WORDAGEDDON

Applicazione per l'Allenamento della Memoria Verbale

Relazione Tecnica del Progetto

Gruppo 02

Anno Accademico 2024-2025

Corso di Programmazione Java Avanzata

DIEM

Indice

1 Introduzione

1.1 Scopo del Progetto

Wordageddon è un'applicazione desktop sviluppata in JavaFX che mira ad allenare la memoria verbale degli utenti attraverso un sistema di quiz basato sull'analisi testuale. L'applicazione presenta documenti testuali per un tempo limitato e successivamente genera domande a risposta multipla relative alla frequenza e distribuzione delle parole nei testi analizzati.

1.2 Obiettivi

Il progetto si pone i seguenti obiettivi principali:

- Sviluppare un sistema di gaming educativo per l'allenamento della memoria verbale
- Implementare algoritmi di analisi testuale efficienti utilizzando le Stream API di Java 8+
- Creare un'interfaccia utente intuitiva e responsiva con JavaFX
- Gestire un sistema multiutente con persistenza dei dati su database relazionale
- Fornire funzionalità amministrative per la gestione dei contenuti

1.3 Tecnologie Utilizzate

- **Linguaggio**: Java 8+ con ampio utilizzo di Stream API e programmazione funzionale
- **GUI Framework**: JavaFX per l'interfaccia utente
- **Database**: Sistema di gestione database relazionale (JDBC)
- **Architettura**: Pattern MVC (Model-View-Controller)
- **Concorrenza**: JavaFX Services per operazioni asincrone

2 Analisi dei Requisiti

2.1 Requisiti Funzionali

2.1.1 Gestione Sessioni di Gioco

- **RF01**: Il sistema deve permettere la configurazione di sessioni di gioco con parametri variabili (numero documenti, lunghezza testi, tempo di lettura) basati su livelli di difficoltà
- **RF02**: Il sistema deve mostrare documenti testuali per un tempo prestabilito
- **RF03**: Il sistema deve generare automaticamente domande a risposta multipla basate sull'analisi dei documenti

2.1.2 Tipologie di Domande

Il sistema deve supportare le seguenti tipologie di domande:

- **Frequenza assoluta**: "Quante volte compare la parola X nel documento Y?"
- **Confronto di frequenze**: "Quale tra le seguenti parole è la più frequente?"
- **Associazione documento-specifica**: "Quale parola compare più spesso nel documento Z?"
- **Esclusione**: "Quale di queste parole non appare mai in nessun documento?"

2.1.3 Gestione Utenti

- **RF04**: Sistema di registrazione e autenticazione utenti
- **RF05**: Associazione punteggi alle sessioni utente
- **RF06**: Persistenza dei dati utente su database relazionale

2.1.4 Funzionalità Amministrative

- **RF07**: Pannello amministratore per caricamento documenti di testo
- **RF08**: Gestione liste di stopwords personalizzabili
- **RF09**: Configurazione parametri di gioco

2.1.5 Funzionalità Avanzate

- **RF10**: Classifiche globali e personali
- **RF11**: Statistiche dettagliate post-gioco
- **RF12**: Supporto multilingua
- **RF13**: Salvataggio e ripristino sessioni interrotte

2.2 Requisiti Non Funzionali

- **RNF01**: **Performance**: Analisi documenti deve completare in tempo reale
- **RNF02**: **Usabilità**: Interfaccia intuitiva e responsive
- **RNF03**: **Scalabilità**: Supporto per documenti di grandi dimensioni
- **RNF04**: **Manutenibilità**: Codice ben strutturato con documentazione Javadoc
- **RNF05**: **Portabilità**: Compatibilità cross-platform JavaFX

3 Progettazione del Sistema

3.1 Architettura Generale

Il sistema adotta un'architettura a tre livelli (*Three-Tier Architecture*), che consente di organizzare il codice in modo ordinato e modulare:

- **Presentation Layer**: rappresenta l'interfaccia grafica dell'applicazione, sviluppata con JavaFX. Include tutte le schermate con cui interagiscono gli utenti e gli amministratori.
- **Business Logic Layer**: contiene la logica dell'applicazione, come la gestione del gioco, l'analisi dei testi e il controllo degli utenti.
- **Data Access Layer**: gestisce il salvataggio e il recupero delle informazioni da fonti persistenti (es. database o file).

