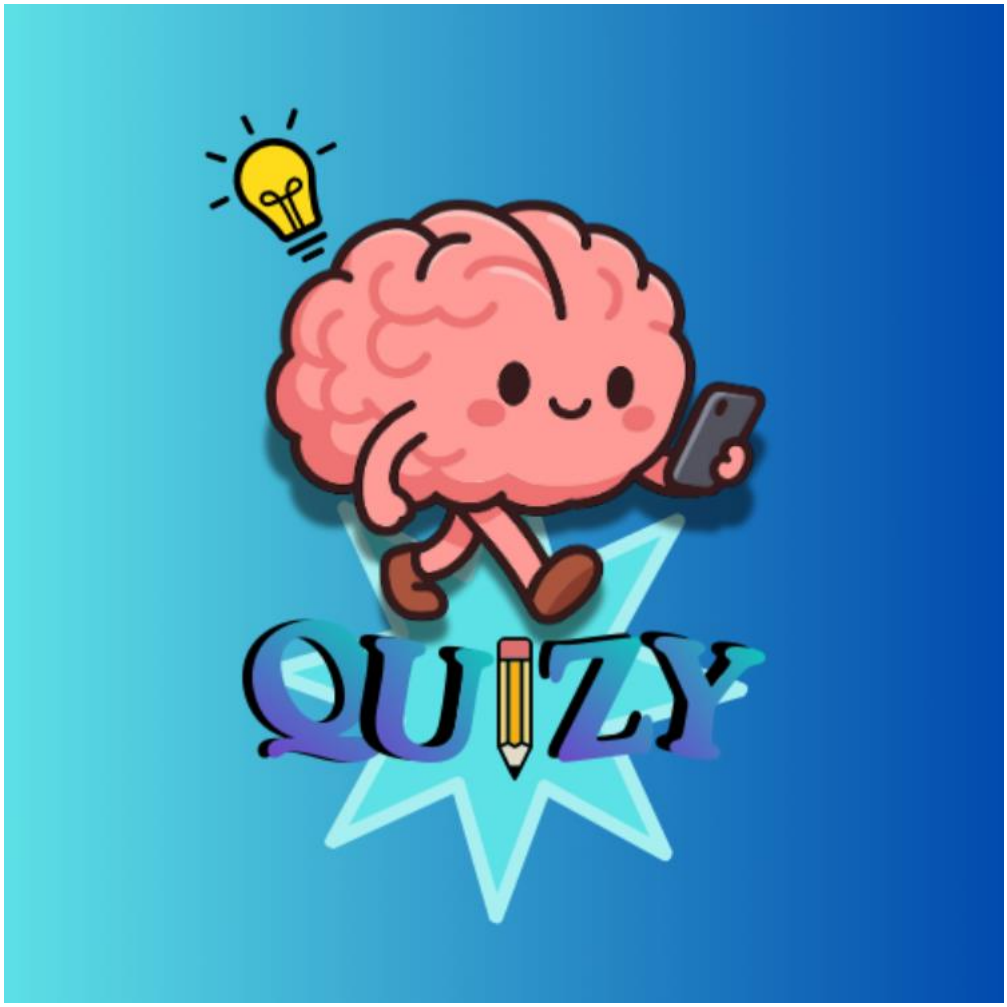
Università degli Studi di Salerno

Corso di Ingegneria del Software

Quizy

Object Design Document

Versione 0.3



Data: 22/12/2025

Progetto: Quizy	Versione: 0.3
Documento: Object Design Document	Data: 22/12/2025

Coordinatore del progetto:

Nome	**Matricola**
Davide Nino Longobardi	0512119971

Partecipanti:

Nome	**Matricola**
Roberto Caiazza	0512116335

Revision History

Data	**Versione**	**Descrizione**	**Autore**
18/12/2025	0.1	Stesura iniziale del documento	Davide Nino Longobardi, Fabiana Paciello, Roberto Caiazza
19/12/2025	0.2	Completamento del documento	Davide Nino Longobardi, Roberto Caiazza
06/12/2025	0.3	Revisione Documento	Davide Nino Longobardi, Roberto Caiazza

1.INTRODUZIONE....3
1.1.Object Design trade-off....4
1.2.Interface documentation guidelines....4
Naming Conventions:....5
Exception Handling:....5
Visibilità e incapsulamento....5
1.3.Definitions, acronyms and abbreviations....5
1.4.References....5
2.PACKAGES....6

1.INTRODUZIONE

Questa sezione analizza i criteri utilizzati per definire l'architettura, focalizzandosi sul raggiungimento di un equilibrio tra presentazioni tecniche sostenibilità del codice e costi operativi.

