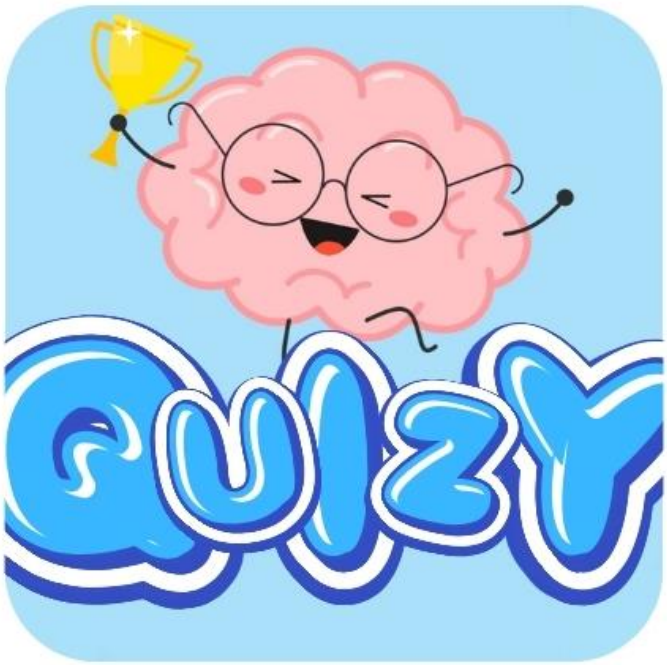
Università degli Studi di Salerno
Corso di Ingegneria del Software

Quizy
Object Design Document
Versione 0.0



Data: 22/12/2025

Coordinatore del progetto:

Nome	Matricola
Davide Nino Longobardi	0512119971

Partecipanti:

Nome	Matricola
Roberto Caiazza	0512116335
Fabiana Paciello	0512119578

Revision History

Data	Versione	Descrizione	Autore
18/12/25	0.1	Inizio	Davide Nino Longobardi, Roberto Caiazza, Fabiana Paciello
19/12/25	0.2	Continuo	Davide Nino Longobardi, Roberto Caiazza, Fabiana Paciello

1. INTRODUZIONE

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

1.1 Object Design trade-off

Nella progettazione di questa applicazione sono stati considerati i seguenti compromessi tecnici:

- **Trasparenza del Modulo vs. Performance:** Per la gestione dei Quiz, abbiamo scelto di utilizzare una struttura dati `List` (o `ArrayList`) per memorizzare le domande. Il trade-off è tra la facilità di iterazione per il calcolo del punteggio e il costo di memoria. Abbiamo privilegiato la **velocità di calcolo** (Performance) per garantire un'esperienza utente fluida durante lo svolgimento.
- **Accuratezza vs. Tempo di Risposta:** Nel calcolo dei risultati (metodo `calcoloRisultato` in `QuizManager`), il sistema processa i dati in tempo reale. Si è scelto di non utilizzare tecniche di caching per i punteggi intermedi per garantire la massima accuratezza dei dati, accettando un leggero carico computazionale sul server al termine di ogni quiz.
- **Information Hiding vs. Efficienza:** Tutti gli attributi delle classi `Quiz` e `Domanda` sono stati definiti `private`. L'accesso avviene esclusivamente tramite interfacce pubbliche. Questo garantisce che modifiche future alla struttura del database non impattino il resto del sistema, a scapito di un minimo overhead dovuto alle chiamate dei metodi getter/setter.

1.2 Interface documentation guidelines

Convenzioni:

- Nome delle classi: Singolare(es: Quiz, RispostaUtente)
- Nomi metodi: Frasi verbali (es: getQuizDisponibili)
- Eccezioni: Restituire tramite meccanismi dedicati

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 database.

