The Knife - Manuale Tecnico

Suigi Davide - 759430 - 15/7/2025

Indice

Installazione

Requisiti di sistema

L'applicazione è programmata in java, dunque l'unico requisito è di avere una versione della JRE compatibile con il progetto.

- Versione build usata durante lo sviluppo: Java 1.8.0 build 481

Per quanto riguarda la JDK usata:

```
> java -version
openjdk version "21.0.6" 2025-01-21 LTS
OpneJDK Runtime Environment Temurin-21.0.6+7 (build 21.0.6+7-LTS)
OpneJDK 64-Bit Server VM Temurin-21.0.6+7 (build 21.0.6+7-LTS, mixed mode sharing)
```

Tutte le versioni diverse potrebbero dare errori inattesi dovuti a possibili cambiamenti a livello di metodi o classi.

Setup Ambiente

L'eseguibile JAR è localizzato nella cartella del progetto.
Per la corretta esecuzione, è necessario lasciare l'eseguibile all'interno della cartella.
Questo perché nelle varie classi del progetto, spesso si fa riferimento ai PATH interni alla cartella /data/.

Installazione del programma

Il programma è reperibile sia sul mio OneDrive che su GitHub:

- OneDrive
  - Su OneDrive si trova la cartella "The Knife".
    Scaricare lo zip al suo interno, e scompattarlo per poter usare l'applicazione
- GitHub
  - Una volta premuto il link, verrà automaticamente messo nei download lo zip del progetto.
    Scompattarlo per poter usare l'applicazione

Esecuzione ed Uso

Setup ed avvio

Una volta scaricato il sorgente, per avviarlo bisogna andare tramite CMD/Shell nella cartella dove il progetto è stato scompattato, e avviarlo tramite riga di comando:

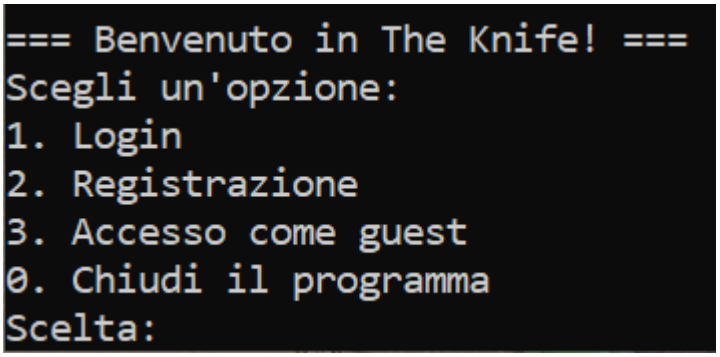
```
> Java -jar TheKnife.jar
```

Se si è dovuto fare delle modifiche, sarà necessario ricompilare il progetto, e ricreare il file JAR per l'esecuzione del codice modificato.
Per la compilazione, seguire i passi:

```
> javac -d out $(find src/main/java -name "*.java") && java -cp out theknife.TheKnife .
> jar cfm TheKnife.jar MANIFEST.MF -C out .
> java -jar TheKnife.jar
```

Uso e funzionalità

L'applicazione si avvierà caricando gli utenti dal file csv e facendo partire un ciclo while (true) per la continua esecuzione, e proponendo un menu di benvenuto:



Il menu di benvenuto è composto da una serie di println, e da una lettura scanner per scegliere l'opzione.
Il valore preso dalla scelta verrà passato in uno Switch Case, che in base alla lettura proporrà all'utente l'opzione scelta.
Nel primo caso, verrà effettuato il login dell'utente: Viene richiesto username e password, che verranno controllati direttamente con tutti gli utenti registrati salvati in utenti.csv.
In caso di esito positivo, l'applicazione rimanderà al menu Cliente o al menu Ristoratore in base al ruolo dell'utente.
Altrimenti avviserà l'utente che uno dei due è errato.

```
switch (scelta) {
    // CASO 1: Login
    case "1":
        //Username e password dell'utente
        System.out.print(s:"\nUsername: ");
        String username = scanner.nextLine();

        String password;
        // Uso della libreria java "Console" per la censura della password durante l'inserimento
        if (console != null){
            char[] pwdArray = console.readPassword(fmt:"Password: ");
            password = new String(pwdArray);
        } else {
            // Fallback se Console non è disponibile (ad esempio in IDE)
            System.out.print(s:"Password: ");
            password = scanner.nextLine();
        }

        // Verifica delle credenziali
        Utente u = LoginManager.login(utenti, username, password);
        if (u != null) {
            System.out.println("Login effettuato come " + u.getRuolo());
            System.out.println("Benvenuto " + u.getNome() + " " + u.getCognome() + "!\n");
            System.out.println("Accesso effettuato come "+ u.getRuolo() + ".");
            if (u.getRuolo() == Ruolo.CLIENTE){
                menuCliente.mostraMenu(scanner, u);
            } else if (u.getRuolo() == Ruolo.RISTORATORE){
                menuRist.mostraMenu(scanner, u);
            }
        } else {
            System.out.println(x:"Username o password errati.");
        }
        break;
```

