# MANUALE TECNICO PROGETTO THEKNIFE


- **Autori**
  Artur Mukhin
  De Giorgi Filippo
  Magrin Nicolò
  Caredda Anna Eleonora

- **Data:** 23/06/2025
- **Versione Documento** v1.1

## INDICE

# 0. INTRODUZIONE

## 0.1 Scopo del sistema

TheKnife è un'applicazione scritta in Java per la gestione di ristoranti, recensioni, preferiti e utenti, che consente agli utenti (clienti, gestori e guest) di interagire con la piattaforma in vari modi.

# 1. ARCHITETTURA DELL'APPLICAZIONE

## 1.1 Panoramica Generale e struttura dei package

L'architettura del software è stata progettata per essere modulare e stratificata, separando le responsabilità in diversi package.

- **theknife**: Contiene la classe TheKnife, punto di ingresso dell'applicazione.
- **menu**: Gestisce l'interfaccia utente da riga di comando. Le classi *UI e GestioneMenu si occuoano dell'interazione con l'utente e invocano i servizi appropriati.
- **repository**: Le classi Service fungono da intermediari tra l'interfaccia utente e il layer di accesso ai dati.
- **gestioneFile**: gestisce la lettura e la scrittura degli oggetti da e verso i file di testo.
- **entità**: Modella gli oggetti principali dell'applicazione (es. Utente, Ristorante, Recensione).
- **eccezioni**: Contiene le eccezioni personalizzate per la gestione di errori specifici dell'applicazione.

## 1.2 Scelte architetturali

- **Uso di classi Service**: rende il codice della UI più snello e focalizzato sulla presentazione. Questa scelta promuove la riusabilità del codice e facilita i test.
- **GestioneFile**: astrae il modo in cui i dati vengono salvati. Questo significa che in futuro si potrebbe sostituire la gestione tramite file con un database senza dover modificare i layer di servizio e di presentazione.
- **Ereditarietà**: Le classi Cliente e Gestore ereditano dalla classe Utente, condividendo attributi comuni come username e password ma differenziandosi per il ruolo. Questa scelta riduce la duplicazione del codice.

# 2. ESECUZIONE E USO

## 2.1 Setup e Lancio del Programma

Una volta che il progetto è stato configurato, eseguire il programma direttamente dall'IDE:

1. **Aprire l'IDE** (NetBeans).
2. **Eseguire il programma**:
    - Selezionare la classe principale (ad esempio, la classe TheKnife) e cliccare su "Esegui" o "Run".

## 2.2 Uso delle Funzionalità

A seconda del ruolo dell'utente (gestore, cliente o guest) ci sono diverse funzionalità:

- I **clienti** possono:
    - registrarsi ed accedere al proprio account;
    - Aggiungere, visualizzare e modificare recensioni;
    - aggiungere e rimuovere ristoranti ai preferiti.
- I **gestori** possono:
    - registrarsi ed accedere al proprio account;
    - inserire nuovi ristoranti;
    - adozione di un ristorante già esistente, se non ha già un proprietario;
    - visualizzazione e gestione dei ristoranti associati.

Ogni ristorante deve essere associato ad un solo gestore.

- I **guest** possono visualizzare i ristoranti tramite dei filtri per locazione, tipologia, prezzo e servizi offerti (delivery, prenotazione).

## 2.3 Data Set di Test

L'applicazione utilizza dei file CSV per gestire i dati degli utenti, dei ristoranti e delle recensioni:

- **utenti.csv**: contiene i dati degli utenti (nome, cognome, username, password, dataNascita, luogodomicilio, ruolo).
- **michelin_my_maps.csv**: contiene informazioni sui ristoranti (nome, indirizzo, locazione, prezzo, cucina, longitudine, latitudine, numero di telefono, url, website url, premio, stella verde, strutture e servizi, descrizione).
- **recensioni_ristoranti.csv**: contiene le recensioni dei clienti sui ristoranti (id, username, stelle, testo, data, ristorante recensito).
- **preferiti.cvs**: contiene i ristoranti preferiti del cliente (username, ristoranti preferiti).
- **username_ristoranti.csv:** associa ogni gestore ai ristoranti che possiede (username, ristoranti posseduti)
- **risposta_recensioni.csv:** serve per memorizzare le risposte che i gestori dei ristoranti forniscono alle recensioni pubblicate dai clien**ti.**

Questi file possono essere modificati.