3.2 Pattern Architetturale Utilizzato: Model-View-Controller (MVC)

L'architettura dell'applicazione segue il pattern **Model-View-Controller (MVC)**, che suddivide il sistema in tre componenti principali, facilitando la manutenzione e l'espansione del progetto.

- **Model**: comprende le classi che rappresentano i dati fondamentali del sistema, come `Utente`, `Sessione`, `Documento` e `Domanda`. Queste classi contengono lo stato e il comportamento associato ai dati, e costituiscono il cuore informativo dell'applicazione.
- **View**: è costituita dai file FXML e dalle risorse grafiche utilizzate per costruire l'interfaccia utente. Le viste sono organizzate in schermate separate e si occupano esclusivamente della presentazione dei dati, senza includere logica applicativa.
- **Controller**: i controller sono raggruppati in un `package` dedicato e gestiscono l'interazione tra vista e modello. Ogni controller riceve gli input dell'utente dalle varie schermate, aggiorna i dati nel modello e coordina i cambiamenti della vista. I controller lavorano in stretta connessione con i servizi della logica applicativa, ma restano focalizzati sul controllo del flusso dell'interfaccia.

L'organizzazione a pacchetti (ad esempio `model`, `view`, `controller`) riflette questa suddivisione, rendendo chiara la responsabilità di ogni componente. Questo approccio rende il sistema più modulare, facilitando modifiche, test e riutilizzo del codice.

3.3 Gestione dei File di Input

Il sistema consente l'inserimento dei documenti da analizzare esclusivamente in formato `.txt`. Le stopwords, necessarie per l'elaborazione del testo, possono essere fornite sia in formato `.txt` (con parole separate da , o ; oppure tabulazione o ancora spazio), sia in formato `.csv`.

3.4 Configurazione del Database

L'applicazione utilizza PostgreSQL come sistema di gestione. La connessione al database è gestita tramite l'interfaccia `ConnectionConfig`, nella quale sono definite variabili statiche per l'utente (`USER`), la password (`PASSWORD`) e l'URL di connessione (`URL`). La modifica di questi parametri consente di adattare facilmente l'applicazione a diversi ambienti di esecuzione.

4 Implementazione

4.1 Analisi Testuale

4.1.1 Algoritmo di Tokenizzazione

Il processo di analisi testuale implementa le seguenti fasi:

1. **Preprocessing**: Rimozione punteggiatura e normalizzazione case
2. **Tokenizzazione**: Suddivisione in token utilizzando regex
3. **Filtering**: Rimozione stopwords(fatta lato query SQL)
4. **Frequency Analysis**: Calcolo frequenze utilizzando Stream API

4.1.2 Implementazione con Stream API

```
1 public Map<String, Integer> setMappaturaQuiz(){
2         Stream<String> parole = Arrays.stream(contenuto.
             toLowerCase().split("[\\p{Punct}\\s]+")).filter(s->!s.
             isEmpty()); //Splitta il contenuto anche per
             punteggiatura
3         return parole.collect(Collectors.groupingBy(String::
             toString,
4                   Collectors.summingInt(p->1)));
5     }
```

Listing 1: Analisi Frequenza Parole

4.2 Generazione Domande

4.2.1 Tipologie di Domande

Nel processo di generazione automatica delle domande, il sistema seleziona casualmente una delle quattro tipologie disponibili nel database. A ciascuna tipologia corrisponde uno specifico set di risposte.

Per ogni domanda, viene selezionato casualmente un documento di riferimento, da cui si estrae anche una parola chiave. Anche le risposte vengono generate in modo casuale, coerentemente con la logica della tipologia specifica.

Nel caso della tipologia di domanda di *esclusione*, la risposta corretta è scelta attraverso la tabella dizionario , che contiene cioè tutte parole fuori dai contesti dei documenti mostrati.

4.2.2 Algoritmi di Generazione

- **Frequenza Assoluta**: Selezione casuale parola e documento con calcolo frequenza esatta
- **Confronto**: Selezione multipla di parole con ordinamento per frequenza
- **Esclusione**: Identificazione parole presenti/assenti nell'intero corpus
- **Documento-Specifica**: Associazione parole ai documenti di origine

