Università degli Studi dell'Insubria

Laurea triennale in Informatica

Laboratorio Interdisciplinare B

a.a. 2024/2025

TheKnife

Manuale Tecnico

Piattaforma per la Ricerca di Ristoranti: TheKnife

Guida Completa allo sviluppo della soluzione

Autore:
Sergio Enrico Pisoni
Matricola: 755563
Sede: VA

Indice

Capitolo 1

Progettazione della soluzione

In seguito a un attento studio dei requisiti dell'applicazione, ho iniziato a progettare la soluzione dalle basi.

1.1 Progettazione del Database

Di fondamentale importanza è comprendere come strutturare i propri dati, proprio per questo lo sviluppo è partito dalla definizione di uno schema Entità-Relazione, dove ho modellato tutti i dati di cui avrebbe avuto bisogno l'applicazione.

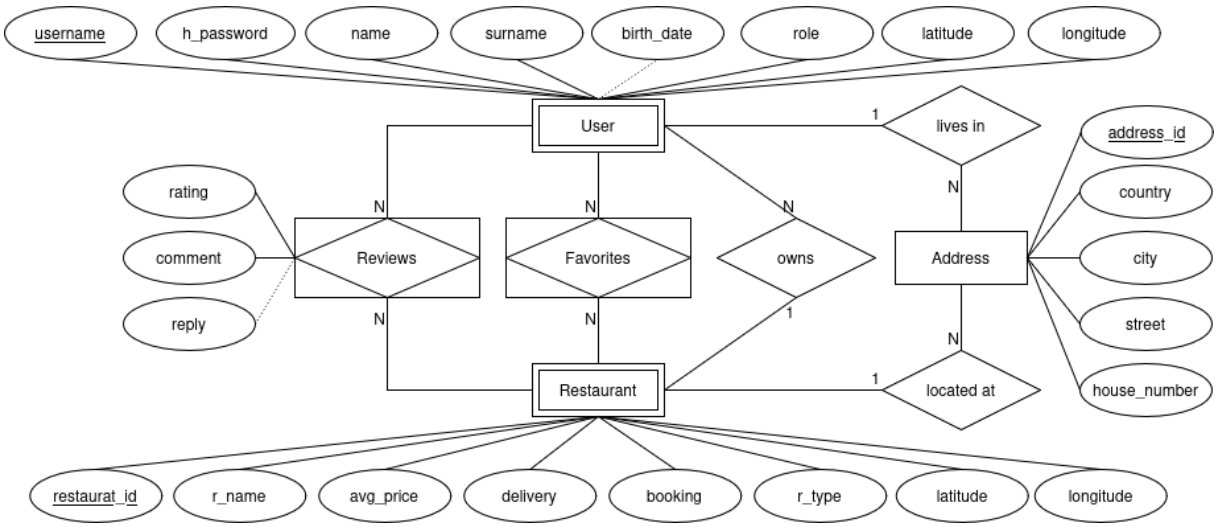

Figura 1.1: Schema ER del database

1.2 Use Case Diagram

Una volta chiarito come sarebbero stati salvati i dati è stato necessario definire come essi venissero utilizzati nel concreto dalle varie tipologie di utente, aggiungendo ulteriore chiarezza su come sarebbe stato necessario sviluppare il tutto.
Da questa necessità è nato il seguente Use Case Diagram con esempi tipo di interazione con il sistema.