Nel secondo caso, ovvero la registrazione, la struttura è bene o male la stessa, con la differenza che il codice controlla che lo username non sia già stato preso(e in quel caso esce dal Case 2), che non vengano lasciati campi vuoti, e che il ruolo dell'utente sia o Cliente o Ristoratore.

```java
// CASO 2: Registrazione
case "2":
    //Inserimento dei dati dell'utente
    System.out.print(s:"\nNome: ");
    String nome = scanner.nextLine();
    // Controllo che il campo non sia vuoto
    while (nome.isBlank()){
        System.out.print(s:"Il nome non può essere vuoto. Inserisci il nome:");
        nome = scanner.nextLine();
    }
    System.out.print(s:"Cognome: ");
    String cognome = scanner.nextLine();
    // Controllo che il campo non sia vuoto
    while (cognome.isBlank()){
        System.out.print(s:"Il cognome non può essere vuoto. Inserisci il cognome:");
        cognome = scanner.nextLine();
    }
    System.out.print(s:"Username: ");
    String newUsername = scanner.nextLine();
    // Controllo che il campo non sia vuoto
    while (newUsername.isBlank()){
        System.out.print(s:"Lo username è necessario per l'accesso. Inserisci lo username:");
        newUsername = scanner.nextLine();
    }
    // Controllo se lo username esiste già. Se esiste, esce dalla registrazione
    if (LoginManager.user
        System.out.println
        break;
    }
```

```java
String newpassword;
// Uso della libreria java "Console" per la censura della password durante l'inserimento
if (console != null){
    char[] pwdArray = console.readPassword(fmt:"Password: ");
    newpassword = new String(pwdArray);
} else {
    // Fallback se Console non è disponibile (ad esempio in IDE)
    System.out.print(s:"Password: ");
    newpassword = scanner.nextLine();
}
// Controllo che il campo non sia vuoto
while (newpassword.isBlank()){
    System.out.print(s:"Per una totale sicurezza dell'account, inserisci una password:");
    // Uso della libreria java "Console" per la censura della password durante l'inserimento
    if (console != null){
        char[] pwdArray = console.readPassword(fmt:"Password: ");
        newpassword = new String(pwdArray);
    } else {
        // Fallback se Console non è disponibile (ad esempio in IDE)
        System.out.print(s:"Password: ");
        newpassword = scanner.nextLine();
    }
}
System.out.print(s:"Luogo di domicilio: ");
String luogo = scanner.nextLine();
System.out.print(s:"Ruolo (1=Cliente, 2=Ristoratore): ");
int ruoloScelto = Integer.parseInt(scanner.nextLine());
Ruolo ruolo = ruoloScelto == 2 ? Ruolo.RISTORATORE : Ruolo.CLIENTE;

// Salcataggio nel file CSV indicato a inizio file
Utente nuovo = new Utente(nome, cognome, newUsername, LoginManager.hashPassword(newpassword), dataNascita:null, luogo, ruolo);
utenti.add(nuovo);
UtilsCSV.salvaUtente(pathUtenti, utenti);
System.out.println(x:"Registrazione completata!");
break;
```

Durante l'immissione della password, viene usato "console.readPassword("Password: "). Questo è un metodo della classe java.io.Console, che permette di nascondere la password durante la digitazione a terminale.

Infine, abbiamo gli ultimi 3 casi, ovvero Guest, Chiudi il programma e se viene messo un valore inaspettato.
Per quanto riguarda Guest, non essendo necessari controlli per username o password, il codice carica il menu Guest e le sue interazioni.
Il caso 0 chiude il programma, ponendo un break al ciclo while (true) usato all'inizio per mantenere il programma in esecuzione.
Se viene messo un valore inatteso, verrà selezionato il Default, restituendo un avviso e facendo ripartire il ciclo da zero, ristampando a schermo le opzioni del menu.

```
// CASO 3: Accesso come guest
case "3":
    System.out.println(x:"Accesso come guest. Funzionalità limitate.");
    menuGuest.mostraMenu(scanner);
    break;

// CASO 0: Chiudi il programma
case "0":
    System.out.println(x:"Chiudi il programma.");
    scanner.close();
    return;

// CASO DEFAULT: Scelta non valida
default:
    System.out.println(x:"Scelta non valida.");
}
```

Registrazione

Come visto poco fa, la fase di registrazione fa dei controlli sulle informazioni richieste.
In particolare:

```
// Controllo che il campo non sia vuoto
while (nome.isBlank()){
    System.out.print(s:"Il nome non può essere vuoto. Inserisci il nome:");
    nome = scanner.nextLine();
}
```

Questo controllo viene ripetuto per le prime 5 richieste, ed è necessario per evitare di salvare un utente… senza nome, cognome, username, password o luogo di domicilio.
nome.isBlank() è un metodo che controlla la Stringa digitata durante lo scanner non sia vuota. Se lo è, entra nel ciclo, e lo ripete finché il valore richiesto non sia vuoto.
Per lo username, il controllo avviene invocando il metodo LoginManager.usernameEsiste, che controlla se nel file csv esistono già username come quello appena inserito.
Se esiste, break.

```
// Controllo se lo username esiste già. Se esiste, esce dalla registrazione
if (LoginManager.usernameEsiste(utenti, newUsername)) {
    System.out.println(x:"Username già esistente.");
    break;
}
```

```
/**
 * Verifica se un username esiste già nella lista degli utenti.
 * @param utenti La lista degli utenti registrati.
 * @param username Lo username da verificare.
 * @return true se lo username esiste, false altrimenti.
 */
public static boolean usernameEsiste(List<Utente> utenti, String username) {
    return utenti.stream().anyMatch(u -> u.getUsername().equals(username));
}
```

Per il ruolo scelto, selezionato scegliendo tra 1 o 2, un simil-controllo avviene in Utente.java, dove il numero scelto corrisponde a UTENTE o RISTORATORE.

Infine, le informazioni dell'utente vengono salvate in una lista, e poi nel file utenti.csv.

```
Utente nuovo = new Utente(nome, cognome, newUsername, LoginManager.hashPassword(newpassword), dataNascita:null, luogo, ruolo);
utenti.add(nuovo);
UtilsCSV.salvaUtente(pathUtenti, utenti);
System.out.println(x:"Registrazione completata!");
break;
```

La classe Utente contiene la definizione e i getter/setter necessari.
Inoltre, durante la creazione di un nuovo Utente, viene usata la classe LoginManager, che contiene i metodi utili per il login, o in questo caso, per la cifrazione della password.

```java
/**
 * Cifra la password utilizzando l'algoritmo SHA-256.
 * @param password La password da cifrare.
 * @return La password cifrata in formato esadecimale.
 */
public static String hashPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance(algorithm:"SHA-256");
        byte[] hash = md.digest(password.getBytes());
        StringBuilder sb = new StringBuilder();
        for (byte b : hash) sb.append(String.format(format:"%02x", b));
        return sb.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(message:"Algoritmo di hash non trovato", e);
    }
}
```

Il metodo riceve la password come argomento, e tramite l'uso della libreria java.security.MessageDigest, converte la password in un messaggio cifrato con algoritmo SHA-256.

