UNIVERSITÀ DEGLI STUDI DI SALERNO

DIEM - Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea Triennale in Ingegneria Informatica

Progetto - Programmazione Java Avanzata
Wordageddon

Gruppo 16:
Emanuele Tocci - 0612707488
Francesco Lanzara - 0612707459
Rossella Pale - 0612707284
Claudia Montefusco - 0612707404

ANNO ACCADEMICO 2024/2025

Indice

1 Introduzione

Wordageddon è un'applicazione desktop ludico-educativa sviluppata in JavaFX, progettata per allenare la memoria dell'utente attraverso quiz basati sulle parole più frequenti presenti in documenti testuali.

Figura 1: Logo

1.1 Panoramica

L'applicazione **sfida l'utente nella memorizzazione** e riconoscimento delle parole più frequenti in uno o più documenti mostrati per un tempo limitato.
L'utente visualizza uno o più testi per un **tempo limitato**, in base al livello di **difficoltà** scelto, e al termine affronta una serie di **domande** a risposta multipla basate sull'analisi dei documenti. Le domande spaziano dalla frequenza di termini specifici al confronto tra parole, fino all'identificazione di parole mai apparse.
L'app supporta la gestione **multiutente**, con registrazione, login e salvataggio dei punteggi in un **database relazionale** `SQLite`. Un **pannello amministrativo** permette di caricare nuovi documenti e liste di **stopwords**, oltre a offrire funzioni opzionali come classifiche, statistiche dettagliate post-gioco e ripresa delle sessioni interrotte.

1.2 Descrizione del Gameplay

Ogni partita inizia con la selezione di una **difficoltà** e la presentazione di uno o più documenti testuali che l'utente deve leggere in un tempo limitato. Terminata la fase di lettura, viene avviata una serie di **domande** a risposta multipla che verificano la memoria e la capacità di analisi del giocatore. Le domande possono riguardare:

- La frequenza con cui una parola appare in uno o più documenti
- Quale fra alcune parole è la più o la meno frequente
- L'associazione di parole a uno specifico documento
- L'identificazione di una parola che non compare in nessun documento.

Il numero di documenti mostrati, la lunghezza dei testi, il numero di domande e il tempo disponibile sono parametri che variano in base al **livello di difficoltà** scelto. Abbiamo deciso di rimanere "più generosi" riguardo il tempo di lettura ma implementare un pulsante di skip di tale fase qualora l'utente si sentisse pronto di affrontare il quiz prima dello scadere del timer.

In particolare, a seconda del livello di difficoltá scelto:

Difficoltá	Punteggio MAX	MAX Documenti	MAX Domande
Facile	100	3	10
Media	200	5	15
Difficile	300	7	20

Il sistema classifica automaticamente i documenti caricati dall'utente in base al numero di parole in esso contenuto. Nelle varie diffi verranno quindi mostrati solamente documenti che rispettano la classificazione.
Dopo la fase di quiz, il punteggio ottenuto viene mostrato in una schermata di resoconto (3f) e **salvato**.

2 Progettazione

2.1 Analisi dei Requisiti

ⓘ **Legenda** **Categorie** requisiti:

- **IF:** Funzionalità individuali
- **DF:** Dati e formato dei dati
- **UI:** Interfaccia utente
- **NF:** Requisiti non funzionali
- **PC:** Vincoli di progetto

Livelli di **priorità** (Business value/Rischio tecnico):

- **H:** Must-have / alto
- **M:** Should-have / medio
- **L:** Nice-to-have / basso

Tipo	Nome	Business Value	Rischio Tecnico
IF-1	Fase di lettura	H	H
IF-2	Fase quiz	H	H
IF-3	Selezione della difficoltà	H	L
IF-4	Autenticazione utenti	H	M
IF-5	Stop words	H	L
IF-6	Privilegi di amministratore	H	L
IF-7	Leaderboard	M	M
IF-8	Statistiche post-partita	M	L
IF-9	Gestione sessioni interrotte	L	M
IF-10	Logging	L	M
DF-1	Persistenza dati utente	H	M
NF-1	Varietà delle domande	H	M
NF-2	Styling CSS	M	-
PC-1	Implementazione in Java+JavaFX	H	-
UI-1	Menu principale	H	L
UI-2	Pannello utente	M	M

2.1.1 Descrizione dei requisiti

2.1.1.1 IF-1: Fase di lettura Costituisce la prima fase di gioco vera e propria. Il sistema seleziona, in base alla difficoltà scelta e ai documenti a disposizione, le **opzioni** di gioco:

- quali documenti mostrare
- quanti documenti mostrare

- il valore del **timer**: una volta istanziato il **contesto** della partita, il timer si avvia e vengono mostrati a schermo i documenti, in un'interfaccia che agevola la navigazione tra le pagine e consente di terminare la lettura prima dello scadere del timer

2.1.1.2 IF-2: Fase di Quiz Entro il termine della fase di lettura, il sistema genera le domande da porre nel quiz che costituisce la fase successiva del gioco. Viene dunque mostrato un **elenco di domande a risposta multipla** (4 risposte possibili), con domande relative alla presenza e frequenza di parole all'interno dei documenti precedentemente visualizzati, un timer e un pulsante Fine. Allo scadere del timer o alla pressione del pulsante, la partita termina e vengono calcolati e visualizzati i risultati.

