TheKnife

Manuale Tecnico

Autori: Andrea De Nisco (752452 CO) & Antonio De Nisco (752445 CO)

Versione: 1.0 | **Data:** Ottobre 2025

Indice

1. Architettura del Sistema

Architettura Client-Server

TheKnife implementa un'architettura client-server distribuita con separazione netta delle responsabilità:

CLIENT LAYER

Responsabilità:

- Interfaccia utente JavaFX
- Gestione eventi UI
- Validazione input lato client
- Navigazione tra schermate

SERVER LAYER

Responsabilità:

- Logica di business
- Gestione autenticazione
- Validazione server-side
- Orchestrazione operazioni

DATA LAYER

Responsabilità:

- Persistenza dati
- Gestione transazioni
- Integrità referenziale

- Ottimizzazione query

Flusso Architetturale

Comunicazione tra Layer

```
CLIENT (JavaFX)
    ↓ Richieste UI
NAVIGATION MANAGER
    ↓ Coordinate azioni
SERVER SERVICE
    ↓ Chiamate business
DAO LAYER
    ↓ Query SQL
DATABASE (PostgreSQL)
```

Sicurezza dell'Architettura

Sistema di Sicurezza

- **Password Encryption:** SHA-256 con salt randomico
- **SQL Injection Prevention:** PreparedStatements
- **Session Management:** Gestione sicura dello stato utente
- **Input Validation:** Doppia validazione client/server

Implementazione Salt e Encryption

```
public class PasswordUtils { private static final SecureRandom random = new SecureRandom(); public static String generateSalt() { byte[] salt = new byte[16]; random.nextBytes(salt); return Base64.getEncoder().encodeToString(salt); } public static String hashPassword(String password, String salt) { try { MessageDigest md =
```

```
MessageDigest.getInstance("SHA-256");
md.update(Base64.getDecoder().decode(salt)); byte[] hashedPassword =
md.digest(password.getBytes(StandardCharsets.UTF_8)); return
Base64.getEncoder().encodeToString(hashedPassword); } catch
(NoSuchAlgorithmException e) { throw new RuntimeException("Errore
nell'hashing della password", e); } } }
```

2. Requisiti di Sistema

Requisiti di Sviluppo

Componente	Versione Minimum	Versione Consigliata	Note
Java JDK	21	21 LTS	Supporto moduli e Records
Apache Maven	3.6.0	3.9.x	Gestione dipendenze
JavaFX	21	24.0.2	UI Framework
PostgreSQL	12	16.x	Database relazionale

Dipendenze Maven

```
<dependencies> <!-- JavaFX Controls --> <dependency>
<groupId>org.openjfx</groupId> <artifactId>javafx-
controls</artifactId> <version>24.0.2</version> </dependency> <!--
PostgreSQL JDBC Driver --> <dependency>
<groupId>org.postgresql</groupId> <artifactId>postgresql</artifactId>
<version>42.7.8</version> </dependency> <!-- SLF4J + Logback Logging
--> <dependency> <groupId>org.slf4j</groupId> <artifactId>slf4j-
api</artifactId> <version>2.0.16</version> </dependency>
</dependencies>
```

3. Setup Ambiente di Sviluppo

Configurazione IDE

IDE Consigliati

- **IntelliJ IDEA:** Plugin JavaFX, Maven integration
- **Eclipse:** e(fx)clipse plugin, M2Eclipse
- **Visual Studio Code:** Extension Pack for Java, JavaFX support

Struttura Progetto Maven

CONFIGURAZIONE

pom.xml	Maven principale
dependency-reduced-pom.xml	POM ottimizzato
README.txt	Documentazione
autori.txt	Info sviluppatori
module-info.java	Modulo JPMS

CODICE SORGENTE

src/main/java/com/example/

MainApp.java	Entry point

ClientApp.java	Client logic
NavigationManager.java	Singleton
ServerService.java	Business logic
DBConnection.java	DB manager
LoginForm.java	Form login
RegisterForm.java	Form registro
GuestPage.java	Pagina ospiti
Ristorante.java	Model
RistoranteDAO.java	Data Access
Recensione.java	Model
RecensioneDAO.java	Data Access

RISORSE

src/main/resources/

restaurant_owners.properties	Config DB
primary.fxml	Layout principale
secondary.fxml	Layout secondario

Diagrammi:

uml_diagram.png	UML classi
er_diagram.png	ER database

LIBRERIE

lib/

JavaFX 21	UI Framework
• javafx-base-21.jar	Core
• javafx-controls-21.jar	UI Controls
• javafx-fxml-21.jar	FXML
• javafx-graphics-21.jar	Graphics
postgresql-42.7.3.jar	JDBC Driver
checker-qual-3.42.0.jar	Annotations