1.1.Object Design trade-off

Nel processo di progettazione del sistema sono state adottate alcune scelte architetturali consapevoli, ciascuna delle quali introduce compromessi tra prestazioni, complessità e sicurezza.

- **Riduzione delle Query vs Carico del server:** Per evitare di eseguire troppe volte le stesse query verso il database, è stato creato un componente chiamato QuizLog. Questo componente tiene traccia delle informazioni che sono già state derivate ed elaborate. In questo modo, il sistema non deve più accedere direttamente al database ogni volta, il che migliora i tempi di risposta, specialmente quando gli utenti stanno eseguendo i quiz.
  Il rovescio della medaglia è che il server deve lavorare di più e gestire lo stato dell'applicazione in modo più complesso. Tuttavia, questi svantaggi sono considerati accettabili se paragonati ai benefici che si ottengono in termini di prestazioni del sistema.
- **Manutenibilità del codice vs Complessità del sistema:** Il sistema utilizza un approccio di Object-Relational Mapping tramite JPA e Hibernate, anziché collegarsi direttamente al database con JDBC. Questo metodo aggiunge un livello di astrazione e complessità, ma rende il codice più leggibile, separa meglio le responsabilità e aumenta la sicurezza quando si salvano i dati. Il prezzo da pagare è che il sistema diventa più strutturato e potenzialmente più lento, ma in cambio è più facile da gestire e aggiornare il modello dei dati.
- Rigidità delle sessioni vs Sicurezze: la gestione delle sessioni è stata progettat in modo restrittivo, impedendo l'accesso simultaneo dello stesso utente da più sessioni attive. Questa decisione riduce la flessibilità d'uso del sistema ma aumenta significativamente il livello di sicurezza, prevedendo accessi concorrenti non autorizzati e riucendo il livello di sessio hijacking.

1.2.Interface documentation guidelines

Per garantire la coerenza nello sviluppo del codice e facilitare l'integrazione tra il client Android e il backend Java, ci atteniamo alle seguenti convenzioni di codifica e documentazione:

Naming Conventions:

- **Classi**: Utilizza del PascalCase con sostantivi singolari. I nomi descrivono il dominio del problema (es. Quiz, Utente ...)
- **Metodi**: Utilizzo del camelCase. I nomi devono iniziare con un verbo che descrive l'azione eseguita (es. CreaQuiiz, getPunteggio)
- **Costanti**: Utilizzo odell'UPPER_SNAKE_CASE. Tutte le lettere maiuscole separate da underscore
- **Variabili**: Utilizzo del camelCase per variabili locali e parametri

Exception Handling:

La gestione degli errori non avviene tramite codici di ritorno, per esempio *null*, ma attraverso il lancio di Eccezioni specifiche.

La gestione delle sessioni è stata progettata in modo restrittivo, impedendo l'accesso simultaneo dello stesso utente da più sessioni attive. Questa decisione riduce la flessibilità d'uso del sistema, ma aumenta significativamente il livello di sicurezza, prevenendo accessi concorrenti non autorizzati e riducendo il rischio di session hijacking. Il sistema privilegia quindi la sicurezza rispetto alla comodità dell'utente.

Visibilità e incapsulamento

- Tutti gli attributi delle classi Entitty devono essere private
- L'accesso ai dati avviene tramite metodi accessori public (Getters e Setters) per garantire l'incapsulamento

1.3.Definitions, acronyms and abbreviations

- **MVC (Model-View-Control):** Pattern architettura utilizzato.
- **DAO (Data Access Object)**: Pattern utilizzato per astrarre e incapsulare l'accesso al dattabase.

