TheKnife

Manuale Tecnico

Lorenzo Radice
Laurea Triennale in Informatica, Università degli Studi dell'Insubria
Matricola: 753252 — Sede Como

Anno Accademico 2024/2025

Contents

1 Progettazione

1.1 Database

L'analisi dei requisiti ha portato alla definizione di uno diagramma Entità-Relazione normalizzato per definire una Base di Dati che gestisse utenti, ristoranti, recensioni e preferiti.

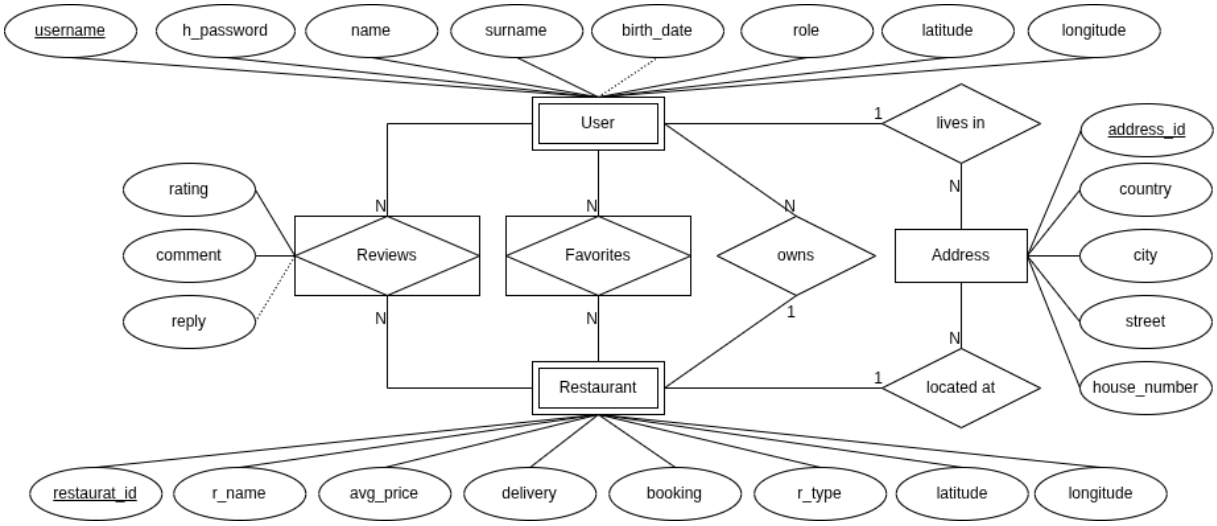

Figure 1: Diagramma ER

1.2 Use-Case Diagram

Per ciascuna tipologia di utente (ospite, cliente, ristoratore) sono stati individuati i casi d'uso principali, dalla ricerca di ristoranti alla gestione di recensioni e risposte.