OUTPUT BUILD

target/ (generato)

classes/	Bytecode
generated-sources/	Auto-generati
maven-status/	Status build
test-classes/	Test compilati

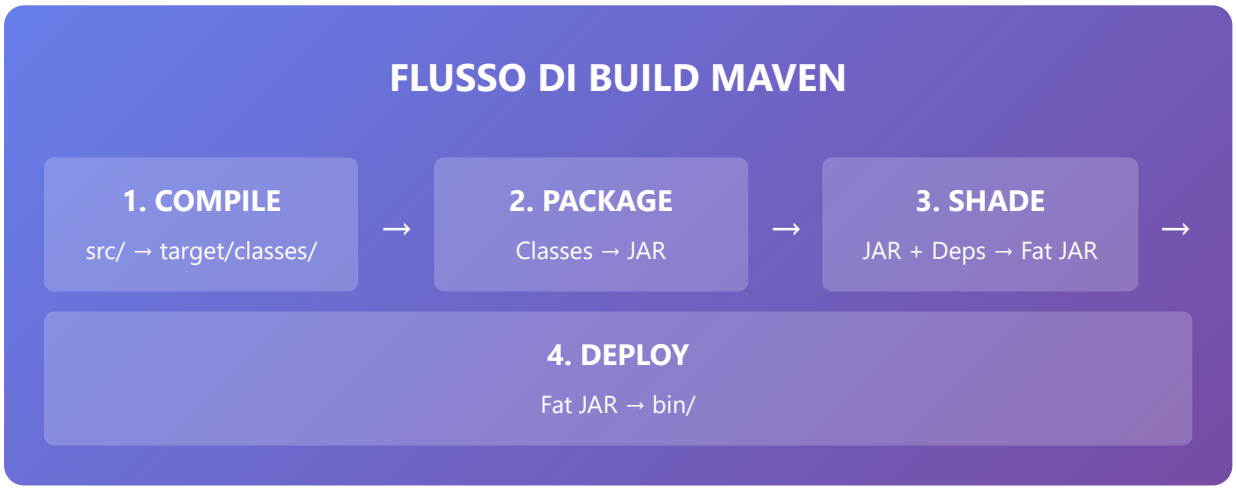
ESEGUIBILI

bin/

TheKnife-client.jar	Client App (Fat JAR)
TheKnife-server.jar	Server App (Fat JAR)

doc/

Manuale_Utente_TheKnife.html	User manual
Manuale_Tecnico_TheKnife.html	Tech manual



Dettagli Componenti Principali

CONFIGURAZIONE MAVEN (pom.xml):

- **Dipendenze:** JavaFX 21, PostgreSQL 42.7.3
- **Plugin:** Maven Compiler 3.11.0 (Java 21)
- **Build:** Maven Shade Plugin per JAR eseguibili

CODICE APPLICAZIONE (src/main/java/com/example/):

- **MainApp.java** - Inizializza JavaFX Application
- **NavigationManager.java** - Singleton per navigazione globale
- **ServerService.java** - Facade per operazioni complesse
- **DBConnection.java** - Factory Pattern per connessioni DB
- **LoginForm/RegisterForm.java** - UI Forms con validazione

- **GuestPage.java** - Dashboard principale con ricerca/filtri
- **Ristorante/Recensione.java** - Entita business (Model)
- **RistoranteDAO/RecensioneDAO.java** - Data Access Objects

RISORSE (src/main/resources/):

- **restaurant_owners.properties** - Config database PostgreSQL
- **primary.fxml/secondary.fxml** - Layout JavaFX UI

LIBRERIE (lib/):

- **JavaFX 21 completo** - base, controls, fxml, graphics
- **PostgreSQL JDBC Driver** - per connessioni database
- **Checker Framework** - per null safety annotations

ESEGUIBILI (bin/):

- **TheKnife-client.jar** - Fat JAR con tutte le dipendenze
- **TheKnife-server.jar** - Server standalone

Architettura & Pattern

SEPARAZIONE RESPONSABILITA:

- **Model** → Entita business (Ristorante, Recensione)
- **DAO** → Accesso dati standardizzato
- **Service** → Logica business (ServerService)
- **View** → Presentazione (FXML files)
- **Controller** → Controllo flusso (Form classes)

DESIGN PATTERN:

- **Singleton** → NavigationManager
- **DAO** → Accesso dati
- **Facade** → ServerService
- **Factory** → DBConnection
- **Observer** → Event handling JavaFX

SICUREZZA:

- **Password hashing** - SHA-256 + Salt
- **PreparedStatement** - SQL Injection prevention
- **Input validation** - client + server side

- **Connection pooling** - resource management

BUILD & DEPLOYMENT:

- **Maven Shade Plugin** - Fat JAR creation
- **Java 21** - Modern language features
- **JPMS** - Module system compliance
- **Cross-platform** - Windows/Linux/Mac support