La classe UtilsCSV contiene i metodi utili alla scrittura e lettura da un file csv.
In questo caso, uso il metodo salvaUtente, che salva le informazioni appena inserite nel file utenti.csv.

```java
/**
 * Salva la lista di utenti in un file CSV.
 * @param path Il percorso del file CSV.
 * @param utenti La lista di utenti da salvare.
 */
public static void salvaUtente(String path, List<Utente> utenti) {
    try (PrintWriter pw = new PrintWriter(new FileWriter(path))) {
        pw.println(x:"Name,Surname,Username,Password,Birthdate,Address,Type");
        for (Utente u : utenti) {
            pw.printf(format:"%s,%s,%s,%s,%s,%s,%s%n",
                    u.getNome(),
                    u.getCognome(),
                    u.getUsername(),
                    u.getPasswordCifrata(),
                    u.getDataNascita() != null ? u.getDataNascita().toString() : "",
                    u.getLuogoDomicilio(),
                    u.getRuolo().toString());
        }
    } catch (IOException e) {
        System.err.println("Errore durante la scrittura del file: " + e.getMessage());
    }
}
```

(getDataNascita viene salvato come "" perchè durante la registrazione non viene mai richiesta. Questa è una mia svista, e ho intenzione di risolvere in un secondo momento)

Login

La fase di login avviene chiedendo username e password.
Così come per la registrazione, la password viene nascosta durante la digitazione su riga di comando, tramite l'uso della libreria java.io.Console.
Il controllo effettuato verifica che lo username e la password coincidano con uno degli utenti salvati nel file csv.

```
    // Verifica delle credenziali
    Utente u = LoginManager.login(utenti, username, password);
    if (u != null) {
        System.out.println("Login effettuato come " + u.getRuolo());
        System.out.println("Benvenuto " + u.getNome() + " " + u.getCognome() + "!\n");
        System.out.println("Accesso effettuato come "+ u.getRuolo() + ".");
        if (u.getRuolo() == Ruolo.CLIENTE){
            menuCliente.mostraMenu(scanner, u);
        } else if (u.getRuolo() == Ruolo.RISTORATORE){
            menuRist.mostraMenu(scanner, u);
        }
    } else {
        System.out.println(x:"Username o password errati.");
    }
    break;
```

In ordine, LoginManager.Login converte la password cifrata in non cifrata, e controlla per ogni riga nel file csv se esistono corrispondenze in username e password, e restituisce l'utente se la verifica ha successo, null altrimenti.

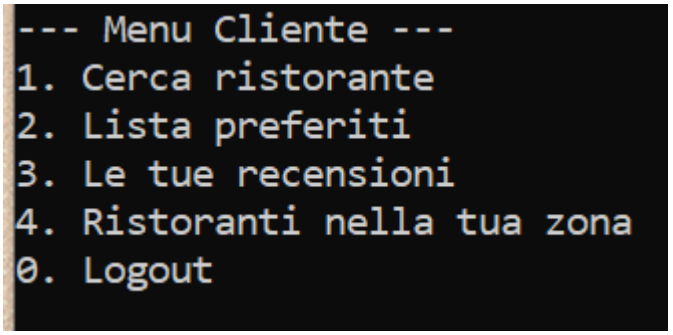
```
/**
 * Verifica le credenziali dell'utente e restituisce l'oggetto Utente se il login ha successo.
 * @param utenti La lista degli utenti registrati.
 * @param username Lo username dell'utente.
 * @param password La password dell'utente.
 * @return L'oggetto Utente se le credenziali sono corrette, altrimenti null.
 */
public static Utente login(List<Utente> utenti, String username, String password) {
    String hashed = hashPassword(password);
    for (Utente u : utenti) {
        if (u.getUsername().equals(username) && u.getPasswordCifrata().equals(hashed)) {
            return u;
        }
    }
    return null;
}
```

Una volta confermato l'accesso dell'utente, il codice controlla quale sia il suo ruolo, e carica il menu appopriato.

```
if (u.getRuolo() == Ruolo.CLIENTE){
    menuCliente.mostraMenu(scanner, u);
} else if (u.getRuolo() == Ruolo.RISTORATORE){
    menuRist.mostraMenu(scanner, u);
}
```

Funzionalità da Cliente

Una volta effettuato l'accesso da cliente, viene caricato il seguente menu:



In particolare, viene caricata la classe menuCliente, che gestisce tutta l'interazione col cliente.
Anche in questo caso, essendo che la classe non deve chiudersi a meno che non lo decida l'utente, ho adottato l'uso di un while (true), con return per il caso 0, ovvero uscita solo dal ciclo e non dall'intero programma.
A ogni iterazione, la prima cosa che verrà stampata è il menu dell'immagine sopra.

```java
public static void mostraMenu(Scanner scanner, Utente utente) {
    while (true) {
        System.out.println(x:"\n--- Menu Cliente ---");
        System.out.println(x:"1. Cerca ristorante");
        System.out.println(x:"2. Lista preferiti");
        System.out.println(x:"3. Le tue recensioni");
        System.out.println(x:"4. Ristoranti nella tua zona");
        System.out.println(x:"0. Logout");
        String scelta = scanner.nextLine();

        switch (scelta) {
```

```java
// CASO 0: Logout
case "0":
    System.out.println(x:"Logout effettuato.");
    return;
```

Come è possibile vedere, il sistema di autenticazione è interamente gestito da un semplice ciclo while con Switch case al suo interno.

Cerca Ristorante

La ricerca avviene caricando un menu di selezione del tipo di filtro

```
// CASO 1: Cerca ristorante
case "1":
    boolean esciRicerca = false;
    while (!esciRicerca) {
        System.out.println(x:"\n=== Ricerca ristorante ===");
        System.out.println(x:"1. Ricerca per luogo");
        System.out.println(x:"2. Ricerca per tipo di cucina");
        System.out.println(x:"3. Ricerca per fascia di prezzo");
        System.out.println(x:"4. Ricerca per asporto");
        System.out.println(x:"5. Ricerca per prenotazione online");
        System.out.println(x:"6. Ricerca per valutazione media");
        System.out.println(x:"7. Ricerca combinata");
        System.out.println(x:"0. Torna al menu principale");
        String sceltaRicerca = scanner.nextLine();

        List<Ristorante> ristoranti = UtilsCSV.caricaRistoranti(UtilsCSV.PATH_RISTORANTI);
        List<Ristorante> risultati = new ArrayList<>();
```

e caricando la lista dei ristoranti dal file csv attraverso il metodo UtilsCSV.caricaRistoranti.
Anche in questo caso, la selezione è gestita da uno Switch case, che in base alla selezione, carica il metodo adatto della classe UtilsRicerca.
In tutti i casi, viene richiesto di inserire il campo del filtro, e dopodichè viene chiamato il metodo del filtro, come ad esempio:

```
case "7":
    System.out.print(s:"Nazione (invio per ignorare): ");
    String naz = scanner.nextLine();
    if (naz.isBlank()) naz = null;
    System.out.print(s:"Città (invio per ignorare): ");
    String cit = scanner.nextLine();
    if (cit.isBlank()) cit = null;
    System.out.print(s:"Tipo cucina (invio per ignorare): ");
    String tipo = scanner.nextLine();
    if (tipo.isBlank()) tipo = null;
    System.out.print(s:"Fascia prezzo (invio per ignorare): ");
    String fascia = scanner.nextLine();
    if (fascia.isBlank()) fascia = null;
    risultati = UtilsRicerca.filtraCombinato(ristoranti, naz, cit, tipo, fascia);
    break;
```