> ⓘ **Nota** La scelta di generare le domande **prima della fase del quiz** consente di **minimizzare l'overhead** nel caricamento del quiz stesso al termine della lettura, poiché durante quest'ultima al programma sono richieste poche risorse, e può quindi "portarsi avanti" eseguendo operazioni di preparazione alla fase successiva.

2.1.1.3 IF-4: Autenticazione Utenti Il programma supporta un sistema di gestione degli account, permettendo quindi di effettuare **sessioni di gioco** come utenti diversi. All'avvio, all'utente è richiesto di autenticarsi o registrarsi. Una volta eseguito il sign in/up, l'account viene identificato come **attivo**, ossia agli avvii successivi del programma l'utente sarà automaticamente loggato con quell'account, finché non esegue manualmente il logout (ad esempio per cambiare utente). In seguito alla fase di autenticazione (che può anche corrispondere alla confermata presenza di un account attivo implicita al sistema), l'utente viene indirizzato al **menu principale**.

2.1.1.4 IF-5: Stop words Il motore di gioco prevede la gestione di una **lista di stop words**: le parole inserite in questa lista saranno **ignorate** dall'algoritmo di generazione delle domande. È utile aggiungere a questa lista parole come articoli o preposizioni, che non sono di interesse al dominio descritto dai documenti, e di cui risulta inutilmente difficile memorizzarne la frequenza.

2.1.1.5 Selezione livello difficoltà All'avvio di una partita, all'utente è richiesto di scegliere fra **3 livelli di difficoltà**: facile, medio e difficile. La scelta della difficoltà impatterà sulla **lunghezza e il numero di documenti**, ed eventualmente sul **valore del timer**.

2.1.1.6 IF-7: Leaderboard Dal menu principale è possibile accedere ad una pagina del menu contenente la leaderboard del gioco. Questa contiene:

- **classifica globale**, relativa a tutte le partite giocate sul dispositivo da ogni utente
- classifica per **difficoltà**, che permette di filtrare i risultati in base alle partite giocate dagli utenti in una determinata difficoltà.

2.1.1.7 IF-8: Statistiche post-partita Al termine di una partita, prima di ritornare al menu principale, l'utente può consultare **informazioni dettagliate** sulla partita terminata, sia relative alle risposte del quiz (con un **confronto** tra le risposte scelte e quelle corrette), sia statistiche sulle parole presenti nei documenti.

2.1.1.8 IF-9: Gestione sessioni interrotte Quando l'utente chiude il programma durante il corso di una sessione di gioco, quest'ultimo **registra il contesto** della partita e, all'avvio seguente, propone all'utente tramite un popup di riprendere la sessione interrotta. In caso di risposta affermativa, verrà **caricato il contesto** salvato dell'ultima partita, altrimenti si verrà reindirizzati al menu principale.

2.1.1.9 IF-10: Logging Come strumento di diagnostica al fine di aumentare la manutenibilità del software, è possibile, dalle impostazioni del programma, **abilitare la funzione di logging**, che prevede la memorizzazione di eccezioni ed errori verificatisi a runtime in un file testuale consultabile dalla cartella contenente i file di gioco.

2.1.1.10 DF-1: Persistenza dati utente Il sistema mantiene un **database locale** su file (`SQLite`) per garantire la persistenza delle informazioni relative agli utenti registrati, sia quelle di autenticazione, sia **statistiche** relative alle partite giocate in precedenza.

2.1.1.11 NF-1: Varietà delle domande Il sistema propone automaticamente una serie di domande di **varie categorie**, al fine di aggiungere allo stesso un grado di **imprevedibilità**. Le tipologie di domande possibili sono:

- **Frequenza assoluta** (quante volte appare una parola)
- **Confronto** tra frequenze di più parole
- Associazione di parole a documenti specifici (solo se ne è stato presentato più d'uno)
- Individuazione di parole mai comparse

2.1.1.12 UI-1: Menu Principale In seguito alla (eventuale) schermata di autenticazione, viene visualizzato a schermo il **menu principale**, da cui è possibile:

- iniziare una nuova partita
- visualizzare la pagina della leaderboard
- visualizzare il pannello utente (admin)
- uscire dall'applicazione

2.1.1.13 UI-2: Pannello Utente Dal menu principale è possibile accedere al pannello utente, che permette di:

- visualizzare le informazioni di **autenticazione** dell'utente
- consultare statistiche sulle partite precedentemente giocate (punteggio medio, miglior punteggio, ultime partite)
- modificare la password di accesso
- effettuare il **logout**
- **cancellare** l'account