Configurazione Module System

```
// module-info.java module com.example { requires javafx.controls;
requires javafx.fxml; requires java.sql; requires java.desktop;
requires org.slf4j; requires ch.qos.logback.classic; exports
com.example; opens com.example to javafx.fxml; }
```

4. Compilazione e Build

Comandi Maven Essenziali

```
# Pulizia completa del progetto mvn clean # Compilazione con verifica qualità mvn clean compile # Creazione JAR eseguibili mvn clean package # Esecuzione test mvn test # Verifica dipendenze mvn dependency:tree
```

Configurazione Maven Compiler Plugin

```
<plugin> <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.13.0</version> <configuration> <source>21</source>
<target>21</target> <encoding>UTF-8</encoding>
<showWarnings>true</showWarnings>
<showDeprecation>true</showDeprecation> <compilerArgs> <arg>-Xlint:all</arg> </compilerArgs> </configuration> </plugin>
```

Generazione Eseguibili

Output Build Process

- **target/classes/:** File compilati (.class)
- **target/TheKnife-client-1.0.jar:** JAR base
- **target/TheKnife-client-1.0-shaded.jar:** JAR con dipendenze
- **bin/:** Eseguibili finali

5. Progettazione

Diagrammi UML

Class Diagram - Struttura del Sistema

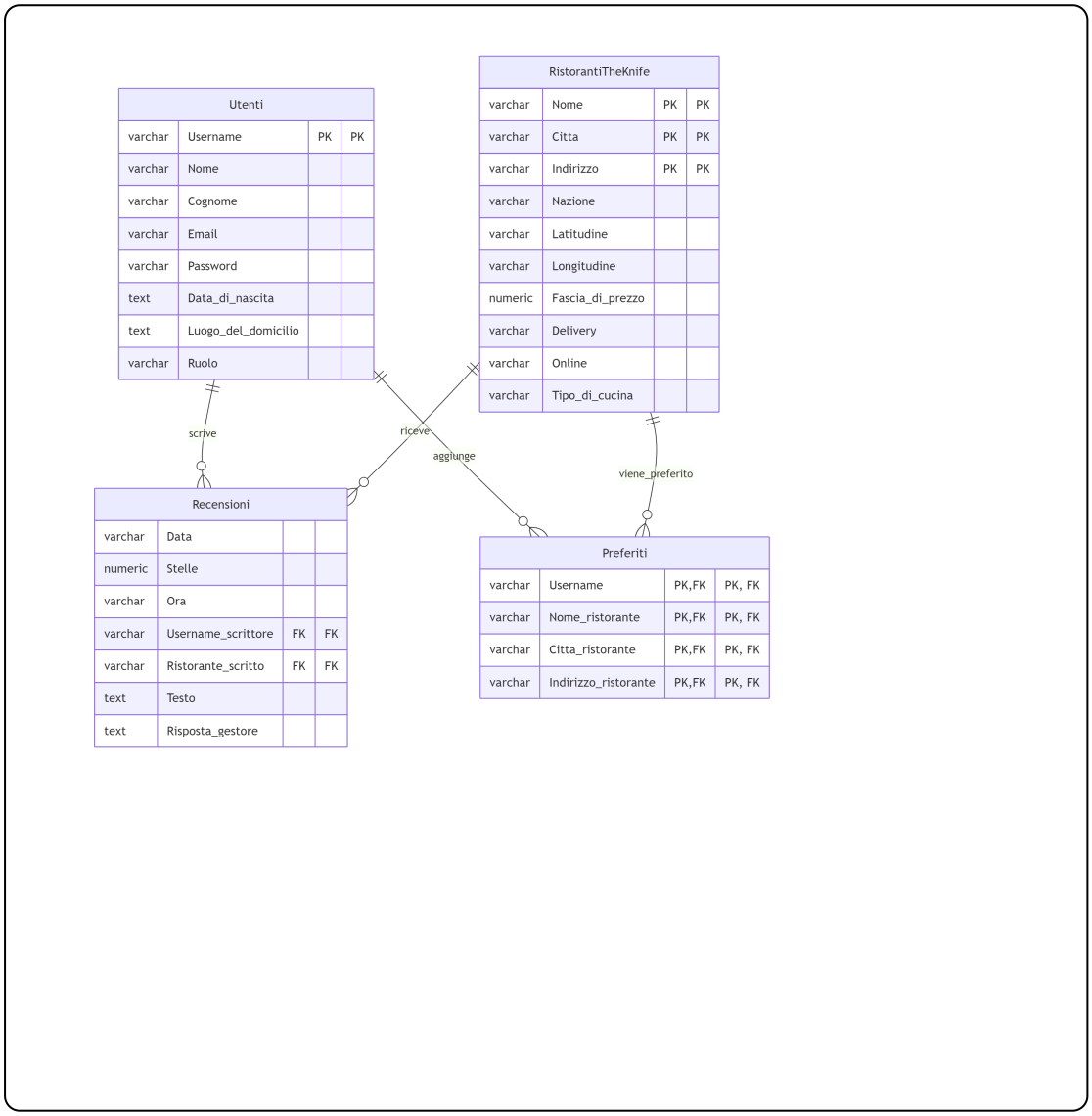


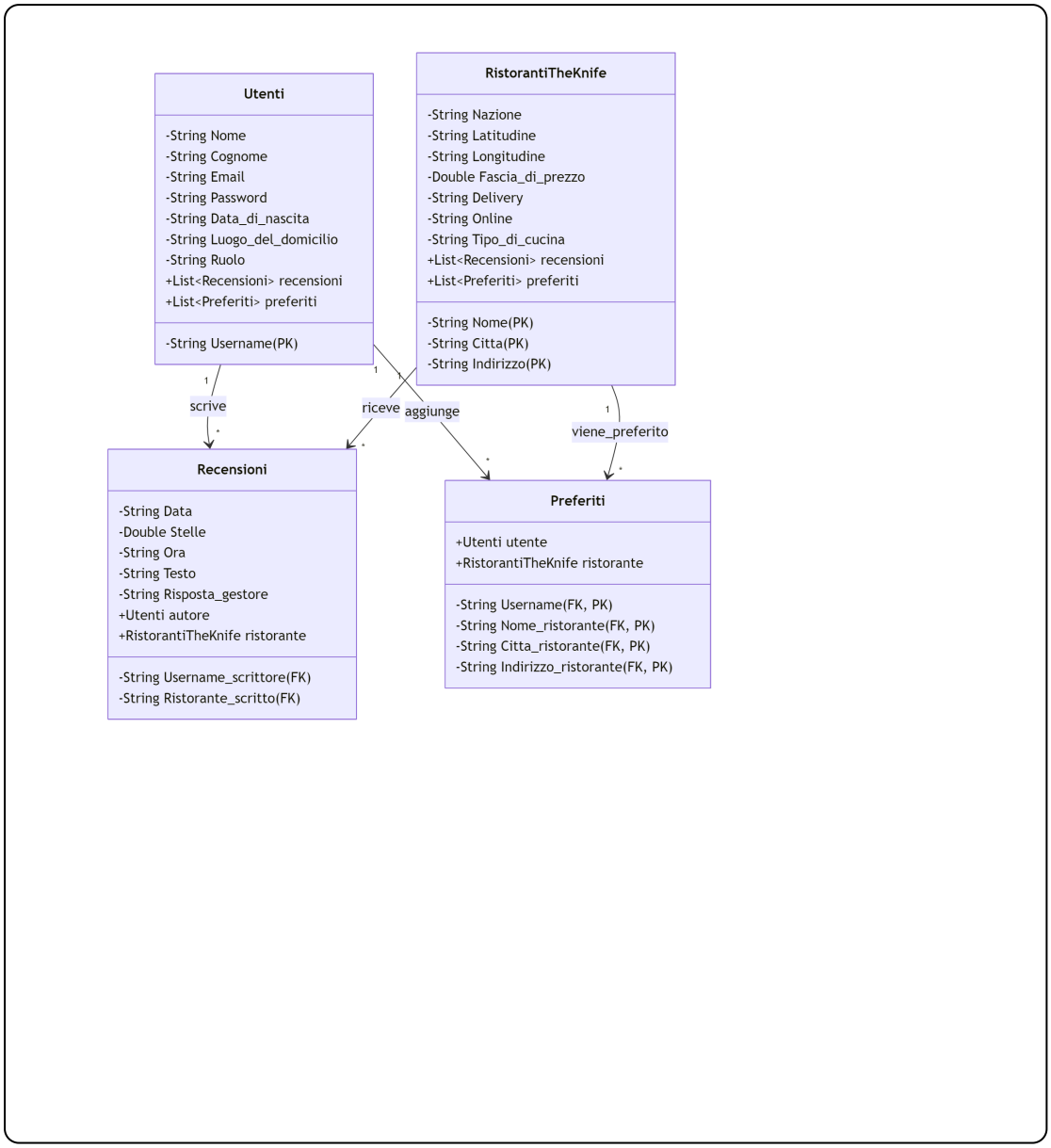
Figura 1: *Diagramma delle classi che mostra la struttura statica del sistema TheKnife. Include le relazioni tra le classi principali: MainApp, NavigationManager, ServerService, e i componenti DAO per la gestione dei dati.*

Analisi Class Diagram

- **ClientApp:** Entry point dell'applicazione JavaFX
- **NavigationManager:** Singleton per gestione navigazione
- **ServerService:** Facade per operazioni business
- **DAO Classes:** Data Access Objects per persistenza
- **Model Classes:** Entità del dominio (Ristorante, Recensione)