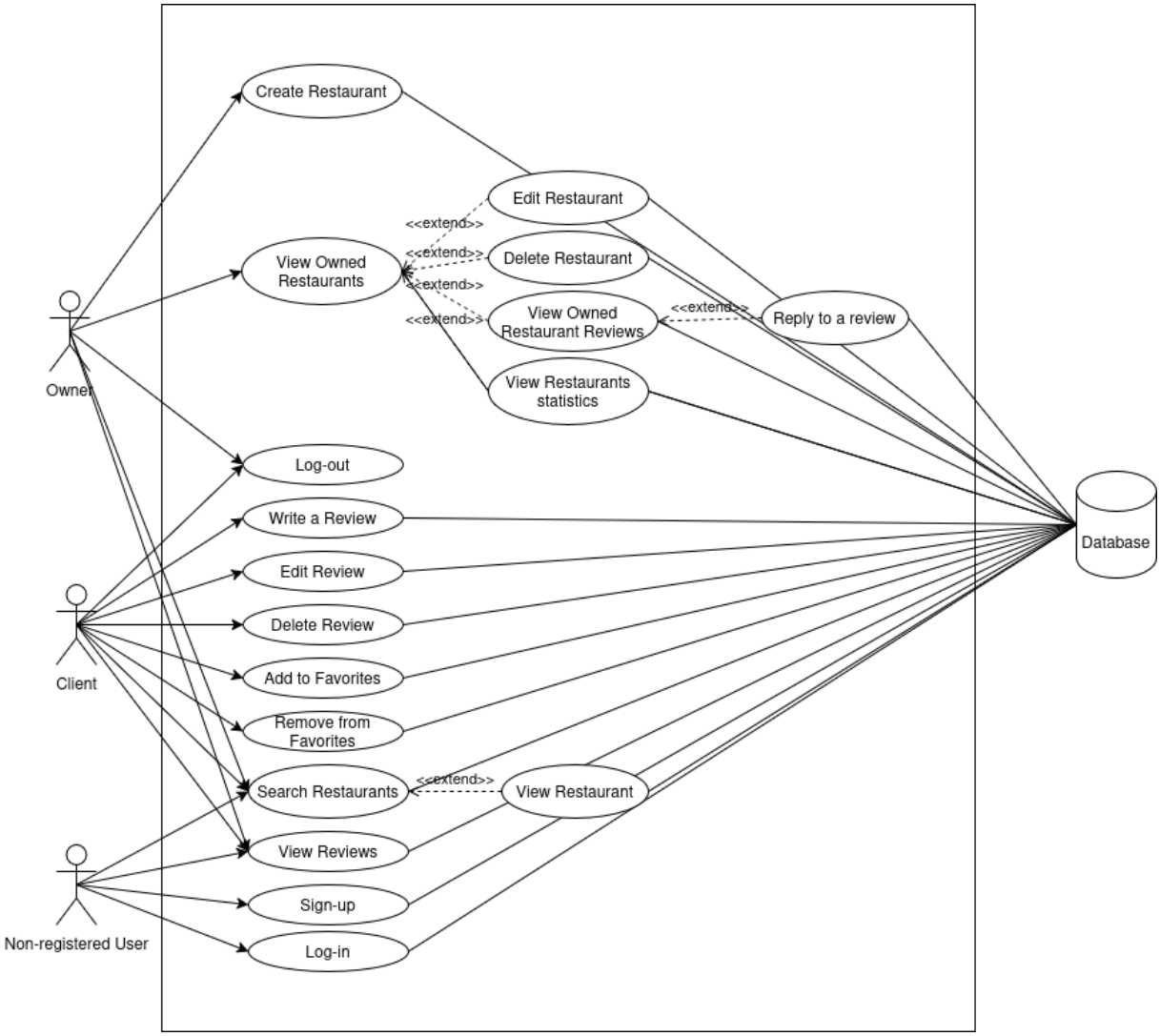

Figura 1.2: Schema ER del database

Dopo aver quindi ben definito la struttura base di dati e interazione ero in grado di capire cosa sarebbe stato necessario per implementarla, in seguito un approfondimento dell'implementazione.

Capitolo 2

Sviluppo della Soluzione

Anche se lo sviluppo reale è partito dallo stabilire la comunicazione fra client e server e poi implementazioni rispettivamene dei due moduli
Per chiarezza e ordine procederò però a parlare dell'implementazione a partire dalle fondamenta, partirò dal database per arrivare all'ui lato client.

2.1 Database

2.1.1 Gestione del Database

Architettura di Connessione

Il sistema TheKnife utilizza PostgreSQL come sistema di gestione del database relazionale (DBMS). La connessione al database viene gestita attraverso il driver JDBC PostgreSQL, che permette l'interazione tra l'applicazione Java server-side e il database.

Configurazione del Database Il database `theknife_db` viene configurato con le seguenti credenziali:

- **Database**: `theknife_db`
- **Utente**: `theknife`
- **Password**: `password`
- **Host**: `localhost`
- **Porta**: `5432`

Struttura delle Classi di Connessione

La gestione della connessione al database è implementata attraverso due classi principali:

DBConnection Classe responsabile della gestione delle connessioni al database. Implementa il pattern Singleton per garantire un controllo centralizzato delle risorse di connessione e prevenire la creazione di multiple istanze non necessarie.

DBConnector Classe di utilità che fornisce metodi specifici per l'apertura, la chiusura e la gestione delle transazioni con il database. Questa classe incapsula la logica di connessione JDBC e fornisce un'interfaccia semplificata per le operazioni di database.

Schema delle Tabelle

Il database è strutturato secondo il seguente schema relazionale:

Tabella `addresses` Memorizza gli indirizzi geografici con coordinate di latitudine e longitudine. Questa tabella serve come riferimento per la localizzazione sia degli utenti che dei ristoranti.