| Entità | File CSV | Classe di gestione file |
|---|---|---|
| Utenti | utenti.csv | FileUtenti |
| Ristoranti | michelin_my_maps.csv | FileRistorante |
| Recensioni | recensioni_ristoranti.csv | FileRecensioni |
| Associazioni Gestore-Ristorante | username_ristoranti.csv | FileGestoreRistorante |
| Preferiti Cliente | preferiti.csv | FilePreferitiCliente |
| Risposte alle recensioni | risposta_recensioni.cvs | FileRispostaRecensioni |

## 3. ARCHITETTURA DEL SISTEMA

### 3.1 Struttura generale dell'applicazione

La struttura dell'applicazione è costituita da tre moduli principali:

1. **Interfaccia Utente (UI)**

   Il primo livello dell'applicazione è quello che gestisce l'interazione diretta con l'utente finale. Funzioni principali:

   - Registrazione e login degli utenti.
   - Visualizzazione e gestione delle informazioni sui ristoranti.
   - Inserimento delle recensioni (per i clienti) e gestione dei preferiti.
   - Associazione dei ristoranti ai gestori (per i gestori).
   - I gestori possono rispondere alle recensioni lasciate dai clienti.

   Componenti principali:

   - **UtenteUI**: gestisce le operazioni di registrazione, login degli utenti. Permette agli utenti di creare un nuovo account o di accedere a un account esistente.
   - **RistoranteUI**: gestisce la visualizzazione dei ristoranti e della lista dei ristoranti preferiti (per i clienti) o l'associazione a un gestore (per i gestori). Per i gestori, possibilità di aggiungere, modificare e rimuovere ristoranti.
   - **RecensioneUI**: permette ai clienti di inserire recensioni sui ristoranti e visualizzare le recensioni già pubblicate.
   - **RispostaRecensioniUI**: consente ai gestori di rispondere alle recensioni lasciate dai clienti. Le risposte sono visualizzate accanto alle recensioni, e ogni risposta è associata a un'ID recensione.
   - **PreferitiClienteUI**: consente ai clienti di aggiungere o rimuovere ristoranti dalla loro lista dei preferiti.
   - **AssGestoreRistorantiUI**: gestisce l'associazione tra i gestori e i ristoranti che possiedono o gestiscono; permette di aggiungerne di nuovi e visualizzare i ristoranti che un gestore possiede.
   - **GestoreUI**: permette di visualizzare un menu all'utente, raccogliere le scelte dell'utente, eseguire azioni, come aggiungere nuovi dati o visualizzare l'elenco dei dati già esistenti.

- **ComandiBaseUI**: definisce due metodi per la visualizzazione dei dati nell'interfaccia utente:
  - visualizza(): mostra i dati all'utente.
  - visualizza(V valore): visualizza i dettagli di un'entità specifica che viene passata come parametro (valore).
- **ComandiUI**: estende ComandiBaseUI e Dominio. Gestisce le entità attraverso l'interfaccia utente con i metodi (add, get, put, remove) come l'aggiunta, la visualizzazione, la modifica e la rimozione di ristoranti, recensioni, preferiti.
- **ComandiUISenzaParametri:** è una versione di ComandiUI che non richiede parametri per eseguire le operazioni con metodi add, get, put, remove per le entità, ma esegue queste operazioni senza la necessità di passare una "chiave" o "valore".

L'interfaccia utente riceve gli input da parte dell'utente, i quali vengono elaborati e presenta i risultati all'utente.

All'interno del pacchetto menu è contenuta la classe **GestioneMenu**: gestisce l'interazione dell'utente (gestore, cliente, guest) tramite un menu.

- ristoranteServ, utenteServ, preferitiClienteServ, recensioneServ: sono i servizi che interagiscono con le entità (ristorante, utente, preferiti del cliente e recensioni) per effettuare operazioni CRUD (create→ add, read→ get, update→ put, delete→remove).
- UtenteUI, RistoranteUI, PreferitiClienteUI, RecensioneUI, AssGestoreRistoranteUI: interfacce utente (UI) che gestiscono l'interazione dell'utente con i rispettivi servizi, come registrazione, ricerca ristoranti, gestione dei preferiti e recensioni.

**Metodi:**

1- **benvenuto():** mostra un menu con le opzioni principali per l'utente:
   - Registrarsi
   - Effettuare il login
   - Entra come guest
   - Esci

   Se l'utente sceglie "Registrati", verrà chiamato il metodo utenteUI.add() per aggiungere un nuovo utente. Se sceglie "Login", verrà chiamato utenteUI.get() per recuperare l'utente esistente. Se sceglie "Guest", verrà chiamato il metodo benvenutoGuest(). Se sceglie "Esci" il programma termina il ciclo di esecuzione del menu.

