

UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA
FACOLTÀ DI INFORMATICA (CORSO DI LAUREA TRIENNALE)

MANUALE TECNICO

The Knife

Bonci Claudi, Crippa Filippo,

Filizzola Lorenzo e Tonolo Claudio

Contenuti basati sul corso di: *Rizzardi Alessandra*

Contents

o	Introduzione a The Knife	3
1.	Capacità operative	3
2.	Concetti chiave.....	3
o	Architettura del progetto	4
1.	Struttura.....	4
2.	Pattern architetturale	4
.2.1	Funzionamento.....	4
3.	Persistenza dei dati	4
4.	Flusso dati.....	4
o	File di tipo JAVA.....	5
1.	Gestore	5
.1.1	GestoreDataset	5
.1.2	GestoreUtenti:	7
2.	Controller.....	8
.2.1	HomeNotloggedController.....	8
.2.2	ControllerViewRistorante.....	9
.2.3	ControllerVisualizzaRecensioneSenzaRisposta.....	11
.2.4	ControllerVisualizzaRecensione.....	12
.2.5	ControllerRecensisci.....	13
.2.6	ControllerChangeDataUser	13
.2.7	ControllerChangePasswordUser	15
.2.8	ControllerCreaRistorante.....	16
.2.9	ControllerCreateUser	21
.2.10	ControllerModUser	22
.1.1	ControllerModRistoratore	28
.1.2	ControllerModVisualizzazioneRecensioneSenzaRisposta	30

○ Introduzione a The Knife¹

The Knife è un progetto sviluppato con l'utilizzo del linguaggio di programmazione Java, con lo scopo di simulare le funzionalità e l'utilità dell'applicazione, di guida ai ristoranti, *The Fork*.

Lo scopo della piattaforma The Knife è la ricerca dei ristoranti più vicini all'utente, seguendo delle liste di filtraggio desiderate dallo stesso.

1. Capacità operative

Le capacità principali fornite dalla piattaforma sono:

- Registrazione e Accesso del cliente nella piattaforma, tramite algoritmi per:
 - a) Registrazione unificata del profilo sulla piattaforma, con nome utente unico seguito da ID, crittazione della password e controllo sistematico nella creazione utente.
 - b) Controllo tramite autenticazione delle credenziali utente, durante le operazioni di accesso all'applicazione.
 - c) Gestione dello stato utente, durante l'accesso ed uscita da The Knife.
- Gestione dei profili:
 - a) Distinzione tra utente, ristoratore e utente non loggato (ospite).
 - b) Gestione dei ristoranti preferiti dell'utente.
- Filtri di ricerca ristoranti:
 - a) Tipi di cucina: L'utente può ricercare il ristorante più adatto alle sue esigenze in base al tipo di cucina culinario desiderato.
 - b) Valutazione media: La valutazione dei diversi ristoranti vengono calcolate automaticamente, dando la possibilità all'utente di cercare il ristorante che più preferisce.
 - c) Città e stato: L'utente ha la possibilità di trovare il ristorante più vicino alla sua posizione, oppure in una città, o stato, a sua scelta (mostrato solamente all'ospite, in quanto il programma prende la posizione in base allo stato ed alla città inserite nei dati utente).
 - d) Servizi offerti: Come ad esempio la possibilità di Delivery o servizi all'interno del ristorante.
 - e) Fascia di prezzo: L'utente può ricercare il ristorante tramite il filtro Prezzo, dove i ristoranti vengono indicati con €, il più economico, e €€€€ il più alto.
 - f) Stelle Michelin o riconoscimenti: Ogni ristorante ha una sezione dedicata ai riconoscimenti e stelle ricevute, questo permette all'utente di scegliere il ristorante più adatto.
- Creazione, gestione e risposta delle recensioni.
- Creazione, cancellazione, gestione e modifica ristoranti (esclusivo alla tipologia di account Ristoratore).

2. Concetti chiave

L'applicazione divide l'esperienza d'utilizzo in base alla tipologia di account utente:

- Utente: Permette all'utente di lasciare/eliminare/modificare recensioni, valutazioni e ci ricercare i ristoranti desiderati.

- Ristoratore: Permette la creazione/eliminazione dei ristoranti, la modifica dello stesso e la gestione dei feedback degli utenti. Inoltre può effettuare tutte le operazioni accessibili allo stato account utente.
- Ospite: Esclusivo solo per chi non ha effettuato il Login. Permette solo la ricerca dei ristoranti e la visualizzazione delle recensioni degli stessi.

○ Architettura del progetto

1. Struttura

Il progetto è modulare, ogni file svolge una determinata funzione, suddiviso in:

- Java: File core dell'applicazione, comprendono file di gestione e controllo dei componenti.
- CSV: File di dataset dove vengono immagazzinati ristoranti, le città e gli account creati.
- Fxml: File di gestione della grafica, importati da Java tramite JavaFX. Sono i file che gestiscono la visualizzazione delle finestre di The Knife.

2. Pattern architetturale

Il pattern utilizzato durante la programmazione dei file gestori di JAVA, è l'architettura Singleton:

La quale consiste nella classe stessa che gestisce la propria istanza, senza dare modo ad altre classi di gestirla. In questo modo tutte le classi che richiamano un gestore usano la stessa istanza condivisa.

.2.1 Funzionamento

Il Singleton si implementa creando solo costruttori privati richiamati da metodi apositi, all'interno della classe stessa, salvandone l'istanza in una variabile di classe.

3. Persistenza dei dati

I dati vengono salvati dentro dei file CSV in maniera ordinata, gestiti in seguito da gestori.

Un esempio è il DatasetUtenti.CSV.

4. Flusso dati

L'applicazione viene avviata dal Launcher, successivamente la classe gestore, richiama tutti i gestori, per prendere i dati CSV, i quali vengono manipolati ed utilizzati dai controller.

○ File di tipo JAVA

I tipi java sono:

- main per l'avvio del progetto.
 - Launcher
 - App
- Gestori: File indirizzati nella gestione dei Dataset.
- Controller: Gestiscono la grafica (FXML)
- Dati presenti nel dataset:
 - Recensione
 - Utente

1. Gestore

- Start: Il file Gestore.java, invoca tutti i gestori creandone le istanze condivise.

.1.1 GestoreDataset

- Metodo inserimentoDati(): Riempie l'*ArrayList<String[]> dataSet*, il quale contiene ristoranti.

```
878 /**
879 * Metodo che carica il dataset dei ristoranti dal file CSV nella variabile
880 * dataSet.<br>
881 * Utilizza la libreria BufferedReader per la lettura del file CSV.<br>
882 */
883 private void inserimentoDati() {
884     String[] appoggio;
885     int iRow = 0;
886     try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
887         String line;
888         while ((line = reader.readLine()) != null) {
889             dataSet.add(new String[17]);
890             appoggio = line.split(";", -1);
891             for (int i = 0; i < 17; i++) {
892                 String value = "";
893                 if (i < appoggio.length && appoggio[i] != null) {
894                     value = appoggio[i].trim();
895                 }
896                 dataSet.get(iRow)[i] = value;
897             }
898             iRow++;
899         }
900     } catch (IOException e) {
901         System.out.println("File non trovato." + e);
902     }
903 }
```

- Metodo inserimentoDatiCucina(): Riempie l'*ArrayList<String> dataSetCucina*, il quale contiene tutti i tipi di cucina presenti nel database.

```
556 /**
557 * Metodo che carica il dataset dei tipi di cucina dal file CSV nella variabile
558 * dataSetCucina.<br>
559 * Utilizza BufferedReader per la lettura del file CSV.<br>
560 */
561 private void inserimentoDatiCucina() {
562     try (BufferedReader reader = new BufferedReader(new FileReader(cucinePath))) {
563         String line;
564         while ((line = reader.readLine()) != null) {
565             dataSetCucina.add(line);
566         }
567     } catch (IOException e) {
568         System.out.println("File non trovato." + e);
569     }
570 }
```

- Metodo ControlloDatasetRecensioni(): Riempie l'*ArrayList<Recensione> recensioni* con le recensioni presenti nel *dataSet*. Il quale richiama *GestoriRecensioni*.

```

222  /**
223   * Metodo che modifica e aggiunge le stelle controllando le recensioni del
224   * ristorante.<br>
225   * Utilizzato per controllare le valutazioni già presenti nel dataset delle
226   * recensioni.<br>
227   */
228  private void controlloDatasetRecensioni() {
229      GestoreRecensioni gestoreRecensioni = GestoreRecensioni.getGestoreRecensioni();
230      ArrayList<Recensione> recensioni = gestoreRecensioni.getRecensioni();
231      for (Recensione line : recensioni) {
232          String idR = line.getId();
233          for (String[] row : dataSet) {
234              if (row[16].equals(idR)) {
235                  addStelle(String.valueOf(line.getStelle()), idR);
236              }
237          }
238      }
239  }

