TheKnife

Manuale Tecnico

Università degli Studi dell'Insubria - Laurea Triennale in Informatica

Erik Mirashaj, 760453
Lorenzo Mujeci, 757597
Igor Gorchynskyi, 757184
Matteo Nika, 762540

Sommario

Introduzione

TheKnife è un sistema di gestione per ristoranti sviluppato in JavaFX che permette agli utenti di visualizzare, recensire e prenotare tavoli presso diversi ristoranti. Il sistema implementa un'architettura MVC completa con gestione delle sessioni utente, persistenza dati tramite file CSV e interfaccia grafica moderna svolto nell'ambito del progetto di Laboratorio A per il corso di laurea in Informatica dell'Università degli Studi dell'Insubria.

Il progetto è sviluppato in Java 23, utilizza un'interfaccia grafica costruita con JavaFX 24.0.1 ed è stato sviluppato e testato sul sistema operativo Windows 10.

Architettura del Sistema

Il progetto segue l'architettura Model-View-Controller e comprende 28 classi totali organizzate in tre categorie principali. L'architettura è progettata per separare chiaramente la logica di business, la gestione dei dati e l'interfaccia utente, garantendo manutenibilità e scalabilità del codice.

Tecnologie e Librerie esterne utilizzate

- **JavaFX 24.0.1 per l'interfaccia utente**
  JavaFX è il framework principale utilizzato per creare l'interfaccia utente moderna e responsiva. Fornisce componenti grafici avanzati come ListView, TableView, e un sistema di layout flessibile basato su FXML. La scelta di JavaFX permette la creazione di applicazioni desktop con interfacce accattivanti e user-friendly, supportando anche funzionalità avanzate come binding dei dati e animazioni.

- **Java Collections Framework per la gestione delle strutture dati**
  Il framework delle collezioni Java fornisce le strutture dati fondamentali utilizzate nel progetto. ArrayList viene utilizzato per liste dinamiche di ristoranti e prenotazioni, garantendo accesso rapido O(1) per indice. HashMap è impiegato per la gestione efficiente dei dati utente e delle cache, offrendo operazioni di ricerca O(1) in media. List e Map sono le interfacce principali che permettono flessibilità nell'implementazione.

- **Pattern Singleton per la gestione delle sessioni**
  Il pattern Singleton è implementato nella classe UserSession per garantire che esista una sola istanza della sessione utente durante l'intera esecuzione dell'applicazione. Questo pattern assicura la coerenza dei dati e facilita l'accesso globale alle informazioni di sessione

da qualsiasi punto dell'applicazione, evitando la duplicazione di dati e conflitti di stato.

- **File CSV per la persistenza dei dati**
  I file CSV sono utilizzati come sistema di persistenza semplice ed efficace. Questa scelta permette di memorizzare i dati in formato leggibile e facilmente modificabile, senza richiedere database complessi. Il formato CSV è ideale per applicazioni di piccole-medie dimensioni e facilita il debug e la manutenzione dei dati.

- **LocalDateTime per la gestione temporale**
  La classe LocalDateTime di Java 8+ è utilizzata per gestire date e orari delle prenotazioni e recensioni. Questa API moderna offre operazioni temporali sicure e intuitive, eliminando i problemi delle vecchie classi Date e Calendar. Supporta formattazione, parsing e calcoli temporali avanzati necessari per la gestione delle prenotazioni.

- **Maven per la gestione delle dipendenze e build**
  Maven è il sistema di build e gestione delle dipendenze che automatizza la compilazione, il packaging e la distribuzione del progetto. Il file pom.xml definisce le dipendenze JavaFX e configura il processo di build per creare JAR eseguibili. Maven semplifica la gestione delle librerie esterne e garantisce build riproducibili su diversi ambienti.

Classi Principali

Il nucleo del sistema è formato da 6 classi fondamentali che gestiscono la logica di business e i modelli dati. TheKnife rappresenta il punto d'ingresso principale dell'applicazione JavaFX e coordina l'inizializzazione dell'intero sistema. Main fornisce un entry point alternativo per garantire compatibilità con diversi ambienti di esecuzione. UserSession costituisce la classe più importante del progetto, implementando tutte le operazioni CRUD sui dati e la gestione della sessione utente attraverso pattern Singleton.

UserSession

UserSession è sicuramente la classe più importante di tutto il progetto: si occupa delle operazioni fondamentali svolte dal programma, cioè la gestione della sessione utente, l'autenticazione, e tutte le operazioni CRUD sui dati dell'applicazione, avvalendosi di strutture dati ottimizzate per garantire performance elevate.