2- **benvenutoUtente(Utente utente):**
   Quando l'utente si registra o effettua il login, viene reindirizzato a questo metodo. Qui si distingue tra un Gestore e un Cliente.
   - Se l'utente è un Gestore, il metodo invoca benvenutoGestore(Gestore utente).
   - Se l'utente è un Cliente, il metodo invoca benvenutoCliente(Cliente utente).

3- **benvenutoGuest():**
   Questa parte gestisce la modalità "guest", ovvero quando un utente entra nell'applicazione senza registrarsi o fare il login. L'utente può visualizzare i ristoranti, ma non può fare altre operazioni come aggiungere preferiti o scrivere recensioni. Le opzioni presentate sono:
   - Visualizza ristoranti
   - Esci

4- **benvenutoGestore(Gestore utente):**
   Quando un gestore accede al sistema, viene mostrato un menu:
   - Aggiungi un ristorante
   - Ricerca ristoranti
   - Visualizza media valutazioni delle recensioni
   - Rispondi alle recensioni

   A seconda della scelta, vengono invocati i metodi corrispondenti per aggiungere e ricercare ristoranti, visualizzare la media delle recensioni o rispondere ad esse.

5- **benvenutoCliente(Cliente utente):**

Le opzioni per il cliente sono:

- Aggiungi un ristorante ai preferiti
- Rimuovi un ristorante dai preferiti
- Visualizza i preferiti
- Aggiungi, modifica o elimina recensioni
- Ricerca ristoranti

A seconda della scelta, vengono invocati i metodi corrispondenti per aggiungere, rimuovere, visualizzare preferiti o aggiungere/modificare/eliminare recensioni.

## 2. Entità e operazioni principali

Questa parte si occupa della gestione delle entità e delle loro operazioni.

**Entità principali:**

- **Utente** (nome, cognome, username, password, dataNascita, luogoDomicilio, ruolo)
- **Ristorante** (nome, indirizzo, locazione, prezzo, cucina, longitudine, latitudine, delivery, url, webSiteUrl, prenotazione, descrizione)
- **Recensione** (ID, username, stelle, testo, data, ristoranteRecensito)
- **PreferitiCliente** (usernameCliente, List<Ristorante> ristorantiPreferiti). Mappa i clienti ai loro ristoranti preferiti.
- **AssGestoreRistoranti**: (usernameRistoratore, List<Ristorante> ristorantiList). Mappa ogni gestore ai ristoranti che gestisce.
- **RispostaRecensioni** (ID, idRif, username, testo, data). Risposta di un gestore a una recensione lasciata da un cliente. Ogni risposta è associata a una recensione e contiene informazioni sul testo della risposta e sul gestore che ha risposto.

**Servizi principali**: i file service gestiscono l'accesso, la lettura, la scrittura e la modifica dei dati contenuti in file di formato CSV, che sono utilizzati per salvare le informazioni delle entità, come utenti, ristoranti, recensioni, preferenze e gestore ristorante.

- **UtenteService**: Gestisce l'autenticazione e la gestione degli utenti gestori o clienti.
  - Legge i dati degli utenti da un file CSV e li trasforma in oggetti di tipo Utente (e sottoclassi Gestore e Cliente). Vengono estratti tutti i campi necessari dal file CSV (nome, cognome, username, password, ruolo ("gestore" o "cliente", data di nascita, domicilio) e convertiti in oggetti di tipo associato. Permette che i dati siano sempre aggiornati.
  - Quando ci sono delle modifiche come aggiornamenti o aggiunte di nuovi utenti, i dati vengono salvati nuovamente nel file CSV.
  - Il metodo **containsKey** verifica se un determinato username esiste già nel sistema. Se un utente tenta di registrarsi con un nome utente già presente nel file, il sistema impedisce la registrazione, evitando così duplicati e garantendo l'unicità degli username.
  - Ogni riga del file rappresenta un utente, con ciascuna colonna corrispondente ai rispettivi dati.
- **RistoranteService**: gestisce la creazione, modifica e visualizzazione dei ristoranti:
  - All'avvio, i dati dei ristoranti vengono letti da un file CSV e convertiti in oggetti Ristorante. Ogni record contiene le informazioni sul ristorante (nome, indirizzo, locazione, tipo cucina, prezzo medio, numero di telefono, coordinate geografiche, URL, prenotazione, delivery, descrizione).
  - Metodi:
    - **containsKey**: verifica se esiste già un ristorante con quel nome.
    - **ristoranteGiaPossedutoDalGestore**: verifica se il gestore possiede già il ristorante.
    - **ristoranteHaAltroProprietario**: impedisce che un ristorante venga associato a più gestori.
- **RecensioneService**: gestisce l'inserimento e la gestione delle recensioni.