```

- Metodo inserimentoDatiStatoCitta(): Riempie l'*ArrayList<String[]> dataSetStatiCitta* il quale contiene gli stati collegati alle città.

```

572  /**
573   * Metodo che carica il dataset degli stati e delle città dal file CSV nella
574   * variabile datasetStatiCitta.<br>
575   * Utilizza BufferedReader per la lettura del file CSV.<br>
576   */
577  private void inserimentoDatiStatoCitta() {
578      int iRow = 0;
579      try (BufferedReader reader = new BufferedReader(new FileReader(statiCittaPath))) {
580          String line;
581          while ((line = reader.readLine()) != null) {
582              if (line.trim().isEmpty()) {
583                  continue;
584              }
585              String[] appoggio = line.split(";", -1);
586              if (appoggio.length >= 2) {
587                  dataSetStatiCitta.add(new String[] { appoggio[0], appoggio[1] });
588              } else if (appoggio.length == 1) {
589                  dataSetStatiCitta.add(new String[] { appoggio[0], "" });
590              } else {
591                  System.out.println("Riga malformata ignorata: " + line);
592                  continue;
593              }
594              iRow++;
595          }
596      } catch (IOException e) {
597          System.out.println("File statoCitta non trovato - verrà creato.");
598      }
599  }

```

.1.2 GestoreUtenti:

- Metodo inserimentoNewDati: Riempie l'*ArrayList<Utente> utenti*, il quale contiene utenti.

```
415  /**
416  * Metodo che inserisce i dati dal dataset nella lista di utenti.<br>
417  * Utilizza la libreria BufferedReader per la lettura del file CSV.<br>
418  */
419 private void inserimentoNewDati() {
420     String[] appoggio;
421     int iRow = 0;
422     try (BufferedReader reader = new BufferedReader(new FileReader(fileUserPath))) {
423         String line;
424
425         while ((line = reader.readLine()) != null) {
426             utenti.add(new Utente());
427             appoggio = line.split(";");
428             if(appoggio.length!=1) {
429                 utenti.get(iRow).setId(appoggio[0]);
430                 utenti.get(iRow).setUsername(appoggio[1]);
431                 utenti.get(iRow).setPasswordHash(appoggio[2]);
432                 utenti.get(iRow).setEmail(appoggio[3]);
433                 utenti.get(iRow).setNome(appoggio[4]);
434                 utenti.get(iRow).setCognome(appoggio[5]);
435                 utenti.get(iRow).setStato(appoggio[6]);
436                 utenti.get(iRow).setCittà(appoggio[7]);
437                 utenti.get(iRow).setIndirizzo(appoggio[8]);
438                 if (appoggio[9].equals("1")) {
439                     utenti.get(iRow).setRistoratore(true);
440                 } else {
441                     utenti.get(iRow).setRistoratore(false);
442                 }
443             }
444             iRow++;
445         }
446     } catch (IOException e) {
447         System.out.println("File non trovato.");
448     }
449 }
```

- Metodo inserimentoFavouriteDati(): Riempie l'*ArrayList<String[]> dataSetFavourite* tutti preferiti degli utenti collegandoli ai ristoranti.

```
450 /**
451 * Metodo che inserisce i dati dal dataset dei ristoranti nella lista dei preferiti.<br>
452 * Utilizza la libreria BufferedReader per la lettura del file CSV.<br>
453 */
454 private void inserimentoFavouriteDati() {
455     int iRow = 0;
456     try (BufferedReader reader = new BufferedReader(new FileReader(pathFavourite))) {
457         String line;
458         while ((line = reader.readLine()) != null) {
459             if (line.trim().isEmpty()) {
460                 continue;
461             }
462             String[] appoggio = line.split(";", -1);
463             if (appoggio.length >= 2) {
464                 dataSetFavourite.add(new String[]{appoggio[0], appoggio[1]});
465             } else if (appoggio.length == 1) {
466                 dataSetFavourite.add(new String[]{appoggio[0], ""});
467             } else {
468                 System.out.println("Riga malformata ignorata: " + line);
469                 continue;
470             }
471             iRow++;
472         }
473         //System.out.println("Caricati " + iRow + " favourites dal file");
474     } catch (IOException e) {
475         System.out.println("File favourite non trovato - verrà creato.");
476     }
477 }
```

- Metodo inserimentoPersoneRistorantiDati(): Riempie l'ArrayList<String[]> personeRistoranti il quale collega i ristoratori ai ristoranti posseduti.

```

482 private void inserimentoPersoneRistorantiDati() {
483     int iRow = 0;
484
485     try (BufferedReader reader = new BufferedReader(new FileReader(personeRistorantiPath))) {
486         String line;
487         while ((line = reader.readLine()) != null) {
488             if (line.trim().isEmpty()) {
489                 continue;
490             }
491             String[] appoggio = line.split(";", -1);
492             if (appoggio.length >= 2) {
493                 personeRistoranti.add(new String[]{appoggio[0], appoggio[1]});
494             } else if (appoggio.length == 1) {
495                 personeRistoranti.add(new String[]{appoggio[0], ""});
496             } else {
497                 System.out.println("Riga malformata ignorata: " + line);
498                 continue;
499             }
500             iRow++;
501         }
502     } catch (IOException e) {
503         System.out.println("File favourite non trovato - verrà creato.");
504     }
505 }

```

2. Controller

.2.1 HomeNotloggedController

- Metodo visualizzaRistoranteButtonAction(): Questo metodo prende il ristorante selezionato dall'utente e richiama la classe ViewRistorante.fxml.

```

219 /**
220  * Metodo FXML che visualizza nel dettaglio un ristorante selezionato.<br>
221  * Utilizzato per aprire la scena ViewRistorante usando il metodo setRoot della class App.<br>
222  * @throws IOException
223  */
224 @FXML
225 private void visualizzaRistoranteButtonAction() throws IOException {
226     String selectedItem = listViewRestaurants.getSelectionModel().getSelectedItem();
227     if (selectedItem == null || selectedItem.startsWith("Nessun ristorante trovato")) {
228         // Nessun elemento selezionato o messaggio di nessun ristorante trovato
229         return;
230     }
231     String idRistorante = selectedItem.split(" - ")[0].replace("Ristorante N: ", "").trim();
232     ControllerViewRistorante.getInstance(GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante), false);
233     App.setRoot("ViewRistorante");
234 }

```

- Metodo searchingButtonAction(): Questo metodo seleziona nel dataSet i ristoranti che corrispondono alle caratteristiche inserite dall'utente nei campi Nome, Stato, Città e Descrizione

```

663 /**
664  * Metodo FXML che rimuove i ristoranti che non soddisfano il criterio di ricerca testuale.<br>
665  */
666 @FXML
667 private void searchingButtonAction() {
668     String searchText = searchTextField.getText().toLowerCase();
669     listViewRestaurants.getItems().clear();
670     ArrayList<String> tempList = new ArrayList<>();
671     ArrayList<String> tempList2 = new ArrayList<>(filteredList);
672     for (String[] row : filteredList) {
673         if (row == null || row.length < 10) {
674             tempList2.remove(row);
675             continue;
676         }
677         String col0 = row[0] == null ? "" : row[0].trim().toLowerCase();
678         String col2 = row[2] == null ? "" : row[2].trim().toLowerCase();
679         if (col0.equals("name") || col2.equals("state")) {
680             continue;
681         }
682         String nome = row[0].toLowerCase();
683         String indirizzo = row[1].toLowerCase();
684         String stato = row[2].toLowerCase();
685         String citta = row[3].toLowerCase();
686         String prezzo = row[4].toLowerCase();
687         String tipo = row[5].toLowerCase();
688         String lon = row[6].toLowerCase();
689         String lat = row[7].toLowerCase();
690         String phone = row[8].toLowerCase();
691         String url = row[9].toLowerCase();
692         String award = row[10].toLowerCase();
693         String servizi = row[11].toLowerCase();
694         String descrizione = row[12].toLowerCase();
695         if (!nome.contains(searchText) || !stato.contains(searchText) ||
696             !citta.contains(searchText) || !prezzo.contains(searchText) ||
697             !indirizzo.contains(searchText) || !lon.contains(searchText) ||
698             !lat.contains(searchText) || !phone.contains(searchText) ||
699             !url.contains(searchText) || !award.contains(searchText) ||
700             !servizi.contains(searchText) || !descrizione.contains(searchText) ||
701             !tipo.contains(searchText)) {
702             //tempList.add(row);
703             tempList2.remove(row);
704         }
705     }
706     filteredList = tempList2;
707 }
708
709
710

```

2.2 ControllerViewRistorante

- Metodo `isInPreferiti()`: Controlla se il ristorante visualizzato è presente nei preferiti, dell'utente loggato, ed in base al esito rende disabilitato e visibile il bottone aggiungi preferiti o il bottone rimuovi preferiti.

```
333  /**
334  * Metodo per verificare se il ristorante e' stato già aggiunto ai preferiti.<br>
335  */
336 private void isInPreferiti() {
337     Gestore gestore = Gestore.getGestore();
338     Utente utenteLoggato = gestore.getUtenteLoggato();
339     if(utenteLoggato != null) {
340         ArrayList<String[]> preferitiUtente = utenteLoggato.getPreferiti();
341         for (String[] strings : preferitiUtente) {
342
343             if(preferitiUtente!=null && strings[16].equals(ristorante[16])) {
344                 GestoreUtenti gestoreUtenti = GestoreUtenti.getGestoreUtenti();
345                 gestoreUtenti.printPreferitiUtente();
346                 aggiungiPreferitiButton.setDisable(true);
347                 rimuoviPreferitiButton.setDisable(false);
348                 aggiungiPreferitiButton.setText("Aggiunto ai preferiti");
349                 rimuoviPreferitiButton.setText("Rimuovi dai preferiti");
350                 rimuoviPreferitiButton.setVisible(true);
351                 break;
352             }
353         else {
354             rimuoviPreferitiButton.setDisable(true);
355             aggiungiPreferitiButton.setDisable(false);
356             aggiungiPreferitiButton.setText("Aggiungi ai preferiti");
357             rimuoviPreferitiButton.setText("Rimuovi dai preferiti");
358             rimuoviPreferitiButton.setVisible(false);
359         }
360     }
361     if(preferitiUtente.isEmpty()){
362         rimuoviPreferitiButton.setDisable(true);
363         aggiungiPreferitiButton.setDisable(false);
364         aggiungiPreferitiButton.setText("Aggiungi ai preferiti");
365         rimuoviPreferitiButton.setText("Rimuovi dai preferiti");
366         rimuoviPreferitiButton.setVisible(false);
367     }
368 }
369 }
```

