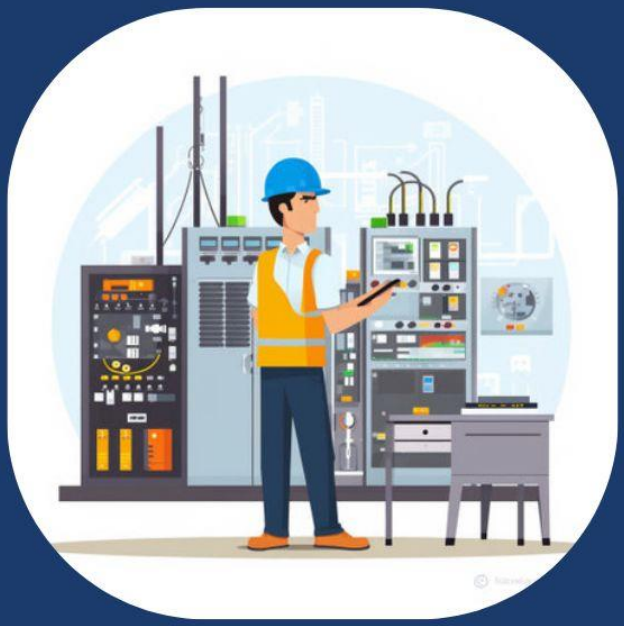
Manuale Tecnico



BOOK RECOMMENDER

App

Indice:

- **Introduzione** 4
- **Key Features** 4
- **Configurazione del Sistema** 5
  - Configurazione PostgreSQL 5
  - Configurazione Client e Server 6
- **Componenti Client-Server** 7
- **Build System e Sicurezza** 11
  - Build System e Dependencies 11
  - Sicurezza 11
- **Classi Principali e Funzionalità** 13
- **Classi Client** 14
- **Gestione Autenticazione e Sessione** 15
  - Procedura di Login 15
  - isValidLogin 15
  - isUserAlreadyLoggedIn 17
  - Registrazione Utenti 19
  - Gestione Race Condition 20
- **Schema del Database** 22
  - Transazioni e Integrità Referenziale 24
  - Inizializzazione del Database 24
- **Funzionalità del Sistema** 25
  - Creazione e Gestione librerie 25
  - Sistema di Valutazione 26
  - Sistema di Raccomandazione 28
  - Visualizzazione Dettagli Libro 28
  - Ricerca Libri 29
- **Gestione della Connettività Remota** 30
  - Implementazione di Ngrok vs RMI 32
  - Connettività 32
  - Sicurezza 32
  - Comportamento con IP dinamici 33
  - Limitazioni di Entrambe le Soluzioni 33

- **Interfaccia Utente con JavaFX e Scene Builder** ____ **34**
- **Architettura dell'Interfaccia** ____ **34**
  - Scene Builder: Sviluppo FXML Dichiarativo ____ 34
  - Injection dei Componenti ____ 36
  - Gestione Avanzata dei Layout ____ 37
  - Responsive Design e Scaling ____ 37
- **Troubleshoting e Gestione Errori** ____ **38**
  - Problematiche Comuni e Soluzioni ____ 38
  - Meccanismi di Recovery ____ 39
- **Elaborazione Dati e Algoritmi** ____ **40**
  - Parsing CSV e Algoritmi di Ricerca ____ 40
- **Sistema di Notifiche** ____ **41**
- **Ulteriori Implementazioni Possibili** ____ **42**
  - Integrazione di ChatGPT per Traduzione e Ricerca Multilingue ____ 42
  - Hashing delle Password ____ 43
  - Sincronizzazione Automatica con Google Drive ____ 44
  - Sistema di Blocco Account dopo Tentativi Falliti ____ 45
- **Conclusioni** ____ **47**
- **Bibliografia e Sitografia** ____ **48**
- **Contatti** ____ **48**

Introduzione

Il Book Recommender System rappresenta un'applicazione strutturata secondo un'architettura client-server che permette agli utenti di gestire collezioni di libri, creare librerie personali, valutare opere letterarie e ricevere consigli personalizzati basati sulle preferenze di lettura.

Il sistema è stato sviluppato in Java, si avvale di JavaFX per l'interfaccia grafica e utilizza PostgreSQL come database relazionale e Ngrok per la comunicazione tra sistemi interconnessi.

Key Features

L'applicazione offre diverse funzionalità chiave per migliorare l'esperienza degli utenti. Gli utenti possono creare e organizzare le proprie librerie digitali, effettuando ricerche rapide per trovare libri in base al titolo, all'autore o all'anno di pubblicazione.

Il nucleo dell'applicazione è costituito dall'algoritmo di raccomandazione che analizza le preferenze dell'utente per suggerire nuove letture potenzialmente interessanti .se non presenti consigli per quel libro da altri utenti

È stata inoltre integrata una funzionalità di connettività remota tramite Ngrok, che consente l'accesso al sistema anche da reti esterne.

Tutte queste funzioni sono accessibili attraverso un'interfaccia utente progettata per essere intuitiva e di facile utilizzo.

Configurazione del Sistema

Configurazione PostgreSQL

Una caratteristica distintiva del Book Recommender è la sua capacità di gestire autonomamente PostgreSQL. Il sistema implementa meccanismi che rilevano la presenza del database, lo avviano se non attivo e possono persino guidare l'utente nell'installazione del software con istruzioni specifiche per ogni piattaforma.

Questa funzionalità si adatta alle diverse configurazioni dei sistemi operativi supportati. Su Windows cerca l'installazione nei percorsi standard, su macOS riconosce le molteplici modalità installative (dall'installer ufficiale a Homebrew fino a PostgreSQL.app), mentre sulle distribuzioni Linux implementa adattamenti specifici per le varie famiglie. Particolare attenzione è stata dedicata ai dispositivi Apple con chip M, con comandi e strategie ottimizzati per questi processori.

Il sistema supporta diverse versioni di PostgreSQL (14, 15, 16), mantenendo compatibilità con le loro differenze strutturali.

Una volta connesso, il sistema verifica la connettività e prepara automaticamente il database, creando schemi e tabelle necessarie, senza nessun intervento manuale necessario.

Questa automazione rappresenta un significativo vantaggio per l'utente finale, eliminando la complessità di configurazione e permettendo l'utilizzo immediato dell'applicazione anche senza conoscenze specialistiche.

- Configurazione Client e Server

Il Book Recommender offre ampie possibilità di personalizzazione sia per il client che per il server, adattandosi a diversi ambienti operativi e requisiti specifici.

Sul lato client, l'interfaccia grafica è stata progettata con caratteristiche di ridimensionabilità dinamica, permettendo all'utente di modificarne le dimensioni attraverso le normali operazioni di ridimensionamento della finestra. L'applicazione risponde in modo responsivo sia all'allargamento che al restringimento dell'interfaccia, adattando proporzionalmente i componenti visuali alle nuove dimensioni. Questa flessibilità garantisce un'esperienza utente ottimale su diverse configurazioni di display, dai monitor più compatti agli schermi ad alta risoluzione.

La connettività è gestita tramite il parametro useNgrok, che determina se la connessione avverrà localmente o remotamente attraverso tunnel sicuri, grazie alla presenza di un token di autenticazione Ngrok che abilita funzionalità avanzate per la connettività remota.

L'accesso al database è configurabile mediante credenziali personalizzabili (dbUser, dbPassword), permettendo l'integrazione con diverse politiche di sicurezza.

Il componente server è progettato con particolare attenzione alla resilienza operativa, per esempio per la gestione delle porte di rete, implementa una strategia di "graceful degradation" con un array ordinato di porte (8888-8892) da tentare sequenzialmente, massimizzando la probabilità di avvio anche in ambienti con restrizioni.

La directory per i file temporanei (TEMP_DATA), per il popolamennto del db, può essere specificata in base alle esigenze di distribuzione dello spazio disco con la relativa cancellazione cosi da non occupare inutilmente spazio una volta spento il lato server.

Attualmente questi parametri sono definiti nel codice, ma in un contesto produttivo sarebbe opportuno estrarli in file di configurazione esterni (JSON, XML, properties), semplificando la gestione di installazioni multiple e l'aggiornamento senza perdita delle personalizzazioni.

▪ Componenti Client-Server

L'architettura del Book Recommender si basa su un modello client-server chiaramente definito con responsabilità ben separate tra i componenti, come illustrato nel diagramma dei casi d'uso in Figura 1.

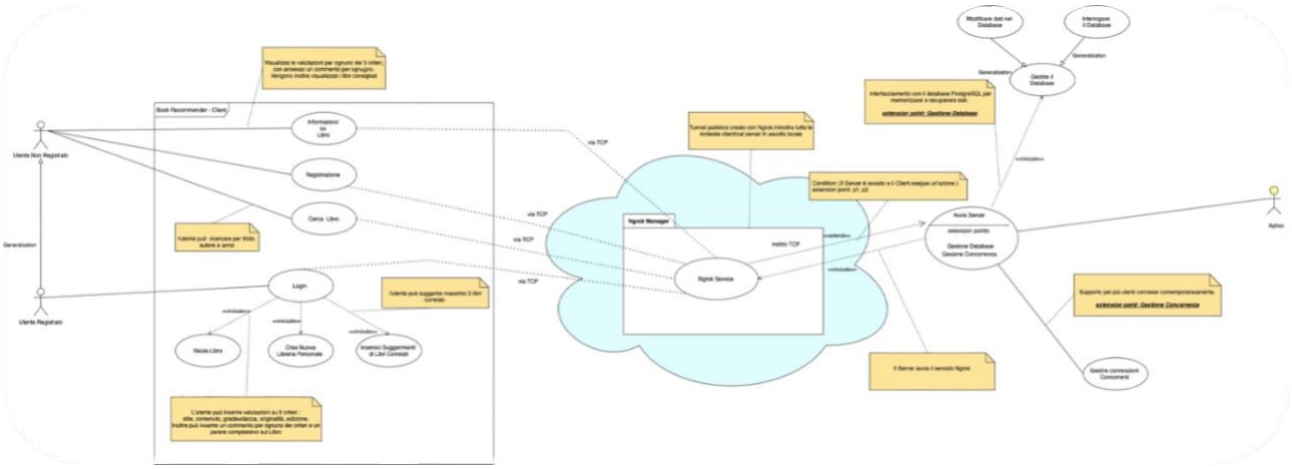


Figura 1: Diagramma dei casi d'uso che mostra le interazioni tra utenti, client e server. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Il server (implementato in Server.java) costituisce il nucleo operativo del sistema. Si occupa dell'inizializzazione del database, gestisce le connessioni client via socket TCP, coordina l'autenticazione e monitora le sessioni attive. Per l'amministrazione, fornisce un'interfaccia completa (ServerInterfaceController) che permette l'avvio/arresto del servizio, il monitoraggio delle connessioni, la gestione di Ngrok e varie funzioni diagnostiche.