1.4.References

- Documento, Problem Statement

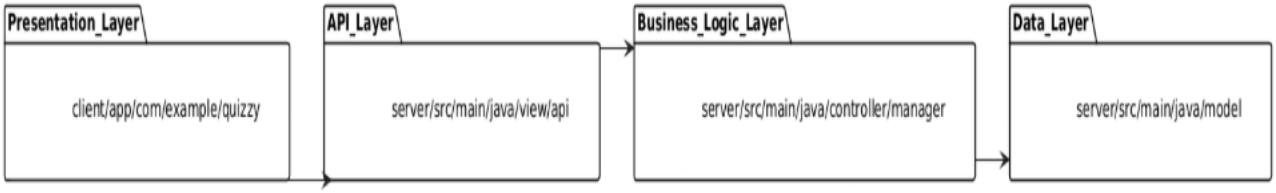
- Documento, Specifiche dei requisiti (RAD)
- Documento, System Designe Document (SDD)

2.PACKAGES

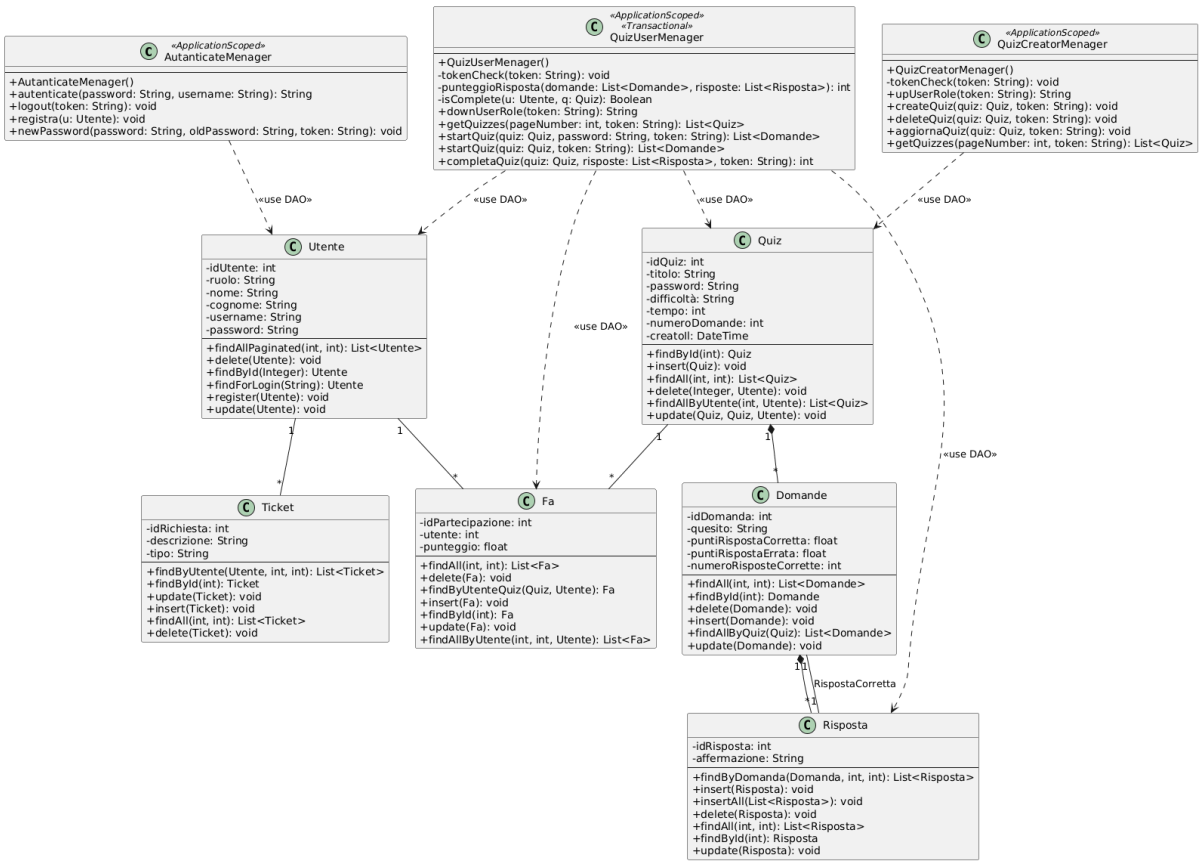
Il progetto è organizzato secondo il pattern architetturale **MVC** (Model–View–Controller), al fine di garantire una chiara separazione delle responsabilità e migliorare la manutenibilità del sistema. La struttura dei pacchetti è organizzata in base a questa suddivisione logica e può essere descritta nel modo seguente:

- **server/src/main/java/controller/menager**:
  Questo pacchetto contiene le classi responsabili della logica applicativa e del coordinamento dei flussi di esecuzione. I manager si trovano a metà strada tra il controller e il model. Lavorano per coordinare le operazioni di business. Inoltre, aiutano a gestire l'accesso ai dati, lasciando che le componenti di persistenza si occupino dei dettagli. Utilizzano anche classi di supporto per semplificare le cose e riutilizzare la logica comune, in modo che non si debba ricominciare da capo ogni volta.
- **server/src/main/java/model**:
  Il pacchetto model rappresenta il livello di dominio e di persistenza del sistema. Contiene le Entity JPA, che modellano le entità persistenti del dominio applicativo, e le componenti incaricate della gestione dell'accesso ai dati (repository/DAO). Questo livello incapsula completamente la logica di interazione con il database, rendendo il resto del sistema indipendente dai dettagli di persistenza.
- **server/src/main/java/view/api**:
  Il pacchetto view/api espone le API dell'applicazione sotto forma di endpoint. Queste componenti ricevono le richieste client, validano i dati in ingresso e delegano l'esecuzione della logica applicativa ai manager, occupandosi esclusivamente della gestione dell'interazione esterna e della restituzione delle risposte.
- client/Quizy/app
  Il pacchetto contiene l'applicazione android

Struttura dei pacchetti:



3.DIAGRAMMA DELLE CLASSI



4.CLASS INTERFACE

4.1. package manager

4.1.1. Classe AutanticateManager

Questa classe gestisce il ciclo di vita della sessione utente e la sicurezza delle credenziali.

1° Metodo:

autenticate(password: String, username: String)

- *Descrizione:* Verifica le credenziali e genera un token di sessione.
- *Pre-condizione:* Username e password non devono essere nulli o vuoti.
- *Post-condizione:* Se le credenziali sono valide, restituisce un token di sessione attivo associato all'utente. Altrimenti lancia un'eccezione.

1° OCL:

context AutanticateMenager::autenticate(password: String, username: String): String

pre: password <> null and password.size() > 0 and username <> null and username.size() >0

post: result <> null and self.logBeble.isAlive(result) = true

-- Oppure lancia LoginFailed se le credenziali non corrispondono

2° Metodo:

logout(token:String)

- Descrizione: Termina la sessione corrente
- Pre – condizione: Il token deve essere valido
- Post – condizione: Il token viene rimosso dalla lista delle sessioni attive

2° OCL

context AutanticateMenager::logout(token: String)

pre: token <> null

post: self.logBeble.isAlive(token) = false

4.1.2. Classe QuizCreatorManager

Gestisce la logica di creazione e modifica dei quiz, assicurando che solo gli utenti con ruolo "Creatore" possano eseguire operazioni di scrittura.

1° Metodo:

createQuiz (quiz: Quiz, token:String)

- *Descrizione:* Memorizza un nuovo quiz nel sistema.
- *Pre-condizione:* Il token deve appartenere a un utente con ruolo "Creatore". Il quiz deve essere valido (non nullo).
- *Post-condizione:* Il quiz viene persistito nel database e associato all'utente corrente.

1° OCL:

context QuizCreatorMenager::createQuiz(quiz: Quiz, token: String)

pre: self.logBeble.isAlive(token) and self.accessControl.checkCreatore(token) -- Verifica ruolo and quiz <> null

post: self.dao.findById(quiz.id) <> null -- Il quiz ora esiste nel DB

2° Metodo

getQuizzes(pageNumber: Integer, token: String)

- *Descrizione:* Restituisce una lista paginata di quiz.
- *Pre-condizione:* Il numero di pagina non deve essere negativo.
- *Post-condizione:* Ritorna una lista di quiz (eventualmente vuota, ma non nulla).

2° OCL:
context QuizCreatorMenager::getQuizzes(pageNumber: Integer, token: String): Sequence(Quiz)

pre: pageNumber >= 0

post: result <> null

4.1.3. Classe QuizUserMenager

Gestisce lo svolgimento dei quiz da parte degli utenti standard (Player) e il calcolo dei punteggi.