- Metodo `isProprietario()`: Controlla se il ristorante visualizzato è posseduto dall'utente loggato.

```
224 /**
225  * Metodo per verificare se l'utente loggato e' proprietario del ristorante.<br>
226  * Chiama il metodo getPersonneRistorantiByIdRistorante del GestoreUtenti per verificare autenticità.<br>
227  * @return true se l'utente loggato e' proprietario del ristorante
228 */
229 private boolean isProprietario() {
230     Gestore gestore = Gestore.getGestore();
231     Utente utenteLoggato = gestore.getUtenteLoggato();
232     GestoreUtenti gestoreUtenti = GestoreUtenti.getGestoreUtenti();
233     if(gestoreUtenti.getPersonneRistorantiByIdRistorante(ristorante[16]).equals(utenteLoggato.getId())){
234         System.out.println(gestoreUtenti.getPersonneRistorantiByIdRistorante(ristorante[16])+" "+utenteLoggato.getId());
235         return true;
236     }
237     return false;
238 }
239 }
```

- Metodo **visualizzaRecensione()**: Serve alla visualizzazione della recensione selezionata dall'utente, ed è stato progettato in modo da aprire **visualizzaRecensioneSenzaRisposta.fxml** se la recensione selezionata non ha una risposta, invece se la risposta è presente, viene richiamato **VisualizzaRecensione**.

```

403 /**
404 * Metodo FXML per aprire la finestra per visualizzare la recensione.<br>
405 * Viene visualizzata la recensione con risposta se presente, altrimenti viene visualizzata la recensione senza risposta.<br>
406 * @throws IOException
407 */
408 @FXML
409 private void visualizzaRecensione() throws IOException {
410     String selectedRecensione = (String) recensioniRistoranteListView.getSelectionModel().getSelectedItem();
411     if(selectedRecensione != null && selectedRecensione.equals("Non ci sono recensioni.")) {
412         String[] partiRecensione = selectedRecensione.split(" - ");
413         String titoloSelezionato = partiRecensione[0].replace("Titolo: ", "").trim();
414         String UtenteSelezionato = partiRecensione[1].replace("By: ", "").trim();
415         String DataSelezionata = partiRecensione[3].trim();
416         String OraSelezionata = partiRecensione[4].trim();
417
418         Recensione recensioneDaVisualizzare = null;
419         for(Recensione rec : recensioni) {
420             if(rec.utenteRecensione.equals(UtenteSelezionato) && rec.getData().equals(DataSelezionata) && rec.getOra().equals(OraSelezionata)) {
421                 recensioneDaVisualizzare = rec;
422                 break;
423             }
424         }
425         if(recensioneDaVisualizzare != null) {
426             FXMLLoader loader = new FXMLLoader(App.class.getResource("VisualizzaRecensione.fxml"));
427             Parent root = loader.load();
428
429             ControllerVisualizzaRecensione controller = loader.getController();
430             controller.setRecensione(recensioneDaVisualizzare);
431
432             Stage stage = new Stage();
433             stage.setTitle("Visualizza Recensione");
434             stage.setScene(new Scene(root));
435             stage.initModality(Modality.APPLICATION_MODAL);
436             stage.show();
437         } else if (recensioneDaVisualizzare != null) {
438             FXMLLoader loader = new FXMLLoader(App.class.getResource("VisualizzaRecensioneSenzaRisposta.fxml"));
439             Parent root = loader.load();
440
441             ControllerVisualizzaRecensioneSenzaRisposta controller = loader.getController();
442             controller.setRecensione(recensioneDaVisualizzare);
443
444             Stage stage = new Stage();
445             stage.setTitle("Visualizza Recensione");
446             stage.setScene(new Scene(root));
447             stage.initModality(Modality.APPLICATION_MODAL);
448             stage.show();
449         } else {
450             System.out.println("Recensione non trovata.");
451         }
452     }
453 }
454 }
```

- Metodo **rispondiButtonAction()**: Questo metodo è collegato al **rispondiButton**, il quale è visibile solo quando è il proprietario a visualizzare un proprio ristorante, questo metodo infatti permette di scegliere da una data lista una recensione, alla quale i proprietario può rispondere se non presente una risposta o modificare la propria recensione se presente, questo avviene indirizzando l'utente alla classe **VisualizzaRecensione.fxml**.

```

371 /**
372 * Metodo FXML per aprire la finestra per rispondere alla recensione.<br>
373 * Questo è possibile solo se si è il proprietario del ristorante.<br>
374 * @throws IOException
375 */
376 @FXML
377 private void rispondiButtonAction() throws IOException {
378     String selectedRecensione = (String) recensioniRistoranteListView.getSelectionModel().getSelectedItem();
379     String[] partiRecensione = selectedRecensione.split(regex: " - ");
380     String titoloSelezionato = partiRecensione[0].replace(target: "Titolo: ", replacement: "").trim();
381     String UtenteSelezionato = partiRecensione[1].replace(target: "By: ", replacement: "").trim();
382     String DataSelezionata = partiRecensione[3].trim();
383     String OraSelezionata = partiRecensione[4].trim();
384
385     Recensione recensioneDaVisualizzare = null;
386     for(Recensione rec : recensioni) {
387         if(rec.utenteRecensione.equals(UtenteSelezionato) && rec.getData().equals(DataSelezionata) && rec.getOra().equals(OraSelezionata)) {
388             recensioneDaVisualizzare = rec;
389             break;
390         }
391     }
392     FXMLLoader loader = new FXMLLoader(App.class.getResource(name: "VisualizzaRecensione.fxml"));
393     Parent root = loader.load();
394
395     ControllerVisualizzaRecensione controller = loader.getController();
396     controller.setRecensione(recensioneDaVisualizzare);
397
398     Stage stage = new Stage();
399     stage.setTitle("Visualizza Recensione");
400     stage.setScene(new Scene(root));
401     stage.initModality(Modality.APPLICATION_MODAL);
402     stage.show();
403 }
```

- Metodo modificaText(): questo metodo viene utilizzato per modificare le caratteristiche del ristorante prendendole dalle TextArea Apposite.

```

583 /**
584  * Metodo per aggiornare i diversi parametri del ristorante.<br>
585  * Tra cui nome, indirizzo, telefono, stato, sito web, citta, award, descrizione.<br>
586  * @return boolean true se tutti i campi sono stati inseriti, false altrimenti
587 */
588 private boolean modificaText() {
589     if (nomeRistoranteField.getText().isEmpty() || indirizzoRistoranteField.getText().isEmpty() || telefonoRistoranteField.getText().isEmpty()
590         || statoRistoranteField.getText().isEmpty() || sitoWebRistoranteField.getText().isEmpty()
591         || cittaRistoranteField.getText().isEmpty() || awardField.getText().isEmpty() || descrizioneRistoranteTextArea.getText().isEmpty())
592     {
593         return false;
594     }
595     ristorante[0] = nomeRistoranteField.getText();
596     ristorante[1] = indirizzoRistoranteField.getText();
597     ristorante[8] = telefonoRistoranteField.getText();
598     ristorante[2] = statoRistoranteField.getText();
599     ristorante[9] = sitoWebRistoranteField.getText();
600     ristorante[3] = cittaRistoranteField.getText();
601     ristorante[10] = awardField.getText();
602     ristorante[12] = descrizioneRistoranteTextArea.getText();
603
604     if (getPrz() == null || getPrz().isEmpty() || serviziAdd == null || serviziAdd.isEmpty())
605     {
606         return false;
607     }
608     ristorante[4] = getPrz();
609     ristorante[11] = serviziAdd;
610     return true;
611 }

```

.2.3 ControllerVisualizzaRecensioneSenzaRisposta

- Metodo setRecensione(): questo metodo prende come input recensioneDaVisualizzare e la proietta negli spazi dedicati.

```

77 /**
78  * Metodo per impostare la recensione da visualizzare.<br>
79  * @param recensioneDaVisualizzare recensione da visualizzare
80  */
81 public void setRecensione(Recensione recensioneDaVisualizzare) {
82     TitoloField.setText(recensioneDaVisualizzare.getTitolo());
83     TestoArea.setText(recensioneDaVisualizzare.getRecensione());
84     ratingRecensione.setRating(recensioneDaVisualizzare.getStelle());
85 }