Tabella `users` Contiene le informazioni degli utenti registrati nel sistema, inclusi clienti e ristoratori. Le password sono memorizzate in forma cifrata per garantire la sicurezza.

Tabella `restaurants` Memorizza i dati dei ristoranti secondo le specifiche del progetto:

- Nome del ristorante
- Informazioni geografiche (nazione, città, indirizzo)
- Coordinate di latitudine e longitudine
- Fascia di prezzo media in euro
- Disponibilità servizio delivery (boolean)
- Disponibilità prenotazione online (boolean)
- Tipo di cucina (enumerazione)

Tabella `reviews` Gestisce le recensioni degli utenti sui ristoranti, includendo:

- Messaggio del cliente
- Valutazione in stelle (1-5)
- Eventuale risposta del ristoratore

Tabella `favorites` Implementa la relazione many-to-many tra utenti e ristoranti preferiti, permettendo agli utenti di mantenere una lista personalizzata di ristoranti.

Inizializzazione del Database

Il processo di inizializzazione del database è automatizzato attraverso Maven e il plugin `sql-maven-plugin`. Il processo comprende tre fasi principali:

1. **Creazione Database e Utente**: Esecuzione dello script `create_database.sql` che crea il database `theknife_db` e l'utente dedicato con i privilegi necessari.
2. **Creazione delle Tabelle**: Esecuzione sequenziale degli script SQL per la creazione delle tabelle:

- `addresses.sql`
- `users.sql`
- `restaurants.sql`
- `favorites.sql`
- `reviews.sql`

3. **Popolamento con Dati di Esempio**: Inserimento di dati di test attraverso gli script nella directory `samples/`, permettendo il testing immediato delle funzionalità dell'applicazione.

2.1.2 Data Access Object

Il sistema TheKnife implementa il pattern Data Access Object (DAO) per separare la logica di accesso ai dati dalla logica di business. Questa architettura garantisce una maggiore manutenibilità del codice e facilita eventuali modifiche al livello di persistenza senza impattare la logica applicativa.

Pattern DAO Implementato L'implementazione segue una struttura gerarchica dove ogni entità del dominio (User, Restaurant, Review) ha una corrispondente classe DAO che estende un'interfaccia base comune. Questo approccio garantisce consistenza nelle operazioni CRUD e facilita l'aggiunta di nuove entità.

Struttura delle Classi DAO

UserDAO La classe `UserDAO` gestisce tutte le operazioni relative agli utenti del sistema, implementando le seguenti funzionalità principali:

- **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

Le query SQL implementate includono:

```
SELECT * FROM users WHERE usr_id = ? AND password_hash = ?
INSERT INTO users (usr_id, password_hash, name, surname,
                   address_id, role) VALUES (?, ?, ?, ?, ?, ?)
UPDATE users SET name = ?, surname = ? WHERE usr_id = ?
```

RestaurantDAO La classe `RestaurantDAO` è responsabile della gestione dei ristoranti e implementa le funzionalità di ricerca avanzata richieste dalle specifiche:

- **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
  - Media delle valutazioni
  - Combinazioni dei criteri precedenti
- **Gestione ristoranti**: Operazioni CRUD per i ristoratori
- **Calcolo metriche**: Aggregazione di dati per rating medio e numero recensioni

Esempio di query parametrica per la ricerca:

```
SELECT r.*, AVG(rv.rating) as avg_rating, COUNT(rv.*) as review_count
FROM restaurants r
LEFT JOIN reviews rv ON r.id = rv.restaurant_id
LEFT JOIN addresses a ON r.address_id = a.id
WHERE 1=1
  AND (? IS NULL OR r.cuisine_type = ?)
  AND (? IS NULL OR r.avg_price BETWEEN ? AND ?)
  AND (? IS NULL OR r.delivery = ?)
  AND ST_DWithin(ST_Point(a.longitude, a.latitude),
                 ST_Point(?, ?), ?)
GROUP BY r.id
HAVING (? IS NULL OR AVG(rv.rating) >= ?)
ORDER BY ST_Distance(ST_Point(a.longitude, a.latitude),
                     ST_Point(?, ?))
```

ReviewDAO La classe `ReviewDAO` gestisce il sistema di recensioni e risposte, implementando:

- **Gestione recensioni**: Operazioni per `addReview()`, `modifyReview()`, `deleteReview()`
- **Risposte ristoratori**: Implementazione di `replyToReview()` con controllo autorizzazioni
- **Visualizzazione**: Recupero recensioni per ristorante con `getReviews()`
- **Aggregazioni**: Calcolo rating medio e conteggio recensioni per ristorante

Query per la gestione delle recensioni:

```
INSERT INTO reviews (user_id, restaurant_id, customer_msg,
                     rating) VALUES (?, ?, ?, ?)
UPDATE reviews SET restaurant_reply = ?
WHERE user_id = ? AND restaurant_id = ?
DELETE FROM reviews WHERE user_id = ? AND restaurant_id = ?
```

FavoritesDAO La classe `FavoritesDAO` implementa la gestione della lista preferiti per gli utenti clienti:

- **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.