- Le recensioni degli utenti sono memorizzate in un file CSV. Ogni recensione ha un ID univoco incrementale. I dati vengono letti e inseriti in una HashMap<Integer, Recensione>, e convertiti nel formato della data, testo, stelle e autore.

- Metodi:

  - **incID()** gestisce l'autoincremento degli identificativi delle recensioni.
  - **put()** aggiorna una recensione esistente.
  - **remove()** elimina una recensione dal sistema e dal file.
  - **mediaStelle(String nomeRistorante):**
    Calcola la media delle stelle per un ristorante specifico. Legge tutte le recensioni per il ristorante e calcola la media delle stelle.

- **PreferitiClienteService**: gestisce i ristoranti preferiti dai clienti.

  - Il file CSV contiene l'associazione tra ogni Cliente e i suoi ristoranti preferiti. I dati vengono letti e memorizzati in una HashMap<String, PreferitiCliente>. Quando un ristorante viene aggiunto o rimosso dai preferiti, la mappa viene aggiornata e salvata, sovrascrivendola se i preferiti vengono modificati.

  - Metodi:
    - **add**: aggiunge un ristorante ai preferiti di un cliente.
      Se il cliente esiste già nella mappa preferitiMap, il ristorante viene aggiunto alla sua lista di ristoranti preferiti (controllando che il ristorante non esiste nella lista).
      Se il cliente non esiste, viene creato un nuovo oggetto PreferitiCliente con il ristorante, e aggiunto alla mappa preferitiMap.
      Dopo aver aggiunto o aggiornato la lista dei preferiti, i dati vengono salvati su file.
    - **remove**: Rimuove un ristorante dai preferiti di un cliente.
    - **getPreferitiMap**: Restituisce la mappa che contiene gli username dei clienti e le loro liste di ristoranti preferiti.
    - **setPreferitiMap**: Imposta o aggiorna la mappa preferitiMap con una nuova versione.

- **AssGestoreRistorantiService**: gestisce l'associazione tra gestori e ristoranti.

  - Viene utilizzato un file CSV che associa ogni Gestore a una lista di Ristorante. I dati vengono caricati in una mappa (HashMap<String, AssGestoreRistoranti>) dove la chiave è lo username del gestore.

  - Metodi:
    - **add**: Aggiunge un ristorante alla lista di un gestore (username). Se il gestore esiste già, il ristorante viene aggiunto alla sua lista e la mappa viene aggiornate e si sovrascrive il file. Se il gestore non esiste, viene creato un nuovo oggetto gestore con il ristorante. Dopo ogni operazione, i dati vengono salvati su file.
    - **getRistorantiMap**: Restituisce la mappa dei gestori con i loro ristoranti.
    - **setRistorantiMap**: Imposta o aggiorna la mappa dei gestori e dei ristoranti.

- **GestoreService**: gestisce i dati relativi ai gestori di ristoranti.
  - utilizza il servizio AssGestoreRistorantiService per ottenere tutti i ristoranti posseduti dal gestore.
  - utilizza il RecensioneService per ottenere tutte le recensioni del ristorante e calcolare la media delle stelle: **mediaStelle**.
- **RispostaRecensioniService**: gestisce le risposte alle recensioni da parte dei gestori dei ristoranti.
  - Metodi:
    - **aggiornaRisposta**: permette di aggiornare una risposta già esistente. Se la risposta con l'ID specificato esiste, viene sovrascritta con i nuovi dati forniti nell'oggetto RispostaRecensioni.
    - **ottieniRispostePerGestore**: permette di ottenere tutte le risposte scritte da un gestore specifico, dato il nome del gestore.
    - **verificaSeRispostaEsiste**: verifica se un gestore ha già risposto a una specifica recensione. Se una risposta esiste già, il metodo restituisce true; altrimenti, false.

