuata da un cliente possa essere trovata nel database. Il test di `deleteReservationTest()` verifica che una prenotazione possa essere eliminata correttamente dal sistema. Di seguito sono riportati i test più rilevanti:

- `loginTest()`:

```
@Test
public void loginTest() throws ClassNotFoundException, SQLException {
    boolean loginSuccess = customerController.login(customer);
    assertTrue(loginSuccess);

    Customer noExistingCustomer = new Customer(null, "nonExistent", "
        user", "0000000000");
    boolean loginFailure = customerController.login(
        noExistingCustomer);
    assertFalse(loginFailure);
}
```

Listing 9: Metodo `loginTest`

- `newReservationTest()`:

```
@Test
public void newReservationTest() throws ClassNotFoundException,
    SQLException {
    int i=reservationDAO.getAllReservation().size();
    Reservation reservation1=new Reservation(null, LocalDateTime.of(2001,
        1, 1, 20, 0), 665, "request", 10,customer.getId());
    customerController.newReservation(reservation1);
    int j=reservationDAO.getAllReservation().size();
    assertEquals(i, j);
}
```

Listing 10: Metodo `newReservationTest`

4.1.3 BrigadeControllerTest

In questa classe sono presenti i test dei metodi della classe `BrigadeController`. Il test del metodo `markOrderTest()` verifica che un ordine possa essere marcato come completato. Inizialmente, lo stato dell'ordine viene verificato come non completato (false). Successivamente, il metodo `markOrder()` viene chiamato per cambiare lo stato dell'ordine a completato (true). Infine, viene verificato che lo stato dell'ordine sia stato aggiornato correttamente.

4.2 Testing DAOs

4.2.1 CustomerDAOTest

In questa classe sono presenti i test dei metodi della classe CustomerDAO. Il test del metodo "addCustomerTest()" verifica che un cliente venga aggiunto correttamente al database controllando i suoi attributi. Il test di "deleteCustomerTest()" verifica che un cliente possa essere eliminato correttamente dal database. Il test di "findClientTest()" verifica che un cliente esistente possa essere trovato e uno non esistente no. Il test di "getAllCustomerTest()" verifica che tutti i clienti vengano recuperati correttamente dal database. Il test di "checkNumberPhoneTest()" verifica che il numero di telefono di un cliente esistente sia presente nel database. Il test di "getIdTest()" verifica che l'ID del cliente possa essere recuperato correttamente tramite il numero di telefono. Il test di "getCustomerTest()" verifica che i dettagli di un cliente possano essere recuperati correttamente tramite il suo ID.

4.2.2 ReservationDAOTest

In questa classe sono presenti i test dei metodi della classe ReservationDAO. Il test del metodo "insertReservationTest()" verifica che una prenotazione venga inserita correttamente nel database, controllando che l'ID del cliente associato sia recuperabile. Il test di "deleteReservationTest()" verifica che una prenotazione possa essere eliminata correttamente dal database. Il test di "getAllReservationTest()" verifica che tutte le prenotazioni vengano recuperate correttamente e che la prenotazione di test sia presente nell'elenco. Il test di "setBillTest()" verifica che il conto associato a una prenotazione venga impostato correttamente.

4.2.3 OrderDAOTest

In questa classe sono presenti i test dei metodi della classe OrderDAO. Il test del metodo "insertOrderTest()" verifica che un ordine venga inserito correttamente nel database, controllando che i dettagli dell'ordine siano corretti. Il test di "deleteOrderTest()" verifica che un ordine possa essere eliminato correttamente dal database. Il test di "getAllOrderTest()" verifica che tutti gli ordini vengano recuperati correttamente e che l'ordine di test sia presente nell'elenco. Il test di "modifyOrderStateTest()" verifica che lo stato di un ordine possa essere modificato correttamente a completato. Il test di "calculateBillTest()" verifica che il conto associato a una prenotazione venga calcolato correttamente.

4.2.4 MenuDAOTest

In questa classe sono presenti i test dei metodi della classe MenuDAO. Il test del metodo "insertDishTest()" verifica che un nuovo piatto venga inserito correttamente nel menù, controllando che i suoi attributi siano corretti. Il test di "deleteDishTest()" verifica che un piatto possa essere eliminato correttamente dal menù e che non sia più presente nel database. Il test di "getAllDishTest()" verifica che tutti i piatti vengano recuperati correttamente dal menù e che il piatto di test sia presente nell'elenco.

4.2.5 TableDAOTest

In questa classe sono presenti i test dei metodi della classe TableDAO. Il test del metodo "testGetTotalSeat()" verifica che il numero totale dei posti a sedere venga recuperato correttamente dal database e che sia maggiore di zero. Il test di "testGetAllTable()" verifica che tutti i tavoli vengano recuperati correttamente dal database, controllando che la lista dei tavoli non sia nulla e che contenga almeno un tavolo. Inoltre, verifica che gli attributi di ciascun tavolo (ID, numero del tavolo e numero dei posti a sedere) non siano nulli e siano valori positivi.

5 Sviluppo tramite IA

In questa sezione analizziamo come l'IA sviluppa la stessa tipologia di software da noi proposta. L'IA presa in considerazione è ChatGPT un'intelligenza artificiale progettata da OpenAI.