```

2.4 ControllerVisualizzaRecensione

- Metodo setRecensione(): Prende come input una recensione selezionata dall'utente e la carica nelle aree dedicate. Inoltre, controlla se l'utente Loggato è il proprietario ed in caso rende visibile il pulsante salva nonché rende editabile la RispostaArea.

```
107  /**
108   * Metodo per impostare la recensione da visualizzare.<br>
109  * Chiama il metodo setRisposta per impostare la risposta.<br>
110  * Chiama il metodo isProprietario per verificare se l'utente loggato è proprietario della recensione.<br>
111  * @param recensioneDaVisualizzare recensione da visualizzare
112  */
113 public void setRecensione(Recensione recensioneDaVisualizzare) {
114     this.recensioneCorrente = recensioneDaVisualizzare;
115     TitoloField.setText(recensioneDaVisualizzare.getTitolo());
116     TestoArea.setText(recensioneDaVisualizzare.getRecensione());
117     ratingRecensione.setRating(recensioneDaVisualizzare.getStelle());
118     RispostaArea.setText(recensioneDaVisualizzare.getRisposta());
119     setRisposta();
120     if(isProprietario()){
121         salvaButton.setDisable(false);
122         salvaButton.setVisible(true);
123         RispostaArea.setEditable(true);
124     } else {
125         salvaButton.setDisable(true);
126         salvaButton.setVisible(false);
127         RispostaArea.setEditable(false);
128     }
129 }
```

- Metodo salvaButtonAction(): questo metodo serve per salvare la risposta alla recensione in caso di modifica; infatti, questo metodo sarà visibile solo se l'utente è il proprietario del ristorante.

```
130 /**
131  * Metodo FXML per salvare la risposta della recensione.<br>
132  * Chiama il metodo closeWindow per chiudere la finestra.<br>
133  */
134 @FXML
135 private void salvaButtonAction(){
136     String testoRecensione = RispostaArea.getText();
137     this.recensioneCorrente.setRisposta(testoRecensione);
138
139     GestoreRecensioni gestoreRecensioni=GestoreRecensioni.getGestoreRecensioni();
140     int id = -1;
141     Recensione rec;
142     for (int i=0; i<gestoreRecensioni.getRecensioni().size(); i++) {
143         rec = gestoreRecensioni.getRecensioni().get(i);
144         if(rec.getUtenteRecensione().equals(recensioneCorrente.getUtenteRecensione())
145             &&rec.getData().equals(recensioneCorrente.getData())
146             &&rec.getOra().equals(recensioneCorrente.getOra())&&rec.equals(recensioneCorrente)){
147
148             id = i;
149             break;
150         }
151     }
152     if (id != -1) {
153         gestoreRecensioni.modificaRecensioneById(id, recensioneCorrente);
154     }
155     closeWindow();
156 }
```

.2.5 ControllerRecensisci

- Metodo inviaRecensione(): In questo metodo prendo titolo, testo e rating e crea una nuova recensione. Ma in caso siano vuoti rende visibile il messaggio di errore: "Per favore, compila tutti i campi.". successivamente aggiunge la recensione al dataset e la scrive nel file recensioni.CSV, aggiornando la media di voti del ristorante.

```
1 /**
2  * Metodo FXML per inviare la recensione.<br>
3  * Chiama il metodo per aggiungere la recensione al GestoreRecensioni.<br>
4  * @throws IOException
5 */
6 @FXML
7 private void inviaRecensione() throws IOException {
8     String titolo = TitoloField.getText();
9     String testo = TestoArea.getText();
10    double rating = ratingRecensione.getRating();
11
12    if(titolo.isEmpty() || testo.isEmpty() || rating == 0) {
13        erroreLabel.setVisible(true);
14        erroreLabel.setText("Per favore, compila tutti i campi.");
15    } else {
16        Gestore gestore = Gestore.getGestore();
17        Utente utenteLoggato = gestore.getUtenteLoggato();
18        Recensione nuovaRecensione = new Recensione(ControllerViewRistorante.getRistorante()[16],utenteLoggato.getUsername(),
19            titolo, testo, rating, addData(), addTime());
20        gestoreRecensioni.aggiungiRecensione(nuovaRecensione);
21        GestoreDataset gestoreDataset = GestoreDataset.getGestoreDataset();
22        gestoreDataset.addStelle(String.valueOf(rating), ControllerViewRistorante.getRistorante()[16]);
23        String idRistorante = ControllerViewRistorante.getRistorante()[16];
24        int idRow = gestoreDataset.getId(idRistorante);
25        String[] ristoranteAggiornato = gestoreDataset.getRiga(idRow);
26        ControllerViewRistorante.getInstance(ristoranteAggiornato, modifica: false);
27        controllerViewRistorante.setRecensioni();
28        controllerViewRistorante.fillListView(gestoreRecensioni.getRecensioniRistorante(idRistorante));
29        gestoreDataset.addStelle(String.valueOf(rating), ControllerViewRistorante.getRistorante()[16]);
30        controllerViewRistorante.setRating();
31        ((Stage) TitoloField.getScene().getWindow()).close();
32    }
33}
```

.2.6 ControllerChangeDataUser

- Metodo saveData è un metodo che viene utilizzato per salvare il parametro che deve essere cambiato in base alla scelta dell'utente che può essere nome, cognome, città o qualunque dato dell'utente, prima di fare ciò però controlla se il campo è vuoto e se lo stato inserito va bene o meno per la città che è già presente

```

72  /**
73  * Metodo FXML che salva i dati dell'utente in base al campo scelto.<br>
74  * Mostra un messaggio di errore se il campo non viene modificato.<br>
75  * Chiama il metodo close per chiudere la stage.<br>
76  * @throws java.io.IOException
77  */
78 @FXML
79 private void saveData() throws java.io.IOException {
80     boolean check = false;
81     String newData = textField.getText();
82     String fieldToChange = textTitle.getText().toLowerCase().trim();
83     errorLabel.setVisible(false);
84
85     if (fieldToChange.equals(anObject: "cambia citta") || fieldToChange.equals(anObject: "cambia stato")) {
86         String citta = fieldToChange.equals(anObject: "cambia citta") ? newData : utenteLoggato.getCittà();
87         String stato = fieldToChange.equals(anObject: "cambia stato") ? newData : utenteLoggato.getStato();
88
89         GestoreDataset gestoreDataset = GestoreDataset.getGestoreDataset();
90         int validazione = gestoreDataset.validaStatoCitta(citta, stato);
91
92         if (validazione == 1) {
93             String statoEsistente = gestoreDataset.findStatoByCitta(citta);
94             errorLabel.setText("Errore: " + citta + " esiste in " + statoEsistente);
95             errorLabel.setVisible(true);
96             return;
97         }
98     }
99
100    switch (fieldToChange) {
101        case "cambia nome":
102            utenteLoggato.setNome(newData);
103            check = true;
104            break;
105        case "cambia cognome":
106            utenteLoggato.setCognome(newData);
107            check = true;
108            break;
109        case "cambia indirizzo":
110            utenteLoggato.setIndirizzo(newData);
111            check = true;
112            break;
113        case "cambia citta":
114            utenteLoggato.setCittà(newData);
115            check = true;
116            break;
117        case "cambia stato":
118            utenteLoggato.setStato(newData);
119            check = true;
120            break;
121        case "cambia username":
122            utenteLoggato.setUsername(newData);
123            check = true;
124            break;
125        case "cambia email":
126            utenteLoggato.setEmail(newData);
127            check = true;
128            break;
129        default:
130            System.out.println(x: "Errore: campo non riconosciuto.");
131            break;
132    }
133
134    if (check) {
135        myStage.close();
136    } else {
137        errorLabel.setVisible(true);
138    }
139}

```

.2.7 ControllerChangePasswordUser

- Metodo saveData: in questo metodo viene utilizzato per salvare la password nuova inserita dall'utente, prima però controlla la validità della nuova password

```
57 /**
58 * Metodo FXML che salva la password dell'utente.<br>
59 * Chiama il metodo controlloPassword per verificare la correttezza della password.<br>
60 * Chiama il metodo close per chiudere la stage.<br>
61 * @throws java.io.IOException
62 */
63 @FXML
64 private void saveData() throws java.io.IOException {
65
66     String data = pswField.getText();
67     String dataNew = pswFieldNew.getText();
68     gestoreUtenti = GestoreUtenti.getGestoreUtenti();
69     labelError.setVisible(false);
70     labelError1.setVisible(false);
71
72     if (data.isEmpty() || !data.equals(utenteLoggato.getPasswordHash())) {
73         labelError1.setVisible(true);
74         if (dataNew.isEmpty()) {
75             labelError.setVisible(true);
76         }
77         return;
78     }
79     else {
80         if (!dataNew.isEmpty() && !dataNew.equals(data) && gestoreUtenti.controlloPassword(dataNew)) {
81             utenteLoggato.setPasswordHash(dataNew);
82             myStage.close();
83         } else {
84             labelError.setVisible(true);
85         }
86     }
87 }
88 }
89 }
```