Relazioni tra le entità

- PreferitiCliente associa uni-molti tra Cliente e Ristoranti.
- AssGestoreRistoranti associa uni-molti tra Gestore e Ristoranti.
- Recensione collega un Utente a un Ristorante tramite ristoranteRecensito.
- RispostaRecensioni si riferisce a una Recensione tramite idRif (Chiave esterna).

## 3. Archiviazione dei dati

Si occupa della gestione dei dati che devono essere memorizzati nel sistema. TheKnife utilizza file CSV per la memorizzazione dei dati. I file CSV sono letti e scritti tramite classi dedicate per ogni tipo di entità.

- **Classi di gestione file**:
  - **GestioneFile**: tutte le altre classi di gestione file usano GestioneFile per eseguire operazioni comuni senza duplicare il codice:
    - Lettura dei file CSV
    - Scrittura di una nuova riga
    - Sovrascrittura del file
  - **FileUtenti**: Gestisce la lettura e la scrittura dei dati degli utenti su un file CSV.
    - La classe legge il file CSV e per ogni riga del file, crea un oggetto Utente (o le sue sottoclassi Gestore o Cliente) utilizzando i dati estratti dal file (nome, cognome, username, password, ruolo, data di nascita e domicilio). I dati vengono letti trasformati in oggetti. Quando nuovi utenti vengono creati o quando vengono aggiornati i dati esistenti, la classe riscrive la riga nel file CSV.
  - **FileRistorante**: Gestisce la lettura e la scrittura dei dati relativi ai ristoranti.
    - La classe FileRistorante legge il file CSV dei ristoranti e crea oggetti Ristorante per ogni riga del file. Quando nuovi ristoranti vengono aggiunti o modificati, i dati vengono aggiornati nel file CSV.
  - **FileRecensioni**: Gestisce la lettura e la scrittura delle recensioni.
    - La classe FileRecensioni legge le recensioni da un file CSV e crea un oggetto Recensione per ogni riga, che contiene informazioni come l'utente che ha recensito il ristorante, le stelle, il testo, la data e il ristorante recensito. Quando un cliente scrive una recensione, i dati vengono salvati nel file CSV.
  - **FileGestoreRistorante**: Gestisce l'associazione dei gestori ai ristoranti; ogni gestore può gestire uno o più ristoranti.
    - La classe FileGestoreRistorante legge il file CSV e per ogni riga crea un oggetto AssGestoreRistoranti, che contiene informazioni sugli username dei gestori e sui ristoranti da loro posseduti. Quando un gestore gestisce un nuovo ristorante, o quando un ristorante viene rimosso, la classe aggiorna il file CSV.
  - **FilePreferitiCliente**: Gestisce l'associazione dei clienti ai loro ristoranti preferiti.
    - La classe FilePreferitiClienti legge il file CSV e per ogni riga crea un oggetto PreferitiCliente, associando il cliente ai suoi ristoranti preferiti. Quando un cliente aggiunge o rimuove un ristorante dalla lista dei preferiti, il file viene aggiornato.
  - **FileRispostaRecensioni**: gestisce l'archiviazione e la gestione delle risposte dei gestori alle recensioni dei clienti-
    - La classe FileRispostaRecensioni legge il file CSV chiamato risposte_recensioni.csv. Ogni risposta è associata a una recensione specifica, consentendo ai gestori di rispondere alle recensioni degli utenti. Ogni risposta viene associata all'ID della recensione a cui risponde e memorizzata in una mappa (Map<Integer, Risposta>).
      - aggiungiRisposta:
        - Permette a un gestore di rispondere a una recensione.
        - La risposta viene associata all'ID della recensione, al nome del gestore, al testo della risposta e alla data corrente.
        - La risposta viene aggiunta alla mappa e i dati vengono successivamente salvati nel file CSV.
      - (saveRisposte):
        - Ogni volta che viene aggiunta o aggiornata una risposta, il metodo salva tutte le risposte nel file CSV sovrascrivendo il contenuto precedente.
        - Ogni risposta viene scritta nel file nel formato: idRecensione, usernameGestore, risposta, dataRisposta.
      - Classe Interna Risposta: rappresenta una singola risposta a una recensione.

Contiene le informazioni: ID della recensione, username del gestore che risponde, testo della risposta e la data della risposta.
Ha un metodo **toCSV()** che converte l'oggetto in una stringa in formato CSV per essere scritta nel file.