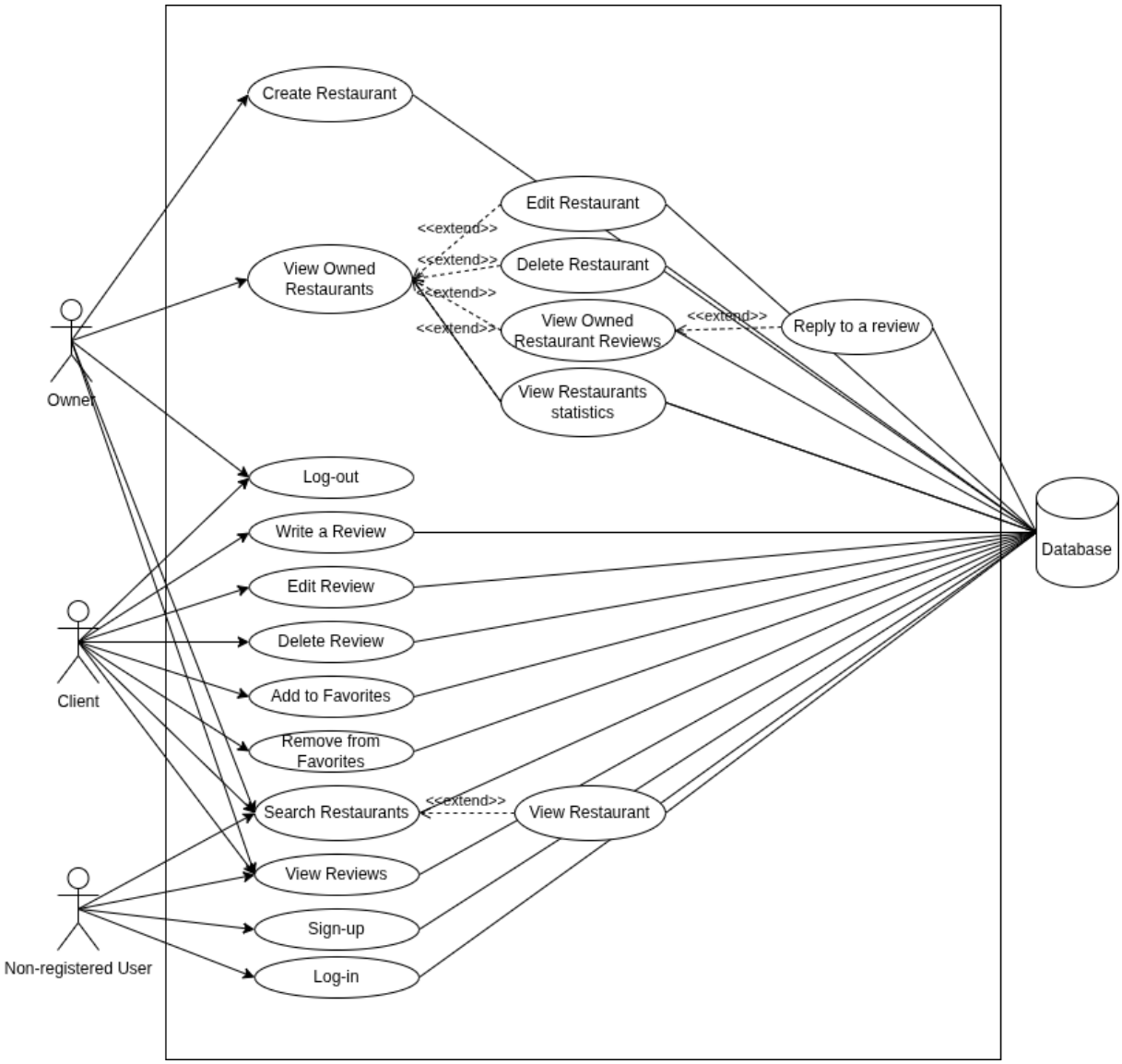

Figure 2: User-Case Diagram

2 Sviluppo

2.1 Architettura

Il sistema utilizza un'architettura client-server basata su RMI (Remote Method Invocation) per la comunicazione tra client e server. Il server si interfaccia con un database PostgreSQL tramite JDBC, gestendo la logica di persistenza tramite DAO (Data Access Object). Il client, sviluppato in JavaFX, implementa il pattern MVC (Model-View-Controller) per la gestione dell'interfaccia utente e delle interazioni.

2.2 Database

2.2.1 Configurazione e connessione

Il sistema utilizza PostgreSQL 13+ con il driver JDBC. Durante la compilazione è richiesto che l'utente *postgre*, con password *postgre*, abbia i permessi per creare un nuovo database e corrispetttivo utente *theknife*.
Il database creato avrà i seguenti parametri di connessione:

- **Database** `theknife_db`
- **User:** `theknife`
- **Password:** `password`
- **Host:** `localhost`
- **Port:** `5432`

2.2.2 Struttura delle tabelle

Di seguito riportata la struttura delle tabelle del database, con le chiavi primarie sottolineate e le chiavi esterne in notazione $\texttt{tabella}^{\texttt{riferimento}}$. Le tabelle sono state create in modo da rispettare le relazioni definite nel diagramma ER.
Per la creazione delle tabelle e il popolamento iniziale del database sono stati utilizzati script SQL.