in questo caso vengono richiesti come campi Nazione, Città, Tipo Cucina e fascia prezzo, che vengono passati al metodo filtroCombinato:

```
/**
 * Filtra i ristoranti in base a una combinazione di criteri.
 * @param ristoranti La lista di ristoranti da filtrare.
 * @param nazione La nazione in cui cercare i ristoranti (può essere null).
 * @param citta La città in cui cercare i ristoranti (può essere null).
 * @param tipoCucina Il tipo di cucina desiderato (può essere null).
 * @param fasciaPrezzo La fascia di prezzo desiderata (può essere null).
 * @return Una lista di ristoranti che corrispondono a tutti i criteri specificati.
 */
public static List<Ristorante> filtraCombinato(List<Ristorante> ristoranti, String nazione, String citta, String tipoCucina, String fasciaPrezzo) {
    return ristoranti.stream()
            .filter(r -> (nazione == null || r.getNazione().equalsIgnoreCase(nazione)) &&
                         (citta == null || r.getCitta().equalsIgnoreCase(citta)) &&
                         (tipoCucina == null || r.getTipoCucina().equalsIgnoreCase(tipoCucina)) &&
                         (fasciaPrezzo == null || r.getFasciaPrezzo().equalsIgnoreCase(fasciaPrezzo)))
            .collect(Collectors.toList());
}
```

Una volta eseguita la ricerca, viene effettuato il controllo della lista risultato:

Viene controllato se nella lista risultati esistono dei valori.
Se non esistono, avviso e esce dallo switch case, e torna al menu cliente iniziale.
Altrimenti, visualizza a schermo i ristoranti nella lista, crea una variabile identificativa e chiede quale ristorante vedere nei dettagli.

```
if (!esciRicerca) {
    if (risultati.isEmpty()) {
        System.out.println(x:"Nessun ristorante trovato con i criteri specificati.");
    } else {
        int j = 0;
        System.out.println(x:"Ristoranti trovati:");
        for (Ristorante r : risultati) {
            j++;
            System.out.println(j + ") " + r.getNome() + " - " + r.getCitta() + ", " + r.getNazione() + " - " + r.getTipoCucina() + " - Prezzo: " + r.get
        }
        System.out.print(s:"Seleziona il numero di un ristorante per vedere i dettagli (0 per tornare indietro): ");
        String input = scanner.nextLine();
        try {
            int sceltaRist = Integer.parseInt(input);
            if (sceltaRist > 0 && sceltaRist <= risultati.size()) {
                Ristorante selezionato = risultati.get(sceltaRist - 1);
                System.out.println(x:"\n--- Dettagli Ristorante ---");
                System.out.println("Nome: " + selezionato.getNome());
                System.out.println("Descrizione: " + (selezionato.getDescrizione() != null ? selezionato.getDescrizione() : "Nessuna descrizione"));
                System.out.println("Prezzo: " + selezionato.getFasciaPrezzo());
                System.out.println("Titolare: " + selezionato.getTitolare());
```

nell'else, la variabile j è l'identificatore in questione. Durante il ciclo for successivo, ogni ristorante stampato a schermo viene aggiunto la posizione j, per permettere all'utente di selezionare il ristorante di cui si vogliono vedere i dettagli.

Una volta stampati i dettagli, all'utente viene presentato il menu per le recensioni:

```
boolean esciMenuRistorante = false;
while (!esciMenuRistorante) {
    System.out.println(x:"\n--- Menu Ristorante ---");
    System.out.println(x:"1. Visualizza recensioni");
    System.out.println(x:"2. Crea recensione");
    System.out.println(x:"3. Modifica recensione");
    System.out.println(x:"4. Cancella recensione");
    System.out.println(x:"5. Aggiungi ai preferiti");
    System.out.println(x:"0. Torna indietro");
    String sceltaRistMenu = scanner.nextLine();

    // Carica recensioni e preferiti
    List<Recensione> recensioni = UtilsCSV.caricaRecensioniPerRistorante(selezionato.getNome());
    Recensione miaRecensione = recensioni.stream()
        .filter(r -> r.getAutore().getUsername().equals(utente.getUsername()))
        .findFirst().orElse(other:null);
```

dove:

- esciMenuRistorante è una variabile booleana di controllo. Quando diventa true, esce dal ciclo while.
- Il ciclo while permette più interazioni nello stesso menu.
  Per tornare allo stato precedente, digitare 0 provoca il cambiamento di valore di esciMenuRistorante in true.
- viene caricato il file recensione.csv tramite il metodo UtilsCSV.caricaRecensioniPerRistorante, con parametro il nome del ristorante, in modo che vengano visualizzate solo le recensioni del ristorante selezionato.
- vengono caricate le possibili recensioni dell'utente loggato in miaRecensione.
  Se non esiste una recensione, la variabile sarà null.
  Questo serve perchè un utente non può avere più di una recensione sullo stesso ristorante.

In base alla selezione, il programma caricherà:

1) Tutte le recensioni del ristorante selezionato, e se ci sono, le risposte dei ristoratori. Il caricamento avviene tramite un ciclo che carica recensione, controlla se esiste una risposta, e la stampa a schermo nel caso.

```
switch (sceltaRistMenu) {
    // CASO 1: Visualizza recensioni
    case "1":
        if (recensioni.isEmpty()) {
            System.out.println(x:"Nessuna recensione presente.");
        } else {
            System.out.println(x:"--- Recensioni ---");
            for (Recensione rec : recensioni) {
                System.out.println(rec.getAutore().getUsername() + " (" + rec.getValutazione() + "/5): " + rec.getCommento());
                String risposta = UtilsCSV.getRispostaRecensione(
                    selezionato.getNome(),
                    selezionato.getTitolare(),
                    rec.getAutore().getUsername()
                );
                if (risposta != null) {
                    System.out.println("  Risposta del ristoratore: " + risposta);
                }
            }
        }
        break;
```

2) Le opzioni di creazione, modifica o cancellazione di una recensione, caricando il metodo adeguato.
   Nel caso della modifica, il programma chiede il reinserimento della valutazione e del commento, per poi sostituirli con la recensione precedente.
3) Il messaggio di successo di caricamento del ristorante nella lista preferiti personale. Ciò avviene tramite il metodo UtilsCSV.aggiungiPreferito, con parametri utente e ristorante selezionato. Per la struttura del file preferiti.csv, vedere Scelta delle strutture dati

Lista Preferiti

La seconda selezione carica la lista preferiti dell'utente loggato.