### 3.2 Organizzazione in pacchetti

- Entita:
  - Associazione: relazione tra oggetti
  - Dominio: metodo equals(Object obj) viene utilizzato per confrontare due oggetti e determinare se sono uguali.
  - Ruolo: utente o gestore.
- entita.dominio: contiene le entità principali:
  - Utente: rappresenta un utente generico:
    - String nome, String cognome, String username, String password, LocalDate Data, String luogoDomicilio, String ruolo
  - Cliente: estende Utente, rappresenta un cliente:
    - String nome, String cognome, String username, String password, LocalDate Data, String luogodomicilio, String ruolo
  - Gestore: estende Utente, rappresenta un gestore di ristoranti:
    - String nome, String cognome, String username, String password, LocalDate data, String luogodomicilio, String ruolo.
  - Ristorante: rappresenta un ristorante:
    - String nome, String indirizzo, String locazione, float prezzo, String cucina, float longitudine, float latitudine, String numeroTelefono, boolean delivery, String url, String webSiteUrl, boolean prenotazione, String descrizione, short stelle

- entita.associazione:
  - AssGestoreRistoranti: associazione tra un gestore e una lista di ristoranti da lui gestiti.
    - String usernameRistoratore, ristorantiList (List<Ristorante>).
  - PreferitiCliente: associazione tra un utente cliente e i ristoranti che ha aggiunto come preferiti.
    - String usernameCliente, ristorantiPreferiti (List<Ristorante>).
  - Recensione: associazione tra un utente, un ristorante e una recensione.
    - int ID, String username, short stelle, String testo, LocalDate data, String ristoranteRecensito
  - RispostaRecensioni: associazione tra una recensione e la risposta del gestore.
    - int ID (identificativo della risposta), int idRif (riferimento alla recensione), String username, String testo, LocalDate data

## 4. ALGORITMI PRINCIPALI

### 1. Lettura da file CSV

Tutti i file vengono letti con una funzione che:

- legge riga per riga
- spezza i campi tenendo conto di eventuali **virgolette** (es. descrizioni con virgole)
- restituisce ogni riga come una List<String>

Implementato in GestioneFile.leggiRiga(String riga).

### 2. Scrittura su file CSV

Le scritture si dividono in due:

- **Aggiunta di una riga**: aggiunge solo la nuova entry in fondo al file
- **Sovrascrittura**: aggiorna l'intero contenuto del file, ad esempio quando una recensione viene modificata o cancellata

Implementato in GestioneFile.scrittura() e GestioneFile.sovraScrivi()

### 3. Autoincremento ID (per Recensioni e Risposte)

Quando si crea una nuova recensione o risposta, viene assegnato un **ID incrementale univoco**.
Implementato in RecensioneService.incID() e RispostaRecensioniService.incID()

### 4. Verifica unicità username/ristorante

Durante registrazioni e creazioni, si verifica se l'elemento esiste già:

- UtenteService.containsKey(String username)
- RistoranteService.containsKey(String nome)

### 5. Associazione e rimozione da HashMap

Per associare clienti a preferiti o gestori a ristoranti:

- Si recupera la lista associata nella HashMap
- Si aggiunge/rimuove l'elemento
- Si aggiorna la mappa e si sovrascrive il file

## 5. GESTIONE DEI FILE

### 5.1) Formato dei file

L'applicazione utilizza file in formato CSV. Caratteristiche dei file CSV usati:

- I campi sono separati da ,.
- I valori contenenti , sono racchiusi tra virgolette (").
- Le righe corrispondono a istanze delle entità.

### 5.2) 5.2 Dettagli sui path di archiviazione

Tutti i file CSV sono memorizzati all'interno della directory archivio situata nella root del progetto.

### 5.3) 5.3 Lettura e scrittura dei file

Tutte le operazioni sui file sono centralizzate nella classe astratta GestioneFile, che fornisce metodi comuni.

- **Lettura (read):** ***public List<String[]> leggiRighe(String path);***
  Legge riga per riga dal file.
  Effettua il parsing tenendo conto di eventuali virgolette.
  Ritorna una lista di array di stringhe.
- Scrittura:
  public void scrittura(String path, String contenuto);
  public void sovraScrivi(String path, List<String> contenuto);→ sovrascrive l'intero file con il nuovo contenuto.

Ogni entità ha una classe *FileXxx.java* che estende **GestioneFile**, ad esempio:

- FileUtenti → crea oggetti Utente, Cliente, Gestore.
- FileRecensioni → crea oggetti Recensione, assegna ID.
- FileRispostaRecensioni → gestisce risposte alle recensioni con idRisposta, idRecensione, testo, username, data.