- **addresses** (address_id, country, city, street, house_number, latitude, longitude)
- **users** (username, h_password, name, surname, birth_date, role, address_id$^{\text{addresses}}$)
- **restaurants** (restaurant_id, r_owner, r_name, avg_price, delivery, booking, r_type, address_id$^{\text{addresses}}$)
- **favorites** (username$^{\text{users}}$, restaurant_id$^{\text{restaurants}}$)
- **reviews** (username$^{\text{users}}$, restaurant_id$^{\text{restaurants}}$, rating, comment, reply)

Le password di tutti gli utenti sono cifrate con Argon2, un algoritmo di hashing sicuro resistente agli attacchi di forza bruta, anche parallelizzati.

2.2.3 Inizializzazione

L'uso di `sql-maven-plugin` permette di gestire le seguenti funzioni.

1. Creazione database e utente (`create_database.sql`).
2. Creazione tabelle (script: `addresses.sql`, ..., `reviews.sql`).
3. Popolamento con dati di esempio (`samples/`).

2.3 Data Access Object - DAO

Si è adottato il pattern DAO per isolare la logica di persistenza.
Le funzionalità garantite da tale pattern sono:

- **UserDAO**: autenticazione, registrazione, gestione profilo.
- **RestaurantDAO**: ricerca avanzata, preferiti, gestione ristoranti.
- **ReviewDAO**: creazione/modifica/cancellazione recensioni e risposte.

UserDAO

- **Autenticazione**: Verifica delle credenziali durante il processo di login, con controllo della password cifrata
- **Registrazione**: Inserimento di nuovi utenti nel sistema con validazione dei dati e cifratura della password
- **Gestione profili**: Operazioni di lettura e aggiornamento delle informazioni utente
- **Controllo ruoli**: Distinzione tra utenti clienti e ristoratori per l'autorizzazione delle operazioni

RestaurantDAO

- **Ricerca parametrica**: Implementazione della funzione `searchRestaurant()` con filtri multipli per:
  - Tipologia di cucina
  - Localizzazione geografica (parametro obbligatorio)
  - Fascia di prezzo
  - Disponibilità servizio delivery
  - Disponibilità prenotazione online
  - Rating minimo
- **Gestione ristoranti**: Creazione dei ristoranti
- **Calcolo metriche**: Aggregazione di dati per rating medio e numero recensioni

ReviewDAO

- **Gestione recensioni**: Operazioni CRUD (Create, Read, Update, Delete) per le recensioni
- **Risposte ristoratori**: Implementazione di `replyToReview()` con controllo autorizzazioni
- **Visualizzazione**: Recupero recensioni per ristorante con `getReviews()`
- **Aggregazioni**: Calcolo rating medio e conteggio recensioni per ristorante

FavoritesDAO

- **Aggiunta preferiti**: Implementazione di `addFavorite()`
- **Rimozione preferiti**: Implementazione di `removeFavorite()`
- **Visualizzazione**: Recupero lista ristoranti preferiti con `getFavorites()`

Prepared Statements Tutte le query utilizzano PreparedStatement per prevenire SQL injection e migliorare le performance attraverso il caching delle query compilate.

2.4 Architettura RMI

Il sistema TheKnife implementa un'architettura distribuita client-server basata su Java RMI (Remote Method Invocation) per gestire la comunicazione tra i client e il server. Questa scelta architetturale permette di separare la logica di presentazione (client) dalla logica di business e accesso ai dati (server), garantendo scalabilit'a e manutenibilit'a del sistema.