2.8 ControllerCreaRistorante

- Metodo creaRistoranteButtonAction: In questo metodo viene utilizzato per fare gli opportuni controlli sui dati del ristorante che l'utente vuole inserire; se vengono tutti validati allora creerà il ristorante nel dataset

```
115 /**
116 * Metodo FXML per creare un nuovo ristorante.<br>
117 * Esegue il controllo che tutti i campi siano compilati e se sono compilati esegue la creazione del nuovo ristorante.<br>
118 * Chiama il metodo setRating per aggiornare la valutazione del ristorante.<br>
119 * Chiama il metodo verificaECreaRistorante per verificare se il ristorante e' già presente nel dataset.
120 * Infine chiama anche il metodo addRistorante per aggiungere il ristorante come proprietario del ristorante.<br>
121 *
122 */
123 @FXML
124 private void creaRistoranteButtonAction() throws IOException {
125     errorFieldcampiVuotiLabel.setVisible(false);
126     errorFieldStatiVuotiLabel.setVisible(false);
127
128     if (!nomeRistoranteField.getText().isEmpty() && !indirizzoRistoranteField.getText().isEmpty() &&
129         !statoRistoranteField.getText().isEmpty() && !cittaRistoranteField.getText().isEmpty() &&
130         (prezzo1CheckMenuItem.isSelected() || prezzo2CheckMenuItem.isSelected() ||
131          prezzo3CheckMenuItem.isSelected() || prezzo4CheckMenuItem.isSelected()) &&
132         !cucineArrayList.isEmpty() && !longitudineRistoranteField.getText().isEmpty() &&
133         !latitudineRistoranteField.getText().isEmpty() && !telefonoRistoranteField.getText().isEmpty() &&
134         !sitoWebRistoranteField.getText().isEmpty() && !awardField.getText().isEmpty() &&
135         !serviziArrayList.isEmpty() && !descrizioneRistoranteTextArea.getText().isEmpty()) {
136
137         GestoreDataset gestoreDataset = GestoreDataset.getGestoreDataset();
138
139         int validazione = gestoreDataset.validaStatoCitta(cittaRistoranteField.getText(), statoRistoranteField.getText());
140         if (validazione == 1) {
141             String statoEsistente = gestoreDataset.findStatoByCitta(cittaRistoranteField.getText());
142             errorFieldStatiVuotiLabel.setText("Errore: " + cittaRistoranteField.getText() + " esiste già in " + statoEsistente);
143             errorFieldStatiVuotiLabel.setVisible(true);
144             return;
145         }
146
147         String[] ristorante = new String[17];
148         ristorante[0] = nomeRistoranteField.getText();
149         ristorante[1] = indirizzoRistoranteField.getText();
150         ristorante[2] = statoRistoranteField.getText();
151         ristorante[3] = cittaRistoranteField.getText();
152
153         if (prezzo1CheckMenuItem.isSelected())
154             ristorante[4] = "€";
155         if (prezzo2CheckMenuItem.isSelected())
156             ristorante[4] = "€€";
157         if (prezzo3CheckMenuItem.isSelected())
158             ristorante[4] = "€€€";
159         if (prezzo4CheckMenuItem.isSelected())
160             ristorante[4] = "€€€€";
161
162         String cucina = "";
163         boolean checkFirst = true;
164         for (String string : cucineArrayList) {
165             if (checkFirst) {
166                 cucina = cucina + string;
167                 checkFirst = false;
```

```

168     } else
169         |   cucina = cucina + "," + string;
170     }
171     ristorante[5] = cucina;
172     ristorante[6] = latitudineRistoranteField.getText();
173     ristorante[7] = longitudineRistoranteField.getText();
174     ristorante[8] = telefonoRistoranteField.getText();
175     ristorante[9] = sitoWebRistoranteField.getText();
176     ristorante[10] = awardField.getText();
177
178     String servizi = "";
179     for (String string : serviziArrayList) {
180         servizi = servizi + "," + string;
181     }
182     ristorante[11] = servizi;
183     ristorante[12] = descrizioneRistoranteTextArea.getText();
184     ristorante[13] = "0.0";
185
186     if (consegnaCheckBox.isSelected())
187         |   ristorante[14] = "1";
188     else
189         |   ristorante[14] = "0";
190
191     if (prenotazioniCheckBox.isSelected())
192         |   ristorante[15] = "1";
193     else
194         |   ristorante[15] = "0";
195
196     ristorante[16] = gestoreDataset.LastId() + "";
197     gestoreDataset.aggiungiRiga(ristorante);
198
199     GestoreUtenti gestoreUtenti = GestoreUtenti.getGestoreUtenti();
200     gestoreUtenti.addNewPersoneRistoranti(utenteLoggato.getId(), ristorante[16]);
201
202     verificaECreaStatoCitta(cittàRistoranteField.getText(), statoRistoranteField.getText());
203     App.setRoot(fxml: "ModRistoratore");
204 } else {
205     errorFieldcampiVuotiLabel.setVisible(true);
206 }
207 }
```

- Metodo popolaMenuCucineConRadio: In questo metodo creiamo il menu con item radio delle cucine disponibili, tra le quali il ristoratore può selezionare da aggiungere al suo nuovo ristorante

```

226 /**
227 * Metodo per popolare il menu di cucine con radio button.<br>
228 * Chiama il metodo checkFilteredList per popolare il menu di cucine con checkbox.<br>
229 */
230 private void popolaMenuCucineConRadio() {
231     cucineFilterComboBox.getItems().clear();
232
233     tutteItem = new javafx.scene.control.CheckMenuItem("Tutte le cucine");
234     tutteItem.setSelected(true);
235     tutteItem.setId("tutteCucine");
236
237     cucineFilterComboBox.getItems().add(tutteItem);
238
239     cucineFilterComboBox.getItems().add(new javafx.scene.control.SeparatorMenuItem());
240
241     boolean checkfirst = true;
242     for (String row : GestoreDataset.getDataSetCucina()) {
243         if (checkfirst) {
244             checkfirst = false;
245             continue;
246         }
247
248         javafx.scene.control.CheckMenuItem checkItem = new javafx.scene.control.CheckMenuItem(row);
249         checkMenuItemsList.add(checkItem);
250         checkItem.setOnAction(e -> {
251             checkFilteredList(checkItem);
252         });
253         cucineFilterComboBox.getItems().add(checkItem);
254     }
255 }
```

- Metodo checkFilteredList: In questo metodo controlliamo quali tipi di cucine vanno aggiunte nella lista delle possibili cucine del ristorante

```

281 /**
282 * Metodo per aggiornare la lista di cucine.<br>
283 * Chiama il metodo fillCucineListView per aggiornare la lista di cucine.<br>
284 * @param checkItem CheckMenuItem della cucina selezionata
285 */
286 @FXML
287 private void checkFilteredList(javafx.scene.control.CheckMenuItem checkItem) {
288     if (checkItem.isSelected()) {
289         cucineArrayList.add(checkItem.getText());
290     }
291     else{
292         cucineArrayList.remove(checkItem.getText());
293     }
294     fillCucineListView();
295 }
```

- Metodo fillCucineListView: In questo metodo mostriamo i tipi di cucine scelte dal ristoratore dalla lista di selezione nella listView

```

297  /**
298  * Metodo per aggiornare la lista di cucine.<br>
299  * Se la lista di cucine è vuota, aggiunge "Nessuna cucina aggiunta" al menu.<br>
300  */
301 private void fillCucineListView(){
302     tipoCucinaRistoranteListView.getItems().clear();
303     boolean checkfirst = true;
304     for (String row : cucineArrayList) {
305         tipoCucinaRistoranteListView.getItems().add(row);
306         tipoCucinaRistoranteListView.refresh();
307     }
308     if (cucineArrayList.isEmpty() || (cucineArrayList.size() == 1 && checkfirst == false))
309         tipoCucinaRistoranteListView.getItems().add(e: "Nessuna cucina aggiunta");
310     tipoCucinaRistoranteListView.refresh();
311 }
312 }
```

- Metodo eliminaCucinaButtonAction: In questo metodo eliminiamo un tipo di cucina dalla lista del ristorante, se non è presente alcun tipo di cucina ci sarà il messaggio “Nessuna cucina aggiunta”

```

313 /**
314 * Metodo per eliminare la cucina selezionata.<br>
315 * Chiama il metodo fillCucineListView per aggiornare la lista di cucine.<br>
316 */
317 @FXML
318 private void eliminaCucinaButtonAction(){
319     String selected=tipoCucinaRistoranteListView.getSelectionModel().getSelectedItem().toString();
320     cucineArrayList.remove(selected);
321     fillCucineListView();
322     for (javafx.scene.control.CheckMenuItem checkMenuItem : checkMenuItemsList) {
323         if(checkMenuItem.getText().equals(selected)){
324             checkMenuItem.setSelected(false);
325             break;
326         }
327     }
328 }
329 }
```

- Metodo eliminaServizioButtonAction: In questo metodo eliminiamo un servizio offerto dal ristorante dalla lista dei servizi, se non è presente alcun tipo di cucina ci sarà il messaggio “Nessun servizio aggiunto”

```

330 /**
331 * Metodo per eliminare il servizio selezionato.<br>
332 * Chiama il metodo fillServiziListView per aggiornare la lista di servizi.<br>
333 */
334 @FXML
335 private void eliminaServiziButtonAction(){
336     String selected=serviziRistoranteListView.getSelectionModel().getSelectedItem().toString();
337     serviziArrayList.remove(selected);
338     fillServiziListView();
339 }
340