Come evidenziato dal diagramma dei casi d'uso, il sistema offre agli utenti non registrati funzionalità di base come la registrazione e informazioni generali sui libri, mentre gli utenti registrati possono accedere a funzionalità avanzate come la creazione di librerie personali, la valutazione dei libri su 5

diversi criteri, e la gestione dei suggerimenti. Il modulo Ngrok Manager agisce come intermediario per connettere client e server attraverso internet, consentendo l'accesso remoto al sistema.

Il processo di inizializzazione del server segue una sequenza sistematica: creazione dell'istanza singleton di DatabaseManager, connessione al database (avviandolo se necessario), configurazione del socket su una porta disponibile, inizializzazione delle strutture dati e avvio del thread di ascolto per le connessioni entranti. Viene attivato automaticamente anche un tunnel Ngrok per l'accesso remoto, come mostrato nella parte destra del diagramma.

Il seguente diagramma di deployment (Figura. 2), illustra l'architettura distribuita del sistema:

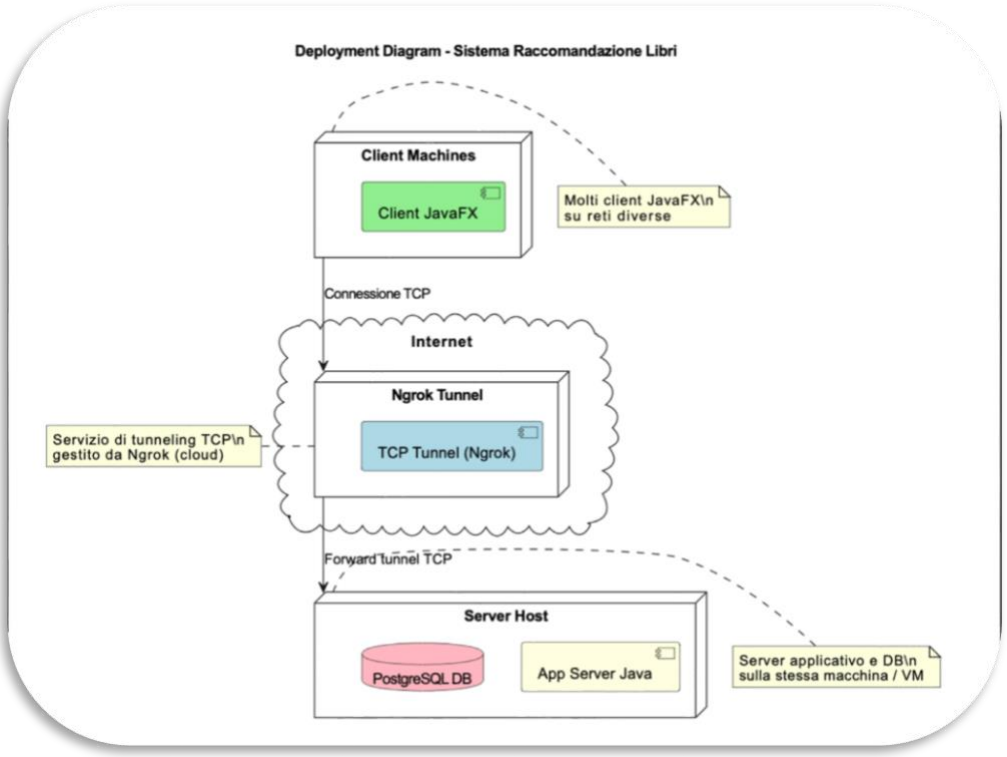


Figura 2: Diagramma di deployment dell'architettura distribuita. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Come evidenziato dal diagramma di deployment (Figura 2), il sistema si articola su tre livelli interconnessi:

1. Client JavaFX distribuiti su reti diverse
2. Servizio cloud Ngrok che gestisce i tunnel TCP come intermediario
3. Server host che mantiene co-locati l'applicazione Java e il database PostgreSQL.

Questa configurazione garantisce connettività universale superando le limitazioni di NAT e firewall se presenti.

La figura seguente illustra il diagramma degli stati (State Machine Diagram), figura 3, che rappresenta il ciclo di vita completo del server:

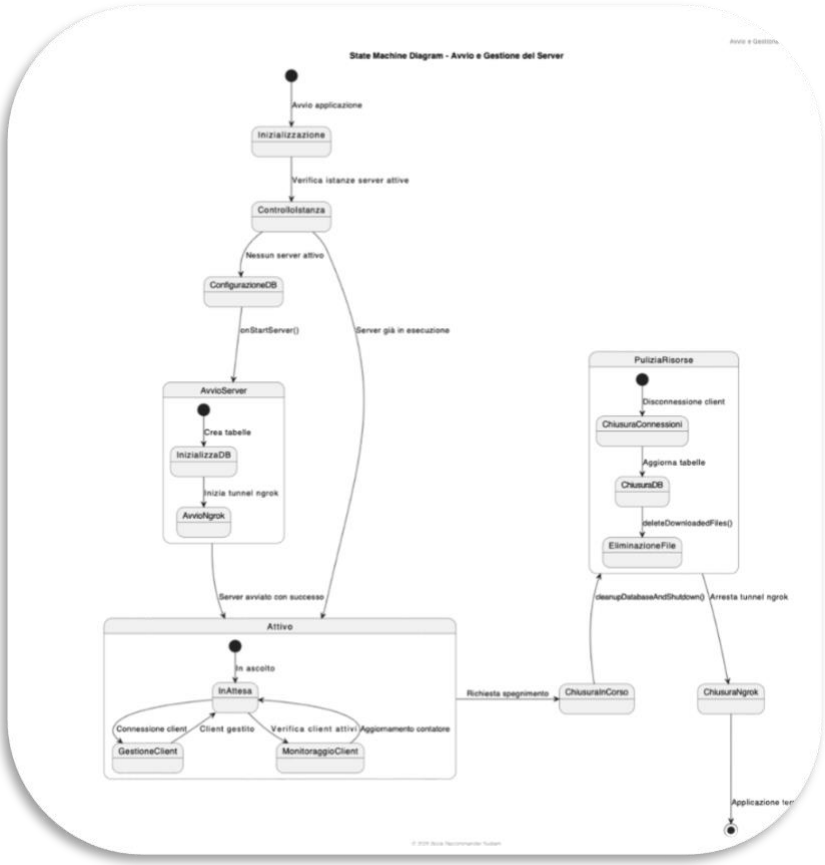


Figura 3: Diagramma degli stati del ciclo di vita del server. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Questo diagramma (Figura 3) evidenzia il flusso di esecuzione del server attraverso i suoi diversi stati operativi. Partendo dall'avvio dell'applicazione, il sistema attraversa una fase di inizializzazione, seguita da un controllo delle istanze server già attive. Se non viene rilevato alcun server in esecuzione, il sistema procede con la configurazione del database e l'avvio di un nuovo server, che include la creazione delle tabelle necessarie e l'inizializzazione del tunnel Ngrok. Una volta raggiunto lo stato "Attivo", il server entra in un ciclo di gestione in cui si alternano attività di attesa, gestione delle connessioni client e monitoraggio delle sessioni attive. In risposta a una richiesta di spegnimento, il sistema esegue una sequenza ordinata di operazioni di pulizia che include la disconnessione dei client, la chiusura delle connessioni, l'aggiornamento delle tabelle, l'eliminazione dei file temporanei e l'arresto del tunnel Ngrok. Questa gestione strutturata degli stati garantisce che il server mantenga sempre un comportamento coerente e prevedibile, anche durante le transizioni tra diverse fasi operative.

Sul versante client (Client.java), le funzionalità principali, visibili nel diagramma dei casi d'uso (Figura 1), includono l'instaurazione e il mantenimento della connessione TCP, la gestione dell'autenticazione utente attraverso login, e l'accesso alle diverse funzionalità come la ricerca di libri, la valutazione su cinque criteri distinti e la creazione di librerie personali. Durante l'operatività, il client trasforma le richieste utente in messaggi per il server e gestisce appropriatamente eventuali disconnessioni. Un thread dedicato monitora costantemente lo stato della connessione, visualizzando immediatamente la schermata di disconnessione in caso di problemi.

L'avvio del client segue un flusso ordinato:

- Connessione al server (locale o remota)
- Presentazione della schermata di login
- Autenticazione
- Creazione della sessione
- Visualizzazione del menu principale e avvio del thread di monitoraggio connessione.

La comunicazione tra i componenti avviene attraverso socket TCP/IP su porte configurabili, con un protocollo che garantisce affidabilità e sicurezza. Dopo l'autenticazione iniziale, la sessione viene registrata centralmente e monitorata con heartbeat periodici (segnali di attività). In caso di interruzioni impreviste, meccanismi specifici notificano l'utente della necessità di riavviare la sessione.

Build System e Sicurezza

Build System e Dependencies

Il Book Recommender si basa su un'architettura di build moderna, adottando Maven come sistema principale per la gestione del progetto e delle dipendenze. Questa scelta tecnologica offre numerosi vantaggi in termini di riproducibilità della build, standardizzazione del processo di sviluppo e semplificazione della gestione delle librerie esterne necessarie al funzionamento dell'applicazione.

La struttura del progetto segue le convenzioni standard di Maven, con una suddivisione ordinata dei file sorgente, delle risorse e delle configurazioni. Questo approccio facilita non solo lo sviluppo collaborativo, ma anche l'integrazione continua e il deployment automatizzato del software.

Tra le dipendenze principali del sistema figura JavaFX, il moderno framework UI che ha sostituito Swing nelle applicazioni Java, offrendo capacità grafiche avanzate e un'architettura basata su CSS per la personalizzazione dell'aspetto visivo. Questa libreria costituisce il fondamento dell'interfaccia utente interattiva del Book Recommender.

Per la connessione al database, il sistema si affida al driver JDBC ufficiale di PostgreSQL, che gestisce efficacemente la comunicazione tra l'applicazione Java e il server database, garantendo prestazioni elevate e supporto completo per le funzionalità avanzate offerte da PostgreSQL.

Sicurezza

La sicurezza rappresenta uno degli aspetti fondamentali su cui ci siamo voluti soffermare durante lo sviluppo di questo progetto, come ad esempio:

1. **Prevenzione SQL Injection**: Utilizzo sistematico di query SQL preparate (Prepared Statements) che separano il codice SQL dai dati forniti dagli utenti, prevenendo efficacemente gli attacchi di SQL Injection.
2. **Prevenzione Login Multipli**: Meccanismo che impedisce sessioni simultanee dello stesso utente attraverso il monitoraggio della tabella active_clients, evitando conflitti di stato e potenziali inconsistenze dei dati.