2.2 Casi d'uso

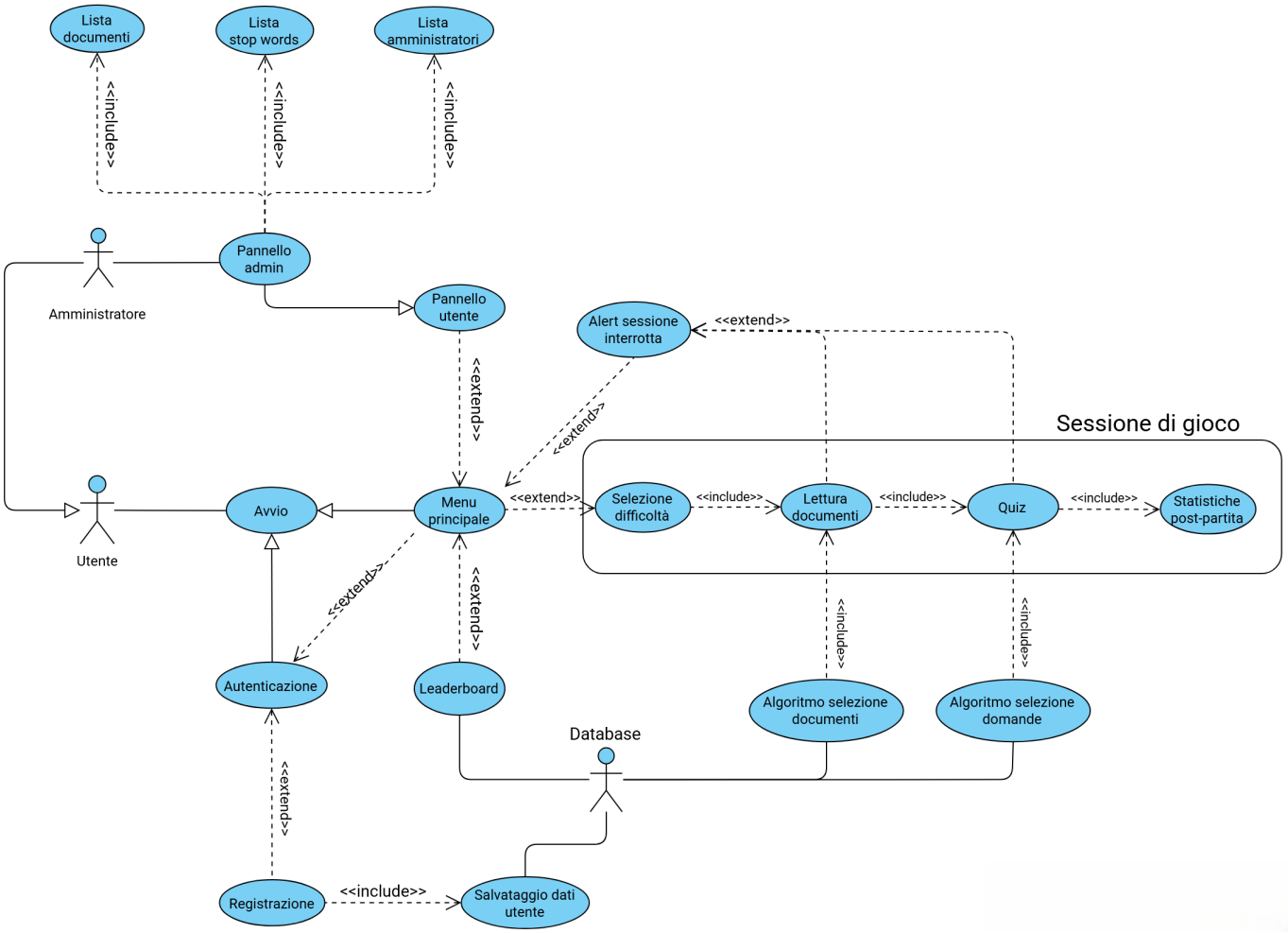


Figura 2: Diagramma dei casi d'uso

2.2.1 Descrizione dei casi d'uso

Vengono descritti i **principali** casi d'uso del sistema:

- Login
- Registrazione
- Avvio Gioco
- Visualizzazione Classifica
- Upload Documenti
- Upload Stopwords
- Retrocessione/Promozione utente

2.2.1.1 Login

- **Attore:** utente
- **Pre-condizioni:**
  - L'utente deve essere registrato
- **Post-condizioni**
  - L'utente accede alla homepage
- **Flusso standard**:

1. Il sistema mostra un form con i campi necessari al login (3a)
2. L'utente inserisce le credenziali e preme il pulsante "Login"
3. Il sistema verifica la validitá delle credenziali
4. L'utente viene reindirizzato alla schemata principale dell'app e viene salvata la sessione

- **Flusso alternativo**

1. I dati inseriti sono errati: il sistema mostra un warning e consente un nuovo tentativo

2.2.1.2 Registrazione

- **Attore:** utente
- **Post-condizioni**
  - L'utente risulta registrato: il sistema salva i dati dell'utente nel database
- **Flusso standard**:

1. Il sistema mostra un form per la registrazione (3a)
2. L'utente inserisce le credenziali e preme il pulsante "Signup"
3. Il sistema verifica la validitá delle credenziali
4. L'utente viene reindirizzato alla schermata principale dell'app

- **Flusso alternativo**:

1. L'utente inserisce solo username o password: il sistema mostra un messaggio di errore e consente un nuovo tentativo
2. L'utente é giá registrato: il sistema mostra un messaggio di errore

2.2.1.3 Avvio Gioco

- **Attore:** utente
- **Pre-condizioni:** L'utente ha effettuato il login e si trova nel menu
- **Flusso standard**:

1. L'utente avvia la modalitá di gioco cliccando il pulsante "Gioca" presente nel menu
2. Il sistema mostra la schermata per la scelta della difficoltá
3. L'utente seleziona la difficoltá
4. Il sistema inizializza i parametri di gioco in base alla difficoltá scelta e mostra i documenti da leggere
5. Il sistema analizza i documenti, genera le domande e le mostra all'utente
6. L'utente risponde (o salta) alle varie domande
7. Al termine del quiz il sistema mostra la schermata di riepilogo (3f) con il punteggio

2.2.1.4 Visualizzazione Classifica

- **Attore:** utente
- **Pre-condizioni:** L'utente ha effettuato il login
- **Flusso standard**:
  1. L'utente accede alla sezione "Leaderboard" (3e)tramite pulsante presente nella home principale;
  2. Il sistema recupera i dati di punteggio da tutti gli utenti;
  3. Il sistema genera 4 tipi di classifica: una per ogni livello di difficoltà e una globale;
  4. Il sistema ordina i punteggi in ordine decrescente;