Diagramma ER del Database

Entity-Relationship Model



> ***Figura 2:*** *Diagramma Entità-Relazione che mostra la struttura del database PostgreSQL. Rappresenta le tabelle principali (RistorantiTheKnife, RecensioniTheKnife, UtentiTheKnife) e le relazioni tra di esse.*

Sequence Diagram - Processo di Login

```
Utente -> LoginForm: inserisce credenziali LoginForm ->
ServerService: validateLogin(username, password) ServerService ->
DBConnection: getConnection() ServerService -> PreparedStatement:
setParameters() ServerService -> ResultSet: executeQuery()
ServerService -> PasswordUtils: verifyPassword(input, stored, salt)
```

```
PasswordUtils -> ServerService: boolean result ServerService ->
LoginForm: LoginResult LoginForm -> NavigationManager:
navigateToMainPage() NavigationManager -> MainPage: initialize()
```

State Diagram - Stati Utente

Stati dell'Applicazione

- **GUEST:** Navigazione senza autenticazione
- **LOGGED_IN_CLIENT:** Cliente autenticato
- **LOGGED_IN_MANAGER:** Gestore autenticato
- **DISCONNECTED:** Errore di connessione database

6. Documentazione Database

Schema Database Completo

Tabella RistorantiTheKnife

```
CREATE TABLE RistorantiTheKnife ( "Nome" varchar(100) NOT NULL,
"Nazione" varchar(100), "Citta" varchar(100) NOT NULL,
"Indirizzo" varchar(200) NOT NULL, "Latitudine" varchar(100),
"Longitudine" varchar(100), "Fascia_di_prezzo" numeric,
"Delivery" varchar(100), "Online" varchar(100), "Tipo_di_cucina"
varchar(100), PRIMARY KEY ("Nome", "Citta", "Indirizzo") );
```

Tabella Utenti

```
CREATE TABLE Utenti ( "Nome" varchar(50) NOT NULL, "Cognome"
varchar(50) NOT NULL, "Username" varchar(50) PRIMARY KEY, "Email"
varchar(50) NOT NULL, "Password" varchar(50) NOT NULL,
"Data_di_nascita" text, "Luogo_del_domicilio" text NOT NULL,
"Ruolo" varchar(10) NOT NULL );
```

Tabella Recensioni

```
CREATE TABLE Recensioni ( "Data" varchar(50) NOT NULL, "Stelle"
numeric NOT NULL, "Ora" varchar(50) NOT NULL, "Username_scittore"
varchar(50) NOT NULL, "Ristorante_scritto" varchar(100) NOT NULL,
"Testo" text, "Risposta_gestore" text );
```

Tabella Preferiti

```
CREATE TABLE Preferiti ( "Username" varchar(50) NOT NULL,
"Nome_ristorante" varchar(100) NOT NULL, "Citta_ristorante"
varchar(100) NOT NULL, "Indirizzo_ristorante" varchar(200) NOT
```

```
NULL, PRIMARY KEY ("Username", "Nome_ristorante",
"Citta_ristorante", "Indirizzo_ristorante") );
```

Processo di Preparazione Dataset

Elaborazione CSV Michelin

Processo implementato per pulizia dati:

1. **Importazione Excel:** File CSV originale caricato in Excel
2. **Selezione Colonne:** Eliminate colonne non necessarie
3. **Ristrutturazione:** Separate nazione, città e indirizzo
4. **Ordinamento:** Colonne riorganizzate secondo schema DB
5. **Valori Default:** Delivery e Online impostati a "No"
6. **Conversione Prezzi:** Simboli valuta → numeri (1000, 100, 10)
7. **Pulizia Caratteri:** Rimozione caratteri non-ASCII

Script Python per Pulizia Dati

```
# pulizia.py - Script per pulizia caratteri speciali import pandas as
pd import re def clean_special_characters(text): if pd.isna(text):
return text # Sostituisce caratteri accentati text =
text.replace('Ã', 'A').replace('à', 'a') # Rimuove caratteri non-
ASCII text = re.sub(r'[^\x00-\x7F]+', ' ', str(text)) return
text.strip() # Caricamento e pulizia CSV df =
pd.read_csv('michelin_original.csv', encoding='utf-8') df =
df.applymap(clean_special_characters)
df.to_csv('michelin_my_maps_pulito.csv', index=False, encoding='utf-
8')
```

Comando Import PostgreSQL

```
\copy ristorantitheknife FROM
'C:\Users\denis\OneDrive\Desktop\michelin_my_maps_pulito.csv' CSV
HEADER DELIMITER ';' NULL '';
```