3. **Sicurezza Connettività Remota**: Implementazione di Ngrok con token di autenticazione personale, endpoint temporanei che cambiano ad ogni riavvio, e comunicazione crittografata end-to-end attraverso tunnel TCP dedicati.
4. **Transazioni Database**: Utilizzo estensivo di transazioni per garantire l'atomicità delle operazioni complesse, con rollback automatico in caso di errori, preservando la consistenza dei dati anche in situazioni impreviste.
5. **Meccanismi di Recovery**: Sistema di rilevamento delle disconnessioni impreviste con notifica immediata agli utenti e prevenzione di operazioni su connessioni interrotte.
6. **Validazione Input**: Implementazione di controlli di validazione in tempo reale degli input utente, con feedback immediato su errori o incongruenze, limitando i caratteri inseribili in specifici campi.
7. **Gestione Accessi Concorrenti**: Implementazione di locking ottimistico e pessimistico per gestire l'accesso concorrente alle risorse condivise, prevenendo race condition.
8. **Monitoraggio Sessioni**: Sistema di monitoraggio continuo delle connessioni attive con heartbeat periodici per verificare la disponibilità reciproca di client e server.
9. **Generazione ID Sessione Univoci**: Creazione di identificativi univoci per ciascuna sessione utente, combinando l'ID utente con timestamp per garantire unicità assoluta.
10. **Integrità dei File**: Verifica dell'integrità dei file scaricati e calcolo di checksum per garantire l'autenticità dei dati.
11. **Pulizia File Temporanei**: Meccanismi di cleanup automatico delle risorse del filesystem, prevenendo accumuli di dati temporanei potenzialmente sensibili.
12. **Gestione Timeout**: Implementazione di timeout configurabili per i tentativi di connessione e rilascio automatico dei lock dopo periodi definiti per prevenire deadlock.
13. **Registrazione Eventi**: Sistema di logging con timestamping preciso per l'analisi cronologica degli eventi, facilitando l'identificazione di anomalie o tentativi di accesso non autorizzati.
14. **Separazione Responsabilità**: Implementazione del pattern MVC che separa nettamente la logica di business dai dati e dalla presentazione, limitando la superficie di attacco.
15. **Centralizzazione Gestione Database**: Utilizzo del pattern Singleton per DatabaseManager, garantendo un punto unico e controllato di accesso al database.

Sebbene il sistema implementi già robuste misure di sicurezza, sono state identificate alcune aree di miglioramento per versioni future, spiegate in un paragrafo successivo specifico.

Classi Principali e Funzionalità

Classi Server

Figura 4: Diagramma delle classi del lato server. *Diagramma dettagliato dell'architettura server*

Nella Figura 4 è possibile osservare in dettaglio la struttura del lato server dell'applicazione:

- **Server.java**: Al centro del diagramma, questa classe rappresenta il core del server con metodi come start(), stop() e isServerRunning(). Come si può vedere, mantiene un riferimento al controller di tipo ServerInterfaceController e al ngrokManager.
- **ServerInterfaceController.java**: Visibile nella parte destra del diagramma, questa classe gestisce un'ampia serie di elementi dell'interfaccia utente come serverRunning (boolean), ngrokEnabled (boolean) e diversi campi di testo e label per monitorare lo stato del server.
- **NgrokManager.java**: Nella parte sinistra della Figura 4, questa classe contiene metodi specifici per la gestione di Ngrok come startTunneling(), stopTunnel() e getPublicEndpoint(). La classe mantiene riferimenti a publicUrl, ngrokProcess e altri elementi necessari per la gestione della connettività remota.

- **DatabaseManager.java**: Posizionato nella parte destra inferiore della Figura 4, implementa il pattern Singleton con metodi come getInstance() e getConnection(), insieme a costanti come DEFAULT_DB_NAME, DEFAULT_PORT e credenziali di accesso al database.

- Classi Client

Figura 5: Rappresentazione completa dell'architettura client dell'applicazione che illustra la struttura dei controller dell'interfaccia utente.

Nella Figura 5 è possibile osservare l'architettura completa del lato client con diverse classi controller:

- Il diagramma mostra chiaramente la struttura relazionale tra i vari controller come **BookSelectionController**, **HomepageController** e **UserMenuController** che gestiscono la navigazione e le operazioni utente.

- Si notano le classi per la gestione delle librerie digitali come **LibrarySelectionController** e **CreateLibraryController** che si occupano della creazione e gestione delle collezioni di libri.
- **RateBookController** e **RecommendBookController** sono evidenti nella parte centrale del diagramma e gestiscono le funzionalità di valutazione e raccomandazione dei libri.
- **LoginController** e **RegistrationController** sono visibili nella parte inferiore della Figura 4, responsabili dell'autenticazione e della registrazione degli utenti.

Queste figure illustrano chiaramente la separazione dei compiti tra client e server, con il server che si occupa della persistenza dei dati e della gestione delle connessioni, mentre il client offre diverse interfacce utente per l'interazione con il sistema di gestione delle librerie digitali.

Gestione Autenticazione e Sessione

Procedura di Login

Il sistema Book Recommender implementa un meccanismo di autenticazione robusto e sicuro utilizzando query SQL preparate (Prepared Statements) all'interno della classe LoginController.java. Questa implementazione rappresenta un esempio concreto di best practice per la protezione contro attacchi di SQL Injection ad esempio nei metodi isValidLogin, isUserAlreasyLoggedIn e handleLogin

isValidLogin

java

```
private boolean isValidLogin(String userId, String password) {
   String sql = "SELECT * FROM users WHERE user_id = ? AND password = ?";

   try (Connection conn = dbManager.getConnection();
       PreparedStatement pstmt = conn.prepareStatement(sql)) {

      pstmt.setString(1, userId);
```

```
        pstmt.setString(2, password);

        ResultSet rs = pstmt.executeQuery();
        return rs.next(); // True se esiste un utente con quelle credenziali
    } catch (SQLException e) {
        return false;
    }
}
```

Questo metodo rappresenta il nucleo del processo di autenticazione. Analizziamo in dettaglio il suo funzionamento:

1. **Definizione della query parametrizzata**: La query SQL viene definita con placeholder (?) invece di concatenare direttamente i valori delle variabili.

   ```
   java
   String sql = "SELECT * FROM users WHERE user_id = ? AND password = ?";
   ```

2. **Preparazione dello statement**: Il metodo prepareStatement crea un oggetto PreparedStatement precompilato.

   ```
   java
   PreparedStatement pstmt = conn.prepareStatement(sql)
   ```

3. **Binding dei parametri**: I valori specifici forniti dall'utente vengono associati ai placeholder in modo sicuro.

   ```
   java
   pstmt.setString(1, userId);
   pstmt.setString(2, password);
   ```

4. **Esecuzione e valutazione**: La query viene eseguita e il risultato viene valutato per determinare la validità delle credenziali.

   ```
   java
   ResultSet rs = pstmt.executeQuery();
   ```

```
        return rs.next();
```

Un attacco di SQL Injection tipico potrebbe tentare di inserire admin' -- come userId, che in una query concatenata tradizionalmente trasformerebbe:

sql

```
SELECT * FROM users WHERE user_id = 'admin' --' AND password = 'qualsiasi'
```

commentando di fatto la verifica della password. Con le prepared statements, il valore viene trattato letteralmente come parte del dato, non come parte della sintassi SQL.

- isUserAlreadyLoggedIn

java

```
private boolean isUserAlreadyLoggedIn(String userId) {
   String sql = "SELECT 1 FROM active_clients WHERE client_id LIKE ?";

   try (Connection conn = dbManager.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

      pstmt.setString(1, "%" + userId + "%");

      ResultSet rs = pstmt.executeQuery();
      return rs.next();
   } catch (SQLException e) {
      return false;
   }
}
```

Questo metodo implementa una verifica aggiuntiva per prevenire login multipli dello stesso utente:

1. **Pattern matching sicuro**: Utilizza l'operatore LIKE con pattern matching attraverso una prepared statement.

   java

   ```
   pstmt.setString(1, "%" + userId + "%");
   ```

2. **Gestione degli errori**: Implementa un comportamento di fallback sicuro in caso di eccezioni.

   java

   ```
   catch (SQLException e) {
      return false;
   }
   ```

3. **Sicurezza proattiva**: Anche se il pattern contiene caratteri speciali (%), la prepared statement garantisce che vengano interpretati come parte del pattern di ricerca e non come istruzioni SQL.

- handleLogin

java

```
@FXML
public void handleLogin(ActionEvent event) {
   String userId = userIdField.getText();
   String password = passwordField.getText();

   if (isValidLogin(userId, password)) {
      errorMessage.setVisible(false);

      if (isUserAlreadyLoggedIn(userId)) {
         showAlreadyLoggedInAlert(event);
      } else {
```

```
            registerUserConnection(userId, true);
            navigateToUserMenu(event, userId);
        }
    } else {
        errorMessage.setVisible(true);
    }
}
```

Questo metodo illustra la sequenza completa di autenticazione:

1. Acquisisce le credenziali dall'interfaccia utente
2. Verifica la loro validità tramite la query preparata in isValidLogin
3. Se valide, controlla se l'utente è già connesso con isUserAlreadyLoggedIn
4. Se non è già connesso, registra la connessione e naviga al menu utente

La combinazione di prepared statements e controlli di sicurezza aggiuntivi (come la prevenzione di login multipli) crea un sistema di autenticazione che non solo protegge contro attacchi di iniezione SQL, ma implementa anche misure di protezione contro altre vulnerabilità potenziali come la condivisione non autorizzata di account.

- Registrazione Utenti

Il processo di registrazione degli utenti rappresenta un elemento gestito dalla classe RegistrationController.java. Questa componente implementa una serie di controlli rigorosi per garantire che solo utenti con dati validi e completi possano accedere all'applicazione.

La procedura inizia con una validazione approfondita dei dati di input forniti dall'utente. Questi includono l'identificativo utente (user_id), il nome completo, il codice fiscale italiano, l'indirizzo email e la password. Ciascun campo viene sottoposto a controlli specifici per verificarne la conformità ai requisiti del sistema.

Una verifica particolarmente importante riguarda l'unicità dell'ID utente nel database. Questo controllo previene la creazione di account duplicati e garantisce che ogni utente sia univocamente

identificabile all'interno del sistema. Il sistema verifica la disponibilità dell'ID richiesto prima di procedere con gli altri controlli prima dell' avvenuta registrazione.

La validazione delle password viene effettuata attraverso espressioni regolari (regex) che verificano la conformità dei requisiti di sicurezza. Il sistema garantisce che le password soddisfino standard elevati come una lunghezza minima di 8 caratteri e la presenza di caratteri maiuscoli, minuscoli, numeri e caratteri speciali.