2.5 Struttura

Registry RMI Il server TheKnife utilizza il registry RMI per registrare i servizi disponibili ai client. Il registry viene avviato sulla porta standard 1099 e permette ai client di ottenere riferimenti remoti ai servizi tramite lookup per nome.

Interfacce Remote Tutte le interfacce di servizio estendono `java.rmi.Remote` e dichiarano che i metodi possono lanciare `RemoteException`. Questa struttura garantisce la trasparenza della distribuzione per i client che invocano metodi remoti.

2.6 Servizi implementati

`UserService` Autenticazione, registrazione, gestione profilo.

`RestaurantService` Ricerca avanzata, preferiti, gestione ristoranti.

`ReviewService` Creazione/modifica/cancellazione recensioni e risposte.

2.7 Data Transfer Objects - DTO

Tutti i DTO implementano `Serializable` per permettere il trasferimento attraverso RMI. Gli oggetti vengono serializzati automaticamente dal meccanismo RMI durante il trasporto tra client e server.

UserDTO Incapsula i dati dell'utente con campi pubblici per accesso diretto:

- Informazioni di base: `username`, `name`, `surname`
- Credenziali: `password` (solo per operazioni di autenticazione)
- Localizzazione: `address`, `latitude`, `longitude`
- Metadati: `birth_date`, `role` (cliente/ristoratore)

RestaurantDTO Contiene tutte le informazioni del ristorante secondo le specifiche:

- Identificazione: `restaurant_id`, `r_name`
- Localizzazione: `address`, `latitude`, `longitude`
- Caratteristiche: `avg_price`, `avg_rating`, `r_type` (tipologia cucina)
- Servizi: `delivery`, `booking`
- Recensioni: `ArrayList<ReviewDTO> reviews`

SearchCriteriaDTO Implementa il pattern Builder per costruire criteri di ricerca flessibili:

```
SearchCriteriaDTO criteria = SearchCriteriaDTO.builder()
    .coordinates(45.8983, 8.8289)
    .cuisineType(CuisineType.ITALIAN)
    .priceRange(215.0, 40.0)
    .deliveryAvailable(true)
    .minRating(3.0)
    .build();
```

2.8 Architettura del Client

Il modulo client di TheKnife implementa un'architettura basata sul pattern Model-View-Controller (MVC) utilizzando JavaFX come framework per l'interfaccia grafica. Questa scelta architetturale garantisce una separazione netta tra la logica di presentazione, la gestione degli eventi e la comunicazione con i servizi remoti.

2.9 Tecnologie e Framework Utilizzati

JavaFX e FXML L'interfaccia utente è sviluppata utilizzando JavaFX con file FXML per la definizione dichiarativa delle viste. Questa approccio offre diversi vantaggi:

- **Separazione design-logica**: I file FXML permettono di definire la struttura e l'aspetto dell'interfaccia separatamente dalla logica applicativa
- **Manutenibilità**: Le modifiche all'interfaccia possono essere effettuate nei file FXML senza toccare il codice Java
- **Designer-friendly**: I file FXML possono essere creati e modificati con tool grafici come Scene Builder
- **Riusabilità**: Componenti UI possono essere facilmente riutilizzati tra diverse viste

Ogni vista dell'applicazione è definita in un file FXML separato (es. `login-view.fxml`, `search-view.fxml`) che specifica la struttura dei componenti, il layout e le proprietà di styling.

Pattern MVC nel Client L'architettura client implementa il pattern MVC dove:

- **Model**: Rappresentato dai DTO ricevuti dai servizi RMI
- **View**: Definita nei file FXML con componenti JavaFX
- **Controller**: Classi Java che gestiscono la logica di presentazione e l'interazione utente

2.10 Gestione sessione utente

Implementata con un Singleton (`UserSession`) che mantiene stato di login, dati utente e controlli di autorizzazione.

3 Sitografia

- https://fxdocs.github.io/docs/html5/
- https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html
- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html