```

- Metodo fillServiziListView: In questo metodo mostriamo i servizi offerti nel suo ristorante scelti dal ristoratore

```

341 /**
342 * Metodo per aggiornare la lista di servizi.<br>
343 * Se la lista di servizi è vuota, aggiunge "Nessun servizio aggiunto" al menu.<br>
344 */
345 private void fillServiziListView() {
346     serviziRistoranteListView.getItems().clear();
347     for (String row : serviziArrayList) {
348         serviziRistoranteListView.getItems().add(row);
349         serviziRistoranteListView.refresh();
350     }
351     if (serviziArrayList.isEmpty()) {
352         serviziRistoranteListView.getItems().add(e: "Nessun servizio aggiunto");
353         serviziRistoranteListView.refresh();
354     }
355 }

```

- Metodo AggiungiServizioButtonAction: In questo metodo il ristoratore aggiunge un nuovo servizio al suo possibile ristorante

```

356 /**
357 * Metodo per aggiungere un servizio.<br>
358 * Se il campo di servizio non è vuoto, aggiunge il servizio alla lista di servizi.<br>
359 * Chiama il metodo fillServiziListView per aggiornare la lista di servizi.<br>
360 */
361 @FXML
362 private void AggiungiServiziButtonAction(){
363     if(!servizioField.getText().isEmpty()){
364         serviziArrayList.add(servizioField.getText());
365         errorFieldServiziVuotoLabel.setVisible(false);
366         servizioField.setText("");
367         fillServiziListView();
368     }
369     else{
370         errorFieldServiziVuotoLabel.setVisible(true);
371     }
372 }

```

.2.9 ControllerCreateUser

- Metodo createUser: In questo metodo si crea il nuovo utente e si aggiunge al dataset utenti, prima però controlla che tutti i campi non siano vuoti, controlla la validità della password e del suo stato che non vada in conflitto con la sua città

```
107  /**
108  * Metodo FXML che crea un nuovo utente.<br>
109 * Crea il nuovo utente controllando che abbia inserito tutti i dati richiesti.<br>
110 * Chiama il metodo creaUtente del GestoreUtenti per aggiungere l'utente al dataset.<br>
111 * @throws IOException
112 */
113 @FXML
114 private void createUser() throws IOException {
115     String username = usernameTextField.getText();
116     String password = passwordField.getText();
117     String email = emailTextField.getText();
118     String nome = nomeTextField.getText();
119     String cognome = cognomeTextField.getText();
120     String stato = statoTextField.getText();
121     String citta = cittaTextField.getText();
122     String indirizzo = indirizzoTextField.getText();
123     boolean isCliente = clienteCheckBox.isSelected();
124     boolean isRistoratore = ristoratoreCheckBox.isSelected();

125
126     GestoreUtenti gestoreUtenti = GestoreUtenti.getGestoreUtenti();
127     GestoreDataset gestoreDataset = GestoreDataset.getGestoreDataset();
128
129     int validazione = gestoreDataset.validaStatoCitta(citta, stato);
130     if (validazione == 1) {
131         creationStatoErrorMessageLabel.setVisible(true);
132         return;
133     } else if (validazione == 2) {
134         creationStatoErrorMessageLabel.setText("La città non può essere vuota");
135         creationStatoErrorMessageLabel.setVisible(true);
136         return;
137     }
```

- Metodo verificaECreaStatoCitta: In questo metodo si verifica che lo stato e la città non esistano già e che non vadano in conflitto tra uno e l'altro

```

170     /**
171      * Metodo che verifica e crea il stato e la città.<br>
172      * @param citta Città da verificare
173      * @param stato Stato da verificare
174      */
175     private boolean verificaECreaStatoCitta(String citta, String stato) {
176         if (citta.isEmpty()) {
177             System.out.println("Città vuota: skip creazione");
178             return false;
179         }
180
181         int validazione = gestoreDataset.validaStatoCitta(citta, stato);
182
183         if (validazione == 0) {
184             String statoTrovato = gestoreDataset.findStatoByCitta(citta);
185             if (statoTrovato == null || statoTrovato.isEmpty()) {
186                 gestoreDataset.addNewStato(stato);
187                 gestoreDataset.addNewCitta(stato, citta);
188             }
189             return true;
190         } else if (validazione == 1) {
191             String statoEsistente = gestoreDataset.findStatoByCitta(citta);
192             System.out.println("ERRORE: " + citta + " esiste già in " + statoEsistente);
193             return false;
194         }
195
196         return false;
197     }

```

.2.10 ControllerModUser

- Metodo fillRecensioniView: In questo metodo si crea la lista di tutte le recensioni che vengono pubblicate dall'utente loggato

```

214     /**
215      * Metodo che riempie la ListView con le recensioni dell'utente.<br>
216      * Se non ci sono recensioni, viene mostrato il messaggio "non ci sono recensioni".<br>
217      * @param list Lista delle recensioni dell'utente
218      */
219     public void fillRecensioniView(ArrayList<Recensione> list) {
220         listRecensioni.getItems().clear();
221
222         for (Recensione row : list) {
223             if (row != null) {
224                 String appoggio ="Titolo: "+row.titolo+" - By: "+row.utenteRecensione + " - Voto: ";
225                 for(int i=0; i<row.stelle; i++) {
226                     appoggio += "★";
227                 }
228                 appoggio += " - "+row.data+" - "+row.ora+"\n";
229                 listRecensioni.getItems().add(appoggio);
230                 listRecensioni.refresh();
231             }
232         }
233         if (list.isEmpty()){
234             listRecensioni.getItems().add(e: "non ci sono recensioni.");
235             listRecensioni.refresh();
236         }
237     }

```

- Metodo `fillListView`: In questo metodo si crea la lista di tutti i ristoranti che rientrano nella categoria dei preferiti dell'utente

```

256 /**
257 * Metodo che riempie la ListView con i ristoranti preferiti dell'utente.<br>
258 * Non visualizza la riga del ristorante se è l'intestazione.<br>
259 * Se ci sono ristoranti preferiti, viene mostrato il numero di ristoranti preferiti.<br>
260 * Se non ci sono ristoranti preferiti, viene mostrato il messaggio "Nessun ristorante trovato nei preferiti".<br>
261 * @param list Lista dei ristoranti preferiti dell'utente
262 */
263 private void fillListView(ArrayList<String[]> list) {
264     listFavourite.getItems().clear();
265     for (String[] row : list) {
266         if (row[0].equals(anObject: "Name") && row[2].equals(anObject: "State")) {
267             continue;
268         }
269         listFavourite.getItems().add("Ristorante N: " + row[16] + " - Nome: " + row[0] + " - Stato: " + row[2] + " - Città: " + row[3] + " - Prezzo: " + row[4] + " - Tipo: " + row[5]);
270     }
271     if (listFavourite.getItems().isEmpty()) {
272         listFavourite.getItems().add("Nessun ristorante trovato nei preferiti.");
273     }
274     listFavourite.refresh();
275 }
```

- Metodo `changeData`: In questo metodo si gestisce il cambio di dati da parte dell'utente per passare alla scheda che permette di cambiare il campo selezionato, controllando sempre la validità dei dati inseriti

```

366 /**
367 * Metodo che gestisce la modifica di un campo dell'utente.<br>
368 * Apre una nuova finestra cambiando il campo da modificare.<br>
369 * Infine cambia i dati dell'utente e riempie la ListView.<br>
370 * @param field Campo dell'utente da modificare
371 */
372 private void changeData(String field, boolean error) throws IOException {
373     FXMLLoader loader = new FXMLLoader(getClass().getResource(name: "ChangeDataUser.fxml"));
374     Parent root = loader.load();
375     Stage smallStage = new Stage();
376     smallStage.setScene(new Scene(root, 433, 482));
377     smallStage.initModality(Modality.APPLICATION_MODAL);
378     ControllerChangeDataUser controller = loader.getController();
379     controller.setMyStage(smallStage);
380     controller.setValue(field, error);
381     smallStage.showAndWait();
382
383     setText();
384     String stato = (statoText.getText() == null) ? "" : statoText.getText().trim();
385     String citta = (cittaText.getText() == null) ? "" : cittaText.getText().trim();
386
387     if (citta.isEmpty()) {
388         System.out.println(x: "Città vuota: impossibile procedere");
389         return;
390     }
391
392     int validazione = gestoreDataset.validaStatoCitta(citta, stato);
393
394     if (validazione == 0) {
395         String statoTrovato = gestoreDataset.findStatoByCitta(citta);
396         if (statoTrovato == null || statoTrovato.isEmpty()) {
397             gestoreDataset.addNewStato(stato);
398             gestoreDataset.addNewCitta(stato, citta);
399         }
400         setCredenziali();
401     } else if (validazione == 1) {
402         if (field.equals(anObject: "Stato")) {
403             String statoCorretto = gestoreDataset.findStatoByCitta(citta);
404             utenteLoggato.setStato(statoCorretto);
405             statoText.setText(statoCorretto);
406             cittaText.setText("");
407             utenteLoggato.setCittà(città: "");
408             setCredenziali();
409             changeData(field, error: true);
410         } else if (field.equals(anObject: "Città")) {
411             changeData(field, error: true);
412         }
413     }
414 }
```