Il programma permette di interagire attraverso un box di testo nel quale abbiamo digitato:

"Crea un software per la gestione di un ristorante. Il software può essere utilizzato da tre tipologie di client: il cliente, l'amministratore e la brigata di cucina. Di seguito specifico le azioni che devono poter compiere i tre diversi attori:

- 1. Customer: il cliente ha la possibilità di creare un proprio profilo attraverso il quale potrà effettuare una prenotazione e visualizzare o modificare quelle passate o attive. Per giunta può consultare il menu del ristorante e nell'eventualità eliminare il proprio profilo.*
- 2. Admin: l'amministratore ha un ruolo centrale nella gestione del ristorante. Tale ruolo potrà visualizzare tutte le prenotazioni, creare l'ordine richiesto dal cliente e farne un resoconto. Inoltre avrà la possibilità di modificare il menu del ristorante.*
- 3. Brigade: la brigata potrà consultare la lista degli ordini inoltrati dall'admin, e segnalarne la loro ultimazione."*

Di seguito è riportato l'estratto della conversazione:

Per creare un software per la gestione di un ristorante con le funzionalità descritte, possiamo strutturare l'applicazione in base alle tre tipologie di utenti: cliente, amministratore e brigata. Di seguito, viene fornita un'architettura di base e una possibile implementazione.

Il database comprende le seguenti entità: **User**, che memorizza i dettagli del cliente, **Reservation** per gestire le prenotazioni, **Menu** per i piatti del ristorante, **Order** per gli ordini inoltrati dall'amministratore alla brigata, e **Profile** per le informazioni degli utenti.

CustomerController gestisce le operazioni del cliente: prenotazioni, visualizzazione e modifica delle prenotazioni, consultazione ed eliminazione del profilo, e consultazione del menu. **AdminController** consente all'amministratore di visualizzare e modificare le prenotazioni, creare ordini e gestire il menu. **BrigadeController** permette alla brigata di consultare gli ordini e segnalare l'ultimazione.

```
public class User {  
    private int id;
```

```

private String name;
private String email;
private String phoneNumber;

public User(int id, String name, String email, String phoneNumber) {
    this.id = id;
    this.name = name;
    this.email = email;
    this.phoneNumber = phoneNumber;
}

// Getter e Setter
}

```

Listing 11: Classe User

```

import java.util.Date;

public class Reservation {
    private int id;
    private int userId;
    private Date date;
    private int numberOfPeople;

    public Reservation(int id, int userId, Date date, int numberOfPeople) {
        this.id = id;
        this.userId = userId;
        this.date = date;
        this.numberOfPeople = numberOfPeople;
    }

    // Getter e Setter
}

```

Listing 12: Classe Reservation

```

public class MenuItem {
    private int id;
    private String name;
    private double price;

    public MenuItem(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    // Getter e Setter
}

```

Listing 13: Classe MenuItem

```

import java.util.List;

public class Order {
    private int id;
    private int userId;
    private List<MenuItem> items;
}

```

```

private String status; // pending, completed

public Order(int id, int userId, List<MenuItem> items, String status) {
    this.id = id;
    this.userId = userId;
    this.items = items;
    this.status = status;
}

// Getter e Setter
}

```

Listing 14: Classe Order

```

import java.util.List;

public class CustomerController {

    public void createProfile(User user) {
        // Codice per creare il profilo del cliente nel database
    }

    public void makeReservation(int userId, Reservation reservation) {
        // Codice per effettuare una prenotazione
    }

    public List<Reservation> viewReservations(int userId) {
        // Codice per visualizzare le prenotazioni attive
        return null; // Restituisce la lista delle prenotazioni
    }

    public List<MenuItem> consultMenu() {
        // Codice per consultare il menu
        return null;
    }

    public void deleteProfile(int userId) {
        // Codice per eliminare il profilo del cliente
    }
}

```

Listing 15: CustomerController

```

import java.util.List;

public class AdminController {

    public List<Reservation> viewAllReservations() {
        // Codice per visualizzare tutte le prenotazioni
        return null;
    }

    public Order createOrder(int userId, List<MenuItem> items) {
        // Codice per creare un ordine
        return new Order(1, userId, items, "pending");
    }

    public void updateMenu(MenuItem item) {

```

```

        // Codice per aggiornare il menu
    }

    public String generateOrderSummary(Order order) {
        // Codice per generare il resoconto di un ordine
        return "Order summary";
    }
}

```

Listing 16: AdminController

```

import java.util.List;

public class BrigadeController {

    public List<Order> viewPendingOrders() {
        // Codice per consultare gli ordini in sospeso
        return null;
    }

    public void markOrderAsCompleted(int orderId) {
        // Codice per segnare un ordine come completato
    }
}

```

Listing 17: BrigadeController

Il cliente può interagire tramite l'interfaccia fornita dal **CustomerController** per gestire il proprio profilo e le prenotazioni. L'amministratore utilizza il **AdminController** per gestire il menu, le prenotazioni e gli ordini. La brigata consulta gli ordini tramite il **BrigadeController**, che comunica l'avanzamento dell'ordine in tempo reale all'amministratore.