4.3 Persistenza Dati

4.3.1 Database Schema

```
CREATE TABLE users (
    username text primary key,
    password text ,
    admin boolean
);

CREATE TABLE documento (
    id serial primary key,
    titolo text,
    contenuto text ,
    difficolta text,
    data_caricamento date
);


CREATE TABLE domanda(
    testo text primary key ,
);

CREATE TABLE mappa(
    id serial primary key ,
    valore text ,
    conteggio integer ,
    documento integer references documento on delete cascade
);



CREATE TABLE sessione (
    id serial primary key ,
    punteggio integer ,
```

```
    utente references users on delete cascade on update cascade
);


CREATE TABLE stopwords(
    id serial primary key ,
    parola varchar(40)
);


CREATE TABLE vocabolario(
    stringa text primary key
);
```

Listing 2: Schema Database

5 Interfaccia Utente

5.1 Schermate Principali

5.1.1 Login

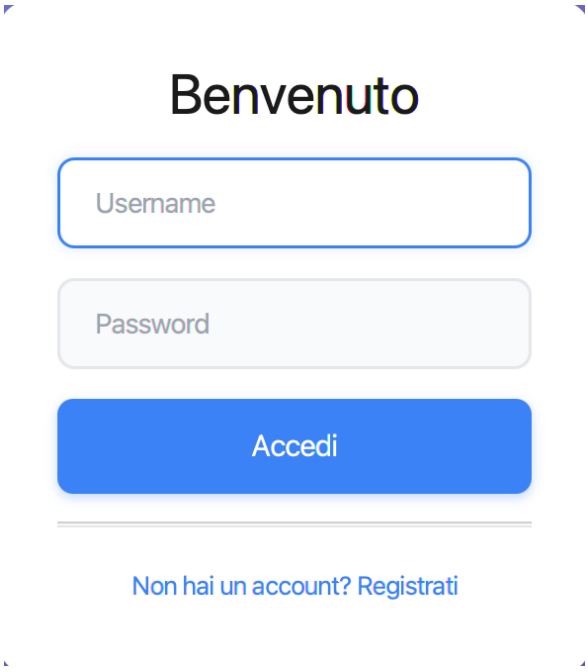

Figura 1: Interfaccia di login

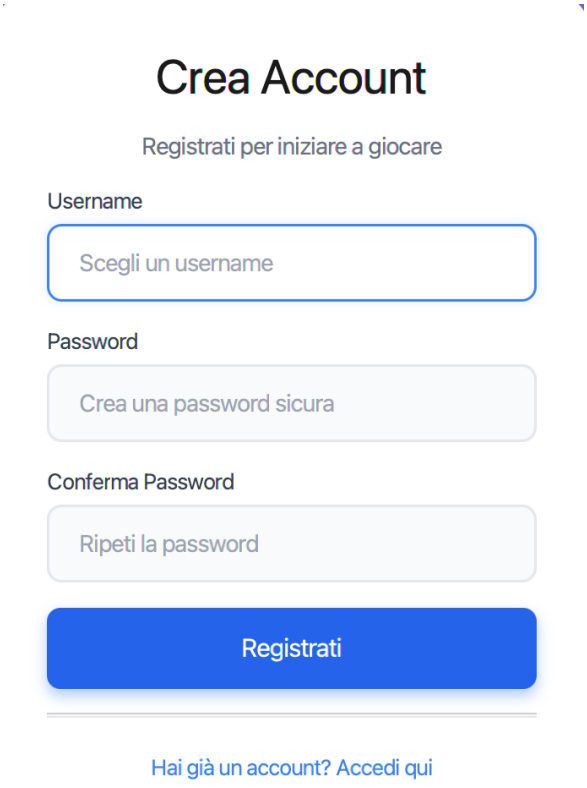

Figura 2: Interfaccia di registrazione

- Interfaccia di autenticazione utente con validazione lato client degli input.
- Gestione degli errori con visualizzazione di messaggi di feedback contestuali.
- Integrazione con sistema di autenticazione sicuro.