1.4 References

- Specifiche dei requisiti (RAD)
- Diagrammi UML correlati
- System Designe Document (SDD)

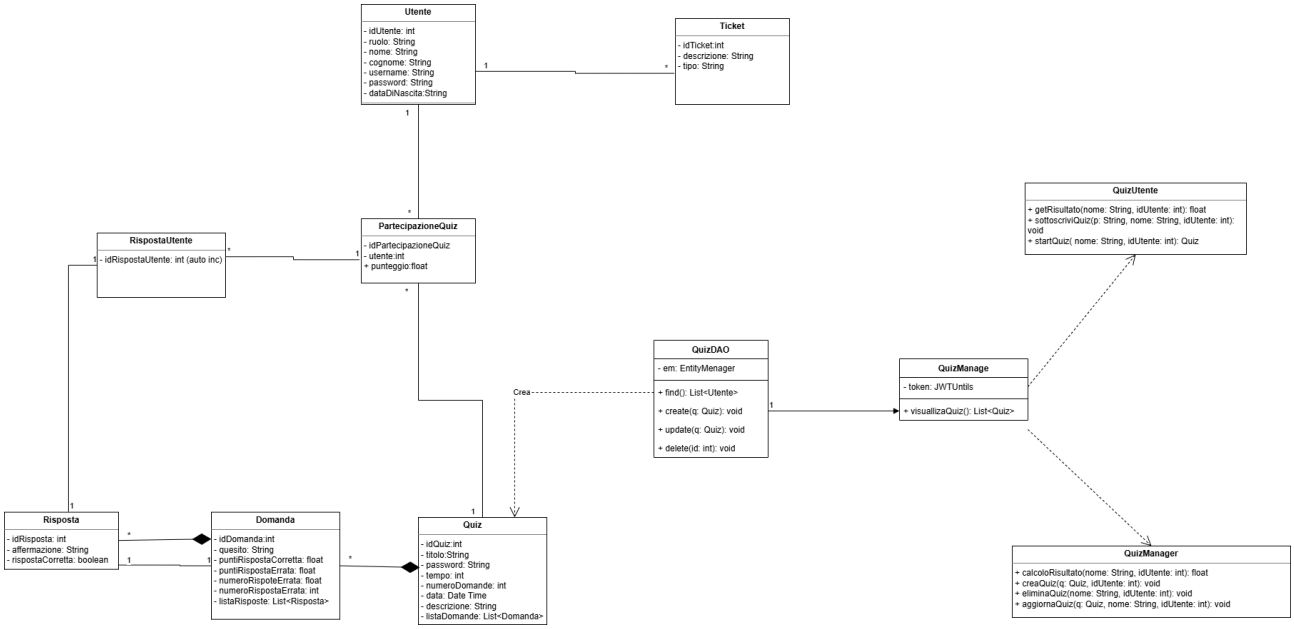
2. PACKAGES

2.1 Struttura dei pacchetti

Nel progetto sono presenti tre pacchetti logici inerenti al pattern **MVC (Model-View-Controller)** utilizzato per la strutturazione del Sistema Software. I pacchetti sono:

- Models: rappresenta il cuore dell'applicazione e l'implementazione del modello di dominio. Comprende le classi Entity (es. Utente, Quiz) che incapsulano i dati e le regole di validazione (Invarianti OCL).
  Nome del package dell'applicazione: ServerQuizy.
- **Views**: Responsabile della presentazione dei dati e dell'interazione con l'utente. Nel contesto del progetto, corrisponde all'applicazione Android (Activity e Layout XML) che visualizza le informazioni fornite dal sistema e cattura gli input dell'utente per inviarli al Controller
  Nome del package dell'applicazione: ViewsQuizy.
- **Controllers**: Intermediario tra Views e Models, è responsabile della gestione della logica applicativa. Le classi preposte ricevono le richieste dalla View, elaborano la business logic e ordinano al Model di aggiornare lo stato dei dati.
  Nome del package dell'applicazione: ControllersQuizy

3. DIAGRAMMA DELLE CLASSI



4. CLASS INTERFACE

In queto capitolo verranno riportate alcune delle classi più importanti del diagramma delle classi.