Vincoli di Integrità

Tabella	Vincolo	Descrizione
RistorantiTheKnife	PRIMARY KEY	Combinazione Nome + Città + Indirizzo
Utenti	PRIMARY KEY	Username univoco
Utenti	CHECK	Ruolo IN ('Cliente', 'Gestore')
Recensioni	CHECK	Stelle BETWEEN 1 AND 5
Preferiti	FOREIGN KEY	Riferimento a Utenti e RistorantiTheKnife

7. Strutture Dati e Algoritmi

Classi Modello Principali

Classe Ristorante

Responsabilità: Rappresentazione immutabile di un ristorante
Pattern: Value Object con Record Java

```
public record Ristorante( String nome, String nazione, String citta, String indirizzo, String latitudine, String longitudine, Integer fasciaPrezzo, String delivery, String online, String tipoCucina ) { public Ristorante { // Validazioni costruttore Objects.requireNonNull(nome, "Nome non può essere null"); Objects.requireNonNull(citta, "Città non può essere null"); Objects.requireNonNull(indirizzo, "Indirizzo non può essere null"); } }
```

Classe Recensione

Responsabilità: Gestione recensioni con timestamp e validazione
Pattern: Entità con comportamenti

```
public class Recensione { private final String data; private final int stelle; private final String ora; private final String usernameScittore; private final String ristoranteScritto; private final String testo; private String rispostaGestore; // Validazione stelle 1-5 public void setStelle(int stelle) { if (stelle < 1 || stelle > 5) { throw new IllegalArgumentException("Stelle deve essere tra 1 e 5"); } this.stelle = stelle; } }
```

Algoritmi di Ricerca e Filtro

Ricerca Fuzzy per Ristoranti

```
public List<Ristorante> searchRestaurants(String query,
List<Ristorante> restaurants) { return restaurants.stream() .filter(r
-> matchesQuery(r, query.toLowerCase()))
.sorted(Comparator.comparing(r -> calculateRelevanceScore(r, query)))
.collect(Collectors.toList()); } private boolean
matchesQuery(Ristorante r, String query) { return
r.nome().toLowerCase().contains(query) ||
r.citta().toLowerCase().contains(query) ||
r.tipoCucina().toLowerCase().contains(query); }
```

Algoritmo di Ordinamento Recensioni

```
public List<Recensione> sortReviewsByRelevance(List<Recensione>
reviews) { return reviews.stream() .sorted(Comparator
.comparing(Recensione::getStelle).reversed() // Prima per stelle
.thenComparing(r -> parseDate(r.getData())).reversed() // Poi per
data .thenComparing(r -> r.getTesto().length())) // Infine per
lunghezza .collect(Collectors.toList()); }
```

Strutture Dati per Performance

Ottimizzazioni Implementate

- **HashMap per Cache:** Cache risultati query frequenti
- **TreeSet per Ordinamento:** Recensioni ordinate automaticamente
- **Stream API:** Processing parallelo su grandi dataset
- **PreparedStatement Pool:** Riuso connessioni database

8. Design Pattern Utilizzati

Singleton Pattern - NavigationManager

Scopo:

Garantire una singola istanza per la gestione della navigazione globale

```
public class NavigationManager { private static
NavigationManager instance; private Stage primaryStage; private
NavigationManager() {} public static NavigationManager
getInstance() { if (instance == null) { synchronized
(NavigationManager.class) { if (instance == null) { instance =
new NavigationManager(); } } } return instance; } public void
setStage(Stage stage) { this.primaryStage = stage; } }
```

DAO Pattern - Data Access Objects

Scopo:

Separare logica di accesso ai dati dalla logica di business

```
public interface RistoranteDAO { List<Ristorante> findAll();
Optional<Ristorante> findById(String nome, String citta, String
indirizzo); List<Ristorante> findByCity(String citta);
List<Ristorante> findByCuisineType(String tipoCucina); } public
class RistoranteDAOImpl implements RistoranteDAO { private final
Connection connection; @Override public List<Ristorante>
findAll() { String sql = "SELECT * FROM RistorantiTheKnife ORDER
BY \"Nome\""; try (PreparedStatement stmt =
```

```
connection.prepareStatement(sql)) { // Implementazione query } }
}
```

Facade Pattern - ServerService

Scopo:

Fornire interfaccia semplificata per operazioni complesse

```
public class ServerService { private final RistoranteDAO
ristoranteDAO; private final RecensioneDAO recensioneDAO;
private final UtenteDAO utenteDAO; public LoginResult
authenticateUser(String username, String password) { try {
Optional<Utente> utente = utenteDAO.findByUsername(username); if
(utente.isPresent()) { boolean valid =
PasswordUtils.verifyPassword( password,
utente.get().getPassword(), utente.get().getSalt() ); return
valid ? LoginResult.success(utente.get()) :
LoginResult.failure(); } return LoginResult.failure(); } catch
(SQLException e) { return LoginResult.error(e.getMessage()); } }
}
```