5.1.2 Dashboard Utente

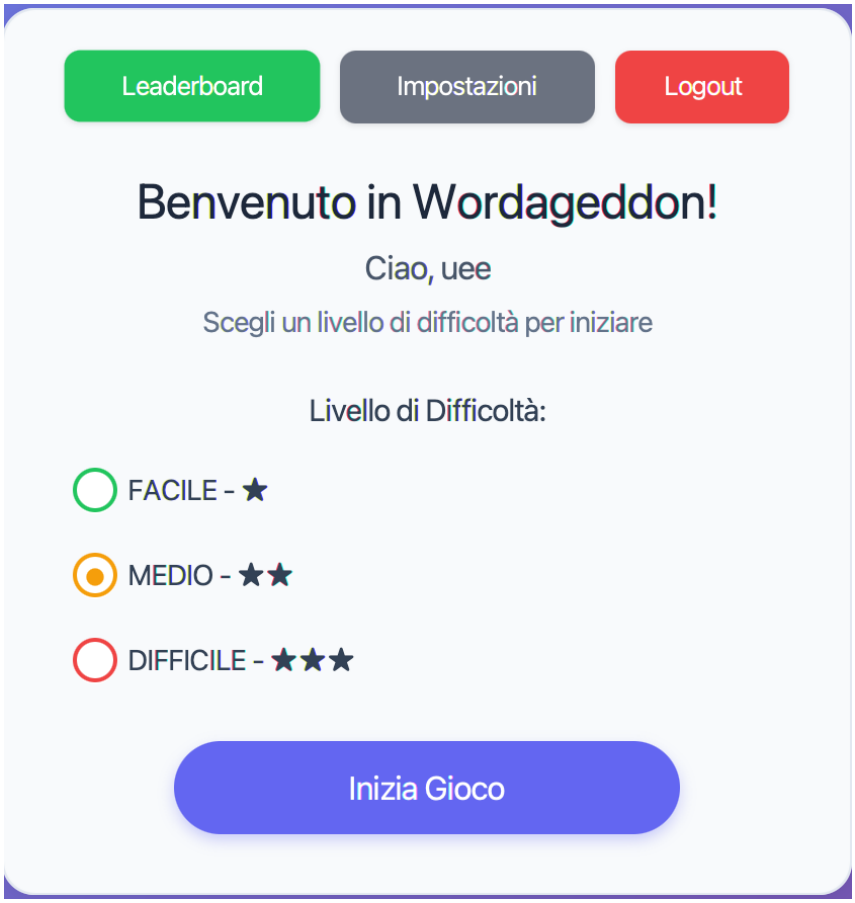

Figura 3: Schermata della dashboard utente

- Selezione del livello di difficoltà tramite interfaccia interattiva.
- Visualizzazione dinamica delle statistiche personali.
- Accesso diretto a classifiche globali e cronologia sessioni.
- Navigazione verso il pannello impostazioni utente.
- Pulsante per esecuzione sicura del logout.
- Avvio immediato di una nuova sessione di gioco.

5.1.3 Sessione di Gioco

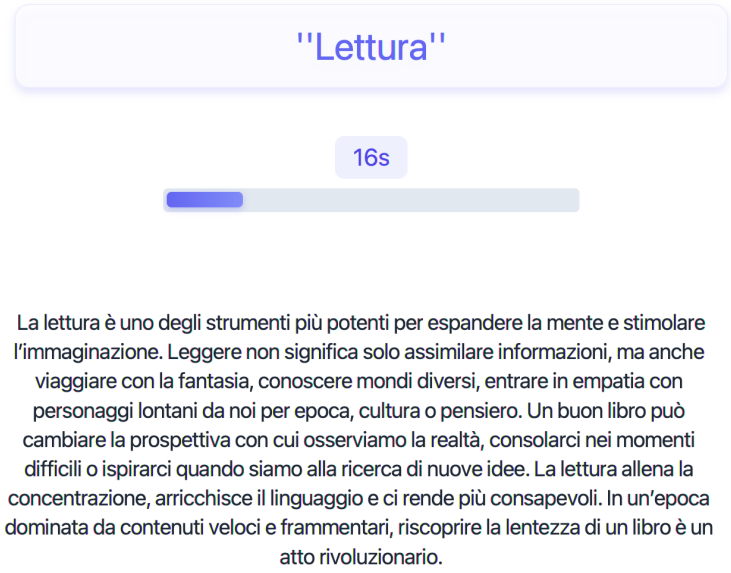

Figura 4: Schermata della sessione di gioco

- Timer countdown per la fase di lettura, sincronizzato con il server.
- Presentazione sequenziale e automatica dei documenti, con intestazione e metadati visibili.