1° Metodo:

completaQuiz(quiz: Quiz, risposteClient: Sequence(Risposta), token: String)

- *Descrizione:* Elabora le risposte inviate dall'utente, calcola il punteggio e salva il tentativo.
- *Pre-condizione:* Liste di risposte non nulle. Sessione attiva.
- *Post-condizione:* Viene creato un nuovo record Fa (tentativo) e il punteggio restituito è la somma corretta dei punti.

1° OCL:

context QuizUserMenager::completaQuiz(quiz: Quiz, risposteClient: Sequence(Risposta), token: String): Integer

pre: quiz <> null and risposteClient <> null and self.logBeble.isAlive(token)

post: result = risposteClient->iterate(r; sum: Integer = 0 |

if r.flagRispostaCorretta then sum + r.domanda.puntiRispostaCorretta

else sum + r.domanda.puntiRispostaSbagliata endif)

and self.daoFa.findByUserAndQuiz(self.logBeble.getUtente(token), quiz) <> null

4.2. package entity

4.2.1. Classe Utente

Rappresenta un utente registrato alla piattaforma (può essere Player, Creatore o Manager).
Attributi:

- private Integer id;

- private String nome;
- private String cognome;
- private String username;
- private String passwordHash;
- private Boolean isCreatore;
- private Boolean isCompilatore;
- private Boolean isManager;

Metodi: Getter e Setter per tutti gli attributi.

Invarianti (OCL):

- Context Utente inv: self.username.size() > 0 and self.username.size() <= 50
- Context Utente inv: self.nome <> null and self.cognome <> null
- Context Utente inv: self.passwordHash <> null

4.2.2. Classe Fa

Rappresenta l'avvenuto svolgimento di un quiz da parte di un utente (Tabella di associazione con punteggio). **Attributi:**

- private Integer id;
- private Utente utente;
- private Quiz quiz;
- private Integer punteggio;
- private List<Risponde> risposteDate;

Metodi: Getter e Setter per tutti gli attributi. **Invarianti (OCL):**

- Context Fa inv: self.utente <> null
- Context Fa inv: self.quiz <> null
- Context Fa inv: self.punteggio <> null

4.3. package utility

4.3.1. Classe SessionLog

Questa classe implementa il pattern **Singleton** per gestire le sessioni attive in memoria, riducendo il carico sul database per la verifica dei token. Mantiene una mappa tra token e utenti loggati.

1° Metodo:

aggiungi(token: String, utente: Utente)

- *Descrizione:* Registra una nuova sessione utente. Se l'utente ha già una sessione attiva, gestisce il conflitto.
- *Pre-condizione:* Il token e l'utente non devono essere nulli.
- *Post-condizione:* Il token viene inserito nella mappa delle sessioni attive (logBible).

1° OCL:

context SessionLog::aggiungi(t: String, u: Utente)

pre: t <> null and u <> null

post: self.logBible.containsKey(t) and self.logBible.get(t) = u

2° Metodo:

isAlive(token: String)

- *Descrizione:* Verifica se un token corrisponde a una sessione valida e non scaduta.
- *Pre-condizione:* Il token deve essere una stringa valida.
- *Post-condizione:* Restituisce true solo se il token è valido crittograficamente e presente in memoria. Altrimenti lancia un'eccezione e rimuove la sessione.

2° OCL:

context SessionLog::isAlive(t: String): Boolean

post: result = self.logBible.containsKey(t) and self.jwtProvider.validateToken(t) -- Implica validità temporale

4.3.2. Classe AccessControlService

Questa classe funge da gatekeeper per la sicurezza, implementando il controllo degli accessi basato sui ruoli (**RBAC - Role Based Access Control**) leggendo i "claim" all'interno dei token JWT.

1° Metodo:

checkCreatore(token: String)

- *Descrizione:* Verifica che l'utente proprietario del token abbia il ruolo di "creatore" per poter modificare o creare quiz.
- *Post-condizione:* Se il ruolo estratto dal token non è "creatore", viene sollevata l'eccezione InvalidRole.

1° OCL:

```
context AccessControlService::checkCreatore(t: String)

    post: self.jwtProvider.getRoleFromToken(t) = 'creatore'
```