2.2 Architettura dei Servizi 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 scalabilità e manutenibilità del sistema.

2.2.1 Struttura dell'Architettura RMI

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.

2.2.2 Servizi RMI Implementati

UserService Il servizio `UserService` gestisce tutte le operazioni relative agli utenti del sistema:

- **Autenticazione**: Il metodo `authenticate(String username, String password)` verifica le credenziali dell'utente e restituisce un oggetto `UserDTO` con i dati dell'utente autenticato
- **Registrazione**: Il metodo `register(UserDTO user)` permette la registrazione di nuovi utenti nel sistema, con validazione dei dati e cifratura della password
- **Gestione profilo**: Metodi per l'aggiornamento delle informazioni utente e la gestione delle preferenze

Implementazione del servizio:

```
@Override
public UserDTO authenticate(String username, String password)
    throws RemoteException {
    try {
        UserDAO userDAO = new UserDAO();
        return userDAO.authenticate(username, password);
    } catch (DAOException e) {
        throw new RemoteException("Errore durante l'autenticazione", e);
    }
}
```

RestaurantService Il servizio `RestaurantService` implementa tutte le funzionalità relative ai ristoranti:

- **Ricerca ristoranti**: Il metodo `searchRestaurants(SearchCriteriaDTO criteria)` implementa la funzionalità `cercaRistorante()` delle specifiche, permettendo ricerche parametriche per:
  - Tipologia di cucina
  - Localizzazione geografica (obbligatoria)
  - Fascia di prezzo
  - Disponibilità servizi (delivery, prenotazione online)
  - Media delle valutazioni
  - Combinazioni dei criteri precedenti
- **Gestione preferiti**: Metodi `addFavoriteRestaurant()` e `removeFavoriteRestaurant()` per implementare `aggiungiPreferito()` e `rimuoviPreferito()`
- **Visualizzazione**: Metodi per `getFavoriteRestaurants()`, `getOwnedRestaurants()`, `getReviewedRestaurants()` che implementano rispettivamente `visualizzaPreferiti()`, gestione ristoranti del ristoratore, e ristoranti recensiti
- **Creazione ristoranti**: Il metodo `createRestaurant()` implementa `aggiungiRistorante()` per i ristoratori

ReviewService Il servizio `ReviewService` gestisce il sistema di recensioni e risposte:

- **Gestione recensioni**:
  - `createOrUpdateReview()` implementa `aggiungiRecensione()` e `modificaRecensione()`
  - `deleteReview()` implementa `eliminaRecensione()`
  - `getReviews()` implementa `visualizzaRecensioni()`
- **Risposte ristoratori**: Il metodo `createOrUpdateReply()` implementa `rispostaRecensioni()` permettendo ai ristoratori di rispondere alle recensioni
- **Aggregazioni**: Metodi per calcolare rating medio e numero recensioni tramite `getAverageRating()` e `getReviewCount()`

2.2.3 Data Transfer Objects (DTO)

Serializzazione 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: `usr_id`, `name`, `surname`
- Credenziali: `password_pt` (solo per operazioni di autenticazione)
- Localizzazione: `address`, `latitude`, `longitude`
- Metadati: `bd` (data di nascita), `role` (cliente/ristoratore)

RestaurantDTO Contiene tutte le informazioni del ristorante secondo le specifiche:

- Identificazione: `id`, `name`
- Localizzazione: `nation`, `city`, `address`, `latitude`, `longitude`
- Caratteristiche: `avg_price`, `avg_rating`, `cuisine`
- Servizi: `delivery`, `online_booking`
- Recensioni: `ArrayList<ReviewDTO> reviews`

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

```
SearchCriteriaDTO criteria = SearchCriteriaDTO.builder()
    .coordinates(45.8183, 8.8239)
    .cuisineType(CuisineType.ITALIAN)
    .priceRange(20.0, 50.0)
    .deliveryAvailable(true)
    .minRating(4.0)
    .build();
```

Gestione della Sicurezza

Autorizzazione I servizi RMI implementano controlli di autorizzazione per garantire che:

- Solo utenti autenticati possano accedere a funzionalità riservate
- I ristoratori possano gestire solo i propri ristoranti
- Gli utenti possano modificare solo le proprie recensioni
- Le operazioni sui preferiti siano limitate agli utenti clienti

Avvio del Server Il server RMI viene configurato all'avvio con:

- Creazione del registry RMI
- Istanziazione delle implementazioni dei servizi
- Binding dei servizi nel registry con nomi univoci
- Configurazione delle politiche di sicurezza RMI

Connessione Client I client si connettono ai servizi tramite:

```
Registry registry = LocateRegistry.getRegistry("localhost", 1099);
RestaurantService restaurantService =
    (RestaurantService) registry.lookup("RestaurantService");
```

2.3 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.