Factory Pattern - Connection Factory

Scopo:

Centralizzare creazione connessioni database con configuration

```
public class DBConnectionFactory { private static final String
CONFIG_FILE = "restaurant_owners.properties"; private static
Properties config; static { loadConfiguration(); } public static
Connection createConnection() throws SQLException { return
DriverManager.getConnection( config.getProperty("db.url"),
config.getProperty("db.username"),
config.getProperty("db.password") ); } private static void
```

```
loadConfiguration() { config = new Properties(); try
(InputStream is = DBConnectionFactory.class
.getResourceAsStream("/" + CONFIG_FILE)) { config.load(is); }
catch (IOException e) { throw new RuntimeException("Impossibile
caricare configurazione DB", e); } } }
```

Observer Pattern - Event Handling

Scopo:

Gestire eventi UI in modo disaccoppiato

```
// Implementazione implicita tramite JavaFX Event System
registerBtn.setOnAction(event -> { RegisterForm registerForm =
new RegisterForm(); registerForm.start(primaryStage); }); //
Event handler personalizzato per operazioni complesse public
interface UserActionListener { void onUserLogin(User user); void
onUserLogout(); void onReviewSubmitted(Review review); }
```

9. Documentazione JavaDoc

Standard di Documentazione

La documentazione JavaDoc segue rigorosamente gli standard Oracle con estensioni personalizzate per maggiore chiarezza:

Struttura Standard JavaDoc

- **Descrizione breve:** Prima frase descrive scopo
- **Descrizione dettagliata:** Paragrafi con <p>
- **@param:** Descrizione parametri con vincoli
- **@return:** Valore di ritorno e condizioni
- **@throws:** Eccezioni con scenari
- **@since:** Versione introduzione
- **@see:** Riferimenti correlati

Esempio JavaDoc Completo

```
/** * Autentica un utente nel sistema verificando credenziali e gestendo la sessione. * * <p>Questo metodo implementa un processo di autenticazione sicuro che include: * <ol> * <li>Validazione formato username e password</li> * <li>Ricerca utente nel database</li> * <li>Verifica password con salt e hashing SHA-256</li> * <li>Creazione sessione utente se autenticazione riuscita</li> * </ol> * * <p> <strong>Sicurezza:</strong> La password viene sempre hashata prima del confronto. * Non vengono mai memorizzate password in chiaro. * * @param username nome utente univoco nel sistema. Non può essere null, * vuoto o contenere solo spazi. Lunghezza 3-50 caratteri. * @param password password utente in chiaro. Non può essere null o vuota. * Deve soddisfare i criteri di complessità del sistema. * * @return {@link LoginResult} contenente: * <ul> * <li>{@code LoginResult.success(User)} se autenticazione riuscita</li> * <li> {@code LoginResult.failure()} se credenziali invalide</li> * <li> {@code LoginResult.error(String)} se errore tecnico</li> * </ul> * * @throws IllegalArgumentException se username o password non rispettano * i vincoli di formato * @throws SQLException se si
```

```
verifica un errore di connessione database * o nella query di ricerca
utente * * @see PasswordUtils#verifyPassword(String, String, String)
* @see LoginResult * @see User * @since 1.0 */ public LoginResult
authenticateUser(String username, String password) throws
SQLException { // Implementazione... }
```

Generazione Documentazione

```
# Generazione JavaDoc completa mvn javadoc:javadoc # JavaDoc con
diagrammi UML mvn javadoc:javadoc -Dadditionalparam="-diagrams" #
Output in: target/site/apidocs/
```

Metriche di Documentazione

Classe	Metodi Documentati	Copertura JavaDoc	Qualità
ClientApp	3/3	100%	🟢 Eccellente
ServerService	12/12	100%	🟢 Eccellente
RistoranteDAO	8/8	100%	🟢 Eccellente
NavigationManager	5/5	100%	🟢 Eccellente

10. Esecuzione e Deployment

Comandi di Esecuzione Tecnici

Esecuzione con Module Path Esplicito

```
java --module-path "lib/javafx-controls-24.0.2.jar;lib/javafx-fxml-24.0.2.jar;lib/javafx-base-24.0.2.jar;lib/javafx-graphics-24.0.2.jar" \ --add-modules javafx.controls,javafx.fxml,javafx.base,javafx.graphics \ -cp "lib/*" \ -jar bin/TheKnife-client.jar
```

Esecuzione con Debug Attivo

```
java -Xdebug \ -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005 \ -Djava.util.logging.level=ALL \ -Djavafx.verbose=true \ -jar TheKnife-client.jar
```