4.1)
Class: QuizDAO

Attributi:

- private EntityMenager em

Metodi:

+ find(): List<Quiz>

Context QuizDAO::find() : Sequence(Quiz)

post:

result->size() = Quiz.allInstances() ->size()

and

Quiz.allInstances()->forAll(q | result->includes(q))

+create(q Quiz): void

Context QuizDAO::create(q: Quiz)

pre:

q <> null and not Quiz.allInstances()->includes(u)

post:

Quiz.allInstances()->includes(q)

And

Quiz.allInstances()->size() = Quiz.allInstances()@pre->size() + 1

+updates(q Quiz): void

Context QuizDAO::updates(q: Quiz)

Pre:

q <> null and Quiz.allInstances()->includes(q)

Post:

Quiz.allInstances()->size() = Quiz.allInstances()@pre->size()

and

Quiz.allInstances()->includes(q)

and

Quiz.allInstances()->excluding(q) = Quiz.allInstances()@pre-
>excluding(q)

+delete(id int): void

Context QuizDAO::delete(id: int)

Pre:

Quiz.allInstances()->exists(q | q.id = id)

Post:

not Quiz.allInstances()->exists(q | q.id = id)

and

Quiz.allInstances()->size() = Quiz.allInstances()@pre->size() - 1

Invarianti:

```
context QuizDAO
            inv EntityManagerNotNull:
                  self.em <> null
```

4.2)

Class: QuizManage

Attributi:

- private JWTUntils token

Metodi:

+ VisualizzaQuiz(): List<Quiz>
Context QuizManage:: VisualizzaQuiz() : Sequence(Quiz)
pre:
Token <> null and jwtToken.size() > 0
and
self.authService.isValid(jwtToken) = true

Post:
-- (Ottenuti delegando al DAO)
result->asSet() = Quiz.allInstances()->asSet()
and
Quiz.allInstances() = Quiz.allInstances()@pre

Invarianti:
Context QuizManage
inv tokenIsValid:
Token <> null and jwtToken.size() > 0
and
self.authService.isValid(jwtToken) = true

4.3)
Class: QuizUtente (Interface)

Metodi:

+ getRisultato(nome String, idUtente int): Real
Context QuizUtente:: getRisultato(nome: String, idUtente: int):

-- PRE-CONDIZIONI (Requisiti di ingresso)
pre ValidInputs:
idUtente > 0 and nome <> ''

pre RecordExists:
QuizUtente.allInstances()->exists(qu |
qu.idUtente = idUtente and qu.nomeQuiz = nome
)

post CorrectResult:
result = QuizUtente.allInstances()->any(qu |
qu.idUtente = idUtente and qu.nomeQuiz = nome
).punteggio

```
        post RangeValido:
            result >= 0.0 and result <= 100.0

        post NoStateChange:
            QuizUtente.allInstances()->size() = QuizUtente.allInstances()@pre->size()


    + sottoscriviQuiz(p String, nome String, idUtente int): void
    Context QuizUtente:: sottoscriviQuiz(p: String, nome: String, idUtente: int): void
    pre ValidInputs:
                    idUtente > 0 and nome <> ''
                pre QuizExistsAndPassMatch:
                    Quiz.allInstances()->exists(q |
                        q.titolo = nome and q.password = p
                    )

                pre NotAlreadySubscribed:
                    not QuizUtente.allInstances()->exists(qu |
                        qu.idUtente = idUtente and qu.nomeQuiz = nome
                    )

                post SubscriptionCreated:
                    QuizUtente.allInstances()->exists(qu |
                        qu.idUtente = idUtente and qu.nomeQuiz = nome
                    )

                post CountIncreased:
                    QuizUtente.allInstances()->size() = QuizUtente.allInstances()@pre-
                    >size() + 1

    +  startQuiz(nome String, idUtente int): Quiz
Context QuizUtente::startQuiz(nome: String, idUtente: int) : Quiz

                pre ValidInputs:

                   idUtente > 0 and nome <> ''

                pre UserSubscribed:
                    QuizUtente.allInstances()->exists(qu |
                        qu.idUtente = idUtente and qu.nomeQuiz = nome
                    )

                pre QuizAvailable:

                    Quiz.allInstances()->exists(q | q.titolo = nome)

                post CorrectQuizReturned:
```