2.2.1.5 Upload Documenti testuali

- **Attore:** Amministratore
- **Pre-condizioni:** L'admin ha effettuato l'accesso
- **Post-condizioni:** Il sistema salva il documento e la relativa analisi (WDM)
- **Flusso standard:**
  - L'admin apre il menu della gestione dei documenti dal pannello "Privilegi Admin" (3b);
  - L'amministratore carica un documento testuale in formato .txt
  - Il sistema effettua l'analisi del documento
  - Il sistema mostra una conferma dell'avvenuto caricamento: il documento é visibile tra quelli caricati
- **Flusso alternativo:** Il caricamento non é andato a buon fine: il sistema mostra un messaggio di errore

2.2.1.6 Upload di stopwords

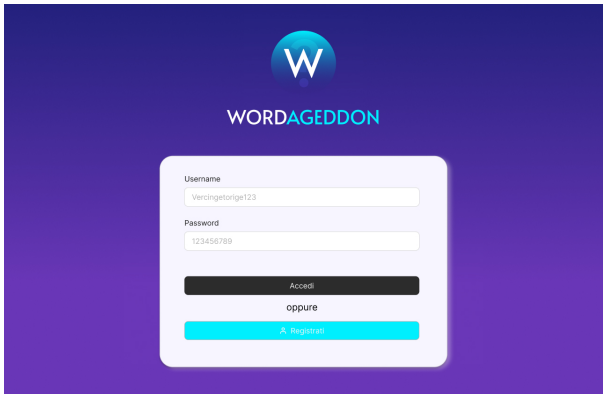
- **Attore:** Amministratore
- **Pre-condizioni:** L'admin ha effettuato l'accesso
- **Post-condizioni:** Il sistema salva le stopwords
- **Flusso standard:**
  - L'admin apre il menu della gestione delle stopwords dal pannello "Privilegi Admin" (3b);
  - L'amministratore carica un documento testuale in formato .txt o scrive le stopwords nel campo di input
  - Il sistema effettua il parsing del file o del `TextInput`
  - Il sistema mostra una conferma dell'avvenuto caricamento e aggiorna la lista delle stopwords caricate (visualizzabile dall'utente)
- **Flusso alternativo:** Il caricamento del file non é andato a buon fine: il sistema mostra un messaggio di errore

2.2.1.7 Promozione/Retrocessione admin

- **Attore:** Amministratore
- **Pre-condizioni:** L'admin ha effettuato l'accesso
- **Post-condizioni:** Il sistema salva il nuovo stato dell'utente selezionato
- **Flusso standard:**
  - L'admin apre il menu della gestione degli altri admin dal pannello "Privilegi Admin" (3b);
  - Il sistema mostra un popup con tutti gli user e la loro tipologia ("user" o "admin");
  - L'utente clicca sul toggleButton per cambiare lo stato;
  - Il sistema aggiorna lo stato dell'utente scelto e lo salva nel database

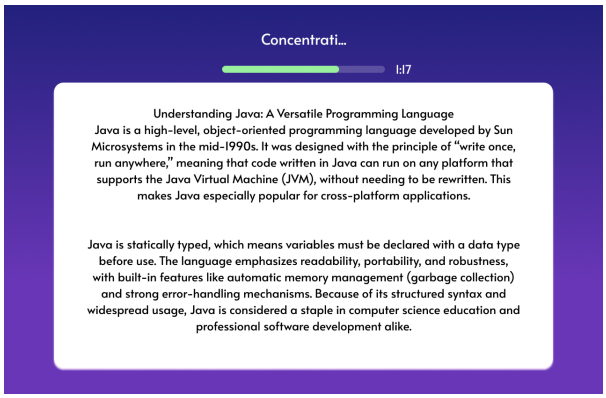
2.3 Design dell'interfaccia

Alla fase di definizione delle funzionalità è seguita la fase di design con la progettazione dei wireframe e dei mockup, utilizzati come bozza iniziale per la struttura dell'applicazione.