Anche il codice fiscale viene sottoposto a una validazione rigorosa tramite regex, assicurando che rispetti il formato standard di 16 caratteri con la corretta sequenza di lettere e numeri. Questa verifica è particolarmente importante per un sistema che opera in contesto italiano.

La validazione dell'indirizzo email avviene attraverso pattern standard che verificano non solo la sintassi generale (presenza di @ e dominio) ma anche l'appartenenza a domini accettati, migliorando così la qualità dei dati registrati nel sistema.

Una volta completate tutte le verifiche con esito positivo, i dati dell'utente vengono registrati nel database, creando così un nuovo account attivo nel sistema.

- Gestione Race Condition

Prima di completare l'autenticazione tramite login, viene implementato un meccanismo per la prevenzione dei login multipli attraverso la tabella active_clients, rappresentando una soluzione per evitare **problemi di concorrenza** nell'accesso alle risorse.

Quando un utente effettua l'accesso, il sistema esegue una verifica per determinare se esiste già una sessione attiva per lo stesso utente (controllando lo user_id), consultando la tabella active_clients e cercando record associati all'identificativo fornito. In caso di rilevamento di una sessione già attiva, il sistema informa l'utente della situazione, impedendo accessi simultanei che potrebbero compromettere l'integrità dei dati.

In caso di autenticazione riuscita, il sistema registra la nuova sessione nella tabella active_clients, memorizzando tutte le informazioni necessarie per tracciare l'attività dell'utente. Durante questa fase viene generato un ID cliente univoco, composto dalla combinazione dell'identificativo utente e di un timestamp, garantendo così l'unicità di ogni sessione. Al termine della procedura, se tutte le verifiche

hanno dato esito positivo, l'utente viene reindirizzato al menu principale dell'applicazione, da cui può accedere a tutte le funzionalità disponibili.

Lo stato di connessione di ciascun utente viene costantemente aggiornato attraverso il metodo DatabaseManager.updateClientConnection(), che mantiene aggiornati i timestamp di attività. Questa operazione permette di distinguere le sessioni attive da quelle potenzialmente abbandonate o interrotte.

Dal lato server, un thread dedicato esegue un monitoraggio periodico delle connessioni attive, verificando la presenza di client inattivi oltre una certa soglia temporale. Quando viene rilevata un'inattività prolungata, il sistema può intervenire automaticamente rimuovendo la sessione dalla tabella active_clients.

In caso di arresto pianificato del server, il sistema provvede alla disconnessione automatica di tutti gli utenti attivi, aggiornando il loro stato e inviando notifiche appropriate. Questo garantisce una chiusura pulita delle sessioni e previene situazioni di inconsistenza dei dati.

Per quanto riguarda le disconnessioni anomale, come quelle causate da problemi di rete o crash dell'applicazione client, il sistema implementa un meccanismo di rilevamento attraverso un thread dedicato di monitoraggio. Questo thread verifica continuamente lo stato della connessione e, in caso di problemi, avvia procedure di ripristino o notifica l'utente della necessità di riavviare la sessione.

L'approccio a due livelli adottato per il monitoraggio rappresenta una soluzione che combina verifiche lato server e lato client. Sul server, il thread periodico identifica e gestisce i client inattivi, mentre sul lato client, un thread dedicato verifica costantemente la connessione al server, garantendo così un rilevamento tempestivo di eventuali problemi di comunicazione.

Questo approccio offre numerosi vantaggi:

- Impedisce l'insorgere di conflitti di stato evitando che più sessioni dello stesso utente possano modificare contemporaneamente gli stessi dati, minimizza il rischio di race condition, situazioni in cui operazioni parallele potrebbero generare risultati imprevisti
- Garantisce l'integrità transazionale delle operazioni, assicurando che le attività vengano completate in modo atomico senza interferenze esterne

- Ottimizza l'allocazione delle risorse evitando duplicazioni non necessarie di processi e connessioni.

Dal punto di vista della sicurezza, impedendo accessi multipli contemporanei, il sistema offre una protezione aggiuntiva nel caso in cui le credenziali di un utente vengano compromesse, limitando potenzialmente l'impatto di accessi non autorizzati.

Schema del Database

Il modello dati del sistema implementa relazioni chiare e ben definite tra le diverse entità che compongono il dominio applicativo, come illustrato nel diagramma ER (Figura 5).

Figura 5: Diagramma ER del database. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Come si può osservare nella Figura 5, l'entità **Users** rappresenta il nucleo centrale del sistema, attorno alla quale si sviluppano tutte le altre relazioni. Gli utenti registrati sono connessi a molteplici funzionalità attraverso diverse relazioni:

1. La relazione **legge** collega gli utenti all'entità **Books**, rappresentando l'accesso degli utenti al catalogo dei libri disponibili nel sistema. I libri contengono tutte le informazioni bibliografiche necessarie, come titolo, autore, anno di pubblicazione e categoria.
2. Attraverso la relazione **crea e gestisce**, gli utenti sono collegati all'entità **libraries**, che rappresenta le collezioni personali create e personalizzate secondo le preferenze individuali.
3. Le **libraries** sono poi collegate all'entità **library_books** tramite la relazione **contiene libro**, implementando un collegamento many-to-many tra librerie e libri, poiché un libro può appartenere a più librerie e una libreria può contenere numerosi libri.
4. La relazione **valuta** collega gli utenti all'entità **book_ratings**, che raccoglie i giudizi e le recensioni degli utenti sui libri letti. Questo sistema implementa una valutazione multicriterio che considera aspetti come stile, contenuto, gradevolezza, originalità e qualità dell'edizione.
5. Tramite la relazione **raccomanda**, gli utenti sono collegati all'entità **book_recommendations**, che rappresenta i consigli di lettura generati in base alle preferenze e alle valutazioni espresse dagli utenti.
6. Infine, la relazione **connessione attiva** collega gli utenti all'entità **active_clients**, che gestisce le sessioni attive nel sistema, impedendo login multipli dello stesso utente e monitorando lo stato delle connessioni.

Il diagramma ER evidenzia chiaramente come l'entità Users sia il fulcro dell'intero sistema, con relazioni che si diramano verso tutte le altre entità principali. Questa struttura centralizzata permette una gestione efficiente delle informazioni e delle operazioni, garantendo al contempo l'integrità referenziale attraverso le relazioni ben definite tra le entità.

- Transazioni e Integrità Referenziale

Come evidenziato dalle linee di connessione nel diagramma ER (Figura 5), il sistema implementa rigorosi meccanismi di gestione delle transazioni e di integrità referenziale. I vincoli di integrità referenziale, rappresentati dalle relazioni a forma di diamante, sono applicati attraverso chiavi esterne con operazioni in cascata per gli eventi di DELETE, assicurando che non rimangano riferimenti orfani quando un'entità principale viene rimossa.

Questo approccio garantisce che operazioni complesse, come la creazione di una libreria con la contestuale aggiunta di libri (rappresentata dalle relazioni "crea e gestisce" e "contiene libro" nel diagramma), siano eseguite in modo atomico. In caso di errore durante l'esecuzione di una di queste operazioni composite, il sistema è in grado di effettuare un rollback completo, ripristinando lo stato precedente del database e prevenendo così inconsistenze nei dati.

- Inizializzazione del Database

L'inizializzazione del database rappresenta la fase nell'avvio del sistema ed è gestita dal metodo initializeDatabase() presente nella classe ServerInterfaceController. Questo processo implementa nella pratica lo schema concettuale mostrato nel diagramma ER (Figura 5).

La procedura inizia con l'eliminazione (drop) delle tabelle esistenti, se presenti. Questa operazione assicura che non vi siano residui di dati precedenti che potrebbero causare conflitti o inconsistenze. La rimozione delle tabelle avviene in un ordine specifico per rispettare i vincoli di integrità referenziale visibili nel diagramma ER, partendo dalle tabelle dipendenti (library_books, book_ratings, book_recommendations, active_clients) fino ad arrivare a quelle principali (libraries, books, users).

Successivamente, il sistema procede alla creazione delle tabelle seguendo lo schema definito nel diagramma ER. Ogni tabella viene costruita con i campi appropriati, specificando tipi di dati, vincoli di unicità, chiavi primarie e chiavi esterne che riflettono le relazioni mostrate nel diagramma.

Per ottimizzare le prestazioni delle query più frequenti, vengono creati indici specifici sulle colonne maggiormente utilizzate nelle operazioni di ricerca e filtraggio, particolarmente importanti per le

entità con molte relazioni come Users. Questi indici migliorano significativamente i tempi di risposta del sistema, soprattutto quando la base dati cresce in dimensioni.

L'ultimo passaggio dell'inizializzazione prevede il popolamento del database con dati iniziali prelevati da file CSV scaricabili da Google Drive mediante un token per ogni file, creando le istanze concrete delle entità descritte nel diagramma ER.

Funzionalità del Sistema

Creazione e Gestione librerie

Il file CreateLibraryController consente agli utenti di creare nuove librerie personali, fornendo un'interfaccia per specificare il nome e altre caratteristiche della collezione. Durante questo processo, il sistema verifica automaticamente la non esistenza di nomi duplicati per lo stesso utente, garantendo **l'unicità delle liberie personali** attraverso il vincolo UNIQUE(user_id, library_name).

Per accedere alle librerie già create, gli utenti utilizzano il file LibrarySelectionController, che visualizzando cosi l'elenco delle collezioni disponibili con informazioni aggiuntive come il numero di libri contenuti e la data di creazione

L'aggiunta di libri alle librerie è gestita dalla classe AddBooksToLibraryController, che offre funzionalità di ricerca nel catalogo generale e permette di selezionare i titoli da includere nella collezione corrente. **Il sistema previene automaticamente l'inserimento di duplicati all'interno della stessa libreria.**

La visualizzazione dei libri contenuti nelle librerie avviene tramite il BookSelectionController, che presenta l'elenco dei titoli con dettagli bibliografici essenziali e offre opzioni per il filtraggio e l'ordinamento in base a diversi criteri.

- Sistema di Valutazione

Il sistema di valutazione implementato in RateBookController rappresenta un approccio sofisticato per la raccolta dei giudizi degli utenti sui libri letti. Come illustrato dal diagramma di sequenza (Figura 6), il processo inizia quando l'utente seleziona una libreria e un libro specifico dall'interfaccia client, attivando una catena di interazioni che attraversa tutti i livelli dell'architettura, dal client fino al database PostgreSQL, passando attraverso il tunnel Ngrok.