5.1.4 Impostazioni

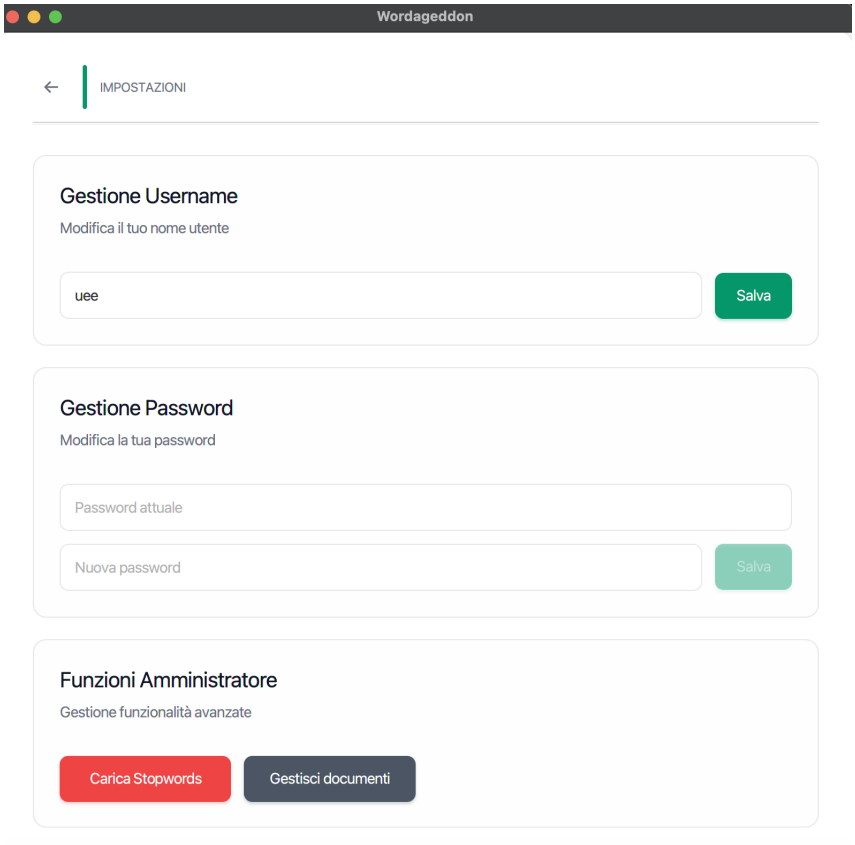

Figura 5: Schermata delle impostazioni

- Form di aggiornamento delle credenziali (username e password).
- Sezione riservata all'amministratore per caricamento e gestione di documenti e stopwords.
- Pulsante per tornare alla schermata principale.

5.1.5 Leaderboard Globale

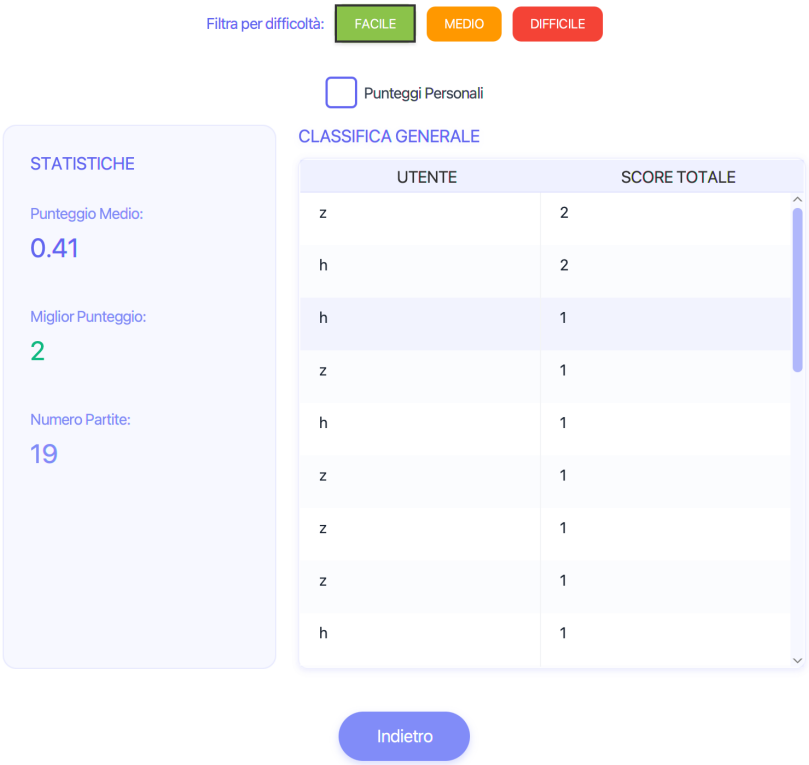

Figura 6: Schermata della classifica globale

- Filtro dei risultati per livello di difficoltà.
- Visualizzazione dei punteggi personali in evidenza.
- Statistiche individuali: punteggio medio, record personale, numero di partite.
- Classifica globale con identificativo utente e punteggio.
- Pulsante per tornare alla schermata principale.