- Metodo changePsw: In questo metodo si gestisce esclusivamente il cambio della password da parte dell'utente passando alla scena che permette di effettuare il cambiamento

```

416 /**
417 * Metodo che gestisce la modifica della password dell'utente.<br>
418 * Apre una nuova finestra per cambiare la password.<br>
419 * Infine cambia i dati dell'utente e li salva nel dataset.<br>
420 * @throws IOException
421 */
422 private void changePsw() throws IOException {
423     FXMLLoader loader = new FXMLLoader(getClass().getResource(name: "ChangePasswordUser.fxml"));
424     Parent root = loader.load();
425
426     Stage smallStage = new Stage();
427     smallStage.setScene(new Scene(root, 433, 545));
428     smallStage.setTitle("Cambiare Password");
429
430     smallStage.initModality(Modality.APPLICATION_MODAL);
431     ControllerChangePasswordUser controller = loader.getController();
432     controller.setMyStage(smallStage);
433
434     smallStage.showAndWait();
435
436     setText();
437     setCredenziali();
438 }

```

- Metodo visualizzaRistoranteButtonAction: In questo metodo viene gestito il passaggio per il ristorante selezionato tra i ristoranti inseriti nei preferiti

```

440 /**
441 * Metodo che gestisce l'azione del pulsante "Visualizza Ristorante".<br>
442 * Visualizza nel dettaglio il ristorante selezionato.<br>
443 * @throws IOException
444 */
445 @FXML
446 private void visualizzaRistoranteButtonAction() throws IOException {
447     String selectedItem = listFavourite.getSelectionModel().getSelectedItem();
448     if (selectedItem == null || selectedItem.startsWith(prefix: "Nessun ristorante trovato")) {
449         return;
450     }
451     String idRistorante = selectedItem.split(regex: " - ")[@0].replace(target: "Ristorante N: ", replacement: "").trim();
452     ControllerViewRistorante.getInstance(GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante), modifica: false);
453     App.setRoot(fxml: "ViewRistorante");
454 }

```

- Metodo rimuoviPreferito: In questo metodo viene gestita la rimozione di un ristorante selezionato tra quelli preferiti e rimosso dalla lista

```

542 /**
543 * Metodo che rimuove un ristorante dai preferiti dell'utente.<br>
544 * Chiama il metodo rimuoviPreferitoUtente per rimuovere il ristorante selezionato.<br>
545 * Chiama i filtri e riempie la ListView.<br>
546 * @throws IOException
547 */
548 @FXML
549 private void rimuoviPreferito() throws IOException {
550     String selectedItem = listFavourite.getSelectionModel().getSelectedItem();
551     if (selectedItem == null || selectedItem.startsWith(prefix: "Nessun ristorante trovato")) {
552         return;
553     }
554     String idRistorante = selectedItem.split(regex: " - ")[0].replace(target: "Ristorante N: ", replacement: "").trim();
555     String[] ristorante = GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante);
556     utenteLoggato.removePreferito(ristorante);
557     dataSetFavourite = gestoreUtenti.getFavouriteByUsername(utenteLoggato.getUsername());
558     filter();
559     fillListView(filteredList);
560 }
561
562 }
```

- Metodo switchToRistorante: In questo metodo viene gestito il passaggio per il ristorante selezionato in base al ristorante a cui è stato fatto una recensione da parte dell'utente loggato

```

564 /**
565 * Metodo che visualizza un ristorante dalla recensione dell'utente selezionata.<br>
566 * Visualizza nel dettaglio il ristorante selezionato.<br>
567 * Se non trova il ristorante, visualizza un messaggio di errore.<br>
568 * @throws IOException
569 */
570 @FXML
571 private void switchToRistorante() throws IOException {
572     String selectedRecensione = (String) listRecensioni.getSelectionModel().getSelectedItem();
573     if(selectedRecensione != null && !selectedRecensione.equals(anObject: "non ci sono recensioni.")) {
574         String[] partiRecensione = selectedRecensione.split(regex: " - ");
575         String titoloSelezionato = partiRecensione[0].replace(target: "Titolo: ", replacement: "").trim();
576         String UtenteSelezionato = partiRecensione[1].replace(target: "By: ", replacement: "").trim();
577         String DataSelezionata = partiRecensione[3].trim();
578         String OraSelezionata = partiRecensione[4].trim();
579
580         Recensione recensioneS = null;
581         String idRistorante = null;
582         for(Recensione rec : recensioni) {
583             if(rec.utenteRecensione.equals(UtenteSelezionato)&& rec.getData().equals(DataSelezionata) && rec.getOra().equals(OraSelezionata)) {
584                 recensioneS = rec;
585                 idRistorante = rec.getId();
586                 break;
587             }
588         }
589         if (recensioneS != null) {
590             ControllerViewRistorante.getInstance(GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante), modifica: false);
591             App.setRoot.fxml: "ViewRistorante";
592         } else {
593             System.out.println(x: "Non c'è nessuna recensione selezionata");
594         }
595     }
596 }
597
598 }
```

- Metodo `visualizzaRecensioneButtonAction`: In questo metodo viene gestita la visualizzazione di una recensione scritta dall'utente loggato; la recensione può essere visualizzata anche con una possibile risposta da parte del ristoratore

```

600  /**
601  * Metodo che visualizza la recensione dell'utente selezionata.<br>
602  * Visualizza nel dettaglio la recensione selezionata.<br>
603  * Se la recensione non esiste, visualizza un messaggio di errore.<br>
604  * @throws IOException
605  */
606 @FXML
607 private void visualizzaRecensioneButtonAction() throws IOException {
608     String selectedRecensione = (String) listRecensioni.getSelectionModel().getSelectedItem();
609     if(selectedRecensione != null && !selectedRecensione.equals(anObject: "non ci sono recensioni.")) {
610         String[] partiRecensione = selectedRecensione.split(regex: " - ");
611         String titoloSelezionato = partiRecensione[0].replace(target: "Titolo: ", replacement: "").trim();
612         String UtenteSelezionato = partiRecensione[1].replace(target: "By: ", replacement: "").trim();
613         String DataSelezionata = partiRecensione[3].trim();
614         String OraSelezionata = partiRecensione[4].trim();
615         Recensione recensioneDaVisualizzare = null;
616         for(Recensione rec : recensioni) {
617             if(rec.utenteRecensione.equals(UtenteSelezionato)&& rec.getData().equals(DataSelezionata) && rec.getOra().equals(OraSelezionata)) {
618                 recensioneDaVisualizzare = rec;
619                 setRecensioneDaCambiare(rec);
620                 setIdRistorante(rec.getId());
621                 break;
622             }
623         }
624         if(recensioneDaVisualizzare != null && recensioneDaVisualizzare.getRisposta()!= null&& !recensioneDaVisualizzare.getRisposta().equals(anObject: " ")) {
625             FXMLLoader loader = new FXMLLoader(App.class.getResource(name: "VisualizzaRecensione.fxml"));
626             Parent root = loader.load();
627             ControllerVisualizzaRecensione controller = loader.getController();
628             controller.setRecensione(recensioneDaVisualizzare);
629             Stage stage = new Stage();
630             stage.setTitle("Visualizza Recensione");
631             stage.setScene(new Scene(root));
632             stage.initModality(Modality.APPLICATION_MODAL);
633             stage.show();
634         }
635         else if (recensioneDaVisualizzare != null) {
636             FXMLLoader loader = new FXMLLoader(App.class.getResource(name: "VisualizzaModRecensioneSenzaRisposta.fxml"));
637             Parent root = loader.load();
638             ControllerModVisualizzaRecensioneSenzaRisposta controller = loader.getController();
639             controller.setRecensione(recensioneDaVisualizzare);
640             controller.setController(this);
641             Stage stage = new Stage();
642             stage.setTitle("Visualizza Oppure Modifica Recensione");
643             stage.setScene(new Scene(root));
644             stage.initModality(Modality.APPLICATION_MODAL);
645             stage.show();
646         }
647         else {
648             System.out.println(x: "Recensione non trovata.");
649         }
650     }
651 }

```

- Metodo rimuoviRecensione: In questo metodo viene gestita la rimozione della recensione selezionata dall'utente tra quelle che sono state pubblicate, non è possibile eliminare una recensione se è presente una risposta da parte del ristoratore

```

689  /**
690  * Metodo per rimuovere una recensione.<br>
691  * Chiama il metodo rimuoviRecensione per eliminare la recensione dal dataset.<br>
692  * Chiama il metodo removeStelle per rimuovere la valutazione dal ristorante.<br>
693  * Infine chiama fillRecensioniView per visualizzare le recensioni aggiornate dopo la rimozione.<br>
694  * @throws IOException
695  */
696 @FXML
697 private void rimuoviRecensione() throws IOException {
698     String selectedRecensione = (String) listRecensioni.getSelectionModel().getSelectedItem();
699     if(selectedRecensione != null && !selectedRecensione.equals(anObject: "non ci sono recensioni.")) {
700         String[] partiRecensione = selectedRecensione.split(regex: " - ");
701         String titoloSelezionato = partiRecensione[0].replace(target: "Titolo: ", replacement: "").trim();
702         String UtenteSelezionato = partiRecensione[1].replace(target: "By: ", replacement: "").trim();
703         String DataSelezionata = partiRecensione[3].trim();
704         String OraSelezionata = partiRecensione[4].trim();
705
706         Recensione recensioneS = null;
707         String idRistorante = null;
708         int idR = -1;
709         ArrayList<Recensione> recensioniArr = gestoreRecensioni.getRecensioni();
710         String rating = null;
711
712         for (int i = 0; i < recensioniArr.size(); i++) {
713             Recensione rec = recensioniArr.get(i);
714             if (rec.utenteRecensione.equals(UtenteSelezionato) &&
715                 rec.getData().equals(DataSelezionata) &&
716                 rec.getOra().equals(OraSelezionata)) {
717                 recensioneS = rec;
718                 idRistorante = rec.getId();
719                 rating = String.valueOf(rec.getStelle());
720                 idR = i;
721                 break;
722             }
723         }
724         if (recensioneS != null && recensioneS.getRisposta() != null &&
725             !recensioneS.getRisposta().equals(anObject: " ")) {
726             System.out.println(x: "Impossibile eliminare la recensione siccome possiede una risposta");
727         } else if (idR >= 0 && idRistorante != null) {
728             gestoreRecensioni.rimuoviRecensione(idR);
729             gestoreDataset.removeStelle(rating, idRistorante);
730         } else {
731             System.out.println(x: "Recensione non trovata");
732         }
733         recensioni = gestoreRecensioni.getRecensioniByUsername(utenteLoggato.getUsername());
734         fillRecensioniView(recensioni);
735     }
736 }
737