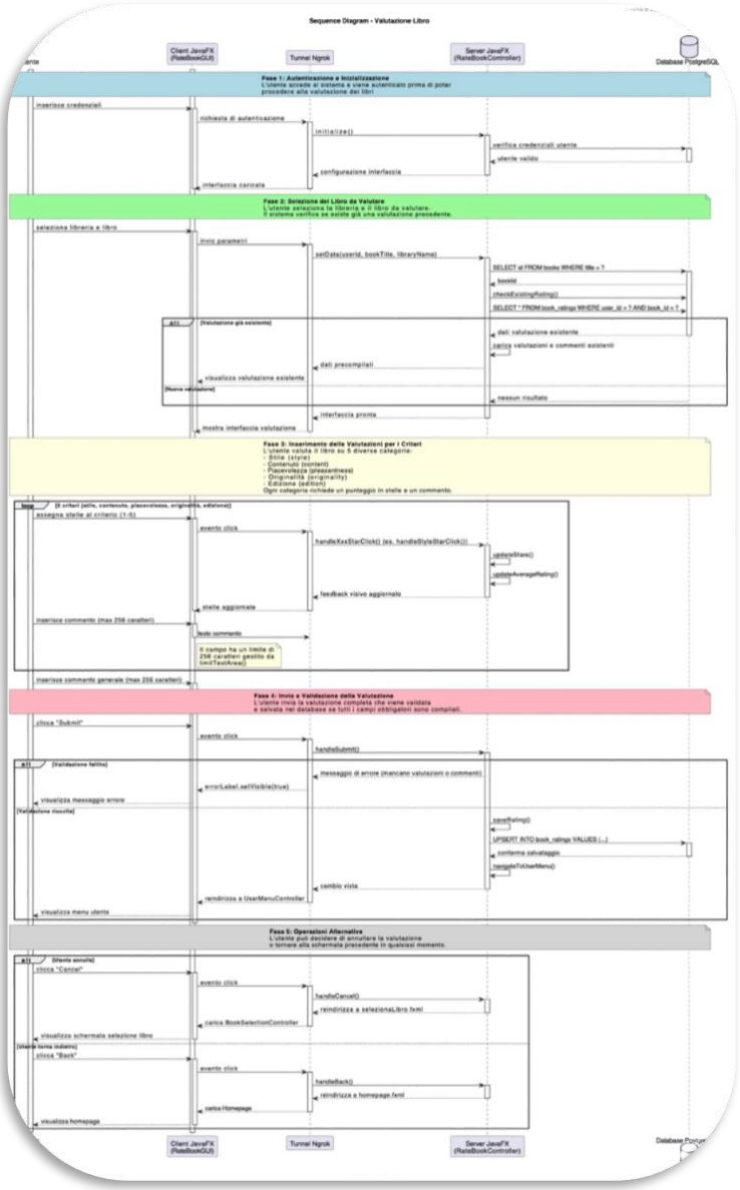


Figura 6: Diagramma di sequenza che illustra il processo di valutazione di un libro. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Come mostrato nella Fase 2 del diagramma (Figura 6), quando l'utente accede alla funzionalità di valutazione, il client invia i parametri necessari (userId, bookTitle, libraryName) al server mediante il metodo setData(). Il server esegue quindi due query principali: la prima verifica l'esistenza del libro nel database, mentre la seconda controlla se l'utente ha già inserito una valutazione per quel titolo. Se una valutazione precedente esiste, il sistema la recupera e la presenta all'utente nell'interfaccia, permettendogli di visualizzare e modificare i giudizi già espressi.

Come evidenziato nella Fase 3 del diagramma (Figura 6), gli utenti possono valutare ciascun libro su cinque diverse caratteristiche: stile di scrittura, qualità dei contenuti, gradevolezza complessiva, originalità dell'opera ed elementi editoriali. Per ogni criterio, l'utente assegna un valore da 1 a 5 attraverso un'interfaccia intuitiva basata su stelle, dove ogni evento di click attiva i metodi handleXxxStarClick() che aggiornano immediatamente lo stato visivo e i valori sottostanti. Il sistema fornisce feedback visivo aggiornando l'interfaccia in tempo reale, offrendo così un'esperienza reattiva e gratificante.

Come si può osservare dal ciclo di interazioni nel diagramma (Figura 6), oltre ai valori numerici, l'utente può inserire commenti specifici per ciascuna caratteristica valutata, con un limite di 256 caratteri gestito dal componente limitTextArea(). È disponibile anche un campo per il commento generale, soggetto allo stesso limite di caratteri. Al completamento della valutazione, l'evento click sul pulsante "Submit" attiva il metodo handleSubmit(), che avvia la sequenza di salvataggio nel database.

La Fase 4 del diagramma (Figura 6) illustra il processo di persistenza dei dati: il sistema calcola automaticamente una valutazione media basata sui punteggi assegnati e utilizza la funzione saveRating() per eseguire un'operazione UPSERT nel database. Questo approccio garantisce che, quando un utente modifica una valutazione esistente, il sistema aggiorni i valori nel database invece di creare record duplicati, mantenendo così l'integrità referenziale. Al termine dell'operazione, come mostrato nella parte finale del diagramma (Figura 6), l'utente riceve una conferma di salvataggio e viene reindirizzato al menu utente, completando così il ciclo di interazione.

- Sistema di Raccomandazione

Il sistema di raccomandazione, implementato, offre funzionalità per la creazione e gestione di consigli di lettura personalizzati. Questo componente consente agli utenti di partecipare attivamente al processo di raccomandazione, arricchendo l'esperienza complessiva della community.

Gli utenti possono selezionare fino a tre libri da consigliare in associazione a un libro di origine, creando così collegamenti tra opere che presentano affinità tematiche, stilistiche o di altro tipo.

Il sistema permette inoltre di visualizzare e modificare i consigli precedentemente inseriti, garantendo così la possibilità di aggiornare i suggerimenti in base a nuove letture o considerazioni. Questa funzionalità favorisce un continuo miglioramento della qualità delle raccomandazioni all'interno del sistema.

Oltre ai consigli manuali, il BookDetailsController implementa un meccanismo di generazione automatica di raccomandazioni basate sulla categoria dei libri in assenza di libri consigliati dalla community. Questo algoritmo analizza le caratteristiche del libro visualizzato e identifica altre opere con attributi simili che potrebbero interessare l'utente.

Il sistema adotta quindi un approccio ibrido che combina raccomandazioni esplicite inserite manualmente dagli utenti e raccomandazioni automatiche generate dall'algoritmo.

- Visualizzazione Dettagli Libro

La visualizzazione dettagliata dei libri, gestita dal BookDetailsController, fornisce una presentazione completa e organizzata delle informazioni relative a ciascuna opera presente nel catalogo. Questa interfaccia rappresenta un punto centrale dell'esperienza utente, offrendo accesso a tutti i dati rilevanti in un'unica schermata.

La visualizzazione include innanzitutto le informazioni bibliografiche essenziali, come titolo, autore, categoria, editore e anno di pubblicazione. Questi dati forniscono un'identificazione immediata del libro e il suo contesto editoriale.

Un elemento particolarmente importante è la presentazione delle valutazioni medie per ogni caratteristica valutata dagli utenti. Queste medie offrono un'indicazione sintetica della qualità percepita dell'opera secondo diverse dimensioni, permettendo un rapido apprezzamento dei suoi punti di forza e debolezza.

Le recensioni degli utenti vengono mostrate con formattazione e colori differenziati per distinguere i vari contributori. Questa visualizzazione facilita la lettura e l'identificazione delle diverse opinioni, arricchendo l'esperienza di consultazione.

La sezione dedicata ai libri consigliati include sia i suggerimenti inseriti manualmente dagli utenti che quelli generati automaticamente dal sistema in assenza di quelli umani, offrendo un'ampia gamma di possibili letture correlate.

Il sistema supporta inoltre una rappresentazione visiva delle valutazioni sotto forma di stelle, utilizzando una tecnica di rendering personalizzato che migliora l'immediatezza della percezione dei giudizi. Questa visualizzazione include anche il supporto per le mezze stelle, permettendo una rappresentazione più accurata.

- Ricerca Libri

La funzionalità di ricerca libri, implementata nel BookService, rappresenta uno strumento fondamentale per la navigazione all'interno del catalogo, offrendo diverse modalità di interrogazione adatte a differenti esigenze degli utenti.

La ricerca per titolo implementa un approccio case-insensitive con supporto per ricerche parziali, permettendo di trovare opere anche conoscendo solo una porzione del titolo. Questa flessibilità migliora significativamente l'usabilità, specialmente quando gli utenti non ricordano con precisione i titoli completi.

Analogamente, la ricerca per autore adotta un approccio case-insensitive con supporto per ricerche parziali, consentendo di localizzare facilmente le opere di un determinato scrittore anche con informazioni incomplete. Questa modalità risulta particolarmente utile per esplorare la bibliografia di specifici autori.

Per ricerche più specifiche, il sistema supporta anche la combinazione di autore e anno, permettendo di identificare con precisione opere pubblicate in un determinato periodo da un autore specifico. Questa funzionalità è implementata combinando la ricerca case-insensitive per l'autore con un controllo esatto sull'anno di pubblicazione.

Una modalità aggiuntiva consente di individuare i libri più valutati, basandosi sulla media delle valutazioni espresse dagli utenti. Questa funzione aiuta a scoprire opere particolarmente apprezzate dalla community, facilitando l'identificazione di letture potenzialmente interessanti.

▪ Gestione della Connettività Remota

La connettività remota rappresenta un aspetto fondamentale del sistema Book Recommender, implementata attraverso il servizio Ngrok per garantire l'accesso al server PostgreSQL anche attraverso NAT (Network Address Translation) e firewall senza necessità di configurazioni complesse di rete.

• Connessione remota con Ngrok

Il sistema integra Ngrok attraverso la classe NgrokManager per esporre il database PostgreSQL locale agli utenti remoti, anche su reti wifi diversi. L'implementazione segue un'architettura a tre livelli:

- Livello Client: Rappresentato dalla classe NgrokManager che gestisce l'eseguibile Ngrok e ne controlla il ciclo di vita.
- Livello Gateway: Il servizio cloud di Ngrok che fornisce endpoint pubblici accessibili globalmente.
- Livello Servizio: Il database PostgreSQL locale che viene esposto attraverso il tunnel.