2.3.1 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

Struttura dei Controller

Architettura Multi-Controller Il client utilizza un'architettura multi-controller dove ogni vista principale ha un controller dedicato che gestisce:

- L'inizializzazione dei componenti UI
- La gestione degli eventi utente (click, input, selezioni)
- La comunicazione con i servizi RMI per il recupero e l'invio dei dati
- L'aggiornamento dinamico dell'interfaccia in base ai dati ricevuti
- La navigazione tra le diverse viste dell'applicazione

2.3.2 Gestione della Sessione Utente

Pattern Singleton per UserSession La gestione della sessione utente è implementata attraverso la classe `UserSession` che utilizza il pattern Singleton per garantire un'unica istanza globale della sessione durante l'intero ciclo di vita dell'applicazione.

Implementazione del Singleton:

```
public class UserSession {
    private static UserSession instance;
    private UserDTO currentUser;
    private boolean isLoggedIn;

    private UserSession() {
        this.isLoggedIn = false;
        this.currentUser = null;
    }

    public static UserSession getInstance() {
        if (instance == null) {
            instance = new UserSession();
        }
        return instance;
    }
}
```

Funzionalità della Sessione La classe UserSession fornisce le seguenti funzionalità principali:

- **Gestione stato autenticazione**: Mantiene informazioni su login/logout dell'utente
- **Controllo autorizzazioni**: Fornisce metodi per verificare ruoli e permessi
- **Accesso ai dati utente**: Centralizza l'accesso alle informazioni dell'utente corrente
- **Persistenza sessione**: Mantiene la sessione attiva durante la navigazione tra viste

Metodi di Controllo Autorizzazioni UserSession implementa metodi specifici per il controllo delle autorizzazioni:

```
public boolean isOwner() {
    return isLoggedIn && currentUser != null &&
           "owner".equalsIgnoreCase(currentUser.role);
}

public boolean isClient() {
    return isLoggedIn && currentUser != null &&
           "client".equalsIgnoreCase(currentUser.role);
}
```

Questi metodi permettono ai controller di adattare dinamicamente l'interfaccia e le funzionalità disponibili in base al ruolo dell'utente.

2.3.3 Componenti UI Personalizzati

Componenti Riutilizzabili Il client implementa componenti UI personalizzati per elementi ricorrenti:

- **Card components**: Per la visualizzazione di ristoranti e recensioni
- **Search components**: Per i filtri di ricerca avanzata
- **Form components**: Per input dati strutturati

Questi componenti incapsulano sia la logica di presentazione che la gestione degli eventi, promuovendo il riuso del codice.

Responsive Design L'interfaccia implementa principi di responsive design attraverso:

- Layout managers flessibili (VBox, HBox, BorderPane)
- Binding properties per adattamento automatico delle dimensioni
- Gestione di multiple risoluzioni e dimensioni finestra

Capitolo 3

Limiti della soluzione

L'implementazione attuale del sistema TheKnife presenta alcune limitazioni tecniche che potrebbero essere affrontate in versioni future dell'applicazione.

Sicurezza

Gestione Password Le password degli utenti vengono memorizzate in chiaro nel database senza applicare algoritmi di hashing sicuri come bcrypt o Argon2. Questa implementazione rappresenta un rischio di sicurezza significativo in caso di compromissione del database.

Autenticazione Il sistema non implementa meccanismi di autenticazione avanzati come:

- Autenticazione a due fattori (2FA)
- Token di sessione con scadenza
- Politiche di complessità password
- Protezione contro attacchi brute-force

Scalabilità del Database

Connessioni Database L'attuale implementazione utilizza un approccio semplificato per la gestione delle connessioni al database che potrebbe limitare la scalabilità, assenza di una pool di connessioni.

Performance Non sono implementate ottimizzazioni avanzate per le query complesse.

Test incompleti Test parziali se non mancanti e poco rigorosi.

Capitolo 4

Riferimenti

https://fxdocs.github.io/docs/html5/
https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html