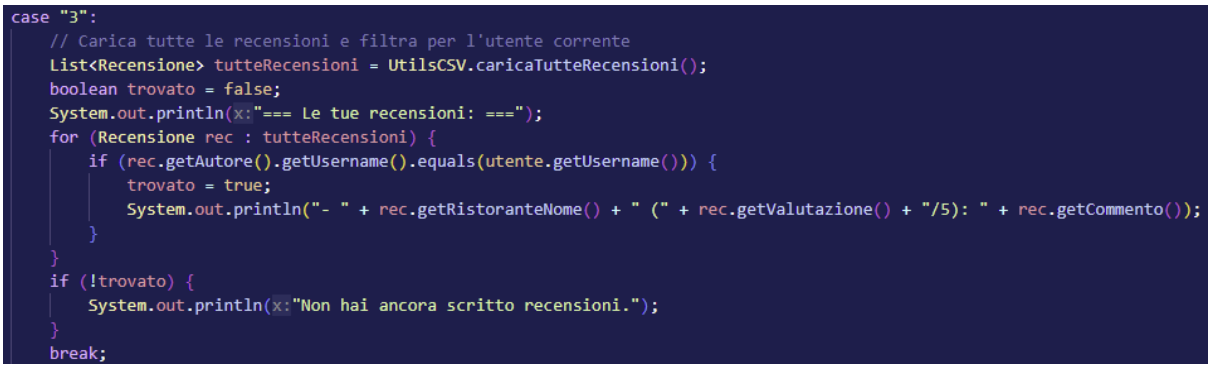
```
case "2":
    System.out.println(x:"=== Lista dei preferiti ===");
    List<String> preferiti = UtilsCSV.caricaPreferiti(utente.getUsername());
    // Se non ci sono preferiti, informa l'utente
    if (preferiti.isEmpty()) {
        System.out.println(x:"Nessun ristorante nei preferiti.");
    }
    // Altrimenti, mostra i preferiti
    else {
        boolean esciPreferiti = false;
        while (!esciPreferiti) {
            System.out.println(x:"I tuoi ristoranti preferiti:");
            for (int k = 0; k < preferiti.size(); k++) {
                System.out.println((k + 1) + ") " + preferiti.get(k));
            }
            System.out.println(x:"0. Torna al menu utente");
```

Per ogni elemento che corrisponde al filtro nel metodo caricaPreferiti, viene stampato a schermo, e numerato come la sezione precedente per i ristoranti.
Una volta caricati i ristoranti preferiti, l'utente può decidere di selezionarne uno, e avere l'opzione di creare, modificare o cancellare una recensione, o di rimuoverlo dai preferiti.
Anche in questo caso, il menu è gestito da uno switch case, per semplificare la gestione della scelta dell'utente.
Una volta che il ristorante viene rimosso dalla lista preferiti, il programma esce dallo switch case e torna al menu cliente

```
// CASO 4: Rimuovi dai preferiti
case "4":
    UtilsCSV.rimuoviPreferito(utente, selezionato);
    System.out.println(x:"Ristorante rimosso dai preferiti!");
    // Aggiorna la lista preferiti dopo la rimozione
    preferiti = UtilsCSV.caricaPreferiti(utente.getUsername());
    esciMenuPrefRist = true;
    break;
```

Le mie recensioni

Questa opzione carica dal file recensioni.csv tutte quelle il cui parametro utente corrisponde a quello attualmente loggato



```
case "3":
    // Carica tutte le recensioni e filtra per l'utente corrente
    List<Recensione> tutteRecensioni = UtilsCSV.caricaTutteRecensioni();
    boolean trovato = false;
    System.out.println(x:"=== Le tue recensioni: ===");
    for (Recensione rec : tutteRecensioni) {
        if (rec.getAutore().getUsername().equals(utente.getUsername())) {
            trovato = true;
            System.out.println("- " + rec.getRistoranteNome() + " (" + rec.getValutazione() + "/5): " + rec.getCommento());
        }
    }
    if (!trovato) {
        System.out.println(x:"Non hai ancora scritto recensioni.");
    }
    break;
```

Altrimenti, mostra un avviso.
Le recensioni sono caricate dal file attraverso l'uso del metodo caricaTutteRecensioni, che poi viene filtrato dal controllo dentro il ciclo for.
Se viene trovata almeno una recensione caricata dall'utente loggato, il valore boolean passa a true, vengono caricate le recensioni, e viene saltato il controllo che stampa l'avviso solo se trovato è false.

Ristoranti nella tua zona

L’ultimo case del menu Cliente carica tutti i ristoranti che corrispondono alla città dell’utente loggato, e li stampa a schermo.

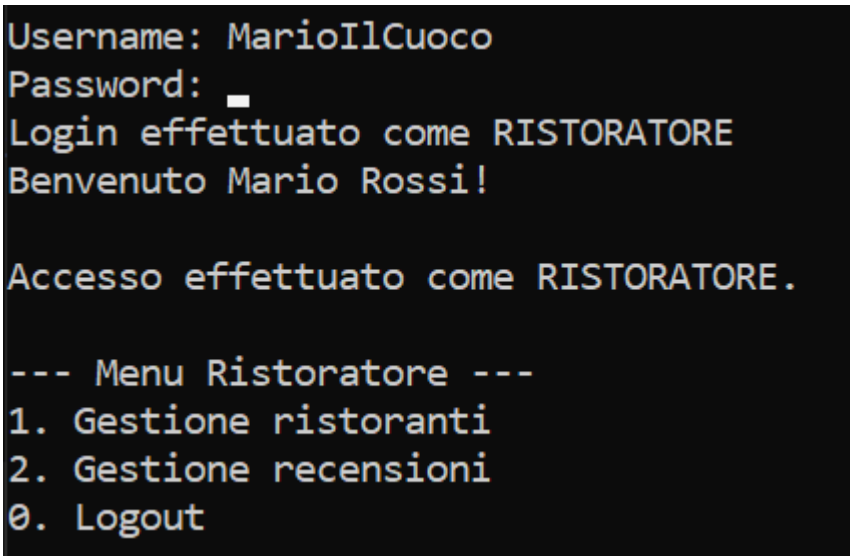
```
case "4":
    // Controlla se l'utente ha impostato un domicilio e lo carica come attrbiuto
    String domicilio = utente.getLuogoDomicilio();
    if (domicilio == null || domicilio.isBlank()) {
        System.out.println(x:"Non hai un domicilio impostato.");
        break;
    }
    // Carica i ristoranti nella zona del domicilio dell'utente
    List<Ristorante> tuttiRistorantiZona = UtilsCSV.caricaRistoranti(UtilsCSV.PATH_RISTORANTI);
    List<Ristorante> filtratiZona = UtilsRicerca.filtraPerLuogo(tuttiRistorantiZona, nazione:null, domicilio, indirizzo:null);

    // Se non sono stati trovati ristoranti, informa l'utente
    if (filtratiZona.isEmpty()) {
        System.out.println("Nessun ristorante trovato nella tua zona (" + domicilio + ").");
    }
```

I ristoranti vengono caricati tramite il metodo caricaRistoranti.
Il filtro avviene tramite il metodo UtilsRicerca.filtraPerLuogo, lasciando null i parametri nazione e indirizzo, per avere la ricerca che corrisponde con il luogo indicato dall’utente durante la registrazione.