5.1.6 Quiz di Valutazione

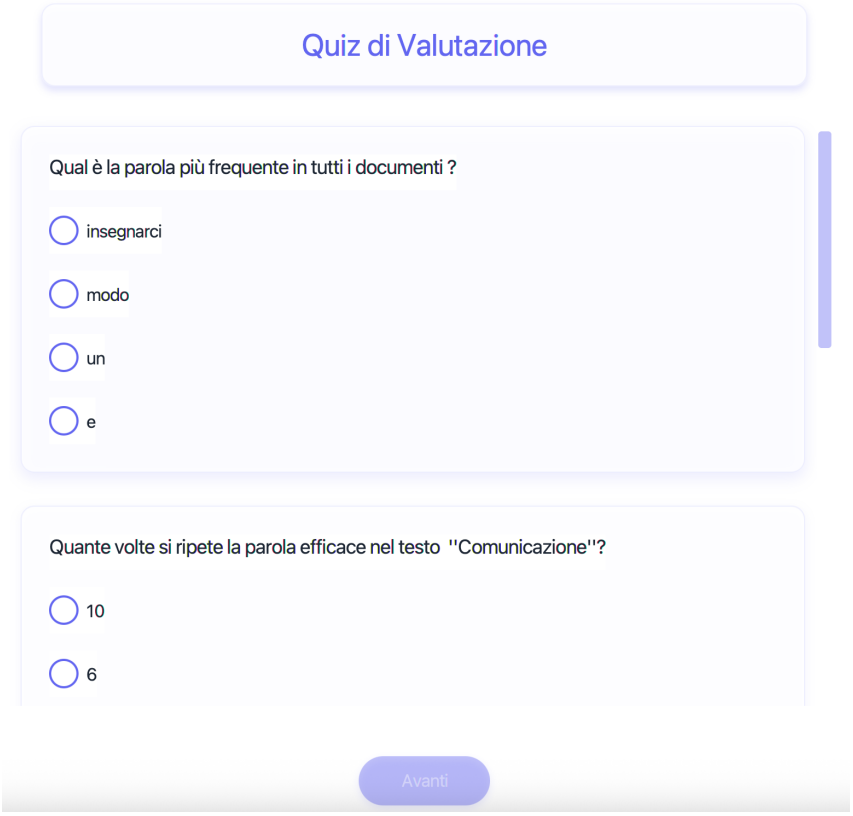

Figura 7: Schermata del quiz di valutazione

- Quattro domande a risposta multipla, ciascuna con quattro opzioni.
- Pulsante per visualizzare il punteggio finale, attivabile solo dopo la risposta a tutte le domande.