Un oggetto UserSession memorizza al suo interno alcune informazioni essenziali:

- L'utente attualmente loggato (username, role, dati personali)
- La lista completa dei ristoranti caricata da “restaurants_list.csv”
- Le collezioni di dati utente: preferiti, prenotazioni, recensioni
- Cache di autenticazione per ottimizzare le operazioni di login
- Strutture dati ottimizzate (HashMap e ArrayList) per accesso rapido

Strutture dati interne

- `List<Restaurant> allRestaurants` - Lista completa ristoranti
- `List<String> userFavorites` - Preferiti dell'utente corrente
- `List<Map<String, String>> userBookings` - Prenotazioni utente
- `Map<String, String> currentUser` - Dati utente attuale
- `List<Map<String, String>> allUsers` - Cache utenti per autenticazione

La strategia generale

La gestione dei dati segue questo workflow:

1. Inizializzazione: Al primo accesso, UserSession carica tutti i dati CSV in memoria
2. Autenticazione: Utilizza la cache degli utenti per validazione rapida O(n)
3. Operazioni CRUD: Mantiene sincronizzazione tra memoria e file CSV
4. Gestione della sessione: Mantiene lo stato dell'utente durante tutta l'esecuzione

Gestione dei file CSV - Strategia duale:

La classe implementa una strategia sofisticata per la lettura dei file CSV:

Lettura (Dual Strategy):
1. Prima strategia - JAR: Per applicazioni distribuite
2. Seconda strategia - Filesystem: Per ambienti di sviluppo

Strategie di ottimizzazione per limitare gli sprechi:
1. Lazy Loading: I dati vengono caricati solo quando necessari
2. Caching intelligente: Una volta caricati, i dati rimangono in memoria
3. Validazione preventiva: Input validation prima delle operazioni sui dati
4. Gestione errori robusta: strategie per prevenire informazioni o dari errati oppure mancanti

Operazioni principali e complessità computazionale:

Autenticazione:
• authenticateUser(username, password): O(n) dove n = numero utenti
• Strategia: Scansione lineare della lista utenti con early termination

Gestione Ristoranti:

- getAllRestaurants(): O(1) - restituisce riferimento alla lista caricata
- getRestaurantById(id): O(n) dove n = numero ristoranti
- Strategia: Scansione lineare con early termination

Gestione Preferiti:

- addFavorite(restaurantId): O(1) - aggiunta a fine lista
- removeFavorite(restaurantId): O(n) dove n = numero preferiti
- isFavorite(restaurantId): O(n) - ricerca lineare
- Strategia: Lista semplice, ottimizzata per pochi elementi

Gestione Prenotazioni:

- addBooking(booking): O(1) - aggiunta a fine lista
- getUserBookings(): O(n) dove n = numero prenotazioni totali
- Strategia: Filtraggio per username dell'utente corrente

Memory Management:

- Strutture dati ottimizzate (ArrayList vs LinkedList)
- Evita creazioni eccessive di oggetti temporanei
- Una sola istanza in memoria per tutta l'applicazione

TheKnife

TheKnife è la classe principale dell'applicazione e rappresenta il punto di ingresso per l'interfaccia grafica JavaFX. Questa classe coordina l'inizializzazione dell'intera applicazione e gestisce il ciclo di vita della finestra principale.

Un oggetto TheKnife eredita da javafx.application.Application e gestisce:

- Stage principale: La finestra principale dell'applicazione
- Scene iniziale: Caricamento della vista home.fxml
- Configurazione UI: Impostazioni di visualizzazione e layout
- Gestione del ciclo di vita: Startup e shutdown dell'applicazione

Responsabilità principali:

- Inizializzazione dell'applicazione JavaFX
- Configurazione della finestra principale (dimensioni, titolo, icona)
- Caricamento della schermata iniziale
- Gestione eventi di chiusura applicazione

Pattern di gestione delle risorse:

- Caricamento sicuro di file FXML tramite ClassLoader
- Gestione robusta di errori durante il caricamento
- Rilascio appropriato delle risorse alla chiusura

Integrazione con UserSession:
TheKnife non gestisce direttamente la logica di business, ma si coordina con UserSession per verificare lo stato di autenticazione, gestire la navigazione iniziale e coordinare delle operazioni di cleanup.

Main

La classe Main fornisce un entry point alternativo fondamentale per la compatibilità con diversi ambienti di esecuzione, specialmente quando JavaFX non è disponibile nel module path o quando l'applicazione viene eseguita tramite JAR.

Questa classe è particolarmente importante per:
1. Esecuzione da JAR: java -jar TheKnife.jar
2. Ambienti senza JavaFX: Quando JavaFX non è nel classpath di sistema
3. Distribuzione semplificata: Un solo punto di ingresso per tutti i contesti

Modelli Dati

La gestione dei file CSV è integrata direttamente in UserSession. Questa scelta architetturale riduce la complessità del sistema, mantiene la coesione dei dati correlati alla sessione e semplifica la gestione dello stato globale.

Restaurant

Restaurant è la classe modello fondamentale che rappresenta l'entità centrale del sistema. Ogni oggetto Restaurant incapsula tutte le informazioni necessarie per descrivere un ristorante e fornisce i metodi essenziali per la gestione e manipolazione dei dati.