Dopodichè vengono visualizzati i ristoranti con il luogo corrispondente a quello dell’utente(altrimenti mostra un avviso che non ha trovato ristoranti), listati come per Cerca Ristorante e con le stesse funzionalità di creazione e modifica recensione, e aggiunta ai preferiti.

Funzionalità dei Ristoratori



Quando l'utente effettua l'accesso, se il controllo del ruolo(menzionato in Login) conferma di essere un ristoratore, il programma carica la classe menuRist.java, che contiene tutti i menu e le funzionalità per un ristoratore.
Anche in questo caso, la classe deve continuare finchè l'utente non effettua il logout(ovvero non digita 0 nel menu soprastante), dunque la gestione dell'accesso dell'utente è gestita da un while (true), contenente lo switch case per il menu dell'immagine.

```
while (true) {
    System.out.println(x:"\n--- Menu Ristoratore ---");
    System.out.println(x:"1. Gestione ristoranti");
    System.out.println(x:"2. Gestione recensioni");
    System.out.println(x:"0. Logout");
    String scelta = scanner.nextLine();

    switch (scelta) {
```

Gestione Ristoranti

Un ristoratore può gestire i ristoranti, nello specifico può:

```
--- Gestione Ristoranti ---
1. Crea nuovo ristorante
2. Visualizza i tuoi ristoranti
3. Cancella un tuo ristorante
0. Torna indietro
```

La creazione di un ristorante avviene similmente alla registrazione di un utente, ovvero vengono richieste le informazioni base per la ricerca sulla piattaforma, come nome del ristorante, nazione, indirizzo, etc.
Una volta digitate tutte le informazioni, il programma le raccoglie e invoca la classe FileWriter per scrivere le informazioni nel file ristoranti.csv, compreso lo username dell'utente ristoratore loggato come titolare.

```
// Salva nel CSV aggiungendo il ristoratore come titolare
try (FileWriter fw = new FileWriter(UtilsCSV.PATH_RISTORANTI, append:true)) {
    fw.write(nome + "," + nazione + "," + citta + "," + indirizzo + "," + lat + "," + lon + "," + fasciaPrezzo + "," + prenotazioni + ","
} catch (IOException e) {
    System.err.println("Errore scrittura ristorante: " + e.getMessage());
}
System.out.println(x:"Ristorante creato!");
break;
```

La visualizzazione dei ristoranti avviene caricando tutti i ristoranti cui titolare corrisponde allo username dell'utente loggato in quel momento.
Se esistono, viene caricato un riepilogo del ristorante, ovvero nome, valutazione media e numero recensioni del ristorante.

```
case "2":
    // Carica i ristoranti dal CSV e filtra per il titolare
    List<Ristorante> tutti = UtilsCSV.caricaRistoranti(path:"data/ristoranti.csv");
    List<Ristorante> miei = new ArrayList<>();
    for (Ristorante r : tutti) {
        if (utente.getUsername().equalsIgnoreCase(r.getTitolare())) {
            miei.add(r);
        }
    }
    // Controlla se l'utente ha ristoranti registrati
    if (miei.isEmpty()) {
        System.out.println(x:"Non hai ristoranti registrati.");
    }
    // Altrimenti, mostra i ristoranti trovati
    else {
        int idx = 1;
        for (Ristorante r : miei) {
            List<Recensione> recs = UtilsCSV.caricaRecensioniPerRistorante(r.getNome());
            double media = recs.stream().mapToInt(Recensione::getValutazione).average().orElse(other:0.0);
            System.out.println(idx++ + ") " + r.getNome() + " | Media valutazioni: " + (recs.isEmpty() ? "N/A" : String.format(format:"%.2f", media)) +
        }
    }
    break;
```

La cancellazione infine avviene caricando tutti i ristoranti che hanno come titolare l'utente loggato, elencandoli con lo stesso sistema di identificazione menzionato in Cerca Ristorante, e digitando il numero identificativo del ristorante che si vuole eliminare.
L'eliminazione è immediata, fare attenzione!

```
String nomeDaCancellare = mieiRist.get(sceltaCanc - 1).getNome();
// Rimuovi dal CSV
List<String> lines = new ArrayList<>();
try (BufferedReader br = new BufferedReader(new FileReader(fileName:"data/ristoranti.csv"))) {
    String line;
    while ((line = br.readLine()) != null) {
        if (!line.startsWith(nomeDaCancellare + ",")) {
            lines.add(line);
        }
    }
```

Gestione Risposte

Il ristoratore ha la possibilità di rispondere alle recensioni dei clienti del proprio ristorante.

```
for (int i = 0; i < tutteRec.size(); i++) {
    Recensione rec = tutteRec.get(i);
    System.out.println((i + 1) + ") " + rec.getRistoranteNome() + " | " + rec.getValutazione() + "/5: " + rec.getCommento() + " (da " + rec.getAu
}
System.out.print(s:"Seleziona una recensione per rispondere (0 per tornare indietro): ");
String inputRec = scanner.nextLine();
int sceltaRec;
try {
    sceltaRec = Integer.parseInt(inputRec);
} catch (NumberFormatException e) {
    System.out.println(x:"Input non valido.");
    break;
}
// Se l'utente sceglie di rispondere a una recensione
if (sceltaRec == 0) break;
if (sceltaRec > 0 && sceltaRec <= tutteRec.size()) {
    Recensione recSelezionata = tutteRec.get(sceltaRec - 1);
    // Controlla se esiste già una risposta
    String rispostaEsistente = UtilsCSV.getRispostaRecensione(
        recSelezionata.getRistoranteNome(),
        utente.getUsername(),
        recSelezionata.getAutore().getUsername()
    );
    // Se esiste, chiede se l'utente vuole sovrascriverla
    if (rispostaEsistente != null) {
```

Il codice controlla se esistono delle recensioni dei ristoranti con titolare corrispondente allo username del ristoratore, e le visualizza a elenco, numerandole con lo stesso algoritmo di identificazione usato precedentemente.