Il diagramma di sequenza del processo è il seguente:

Client Remoto → Endpoint Ngrok Pubblico → Tunnel TCP → PostgreSQL Locale

Il processo di connessione di un client remoto include:

- Avvio del tunnel Ngrok dal server (NgrokManager.startNgrokTunnel)
- Ottenimento dell'URL e porta pubblici (NgrokManager.getPublicUrl, NgrokManager.getPublicPort)
- Generazione della stringa di connessione JDBC (NgrokManager.getJdbcConnectionString)
- Visualizzazione delle informazioni di connessione nell'interfaccia del server
- Connessione del client remoto utilizzando l'URL e la porta forniti

L'integrazione di Ngrok nel sistema prevede diversi componenti:

- **Download e Installazione Automatici**: Il sistema implementa un meccanismo di auto-installazione cross-platform che rileva OS e architettura di sistema (x86, x64, ARM), scarica il binario appropriato dalla CDN ufficiale di Ngrok, estrae e configura i permessi di esecuzione appropriati, supportando Windows, macOS (Intel e Silicon) e distribuzioni Linux.
- **Configurazione e Autenticazione**: Il sistema configura automaticamente Ngrok con un token di autenticazione registrato, permettendo l'utilizzo del servizio con capacità estese.
- **Avvio e Monitoraggio del Tunnel TCP**: Il sistema avvia un tunnel TCP specifico per PostgreSQL.
- **Parsing degli Endpoint Pubblici**: Il sistema implementa algoritmi per estrarre l'host e la porta pubblici dall'API locale di Ngrok.
- **Generazione della Stringa di Connessione JDBC**: Il sistema genera stringhe di connessione JDBC dinamiche per gli endpoint Ngrok.
- **Gestione del Ciclo di Vita**: Il sistema implementa una terminazione controllata del tunnel per evitare processi orfani.

Il sistema automatizza completamente questo processo, rendendo la connettività remota trasparente per gli utenti finali e gli amministratori.

- Implementazione di Ngrok vs RMI

Nel progetto Book Recommender System, la scelta dell'infrastruttura di connettività remota ha rappresentato un elemento cruciale per garantire un'esperienza d'uso efficace e accessibile. L'implementazione ha privilegiato Ngrok rispetto a RMI (Remote Method Invocation) di Java, una decisione supportata da solide considerazioni tecniche e pratiche.

- Connettività

Ngrok offre una connettività universale che supera le limitazioni tipiche delle reti moderne. Mentre i sistemi tradizionali come RMI richiedono configurazioni complesse di porte e regole firewall, Ngrok implementa un approccio innovativo basato su tunnel outbound che si adatta pressoché a qualsiasi ambiente di rete.

Questa caratteristica si rivela particolarmente preziosa in scenari reali dove gli utenti operano da reti aziendali protette, connessioni domestiche con NAT o reti pubbliche con restrizioni. Il meccanismo di bypass delle limitazioni NAT/firewall rappresenta un vantaggio decisivo: anziché richiedere modifiche alla configurazione di rete (spesso impossibili in contesti aziendali), Ngrok stabilisce proattivamente una connessione in uscita verso i propri server cloud, creando un canale bidirezionale attraverso il quale possono transitare le richieste in entrata.

- Sicurezza

Dal punto di vista della sicurezza, il vantaggio di Ngrok è notevole. Il sistema implementa automaticamente protocolli di crittografia moderni senza richiedere interventi manuali, adottando un approccio "security by default" che riduce significativamente la superficie d'attacco. Al contrario, l'implementazione sicura di RMI richiede la corretta configurazione di elementi complessi come keystore, truststore e politiche di sicurezza Java, con un elevato rischio di errori implementativi che potrebbero compromettere l'intera architettura. L'approccio zero-configuration di Ngrok permette inoltre di ridurre drasticamente i tempi di setup e deployment, un fattore cruciale considerando la varietà di competenze tecniche degli utenti finali del sistema.

- Comportamento con IP dinamici

La resilienza operativa di Ngrok costituisce un ulteriore elemento distintivo. In ambienti caratterizzati da IP dinamici o connettività intermittente, RMI mostra significativi limiti, richiedendo riconfigurazioni ogni volta che l'indirizzo pubblico del server cambia. Ngrok, invece, mantiene il tunnel attivo indipendentemente dalle fluttuazioni dell'indirizzo IP, grazie alla connessione persistente verso l'infrastruttura cloud. Questa caratteristica garantisce una continuità di servizio superiore in scenari d'uso reali, dove le disconnessioni e le riconnessioni sono eventi frequenti.

In sintesi, sebbene entrambe le tecnologie presentino vantaggi e limitazioni, l'adozione di Ngrok nel Book Recommender System rappresenta una scelta pragmatica orientata a massimizzare l'accessibilità, la sicurezza e l'affidabilità del servizio. Privilegiando la semplicità d'uso e la robustezza in scenari di rete complessi rispetto alla purezza architettonica di soluzioni più tradizionali come RMI, il sistema offre un'esperienza d'uso fluida e immediata anche per utenti con limitate competenze tecniche o operanti in ambienti di rete restrittivi.

- Limitazioni di Entrambe le Soluzioni

Entrambe le tecnologie presentano limitazioni:

La **dipendenza da un servizio esterno** rappresenta un punto debole di Ngrok, poiché richiede che l'infrastruttura Ngrok sia funzionante e accessibile. D'altro canto, RMI è completamente auto contenuto ma richiede configurazioni di rete specifiche che spesso non sono disponibili o possibili in contesti reali.

La **latenza aggiuntiva** di Ngrok è generalmente compensata dalla sua maggiore affidabilità in scenari reali. Mentre RMI potrebbe offrire prestazioni migliori in ambienti di rete ideali, la sua sensibilità alle configurazioni di rete spesso porta a disconnessioni o inaccessibilità che vanificano il vantaggio teorico.

Il **rate limiting** e gli **endpoint effimeri** della versione gratuita di Ngrok rappresentano limitazioni reali, ma sono problematiche superabili con licenze a pagamento che offrono connessioni simultanee illimitate e domini personalizzati persistenti. RMI non presenta questi limiti specifici, ma le sue

complessità di configurazione e le limitazioni di attraversamento NAT rappresentano ostacoli spesso insormontabili in deployment reali.

In definitiva, la scelta di Ngrok invece di RMI nel Book Recommender riflette una prioritizzazione della facilità di deployment, dell'accessibilità universale e della resilienza in condizioni di rete reali, rispetto alla purezza architetturale o all'autocontenimento teorico di RMI, che nella pratica quotidiana spesso risultano vantaggi illusori a fronte di significative complicazioni implementative.

▪ Interfaccia Utente con JavaFX e Scene Builder

• Architettura dell'Interfaccia

L'interfaccia utente del sistema Book Recommender è stata sviluppata utilizzando JavaFX, il framework di UI moderna per applicazioni Java, e Scene Builder, un tool WYSIWYG (What You See Is What You Get) per la creazione di interfacce JavaFX. Questa combinazione ha permesso di implementare un'interfaccia grafica sofisticata seguendo il paradigma MVC (Model-View-Controller).

• Scene Builder: Sviluppo FXML Dichiarativo

Nello sviluppo dell'interfaccia utente del Book Recommender, Scene Builder ha rappresentato uno strumento fondamentale che ha trasformato radicalmente l'approccio alla creazione delle componenti visuali. Questo ambiente di sviluppo WYSIWYG ha offerto un insieme di vantaggi tecnologici che hanno semplificato notevolmente il processo di implementazione, migliorando al contempo la qualità e la coerenza dell'esperienza utente.

Uno dei principali benefici è stata la netta separazione tra l'interfaccia e la logica applicativa, realizzata attraverso l'utilizzo dei file FXML. Questo formato dichiarativo ha permesso di definire la struttura visuale in modo completamente indipendente dal codice Java che ne gestisce il comportamento. Tale separazione ha notevolmente migliorato la manutenibilità del sistema,

consentendo modifiche all'aspetto visivo senza intervenire sulla logica sottostante, e viceversa. Inoltre, ha facilitato la collaborazione tra sviluppatori, concentrati principalmente sul funzionamento, e designer UI, focalizzati sull'aspetto e l'usabilità.

L'approccio visuale alla progettazione ha rappresentato un'innovazione significativa nel workflow di sviluppo. Invece di codificare manualmente ogni elemento dell'interfaccia, è stato possibile comporre le schermate attraverso semplici operazioni di drag-and-drop, posizionando visivamente i componenti e definendone immediatamente proprietà e relazioni. Questa metodologia ha accelerato drasticamente i tempi di sviluppo, consentendo di sperimentare diverse soluzioni di layout con facilità e riducendo gli errori di implementazione grazie al feedback visivo immediato.

L'integrazione con CSS ha portato un ulteriore livello di sofisticazione all'interfaccia, permettendo di applicare stili personalizzati ai componenti attraverso fogli di stile esterni. Questa caratteristica ha garantito coerenza visiva in tutta l'applicazione, facilitando l'implementazione di un design moderno e professionale. La separazione tra struttura e presentazione ha inoltre permesso di modificare l'aspetto dell'intera applicazione intervenendo solo sui file CSS, senza alterare la struttura funzionale dell'interfaccia.

Il processo di sviluppo dell'interfaccia ha seguito un workflow ben strutturato che ha massimizzato i vantaggi offerti da Scene Builder. Inizialmente, la struttura di ciascuna schermata è stata creata visivamente attraverso l'ambiente grafico, posizionando e configurando i vari componenti UI. Questi layout sono stati poi esportati in file FXML, che rappresentano la definizione dichiarativa dell'interfaccia. Successivamente, attraverso un meccanismo di injection basato su annotazioni, i file FXML sono stati collegati ai rispettivi controller Java, creando il ponte tra definizione visuale e comportamento. Infine, l'implementazione della logica di business nei controller ha completato il ciclo, aggiungendo funzionalità alle strutture visive precedentemente definite.

- Injection dei Componenti

L'architettura dell'interfaccia utente del sistema Book Recommender sfrutta efficacemente il meccanismo di dependency injection offerto da JavaFX attraverso l'uso delle annotazioni @FXML. Questo approccio avanzato consente di creare un collegamento diretto tra gli elementi definiti nell'interfaccia FXML e le corrispondenti variabili nei controller Java, semplificando notevolmente lo sviluppo e la manutenzione dell'applicazione.

L'implementazione di questo sistema di injection garantisce una separazione chiara e netta tra la vista, definita nei file FXML, e la logica di controllo, implementata nelle classi Java. Questa distinzione architettonica promuove una maggiore modularità e facilita la collaborazione tra sviluppatori con competenze diverse, permettendo ai designer di concentrarsi sugli aspetti visivi mentre i programmatori si occupano della logica applicativa.

Un vantaggio significativo di questo approccio è l'auto-wire automatico dei componenti dell'interfaccia utente alle variabili Java, che elimina la necessità di scrivere codice ripetitivo per recuperare gli elementi dell'interfaccia. Il sistema si occupa automaticamente di istanziare e collegare i componenti UI alle variabili annotate con @FXML, riducendo drasticamente la possibilità di errori e semplificando il codice del controller.