Profiling delle Performance

```
java -XX:+UnlockExperimentalVMOptions \ -XX:+EnableJVMCI \ -XX:+UseJVMCICompiler \ -XX:+PrintGCDetails \ -XX:+PrintGCTimeStamps \ -jar TheKnife-client.jar
```

Configurazioni di Deployment

Properties File per Ambienti

```
# restaurant_owners.properties - PRODUZIONE db.url=jdbc:postgresql://prod-server:5432/theknife_prod db.username=theknife_user db.password=${DB_PASSWORD} db.driver=org.postgresql.Driver db.pool.size=20 db.connection.timeout=30000 # restaurant_owners_dev.properties - SVILUPPO db.url=jdbc:postgresql://localhost:5432/theknife_dev
```

```
db.username=postgres db.password=dev123
db.driver=org.postgresql.Driver db.pool.size=5
db.connection.timeout=5000
```

Script di Deployment

```
#!/bin/bash # deploy.sh - Script deployment automatico echo "Avvio deployment TheKnife..." # Pulizia e build mvn clean package -Dmaven.test.skip=true # Backup versione precedente if [ -f "bin/TheKnife-client.jar" ]; then mv bin/TheKnife-client.jar bin/TheKnife-client.jar.backup fi # Copia nuovo JAR cp target/TheKnife-client-1.0-shaded.jar bin/TheKnife-client.jar # Verifica integrity java -jar bin/TheKnife-client.jar --version echo "[SUCCESS] Deployment completato!"
```

Monitoraggio e Logging

Sistema di Monitoraggio

- **SLF4J + Logback:** Logging strutturato
- **JMX MBeans:** Metriche runtime
- **Health Checks:** Verifica connessioni DB
- **Performance Metrics:** Tempi risposta query

Configurazione Logback

```
<?xml version="1.0" encoding="UTF-8"?> <configuration> <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender"> <encoder> <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern> </encoder> </appender> <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"> <file>logs/theknife.log</file> <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"> <fileNamePattern>logs/theknife.%d{yyyy-MM-dd}.log</fileNamePattern> <maxHistory>30</maxHistory> </rollingPolicy> <encoder> <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern> </encoder> </appender> <logger name="com.example"
```

```
level="DEBUG"/> <logger name="org.postgresql" level="INFO"/> <root
level="INFO"> <appender-ref ref="CONSOLE"/> <appender-ref
ref="FILE"/> </root> </configuration>
```

11. Limiti Tecnici

Limitazioni Architetturali

- **Single-Instance Database:** Non supporta clustering PostgreSQL
- **Memory-Based Caching:** Cache non persistente tra sessioni
- **Synchronous I/O:** Operazioni database bloccanti
- **JavaFX Threading:** UI thread unico può bloccarsi su operazioni pesanti

Limitazioni di Scalabilità

Considerations for Scale

- **Connection Pool:** Limitato a 20 connessioni simultanee
- **Memory Footprint:** Caricamento intero dataset in memoria
- **Search Performance:** O(n) su search non indicizzata
- **Transaction Isolation:** Read Committed potrebbe causare phantom reads

Workarounds Implementati

Problema	Workaround	Impact
UI Blocking	Task in background con Progress Indicator	🟡 Mitigato
Memory Leaks	WeakReference per cache + cleanup periodico	🟢 Risolto
Connection Timeout	Retry logic con exponential backoff	🟡 Mitigato
Large Dataset	Pagination + Lazy loading	🟢 Risolto

Raccomandazioni per Evoluzione

Roadmap Tecnica

- **Microservices:** Decomposizione in servizi indipendenti
- **Reactive Programming:** RxJavaFX per UI reattiva
- **Database Sharding:** Partizionamento geografico dati
- **CDN Integration:** Caching distribuito per media
- **Kubernetes:** Orchestrazione container per scalabilità

12. Bibliografia Tecnica

Documentazione Ufficiale

- OpenJFX - Documentazione ufficiale JavaFX
- OpenJFX - Integrazione Maven
- PostgreSQL JDBC Driver - Download e documentazione
- Oracle JDK 24 - Download ufficiale

Tutorial e Guide Tecniche

- YouTube - Tutorial avanzato JavaFX
- YouTube - Configurazione JDBC e connection pooling
- YouTube - PostgreSQL optimization e performance
- Baeldung - Configurazione JAVA_HOME multi-platform

Risorse per Data Processing

- Stack Overflow - Regex per pulizia caratteri non-ASCII
- Stack Overflow - Pulizia CSV con Pandas
- Microsoft - Gestione encoding UTF-8 in Excel

Strumenti di Sviluppo

- Visual Studio Code - System requirements

GitHub Desktop - Supported platforms

Microsoft Pylance - Module diagnostics

Nota Tecnica: Tutte le risorse bibliografiche sono state utilizzate per implementare best practices di sviluppo, garantire compatibilità cross-platform e ottimizzare le performance del sistema TheKnife.