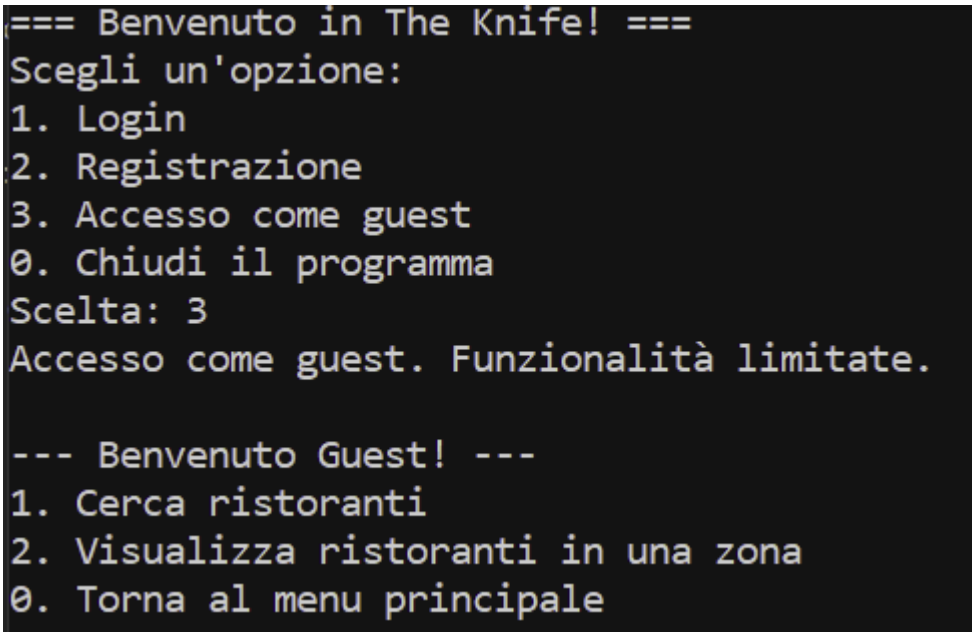
Se il ristoratore sceglie di rispondere a una recensione, viene controllato se esiste già una risposta, e il programma nel caso chiede se vuole sostituirla. Altrimenti chiede di inserire la risposta.
La risposta verrà visualizzata sotto la recensione dell'utente sul ristorante in questione, come mostrato in Cerca Ristorante.

Logout

```
// CASO 0: Logout
case "0":
    System.out.println(x:"Logout effettuato.");
    return;
```

Il sistema di Logout funziona allo stesso modo sia per il cliente che per il ristoratore: essendo le due classi in un ciclo while (true), i menu rispettivi verranno caricati finchè l'utente non sceglie di digitare 0, ovvero la scelta di logout, che è semplicemente un uscita del while (true) attuale, ma non dall'intero programma.

Accesso come Guest



Il programma permette l'accesso come guest.
Il sistema usato è simile al caricamento del menu utente, con la differenza che non ci sono controlli su username e password, e le opzioni sono limitate alla ricerca, il cui funzionamento è lo stesso di Cerca Ristorante, con la differenza che l'utente guest non può accedere al menu recensioni, e vedrà le recensioni di un ristorante in forma anonima, e alla ricerca per zona, il cui funzionamento è simile a Ristoranti nella tua zona, con la differenza che la città su cui filtrare è da indicare al momento, dato che non c'è un utente con cui effettuare il confronto per un eventuale filtro.
Anche in questo caso i ristoranti trovati possono essere selezionati per visualizzarne i dettagli e le recensioni in forma anonima, senza poter effettuare ulteriori azioni.

Dataset di test

Durante lo sviluppo ho creato il file utentiuncensored.txt per la memorizzazione manuale di username e password, dato che le password che vengono salvate nel file utenti.csv sono cifrate.

```
Lista utenti per provare l'accesso(dato che la password è cifrata :) )
Davide Suigi    - Username: dev            - Password: davide
Mario Rossi     - Username: MarioIlCuoco   - Password: marioilcuoco
Tizio Caio      - Username: tiziocaio      - Password: tizio1
Giuseppe Verdi  - Username: giuVerdi       - Password: giuverdi
```

```
Name,Surname,Username,Password,Birthdate,Address,Type
Davide,Suigi,dev,c8a2035c91b566a5dffc34d586408a298abf95ca3d706ead1c677b4979a44c57,,Lecco,CLIENTE
Mario,Rossi,MarioIlCuoco,0aa71a9b28310813eeddd4bf6a76f95b597b69a562d4c16236fe8bcfd1752571,,Lecco,RISTORATORE
tizio,caio,tiziocaio,d5d4d8bc5d7fbf2321f6b2754815d50528c6d60d01d273765e331d9c708bf492,,Roma,CLIENTE
Giuseppe,Verdi,giuVerdi,9d504ed8a5ebd748df71b8deecc79f878fc98bb7a5c6230a8cc9f9fd153d6015,,Milano,CLIENTE
```

Questi sono i due file a confronto, se si vogliono usare degli utenti pronti per l'uso, usare pure questi.
A parte il primo(che sono io), quelli a seguire sono stati creati di pura fantasia, senza una logica specifica.

Inoltre, nel file ristoranti.csv sono stati caricati(tramite intelligenza artificiale) alcuni ristoranti. Da notare, le descrizioni e i titolari potrebbero non corrispondere a quelle effettive.

Scelte Architetturali, algoritmiche e strutture dati

Utilizzo del while (true)
Nella main class e nei file che gestiscono i menu, ogni gestione di accesso è gestita da un while (true).
Usare un loop infinito come metodo di gestione di un menu deriva dalla scelta di usare un “metodo” semplice e diretto nella gestione dell’utente.
Dato che in generale, il programma non richiede mai di accedere con due utenti diversi contemporaneamente, anche di ruoli diversi, o di dover uscire dal programma come conseguenza di una chiusura di un menu, il while (true) permette di gestire i vari switch case successivi e di vedere il menu ogni volta che il ciclo torna all’inizio in maniera semplificata.

Utilizzo degli switch case per la gestione delle opzioni dei menu
Ho pensato che usare uno switch case per gestire i menu piuttosto che una serie di if-else avrebbe lasciato il codice più leggibile per gli esterni al progetto.
Inoltre, posso facilmente gestire i casi output errato(con il Deafult) e l’uscita da un menu, ponendo “0” come opzione finale.