```

.1.1 ControllerModRistoratore

- Metodo searchingButtonAction: In questo metodo viene gestita la ricerca di un dato, presente all'interno delle informazioni di un ristorante, filtrando i ristoranti non validi

```
121 /**
122 * Metodo FXML che esegue la ricerca dei ristoranti.<br>
123 * Chiama fillListView per riempire la ListView con i ristoranti trovati.<br>
124 */
125 @FXML
126 private void searchingButtonAction() {
127     String searchText = searchTextField.getText().toLowerCase();
128     listRestaurants.getItems().clear();
129     boolean checkfirst = true;
130     ArrayList<String[]> tempList = new ArrayList<>();
131     ArrayList<String[]> tempList2 = new ArrayList<>(filteredList);
132     for (String[] row : filteredList) {
133         if (checkfirst) {
134             checkfirst = false;
135         } else {
136             String nome = row[0].toLowerCase();
137             String stato = row[2].toLowerCase();
138             String citta = row[3].toLowerCase();
139             String prezzo = row[4].toLowerCase();
140             String tipo = row[5].toLowerCase();
141
142             if (!(nome.contains(searchText) || stato.contains(searchText) ||
143                   citta.contains(searchText) || prezzo.contains(searchText) ||
144                   tipo.contains(searchText))) {
145                 //tempList.add(row);
146                 tempList2.remove(row);
147             }
148         }
149     }
150     filteredList = tempList2;
151 }
```

- Metodo fillListView: In questo metodo viene mostrata la lista completa di tutti i ristoranti del ristoratore con gli opportuni dati

```
154 /**
155 * Metodo che riempie la ListView con i ristoranti.<br>
156 * Non visualizza la riga del ristorante se è l'intestazione.<br>
157 * Se il ristorante è vuoto viene mostrato il messaggio "Nessun ristorante trovato".<br>
158 * @param list lista di ristoranti
159 */
160 private void fillListView(ArrayList<String[]> list) {
161     listRestaurants.getItems().clear();
162
163     String deliveryValue, prenotationValue;
164     for (String[] row : list) {
165         if (!row[0].equals(anObject: "Name")&&!row[2].equals(anObject: "State")) {
166             deliveryValue = setDeliveryOrPrenotationValue(row[14]);
167             prenotationValue = setDeliveryOrPrenotationValue(row[15]);
168             String valutazione = String.valueOf(gestoreDataset.calcStelle(row[13]));
169             listRestaurants.getItems().add("Ristorante N: "+row[16]+" - Nome: "+row[0] + " - Stato: " + row[2] + " - Città: " + row[3]+ " - Prezzo:" + row[4] +
170             " - Tipo: " + row[5] + " - Consegna: " + deliveryValue + " - Prenotazione: " + prenotationValue + " - Valutazione: " + valutazione);
171             listRestaurants.refresh();
172         }
173     }
174     if (list.isEmpty() || (list.size() == 0)) {
175         listRestaurants.getItems().add(e: "Nessun ristorante trovato con i filtri selezionati.");
176         listRestaurants.refresh();
177     }
178 }
```

- Metodo `visualizzaRistoranteButtonAction`: In questo metodo viene gestita la visualizzazione di un ristorante selezionato tra i ristoranti del ristoratore

```

201 /**
202 * Metodo FXML che visualizza nel dettaglio un ristorante selezionato.<br>
203 * Utilizzato per aprire la scena ViewRistorante usando il metodo setRoot della class App.<br>
204 * @throws IOException
205 */
206 @FXML
207 private void visualizzaRistoranteButtonAction() throws IOException {
208     String selectedItem = listRestaurants.getSelectionModel().getSelectedItem();
209     if (selectedItem == null || selectedItem.startsWith(prefix: "Nessun ristorante trovato")) {
210         return;
211     }
212     String idRistorante = selectedItem.split(regex: " - ")[0].replace(target: "Ristorante N: ", replacement: "").trim();
213     ControllerViewRistorante.getInstance(GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante), modifica: false);
214     App.setRoot(fxml: "ViewRistorante");
215 }
```

- Metodo `rimuovi`: In questo metodo viene gestita la rimozione di un ristorante, con i suoi dati, dalla lista dei ristoranti del ristoratore e dal dataset

```

263 /**
264 * Metodo che rimuove il ristorante selezionato.<br>
265 * <ul>
266 * <li>Chiama il metodo removeRistoranteById per rimuovere il ristorante dal dataset.</li>
267 * <li>Chiama il metodo removePersoneRistorantiByIdRistorante per rimuovere il ristorante dal dataset con i proprietari.</li>
268 * <li>Chiama il metodo rimuoviPreferitoUtente per rimuovere il ristorante dai preferiti dell'utente.</li>
269 * <li>Chiama il metodo filter e fillListView per aggiornare la lista di ristoranti.</li>
270 * </ul>
271 * @throws IOException
272 */
273 private void rimuovi() throws IOException {
274     String selectedItem = listRestaurants.getSelectionModel().getSelectedItem();
275     if (selectedItem == null || selectedItem.startsWith(prefix: "Nessun ristorante trovato")) {
276         return;
277     }
278     String idRistorante = selectedItem.split(regex: " - ")[0].replace(target: "Ristorante N: ", replacement: "").trim();
279
280     gestoreDataset.removeRistoranteById(idRistorante);
281     gestoreUtenti.removePersoneRistorantiByIdRistorante(idRistorante, utenteLoggato.getId());
282     gestoreUtenti.rimuoviPreferitoUtente(utenteLoggato.getUsername(), idRistorante);
283
284     filter();
285     fillListView(filteredList);
286 }
```

- Metodo `modifyRistorante`: In questo metodo viene gestita la modalità di modifica di un ristorante selezionato passando per la opportuna scena di modifica

```

287 /**
288 * Metodo FXML che ritorna alla schermata ModificaRistorante.<br>
289 * @throws IOException
290 */
291 @FXML
292 private void modifyRistorante() throws IOException {
293     String selectedItem = listRestaurants.getSelectionModel().getSelectedItem();
294     if (selectedItem == null || selectedItem.startsWith(prefix: "Nessun ristorante trovato")) {
295         return;
296     }
297     String idRistorante = selectedItem.split(regex: " - ")[0].replace(target: "Ristorante N: ", replacement: "").trim();
298     ControllerViewRistorante.getInstance(GestoreRicerche.getGestoreRicerche().trovaRistorantiID(idRistorante), modifica: true);
299     App.setRoot(fxml: "ViewRistorante");
300 }
301 }
```

.1.2 ControllerModVisualizzazioneRecensioneSenzaRisposta

- Metodo salvaAction: In questo metodo viene gestita la possibilità di pubblicare una propria recensione con anche una valutazione del ristorante

```
104     @FXML
105     private void salvaAction() throws IOException {
106         recensioneVecchia = controller.getRecensioneVecchia();
107         recensioneNuova = recensioneVecchia;
108         String titolo = TitoloField.getText();
109         String testo = TestoArea.getText();
110         double rating = ratingRecensione.getRating();
111         String data = addData();
112         String time = addTime();
113
114         if(titolo.isEmpty() || testo.isEmpty() || rating == 0) {
115             //erroreLabel.setVisible(true);
116             //erroreLabel.setText("Per favore, compila tutti i campi.");
117         } else {
118             recensioneNuova.setTitolo(titolo);
119             recensioneNuova.setRecensione(testo);
120             recensioneNuova.setStelle(rating);
121             recensioneNuova.setData(data);
122             recensioneNuova.setOra(time);
123
124             gestoreRecensioni.modificaRecensione(recensioneNuova, recensioneVecchia);
125             GestoreDataset gestoreDataset = GestoreDataset.getGestoreDataset();
126             if (!controller.getIdRistorante().isEmpty() && controller.getIdRistorante() != null) {
127                 String idR = controller.getIdRistorante();
128                 String ratingOld = String.valueOf(recensioneVecchia.getStelle());
129                 gestoreDataset.changeStelle(String.valueOf(rating), ratingOld, idR);
130             } else {
131                 System.out.println("Non possibile trovare id ristorante per la recensione");
132             }
133         }
134         controller.settingRecensione();
135         Stage stage = (Stage) TitoloField.getScene().getWindow();
136         stage.close();
137     }
```