5.1.7 Riepilogo

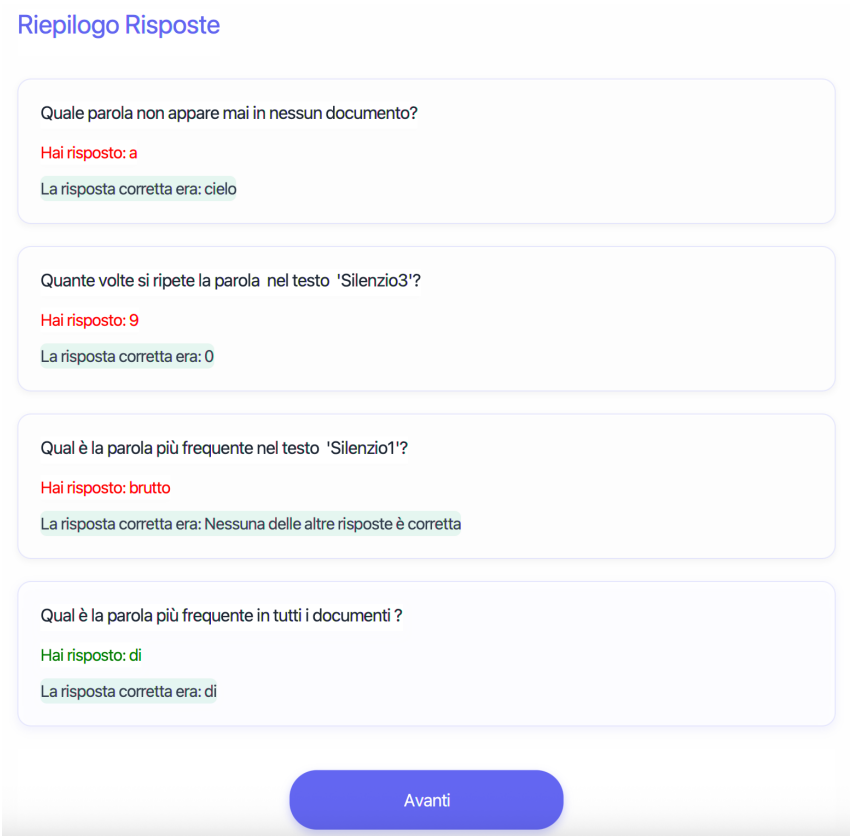

Figura 8: Schermata di riepilogo delle risposte

- Visualizzazione riepilogativa delle risposte fornite dall'utente.
- Evidenziazione delle risposte corrette rispetto a quelle selezionate.
- Pulsante per accedere al punteggio finale.

5.1.8 Sezione Documenti

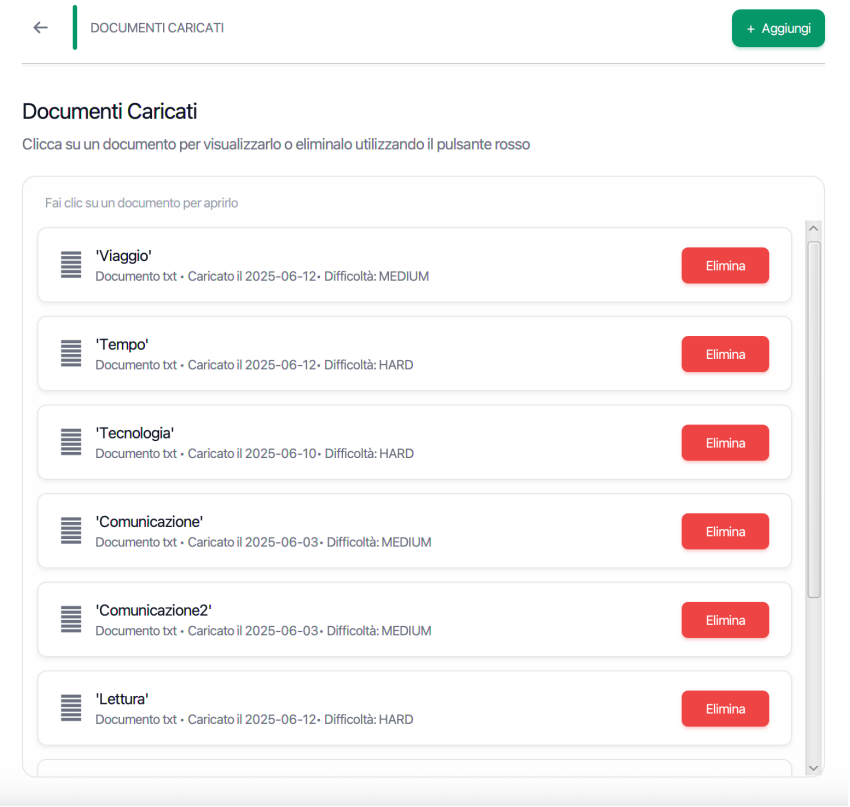

Figura 9: Schermata della gestione documenti

- Visualizzazione tabellare dei documenti caricati con metadati.
- Controlli per aggiunta e rimozione dei documenti (accessibili solo ad amministratori).
- Pulsante per tornare alla pagina delle impostazioni.

6 Risultati e Discussione

6.1 Funzionalità Implementate

Tutte le funzionalità specificate nei requisiti sono state implementate con successo:

- Sistema di gioco multilivello completamente funzionale
- Quattro tipologie di domande con generazione automatica
- Pannello amministrativo per gestione contenuti
- Funzionalità avanzate: classifiche, statistiche

6.2 Qualità del Codice

- Ampio utilizzo Stream API e programmazione funzionale
- Architettura modulare con basso accoppiamento
- Documentazione Javadoc completa
- Gestione eccezioni robusta e logging strutturato