UtilsCSV, UtilsRicerca e LoginManager
Tutte e tre le classi sono servite soprattutto per immagazzinare i metodi generali per quello che servono:

- UtilsCSV contiene tutti i metodi utilizzati per la scrittura e la lettura dei file csv contenuti nella cartella /data/.
  Questa classe è servita soprattutto per evitare la ridondanza nelle varie classi menu, ovunque era necessario caricare i file csv per fare le operazioni.
- UtilsRicerca contiente tutti i metodi filtro per la ricerca dei ristoranti.
  Usata soprattutto nel menuCliente e menuGuest, data la richiesta di cercare i ristoranti dati certe opzioni.
- LoginManager infine contiene i metodi per Login e Cifratura delle password, usate in specifico nella main class.

I file model Utente, Ristorante e Recensione
Questi file sono necessari per la creazione degli oggetti.
Tutti e tre i file contengono la definizione della classe più i rispettivi getter e setter.

I file menu
Questi file sono stati creati per evitare di rendere la main class illeggibile riempiendola di cicli, switch case, menu, variabili etc. ridondanti.
E effettivamente i vari menu gestiscono solo la loro parte di codice, senza mai invadere gli altri menu.

I file CSV

Durante la progettazione, il primo file dato dalla prof.ssa michelin_my_maps.csv è servito come “base” per iniziare da qualche parte.

I file sono strutturati nella seguente maniera:

utenti.csv è stato creato per la memorizzazione degli utenti.

```
Name,Surname,Username,Password,Birthdate,Address,Type
Davide,Suigi,dev,c8a2035c91b566a5dffc34d586408a298abf95ca3d706ead1c677b4979a44c57,,Lecco,CLIENTE
Mario,Rossi,MarioIlCuoco,0aa71a9b28310813eeddd4bf6a76f95b597b69a562d4c16236fe8bcfd1752571,,Lecco,RISTORATORE
tizio,caio,tiziocaio,d5d4d8bc5d7fbf2321f6b2754815d50528c6d60d01d273765e331d9c708bf492,,Roma,CLIENTE
Giuseppe,Verdi,giuVerdi,9d504ed8a5ebd748df71b8deecc79f878fc98bb7a5c6230a8cc9f9fd153d6015,,Milano,CLIENTE
```

ristoranti.csv per la memorizzazione dei ristoranti, sostituendo il michelin_my_maps.csv per averne una mia versione, più semplificato, con solo i valori che mi servivano.

```
Name,Nation,City,Address,Lat,Lon,Price,Delivery,OnlinePrenotation,Cuisine,Description,titolare
Ristorante Bella Italia,Italia,Roma,Viale dei Fori Imperiali,41.8902,12.4923,€€,Yes,Yes,Italian,Traditional Italian cuisine with a modern twist,Giovanni Rossi
Taco Fiesta,Messico,Città del Messico,Paseo de la Reforma,19.4326,-99.1332,$$,Yes,Yes,Mexican,Spicy tacos and burritos,Juan Hernandez
Café de Flore,France,Parigi,Boulevard Saint-Germain,48.8534,2.3499,€€,No,Yes,French,Classic Parisian café with a rich history,Marie Dubois
Pizzeria Napoli,Italia,Napoli,Via Partenope,40.8522,14.2681,€€,Yes,Yes,Italian,Authentic Neapolitan pizza,Luigi Esposito
Nicolin,Italia,Lecco,Località Maggianico,45.835037,9.414134,€€,Yes,Yes,Italian,Family-run restaurant with homemade pasta,Nicolin Family
```

recensioni.csv e risposte.csv servono per la memorizzazione di recensioni e risposte.

```
nomeRistorante,username,valutazione,commento
Ristorante Bella Italia,dev,4,Potevo scegliere tra 11 imperi diversi&#44; e io ovviamente&#44; ho scelto Roma
Nicolin,dev,4,Buon cibo&#44; bella citta
DaMario,dev,1,Descrizione poco credibile
Ristorante Bella Italia,tiziocaio,5,Ottimo ristorante. Il conto potrebbe essere leggermente salato.
```

```
nomeRistorante,ristoratore,cliente,commento
DaMario,MarioIlCuoco,dev,Vieni qua che ti facciamo vedere
```

preferiti.csv infine è per la gestione della lista preferiti dei clienti.

```
username,ristorante
dev,Café de Flore
dev,Nicolin
```

Limitazioni della soluzione sviluppata

Purtroppo l'applicazione sviluppata ha delle limitazioni dovute a problemi di codice e di tempo.

- in Cerca Ristorante, la ricerca tramite "Fascia di prezzo" funziona inserendo un numero di $ o € in base alla posizione del ristorante.
- in Cerca Ristorante selezionare "Aggiungi ai preferiti" sullo stesso ristorante più volte mostrerà un messaggio di successo anche se esiste già nella lista.
  Nonostante questo, quando viene visualizzato Lista preferiti, il ristorante mostrerà correttamente un solo ristorante.
  Questo è dovuto a una limitazione a livello di codice, e potrebbe risultare dannoso alla struttura aggiungere un controllo per questo.
- in Le tue recensioni, l'unico modo per interagire con la propria recensione è cercare il ristorante a cui è stata inserita la recensione e modificarla da lì.
- in Ristoranti nella tua zona, se non esistono ristoranti che sono nella stessa città inserita durante la registrazione o durante la richiesta(come ad esempio per Accesso come Guest), non verranno visualizzati ristoranti.
  Questa è una limitazione a livello di codice, in quanto non esiste un sistema di geolocalizzazione all'interno del progetto.

Sitografia e riferimenti

Javadoc
Il codice è commentato ovunque necessario, e nella cartella doc è stato creato(grazie all'IDE IntelliJ) il documento html contenente il Javadoc del progetto, per una visualizzazione migliore del funzionamento dei metodi presenti nelle classi.

Ambiente di sviluppo
Nonostante abbia appena menzionato IntelliJ per la creazione del documento html per il Javadoc, per l'interezza del progetto ho usato Visual Studio Code.
Questo per una motivazione di comodità e conoscenza dell'IDE, che uso ormai da diversi anni.

Sitografia
Durante lo sviluppo, ho usato i seguenti siti:

- Stackoverflow per la lettura/scrittura da file csv
- W3School per la documentazione Java
- Copilot per l'occasionale controllo del codice e debug in caso di errori inspiegabili