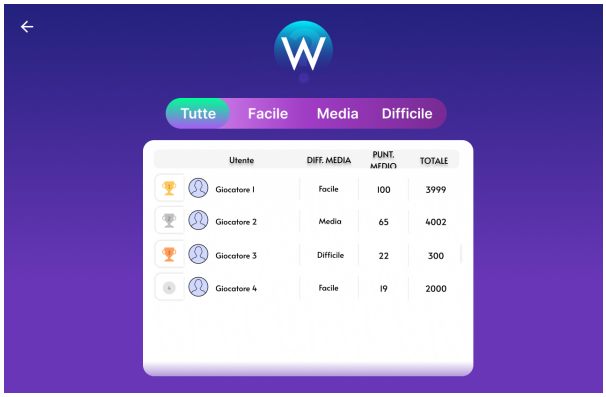
(a) Mockup Accesso

(b) Mockup Pannello Utente

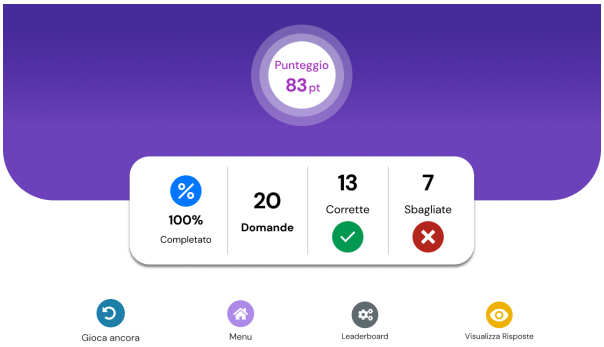


(c) Mockup Lettura

(d) Mockup Domande



(e) Mockup Leaderboard

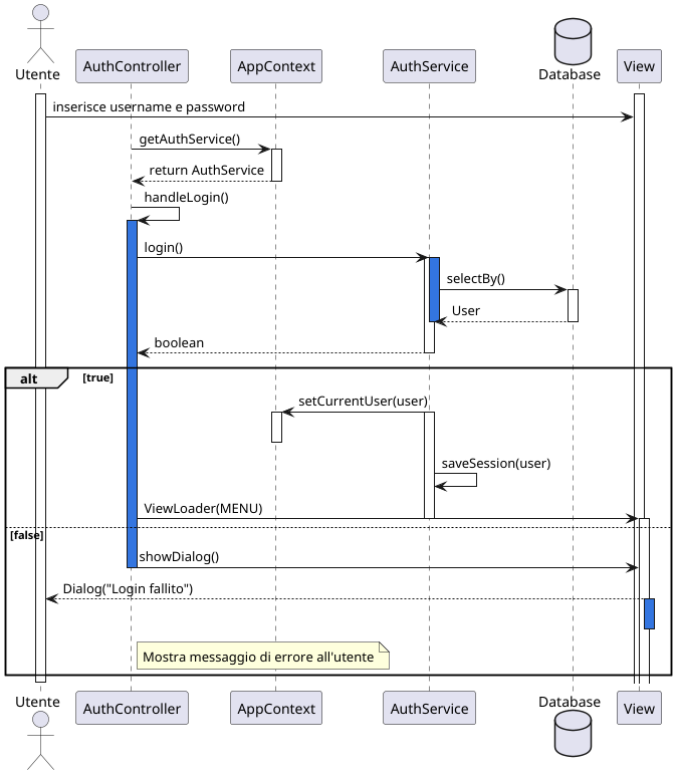


(f) Mockup Risultato

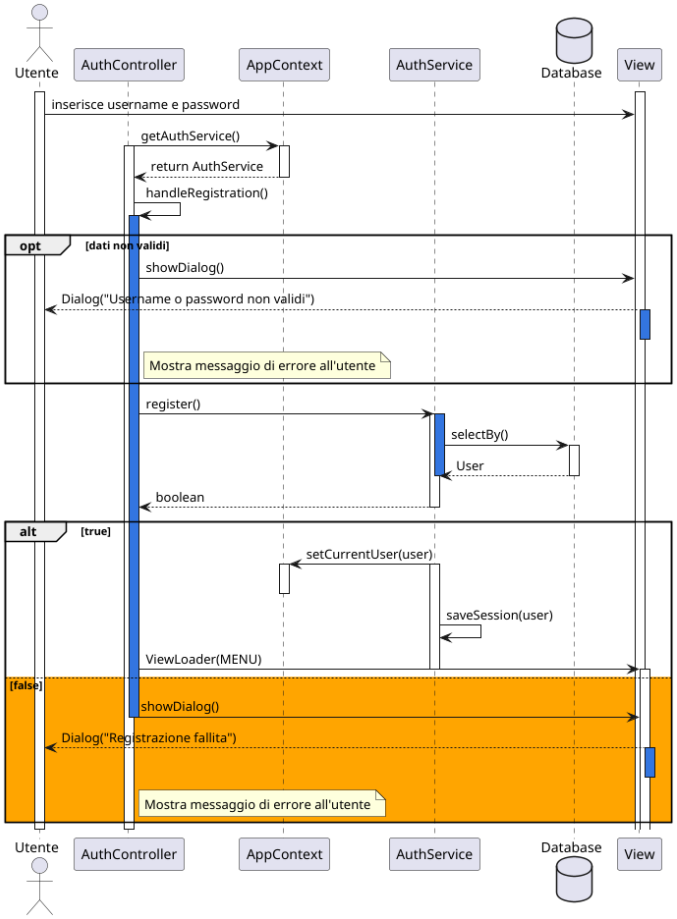
Per non appesantire eccessivamente la documentazione si riportano solamente le **schermate principali**. Tutti i mockups realizzati sono stati caricati all'interno della cartella `docs/mockups/`, presente nella root del progetto... é possibile accedere ad essi anche mediante il seguente link.

2.4 Diagrammi di sequenza

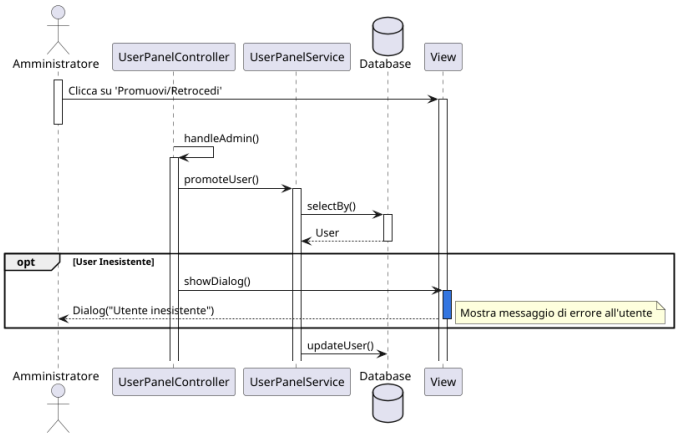
Si mostrano i diagrammi di sequenza del comportamento del sistema negli scenari di utilizzo **principali**. I diagrammi ad alta risoluzione sono stati caricati nella cartella `docs/UML/Sequenza`.