A Appendice A: Diagrammi UML

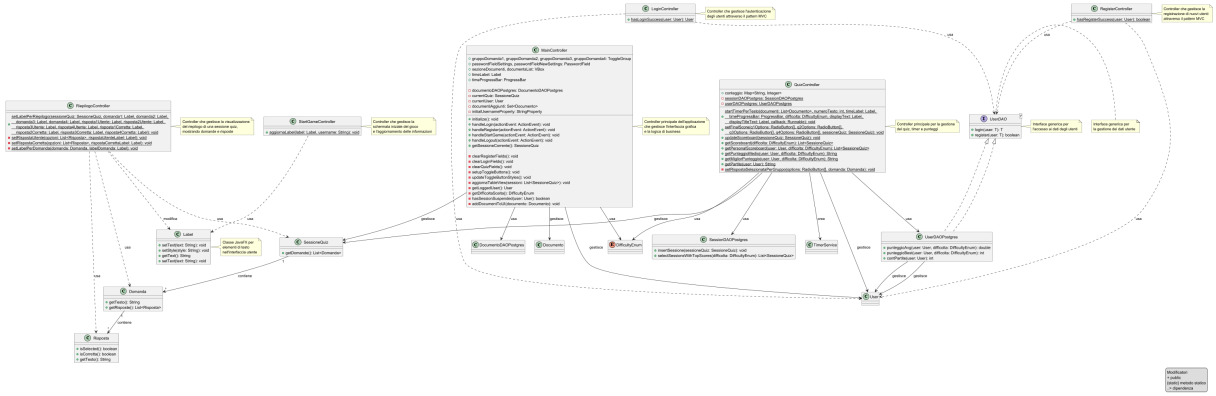

Figura 10: Controllers

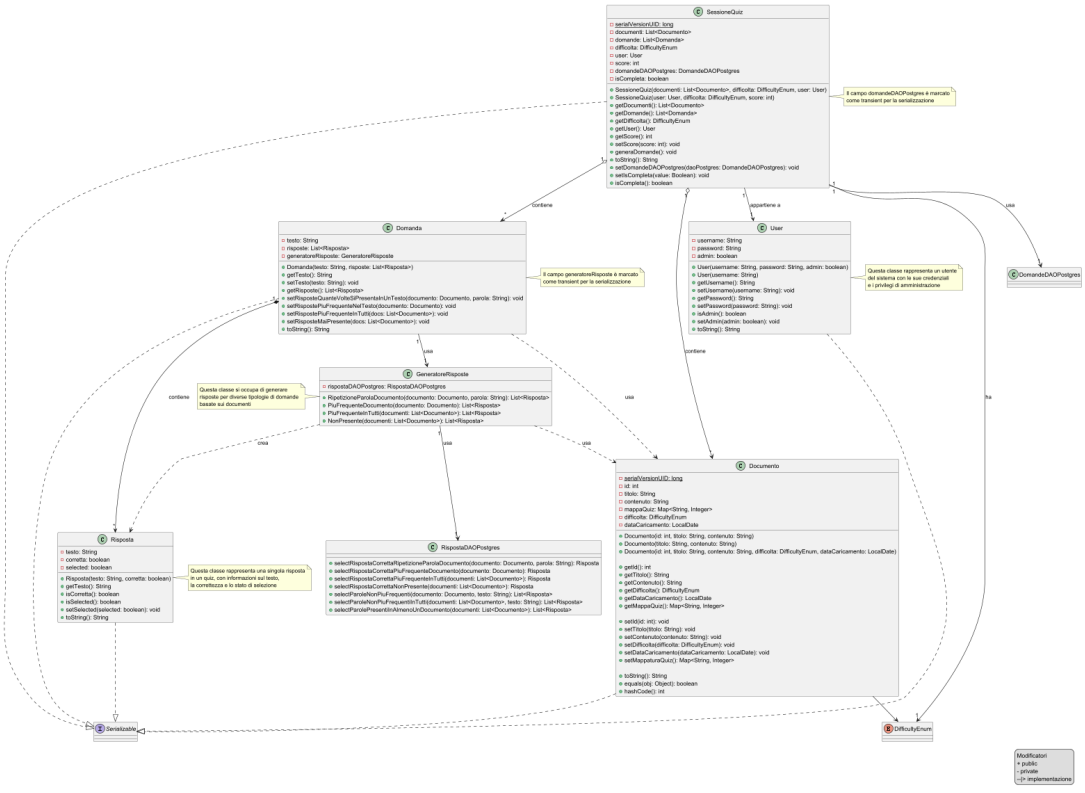

Figura 11: Models