La gestione dichiarativa degli event handler rappresenta un ulteriore beneficio di questa architettura. Gli sviluppatori possono definire i metodi di gestione degli eventi direttamente nel controller e collegarli agli elementi dell'interfaccia tramite Scene Builder, creando un flusso di lavoro intuitivo e visuale che migliora significativamente la produttività durante lo sviluppo.

Questa struttura architetturale facilita inoltre notevolmente le attività di testing, grazie all'isolamento della logica di business dai componenti dell'interfaccia. I controller possono essere testati indipendentemente dalla vista, permettendo l'implementazione di test automatizzati più efficaci e la verifica del comportamento dell'applicazione in condizioni controllate, contribuendo così alla robustezza complessiva del sistema.

- Gestione Avanzata dei Layout

Il Book Recommender utilizza strategicamente diversi layout manager di JavaFX per creare un'interfaccia flessibile e adattabile. Invece di posizionamenti assoluti, l'applicazione combina vari contenitori specializzati per organizzare gli elementi in modo responsivo.

- VBox e HBox dispongono i componenti in sequenze verticali e orizzontali, ideali per menu e pannelli di controllo.
- GridPane organizza gli elementi in griglie configurabili, perfette per form e tabelle con allineamento bidimensionale. Per la struttura principale delle schermate.
- BorderPane divide lo spazio in cinque aree logiche (superiore, inferiore, sinistra, destra e centrale), facilitando l'organizzazione gerarchica dei contenuti. Quando è necessario sovrapporre elementi come dialoghi o indicatori.
- StackPane gestisce il posizionamento lungo l'asse di profondità.

Questi layout manager, utilizzati in strutture nidificate, permettono di costruire interfacce complesse ma coerenti che si adattano dinamicamente alle dimensioni della finestra, garantendo un'esperienza ottimale su diverse configurazioni hardware.

- Responsive Design e Scaling

Il sistema Book Recommender implementa strategie di scaling progettate per adattare l'interfaccia utente a diverse risoluzioni e dimensioni dello schermo. Queste tecniche di responsive design garantiscono che l'applicazione mantenga un'elevata usabilità indipendentemente dal dispositivo utilizzato, distribuendo proporzionalmente lo spazio disponibile tra i vari elementi dell'interfaccia. I componenti si adattano dinamicamente alle dimensioni della finestra, assicurando una visualizzazione ottimale dei contenuti su schermi di diverse dimensioni. Questo approccio flessibile permette di offrire un'esperienza utente coerente e di alta qualità su una vasta gamma di configurazioni hardware, dalle schermate compatte dei laptop ai monitor desktop ad alta risoluzione.

Troubleshoting e Gestione Errori

Problematiche Comuni e Soluzioni

Il Book Recommender implementa meccanismi di risoluzione dei problemi più comuni che gli utenti potrebbero incontrare durante l'utilizzo dell'applicazione. Per quanto riguarda i problemi di connessione al database, il sistema va oltre il semplice rilevamento e avvio automatico di PostgreSQL, gestendo anche la creazione automatica di database e utenti qualora non esistano. Questo approccio proattivo riduce drasticamente la necessità di intervento manuale durante la configurazione iniziale, rendendo l'esperienza di primo utilizzo notevolmente più fluida anche per utenti con limitate competenze tecniche.

La gestione delle problematiche di connessione client-server rappresenta un altro aspetto in cui l'applicazione eccelle. Il sistema è in grado di rilevare proattivamente l'indisponibilità del server, presentando all'utente messaggi di errore chiari e comprensibili. In questi casi, vengono visualizzati dialog interattivi che offrono opzioni intuitive, come ritentare la connessione o uscire dall'applicazione. Particolarmente sofisticato è il sistema di monitoraggio continuo dello stato della connessione, che opera in background e può rilevare interruzioni in tempo reale, permettendo interventi immediati. Per evitare che l'applicazione rimanga bloccata indefinitamente in caso di problemi persistenti, sono stati implementati timeout configurabili per i tentativi di connessione, insieme a una strategia di backoff esponenziale per i retry automatici, che riduce intelligentemente il carico sul server in situazioni problematiche.

Durante le operazioni di download dei dati iniziali, fase particolarmente critica per la corretta inizializzazione del sistema, l'applicazione implementa diverse misure di robustezza. I timeout nelle connessioni HTTP sono gestiti accuratamente per prevenire blocchi indefiniti, mentre l'integrità dei file scaricati viene verificata prima del loro utilizzo effettivo. In caso di errori durante il download, meccanismi automatici si occupano della pulizia dei file temporanei parziali, evitando accumuli di dati inconsistenti. Il sistema supporta, quando tecnicamente possibile, la ripresa dei download interrotti, ottimizzando così l'utilizzo della banda disponibile. Un ulteriore livello di sicurezza è fornito dal calcolo e dalla verifica dei checksum dei file scaricati, garantendo non solo l'integrità ma anche l'autenticità dei dati.

Il sistema affronta in modo particolarmente robusto le problematiche legate al database PostgreSQL. L'applicazione è in grado di rilevare automaticamente l'installazione esistente, con supporto esteso

per percorsi standard e configurazioni personalizzate. In caso di difficoltà nell'avvio del database, vengono effettuati tentativi multipli utilizzando vari comandi specifici per la piattaforma rilevata. Se tutti i tentativi automatici falliscono, l'applicazione non abbandona l'utente ma presenta dialog informativi con istruzioni manuali dettagliate, guidando passo dopo passo attraverso le possibili soluzioni. Il sistema verifica proattivamente la disponibilità della porta standard 5432 e, se occupata, implementa fallback su porte alternative. Questo approccio completo include anche il supporto per installazioni multi-versione, adattandosi intelligentemente all'ambiente specifico dell'utente.

- Meccanismi di Recovery

Per garantire la massima affidabilità, il Book Recommender implementa diversi meccanismi di recovery che intervengono in caso di problemi durante l'esecuzione. Un elemento fondamentale è l'uso estensivo di transazioni database per garantire l'atomicità delle operazioni complesse. Questo approccio assicura che le operazioni vengano eseguite completamente o per nulla, senza stati intermedi che potrebbero compromettere l'integrità dei dati. In caso di errori durante l'esecuzione, il sistema effettua automaticamente il rollback, preservando la consistenza del database anche in situazioni impreviste. Le transazioni garantiscono inoltre un efficace isolamento tra operazioni concorrenti, prevenendo interferenze potenzialmente dannose in ambienti multi-utente.

La gestione dei file temporanei rappresenta un'altra area in cui il sistema implementa meccanismi di recovery sofisticati. Attraverso procedure sistematiche di cleanup, l'applicazione libera le risorse del filesystem al termine delle operazioni, prevenendo l'accumulo di file temporanei che potrebbero saturare lo spazio disco disponibile. Particolarmente intelligente è la gestione delle situazioni in cui i file sono ancora in uso da processi attivi, evitando errori di accesso concorrente. Come meccanismo di fallback, il sistema implementa anche una pulizia posticipata tramite la funzione deleteOnExit(), garantendo che i file temporanei vengano rimossi anche in caso di chiusura anomala dell'applicazione.

Un thread dedicato monitora costantemente lo stato della connessione client-server, offrendo capacità di rilevamento tempestivo delle disconnessioni impreviste. Quando viene rilevata un'interruzione, il sistema visualizza immediatamente una schermata di disconnessione con opzioni chiare per l'utente, migliorando significativamente l'esperienza in caso di problemi di rete. Questo meccanismo previene inoltre operazioni su connessioni già interrotte, che potrebbero causare comportamenti imprevedibili o perdita di dati.

In situazioni particolarmente critiche, il sistema può eseguire un reset completo del proprio stato, implementando una procedura di recovery radicale ma efficace. Questo processo include la chiusura ordinata e sistematica di tutte le risorse allocate, la terminazione controllata di tutti i thread attivi e una completa reinizializzazione dello stato dell'applicazione. Tale approccio previene efficacemente memory leak e resource leak che, accumulandosi nel tempo, potrebbero degradare le prestazioni o causare instabilità.

Elaborazione Dati e Algoritmi

Parsing CSV e Algoritmi di Ricerca

Il Book Recommender implementa un parser CSV, capace di gestire efficacemente vari formati e casi particolari che spesso causano problemi in altri sistemi. Il parser supporta flessibilmente CSV con delimitatori diversi, gestendo correttamente sia file separati da virgole che da tabulazioni. Particolarmente accurata è l'elaborazione dei campi racchiusi tra virgolette, soprattutto quando contengono delimitatori interni che potrebbero confondere parser più semplici. Il sistema gestisce appropriatamente anche l'escape delle virgolette nei campi di testo e implementa un trattamento specifico per i valori NULL espliciti.

Gli algoritmi di ricerca implementati nel sistema sono stati ottimizzati per diversi scenari d'uso, offrendo sia flessibilità che performance elevate. La ricerca per similitudine utilizza operatori SQL LIKE con pattern matching case-insensitive, permettendo una ricerca flessibile di sottostringhe all'interno dei titoli e garantendo un'esperienza utente più intuitiva grazie all'insensibilità alle maiuscole/minuscole. Le prestazioni di queste ricerche sono ottimizzate grazie all'utilizzo dell'indice idx_books_title, che accelera significativamente l'esecuzione delle query.

Per scenari più complessi, il sistema implementa una ricerca multi-campo che combina condizioni su diversi attributi, migliorando notevolmente la precisione dei risultati. Questo approccio consente l'applicazione di filtri combinati su attributi multipli come autore, anno e categoria, producendo risultati notevolmente più precisi rispetto alle ricerche su campo singolo. L'efficienza rimane elevata grazie all'utilizzo strategico di indici multipli, ottimizzati per queste operazioni composite.

L'algoritmo di ricerca con ordinamento implementa criteri basati sulla rilevanza, permettendo un ordinamento intelligente dei risultati in base a metriche di qualità come il rating medio. Per ottimizzare le prestazioni, soprattutto con dataset voluminosi, il sistema consente una limitazione parametrizzabile dei risultati. In determinate condizioni, viene applicato anche un filtraggio automatico dei libri privi di valutazioni, migliorando ulteriormente la pertinenza dei risultati presentati all'utente.

▪ Sistema di Notifiche

Il Book Recommender sfrutta efficacemente il pattern Observer attraverso i listener JavaFX per monitorare in tempo reale le modifiche agli elementi dell'interfaccia utente. Questo meccanismo garantisce una reazione immediata ai cambiamenti nell'interfaccia, migliorando notevolmente la reattività percepita dall'utente. Un caso d'uso particolarmente importante è la validazione in tempo reale dell'input, che fornisce feedback istantaneo durante la digitazione. L'implementazione di questo pattern realizza un efficace disaccoppiamento tra i componenti UI e la logica applicativa sottostante, permettendo sviluppi paralleli e facilitando la manutenzione. Questo approccio consente anche di implementare comportamenti reattivi complessi con un codice minimo e altamente manutenibile.