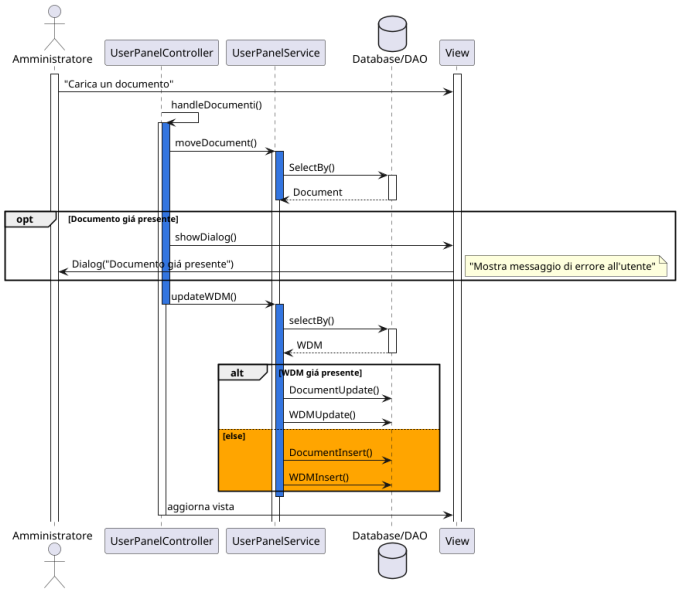
(g) Login



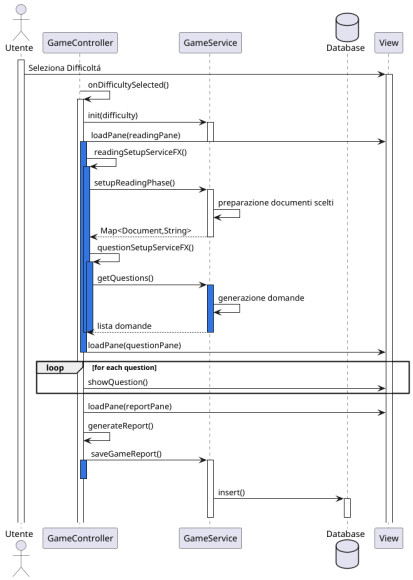
(h) Registrazione



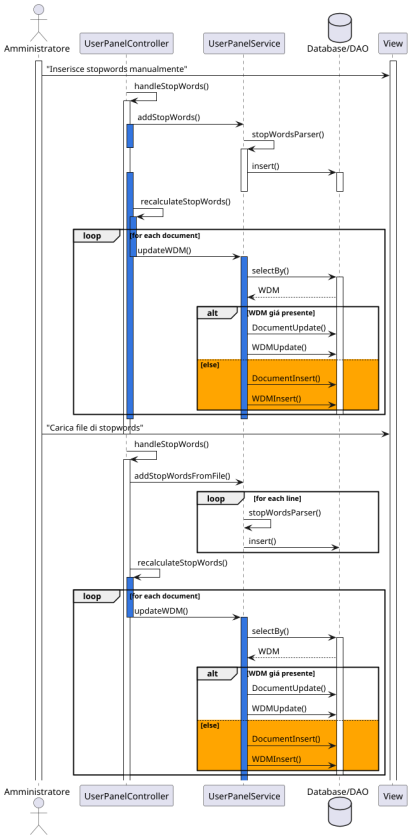
(i) Promote/Demote



(j) Caricamento Documenti



(k) Game



(l) Caricamento Stopwords

3 Dettagli Implementativi

L'architettura del progetto *Wordageddon* si basa su una distribuzione ordinata delle responsabilità, con una chiara separazione tra modelli, servizi, controller e componenti di supporto. L'intera struttura adotta una combinazione del pattern **Model-View-Controller (MVC)** con un **Service Layer** centrale, pensato per incapsulare la **logica di business** dell'applicazione e mediare tra i controller FXML e le implementazioni del pattern **DAO**.

3.1 Organizzazione del codice sorgente

Il codice sorgente é stato suddiviso in package differenti al fine di separare al meglio le varie responsabilitá.

Figura 3: Diagramma dei package

- **controllers**: raccoglie tutti i controller JavaFX associati alle schermate dell'applicazione. Le classi all'interno di questo package fungono da ponte tra l'interfaccia grafica (FXML) e la logica applicativa implementata nei service. Ogni controller ha il compito di gestire gli eventi generati dall'utente, aggiornare la UI e invocare i metodi dei service in risposta alle interazioni. È qui che vengono istanziati popup, avviati task asincroni e manipolata l'interfaccia utente.
- **models**: contiene le classi che rappresentano le entità fondamentali del dominio: `User`, `Document`, `GameReport`, `WDM`, ecc. Fungono da ponte tra database e UI.
- **db**: è il cuore dell'accesso ai dati. Al suo interno, i sottopacchetti `contracts` definiscono le interfacce dei DAO (come `UserDAO`, `DocumentDAO`, `GameReportDAO`), mentre le classi concrete (`JDBCUserDAO`, ecc.) contengono le implementazioni basate su JDBC. Questo modulo incapsula tutte le query SQL e astrae il comportamento di salvataggio, recupero, eliminazione e modifica dei dati.
- **services**: questo livello media tra DAO e controller. Classi come `GameService`, `AuthService`, `LeaderboardService`, `UserPanelService` contengono tutta la logica funzionale dell'applicazione. Ogni operazione significativa (ad esempio la creazione delle domande, la gestione del punteggio, l'upload dei documenti) viene orchestrata da qui, rendendo i controller leggeri e mantenendo la logica centralizzata e testabile.
- **utility**: include classi di supporto trasversali e riutilizzabili. La `Popup` è una classe custom per la creazione di finestre modali coerenti, `SystemLogger` offre una gestione centralizzata e semplificata del logging, `ViewLoader` gestisce la navigazione tra scene FXML e l'iniezione di controller dipendenti da AppContext mentre `Resources` consente l'accesso a percorsi e file usati in più punti del sistema facilmente.
- **WordageddonApp**: la classe di avvio principale. Qui viene configurato il sistema di iniezione dei controller tramite un `ControllerFactory`, e viene lanciata la schermata