result.titolo = nome

post NoStateChange:

Quiz.allInstances()->size() = Quiz.allInstances()@pre->size()

Invarianti:
context QuizUtente

inv UniqueSubscription:
QuizUtente.allInstances()->isUnique(qu |
Tuple{id = qu.idUtente, nome = qu.nomeQuiz}
)

inv ScoreBoundaries:
self.punteggio >= 0.0 and self.punteggio <= 100.0

Inv ValidIdentity:
self.idUtente > 0 and self.nomeQuiz <> ''

4.4)
Class: QuizMenager (Interface)

Metodi:
+calcoloRisultato(nome String, idUtente int): Real
Context QuizMenager:: calcoloRisultato (nome: String, idUtente: int): Real
pre ValidInputs:
idUtente > 0 and nome <> ''

pre RecordExists:
QuizUtente.allInstances()->exists(qu |
qu.idUtente = idUtente and qu.nomeQuiz = nome
)

post CorrectRange:
result >= 0.0 and result <= 100.0

+creaQuiz(q Quiz, idUtente int): void
Context QuizMenager:: creaQuiz (nome: String, idUtente: int): void

pre ValidInputs:
q <> null and idUtente > 0

pre QuizNotExists:
not Quiz.allInstances()->includes(q)

post QuizPersisted:
Quiz.allInstances()->includes(q)

post CountIncreased:
Quiz.allInstances()->size() = Quiz.allInstances()@pre->size()
+ 1

+eliminaQuiz(nome String, idUtente int): void
Context QuizMenager:: eliminaQuiz (nome: String, idUtente: int): void

pre ValidInputs:
idUtente > 0 and nome <> ''

pre QuizExists:
Quiz.allInstances()->exists(q | q.titolo = nome)

pre RecordExists:
QuizUtente.allInstances()->exists(qu |
qu.idUtente = idUtente and qu.nomeQuiz = nome
)

post QuizRemoved:
not Quiz.allInstances()->exists(q | q.titolo = nome)

post CountDecreased:
Quiz.allInstances()->size() = Quiz.allInstances()@pre->size() - 1

post CascadeDelete (Integrità):
not QuizUtente.allInstances()->exists(qu | qu.nomeQuiz = nome)

+aggiornaQuiz(q Quiz, nome String, idUtente int): void
Context QuizMenager:: aggiornaQuiz(q: Quiz, nome: String, idUtente: int):
void

pre ValidInputs:
idUtente > 0 and nome <> '' and q <> null

pre QuizExists:
Quiz.allInstances()->exists(quiz | quiz.titolo = nome)

pre UserAuthorized:
QuizUtente.allInstances()->exists(qu |
qu.idUtente = idUtente and qu.nomeQuiz = nome
)

post QuizUpdated:

Quiz.allInstances()->any(quiz |
quiz.titolo@pre = nome

```
                    ).titolo = q.titolo

        post NoStructuralChange:
        Quiz.allInstances()->size() = Quiz.allInstances()@pre->size()

Invarianti:
Context QuizMenager
    Inv ValidIdentity:
    self.idUtente > 0 and self.nomeQuiz <> ''

    inv AuthorizedManager:
    Utente.allInstances()->exists(u |
            u.id = self.idUtente and u.ruolo = 'menagerQuiz'
        )
```

5. GLOSSARIO