Il sistema di notifiche all'utente è stato progettato con particolare attenzione all'esperienza d'uso, implementando tre principali tipologie di feedback visivo, ciascuna ottimizzata per specifici scenari comunicativi:

Alert Dialog: Utilizzati per notifiche importanti che richiedono attenzione immediata dell'utente. Questi dialog provocano un'interruzione temporanea del flusso applicativo, garantendo la massima visibilità del messaggio. Il problema riscontrato e le possibili soluzioni vengono presentati in modo chiaro e comprensibile. Gli Alert sono completamente personalizzabili in termini di titolo, contenuto e opzioni disponibili, e la loro gestione centralizzata assicura un aspetto e un comportamento standardizzati per tutte le notifiche critiche. Un esempio puo essere l' allert per la connessione al server o login multipli

1. **Status Bar**: Fornisce informazioni non invasive sullo stato corrente dell'applicazione. Questo meccanismo implementa aggiornamenti continui che non interferiscono con l'interazione normale dell'utente. Le notifiche possono essere configurate per scomparire automaticamente dopo un periodo definito, o rimanere visibili fino a un cambiamento di stato. La Status Bar offre un feedback visivo immediato sulla completezza o lo stato di avanzamento delle operazioni, migliorando la comprensione dei processi in corso. Un esempio è la barra per la connessione dal server.

2. **Toast Notifications**: Perfette per notifiche brevi e temporanee di natura non critica. Queste notifiche appaiono e scompaiono gradualmente con effetti di transizione fluidi, minimizzando l'interferenza con il flusso di lavoro dell'utente. Sono particolarmente efficaci per confermare visivamente il completamento positivo delle operazioni, con possibilità di personalizzare durata, posizione e stile visivo in base al contesto applicativo. Un esempio sono tutti gli allert di conferma operazione come il salvataggio di un libro ecc.

Ulteriori Implementazioni Possibili

Il Book Recommender è stato progettato con un'architettura modulare ed estensibile che si presta a numerose evoluzioni future. Tra le potenziali implementazioni che potrebbero arricchire ulteriormente il sistema, quattro direzioni di sviluppo appaiono particolarmente promettenti:

Integrazione di ChatGPT per Traduzione e Ricerca Multilingue

Una delle estensioni più promettenti per il Book Recommender sarebbe l'integrazione di tecnologie di intelligenza artificiale mediante API apposite basate su modelli linguistici avanzati come ChatGPT, che potrebbero arricchire significativamente le funzionalità proposte mediante le seguenti implementazioni:

- **Traduzione di titoli e descrizioni**: Ricerca dei libri in lingue diverse trovando il libro corrispondente nella propria lingua, permettendo agli utenti di esplorare cataloghi internazionali indipendentemente dalla loro conoscenza linguistica.
- **Traduzione di recensioni e valutazioni**: Possibilità di leggere e comprendere commenti e recensioni scritte in lingue diverse dalla propria, ampliando l'accesso al feedback della community globale.
- **Interfaccia multilingue**: Adattamento dinamico dell'interfaccia utente alla lingua preferita, migliorando l'accessibilità per utenti internazionali.

• Hashing delle Password

L'implementazione di un meccanismo di hashing per le password rappresenterebbe un significativo miglioramento della sicurezza rispetto all'attuale memorizzazione in chiaro. Questo potrebbe essere realizzato utilizzando algoritmi moderni come bcrypt o Argon2, di seguito un esempio di implementazione possibile.

java

```
// Esempio di implementazione con bcrypt
public String hashPassword(String plainPassword) {
    return BCrypt.hashpw(plainPassword, BCrypt.gensalt(12));
}

public boolean verifyPassword(String plainPassword, String hashedPassword) {
    return BCrypt.checkpw(plainPassword, hashedPassword);
}
```

Questa modifica richiederebbe l'aggiornamento della tabella utenti nel database per memorizzare gli hash invece delle password in chiaro, e la revisione delle procedure di autenticazione e registrazione per utilizzare le funzioni di verifica appropriate. Un piano di migrazione dovrebbe essere sviluppato per gestire il passaggio dalle password esistenti agli hash senza interrompere l'accesso degli utenti.

- Sincronizzazione Automatica con Google Drive

L'implementazione di un sistema di backup automatico su Google Drive al momento della chiusura del server garantirebbe maggior persistenza tra una sessione e l'altra e faciliterebbe la ripresa da dove interrotto nelle sessioni successive.

All'avvio del server, un meccanismo complementare controllerebbe l'esistenza di backup più recenti su Drive e offrirebbe l'opzione di ripristino. Questa funzionalità richiederebbe l'integrazione con l'API di Google Drive e la gestione appropriata delle credenziali OAuth per l'autenticazione.

- Autenticazione a Due Fattori (2FA)

L'implementazione dell'autenticazione a due fattori rappresenterebbe un significativo miglioramento della sicurezza degli accessi, aggiungendo un secondo livello di verifica oltre alle credenziali standard, di seguito un esempio di codice implementabile.

java

```
public class TwoFactorAuthManager {
    // Genera un codice di verifica temporaneo
    public String generateTOTP(String userSecret) {
        TimeBasedOneTimePasswordGenerator totp = new TimeBasedOneTimePasswordGenerator();
        return totp.generateOneTimePassword(userSecret, Instant.now());
    }

    // Invia il codice via email
    public void sendVerificationCode(String email, String code) {
        EmailService emailService = EmailService.getInstance();
        emailService.sendMail(email, "Codice di Verifica Book Recommender",
                "Il tuo codice di verifica è: " + code + ". Valido per 5 minuti.");
    }

    // Verifica il codice inserito dall'utente
```

```
    public boolean verifyCode(String userSecret, String inputCode) {
        String generatedCode = generateTOTP(userSecret);
        return generatedCode.equals(inputCode);
    }
}
```

Questa funzionalità richiederebbe l'aggiunta di campi per memorizzare il segreto unico di ciascun utente e le preferenze 2FA nel database, insieme a modifiche nell'interfaccia di login per gestire il flusso di autenticazione esteso.

- Sistema di Blocco Account dopo Tentativi Falliti

L'implementazione di un meccanismo di blocco temporaneo degli account dopo multipli tentativi di accesso falliti proteggerebbe contro attacchi di forza bruta, un'implementazione potrebbe essere la seguente:

java

```
public class LoginProtectionService {
    private Map<String, Integer> failedAttempts = new ConcurrentHashMap<>();
    private Map<String, Long> lockoutTime = new ConcurrentHashMap<>();

    private static final int MAX_ATTEMPTS = 5;
    private static final long LOCKOUT_DURATION_MS = 15 * 60 * 1000; // 15 minuti

    public boolean isAccountLocked(String userId) {
        Long lockTime = lockoutTime.get(userId);
        if (lockTime != null) {
            if (System.currentTimeMillis() - lockTime < LOCKOUT_DURATION_MS) {
                return true;
            } else {
                // Sblocco automatico dopo il periodo di lockout
```

```
            lockoutTime.remove(userId);
            failedAttempts.remove(userId);
        }
    }
    return false;
}

public void recordFailedAttempt(String userId) {
    int attempts = failedAttempts.getOrDefault(userId, 0) + 1;
    failedAttempts.put(userId, attempts);

    if (attempts >= MAX_ATTEMPTS) {
        lockoutTime.put(userId, System.currentTimeMillis());

        // Registrazione dell'evento nel sistema di audit
        AuditLogger.log(AuditEvent.ACCOUNT_LOCKED, userId,
                "Account bloccato dopo " + attempts + " tentativi falliti");
    }
}

public void resetFailedAttempts(String userId) {
    failedAttempts.remove(userId);
}

public long getRemainingLockoutTime(String userId) {
    Long lockTime = lockoutTime.get(userId);
    if (lockTime == null) return 0;

    long remainingTime = LOCKOUT_DURATION_MS - (System.currentTimeMillis() - lockTime);
    return Math.max(0, remainingTime);
}
```

}

Questo servizio verrebbe integrato nel processo di login, bloccando temporaneamente gli account dopo un numero predefinito di tentativi falliti e registrando tali eventi nel sistema di audit per ulteriori analisi di sicurezza.

▪ Conclusioni

Il Book Recommender rappresenta un sistema completo per la gestione di collezioni librarie personali, implementato con Java, JavaFX, Ngrok e PostgreSQL secondo un'architettura client-server che bilancia efficacemente funzionalità avanzate e facilità d'uso.

La sicurezza costituisce un pilastro fondamentale di questo progetto, con molteplici livelli di protezione implementati o implementabili sistematicamente: query preparate contro SQL injection, prevenzione di login multipli, gestione sicura delle sessioni e transazioni atomiche per l'integrità dei dati. Questi meccanismi non sono elementi accessori ma componenti intrinseci dell'applicazione, riflettendo un approccio "security-by-design".

Il sistema è progettato con una chiara visione evolutiva della sicurezza, con potenziali implementazioni future già mappate: hashing delle password con algoritmi moderni, autenticazione a due fattori, traduzione ecc.

L'interfaccia utente, sviluppata con approccio *"What You See Is What You Get"* (in italiano: "Quello che vedi è quello che ottieni") tramite Scene Builder, offre un'esperienza coerente e intuitiva, mentre la gestione della persistenza dati è ottimizzata attraverso indici strategici e transazioni.

L'integrazione con Ngrok garantisce connettività remota universale, superando limitazioni tradizionali di accesso.

In sintesi, il Book Recommender dimostra come sia possibile integrare funzionalità avanzate e sicurezza robusta in un'unica soluzione coerente. L'attenzione meticolosa alla protezione dei dati applicata, unita alla chiara roadmap di potenziamenti futuri, crea un sistema che risponde alle esigenze attuali e si prepara ad affrontare le sfide emergenti, esemplificando un approccio moderno allo sviluppo software dove sicurezza e funzionalità coesistono come obiettivi complementari.

Bibliografia e Sitografia

- PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*. https://www.postgresql.org/docs/
- Ngrok. (2024). *Ngrok Documentation*. https://ngrok.com/docs
- Apache Maven Project. (2024). *Maven Documentation*. https://maven.apache.org/guides/
- Oracle. (2024). *Java SE Technical Documentation*. https://docs.oracle.com/en/java/javase/
- Draw.io / diagrams.net. (2024). *Online Diagram Software & Flowchart Maker*. https://www.diagrams.net/.

Contatti

Gervasini Alessio, *Project Manager:* agervasini1@studenti.uninsubria.it