iniziale. Questo punto di ingresso è anche responsabile della configurazione del contesto applicativo (`AppContext`), che funge da container per le istanze dei service e DAO.

3.2 Threads

Per garantire la responsibilità dell'interfaccia utente ed evitare blocchi dell'*Application Thread*, Wordageddon impiega meccanismi asincroni per l'esecuzione in background di operazioni computazionalmente pesanti o I/O-intensive.
Durante operazioni come il caricamento dei documenti, viene avviato un `Task<T>` JavaFX che, in background, elabora il file e aggiorna la relativa WDM. In questo modo, la UI continua a rispondere fluidamente mentre l'elaborazione procede.

Per operazioni più intensive, come il ricalcolo di tutte le WDM al variare delle stopword, si utilizza un `ExecutorService` con **pool fisso**. A ciascun documento è associato un `Callable`, eseguito tramite `invokeAll`, evitando la creazione di thread dedicati e mantenendo così sotto controllo il carico concorrente. L'intero processo di ricalcolo delle WDM, è sincronizzato attraverso due **flag atomici**, che evitano conflitti tra ricalcoli in corso e nuove richieste, garantendo così consistenza e affidabilità nel database e nelle risposte dell'applicazione.

All'avvio della schermata di gioco, l'applicazione sfrutta due servizi asincroni distinti per separare la **logica di caricamento del testo** da quella di **generazione delle domande**. Il primo, `readingSetupService`, carica in background il contenuto dei documenti compatibili con il livello di difficoltà scelto dall'utente. I documenti vengono selezionati tramite una logica dedicata all'interno del `GameService`, e il contenuto viene recuperato in modo asincrono attraverso un `Task` che restituisce una mappa documento-testo. Terminato il caricamento, la UI viene aggiornata per visualizzare il primo documento, mentre in parallelo viene avviato il `questionSetupService`, incaricato di generare le domande basate sul documento mostrato, o sui documenti mostrati.

4 Persistenza dei dati

La logica di persistenza è stata gestita mediante una **gerarchia DAO (Data Access Object)** implementata in JDBC, **centralizzata** attraverso un repository (`JdbcRepository`) che fornisce accesso modulare ai diversi DAO in base alla tipologia di entità (utenti, documenti, partite, ecc.). Tutti i DAO convivono e si coordinano attraverso questo repository, che funge da punto di accesso

unico alla logica di persistenza dell'applicazione. Questo schema ha consentito di isolare il livello di accesso ai dati e mantenere i controller liberi da dettagli di basso livello, rendendo il codice più testabile e scalabile. Il repository si occupa anche della gestione persistente della **singola connessione** al database *SQLite*, condivisa da tutti i DAO. In questo modo si è evitato l'impiego di un **Singleton**, preservando al contempo un controllo centralizzato ed efficiente sull'accesso al database.