Un oggetto Restaurant memorizza al suo interno tutte le informazioni caratterizzanti:

- Identificativo univoco: ID stringa per identificazione rapida
- Informazioni base: Nome, indirizzo, città, telefono
- Dati descrittivi: Descrizione completa del ristorante
- Metriche di qualità: Rating numerico (0.0 - 5.0)

Review

Review rappresenta una recensione lasciata da un utente per un ristorante specifico. Gestisce le relazioni tra utenti e ristoranti attraverso riferimenti chiave.

Un oggetto Review memorizza al suo interno tutte le informazioni caratterizzanti:

- Username utente: stringa per identificazione rapida
- Ristorante: ID stringa per riferimento al ristorante
- Metriche di qualità: Rating numerico (0.0 - 5.0)
- Commento utente: stringa per il commento testuale
- Risposta associata: stringa per la risposta del ristoratore

Gestione delle relazioni:

- Collegamento tramite username (chiave esterna)
- Collegamento tramite restaurantId (chiave esterna)
- Gestione delle risposte associate alla recensione
- Ordinamento cronologico delle recensioni

Strategie di gestione dati:

- Validazione dell'esistenza di user e restaurant
- Eliminazione in cascata delle risposte
- Gestione robusta delle date con LocalDateTime
- Sanitizzazione e validazione del contenuto testuale

Reply

Reply modella le risposte alle recensioni, creando una struttura gerarchica di conversazioni sui ristoranti.

Gestione della gerarchia:

- Legame forte con la recensione padre
- Gestione dei thread di conversazione
- Ordinamento cronologico delle risposte
- Controllo della profondità delle conversazioni

Bookings

Bookings gestisce le prenotazioni effettuate dagli utenti presso i ristoranti, con validazione temporale e gestione delle disponibilità.

Gestione temporale avanzata:

- Validazione che la prenotazione sia nel futuro
- Gestione degli slot temporali disponibili
- Scadenza automatica delle prenotazioni passate

Controller dell'Interfaccia Utente

• LoginController (gestione login)
• RegisterController (gestione registrazione)
• DashboardController (menu principale)
• RestaurantsController (impostazioni ristoranti)
• RestaurantDetailController (dettagli ristorante)
• RestaurantsParametersController (parametri ristorante)
• ReviewsController (gestione recensioni)
• ReviewAddController (aggiunta recensioni)
• RepliesController (gestione risposte)
• ReplyAddController (aggiunta risposte)
• ReplyEditController (modifica risposte)
• BookingsController (gestione prenotazioni)
• BookingsAddController (aggiunta prenotazioni)
• FavoritesController (gestione preferiti)
• HomeController (schermata home)
• AboutController (informazioni app)

L'interfaccia utente è gestita da 16 controller che implementano il pattern MVC, tra i più importanti ci sono:

LoginController

Responsabilità:

- Validazione credenziali utente
- Gestione errori di autenticazione

- Navigazione verso dashboard o registrazione

DashboardController

Responsabilità:

- Visualizzazione del menu principale
- Gestione della visibilità dei pulsanti basata sul ruolo utente
- Coordinamento della navigazione verso funzionalità specifiche

RestaurantsController

Responsabilità:
Visualizzazione lista ristoranti dal file “michelin_my_maps.csv”
Visualizzazione lista dei propri ristoranti
Gestione selezione e navigazione ai dettagli
Possibilità di rispondere a recensioni lasciate da altri utenti nei suoi ristoranti

Ottimizzazioni per performance:
• Uso di ListView con celle personalizzate per rendering efficiente
• Lazy loading dei dati per grandi dataset

Classi Utility

Le Classi Utility hanno la funzione di creare una lista di celle al cui interno verranno poi mostrati vari elementi a seconda della lista.

- BookingListCell (rendering lista prenotazioni)
- FavoritesListCell (rendering lista preferiti)
- RestaurantListCell (rendering lista ristoranti)
- ReviewListCell (rendering lista recensioni)
- ReplyListCell (rendering lista risposte)

Note Architetturali

Tutti i controller seguono un pattern comune:

1. Inizializzazione: Metodo initialize() chiamato automaticamente da JavaFX
2. Metodi annotati con @FXML per gestire eventi UI
3. Metodi per navigare tra diverse schermate
4. Collegamento tra UI e dati dei modelli

Considerazioni Tecniche

Performance e Ottimizzazioni:
• Uso di ListView con celle personalizzate per rendering efficiente
• Lazy loading dei dati per grandi dataset
• Strutture dati ottimizzate (HashMap e ArrayList) per accesso rapido
• Singleton pattern per gestione centralizzata delle sessioni

Scalabilità:
• Architettura modulare che facilita l'aggiunta di nuove funzionalità
• Separazione chiara tra logica di business e presentazione
• Gestione centralizzata dei dati tramite UserSession
• Pattern MVC per mantenere il codice organizzato e manutenibile

Sicurezza:
• Validazione degli input utente prima delle operazioni sui dati
• Gestione robusta degli errori per prevenire crash dell'applicazione
• Controllo degli accessi basato sui ruoli utente
• Filtraggio dei dati prima della persistenza su file