4.1 Database SQLite

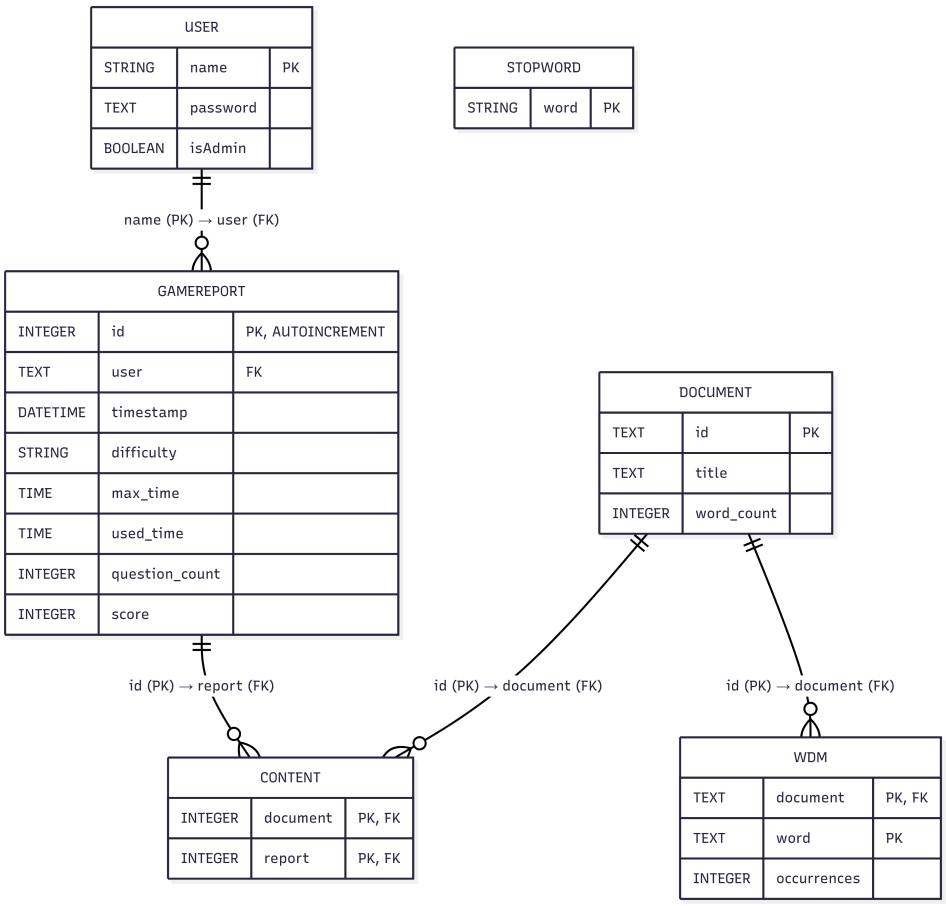


Figura 4: Diagramma ER-UML

Il database `SQLite` dell'applicazione Wordageddon è progettato per garantire la persistenza e l'integrità dei dati relativi agli **utenti**, alle **sessioni di gioco**, ai **documenti** analizzati e alle **stopwords** caricate dagli amministratori.

1. **User:** contiene le informazioni sugli **utenti registrati**, tra cui nome utente (chiave primaria), password e ruolo amministrativo (isAdmin). Sono presenti **trigger** che impediscono la rimozione o la modifica dell'ultimo amministratore, garantendo sempre la **presenza di almeno un admin** nel sistema;

2. **GameReport:** memorizza i **report delle sessioni di gioco**, associando ciascun report a un utente, con dettagli su data/ora, livello di difficoltà, tempo massimo e utilizzato, numero di domande e punteggio ottenuto. Sono presenti **vincoli** per assicurare la correttezza dei dati inseriti (ad esempio, valori ammessi per la difficoltà, formati dei tempi e punteggi non negativi);

3. **Document:** rappresenta i **documenti caricati** nell'applicazione, con identificativo, titolo e conteggio delle parole;

4. **Content:** tabella di relazione che collega i **documenti** alle specifiche **sessioni di gioco** (report), consentendo di tracciare quali documenti sono stati utilizzati in ogni partita.

5. **WDM:** memorizza, per ogni documento, le **parole incontrate** e il numero di occorrenze, permettendo analisi statistiche e la generazione delle domande di gioco;

6. **StopWord:** contiene la lista delle **stopword** utilizzate per filtrare le parole irrilevanti durante l'analisi dei testi.

4.2 Persistenza su file

La persistenza su file viene utilizzata per gestire la **sessione dell'utente**, con l'obiettivo di mantenere l'utente **autenticato** tra diversi avvii dell'applicazione, e per la gestione delle **sessioni interrotte**. Quando un utente effettua il login oppure una nuova registrazione, tutte le informazioni necessarie, comprese le credenziali e il ruolo amministrativo, vengono **salvate localmente** tramite la **serializzazione** dell'oggetto `User`.

Grazie a questo meccanismo, se l'utente riapre l'applicazione in un momento successivo, è possibile **ripristinare automaticamente** la sessione semplicemente leggendo il file precedentemente salvato: non è quindi necessario ripetere il login ogni volta. Infine, al momento del **logout**, la sessione viene **azzerata eliminando il file** dal filesystem, così da garantire che un nuovo avvio dell'applicazione comporti una nuova autenticazione.
Per quanto riguarda il ripristino di una sessione di gioco interrotta invece, il sistema serializza automaticamente l'oggetto `GameSessionState` se l'utente chiude l'applicazione mentre si trova nella fase di quiz. Al successivo avvio dell'app, il sistema controlla la presenza dell'oggetto serializzato e, se presente invita l'utente a continuare la sessione di gioco.

5 Esecuzione dell'app

Per una rapida esecuzione dell'applicazione è stato fornito un file `.jar`. La compilazione dei file sorgenti è stata effettuata utilizzando la `JDK 24` per cui è necessario utilizzare una versione pari o superiore per l'esecuzione del file eseguibile. L'eseguibile fornito, inoltre, **non contiene le dipendenze** di `JavaFX` per cui è necessario scaricare l'sdk dal sito ufficiale e specificare a runtime i moduli necessari:

```
java --module-path $PATH_TO_FX --add-modules javafx.controls,javafx.fxml
-jar Wordageddon.jar
```

Dove `PATH_TO_FX` rappresenta il percorso alla sdk precedentemente scaricata. Si tiene presente che il file .jar deve essere eseguito dalla stessa cartella (root) in cui é presente il database (`db.sqlite`).

5.1 Test dell'app

Nella cartella `docs/testDocs/` sono stati inseriti una serie di **file** che possono essere utilizzati per testare l'applicazione.
Sono forniti 2 **utenti pre-registrati** (un utente standard ed uno con privilegi di admin) con cui è possibile testare direttamente l'applicazione senza doversi necessariamente registrare:

Username	**Password**
admin	admin
demo	demo

Tabella 1: Utenti forniti

6 Documentazione tecnica

La documentazione tecnica del codice sorgente è stata generata mediante **Javadoc** ed è disponibile presso il seguente link: https://bg735.github.io/Wordageddon-Gruppo_16.