JGuesser

Documento di Architettura
(Diagramma delle classi e codice in OCL)

UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione
Gruppo: T24
Membri: Lorenzo D'Ambrosio, Andrea Goldoni, Marco Murru

Indice

Scopo del documento

Nei documento precedente si è posta particolare attenzione sui requisiti del progetto, descritti tramite use-case diagram e tabelle strutturate e sul design descritto tramite diagramma del contesto e diagramma delle componenti.
In questo documento si descrive l'architettura dell'interno sistema. Questo verrà fatto attraverso il diagramma delle classi che è stato costruito a partire dal diagramma delle componenti e del diagramma di contesto e attraverso il linguaggio OCL (Object Constraint Language), un linguaggio che permette di specificare dei vincoli per attributi e metodi delle classi.

1 Diagramma delle classi

In questo capitolo vengono presentate e descritte le classi, che sono richieste per lo sviluppo dell'applicazione JGuesser. Tutte le classi sono costituite da un insieme di attributi e metodi. Gli attributi rappresentano i dati su cui la classe dovrà operare. I metodi sono invece le funzionalità che la classe può offrire. Entrambi possono essere:

- pubblici (+): l'attributo/metodo può essere accessibile dall'esterno della classe in cui è stato definito.
- privati (-): l'attributo/metodo può essere accessibile solo all'interno della classe in cui è stato definito.
- protetti (#): l'attributo/metodo può essere accessibile solo all'interno della classe in cui è stato definito e nelle classi derivate.

Nel nostro caso però non sono stati utilizzati, perchè discutendo ci siamo resi conto, che sono specifiche di programmazione e non di progettazione, esattamente come i getters/setters.
Le classi inoltre possono essere messe in relazione tra di loro ed è possibile associarvi una molteplicità. Una relazione tra due classi rappresenta il fatto che le due classi interagiscono fra di loro e si scambiano dei dati.
Per individuare le classi quello che è stato fatto è partire dal diagramma delle componenti e di contesto. Questo perchè di solito una componente del diagramma delle componente viene mappata in una o più classi e un attore del diagramma di contesto può essere mappato in una classe (es: user). Siccome le classi coinvolte nel diagramma del progetto JGuesser sono tante, si è optato per spezzettare la descrizione in una o gruppi di classi, un po' come è stato fatto per gli use-case diagram.

1.1 Utente

Analizzando il diagramma del contesto del progetto JGuesser è possibile individuare gli attori:

- Guest: non bisogna salvare alcun dato per lui;
- Utente registrato: bisogna salvare una serie di dati;
- Utente premium: non bisogna salvare dati aggiuntivi all'utente registrato, in quanto l'utente premium avrà accesso semplicemente a funzionalità aggiuntive, che non produrranno però dati aggiunitvi da memorizzare.

Per questo motivo si è optato in fase di progettazione del diagramma delle classi, di fare una classe singola e di non utilizzare l'erditarietà (generalizzazione). Oltre ai dati da memorizzare per l'utente questa classe contiene anche una serie di metodi, che servono ad effettuare dei controlli su come i dati sono formattati. Questi metodi sono utili per effettuare dei controlli preventivi nel front-end ed evitare di interrogare il back-end, per cose che si potrebbero fare benissimo lato client.

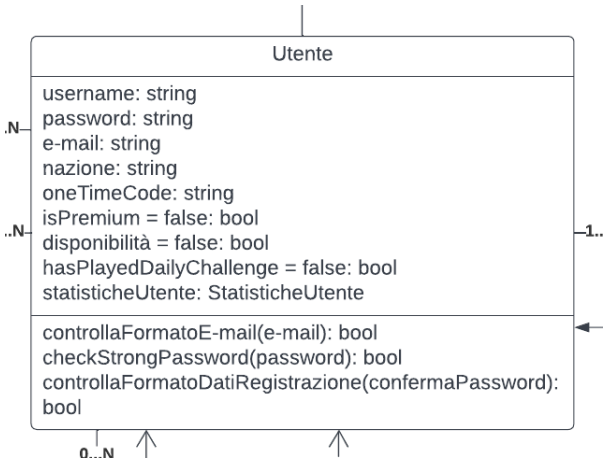

Figura 1: Classe per i dati dell'utente

1.2 Sistema di invio e-mail

Analizzando il diagramma delle componenti del progetto JGuesser è possibile individuare la componente InterfacciaNodemailer, che si interfaccerà con il corrispettivo sistema esterno. Questa componente è stata trasformata nella seguente classe:

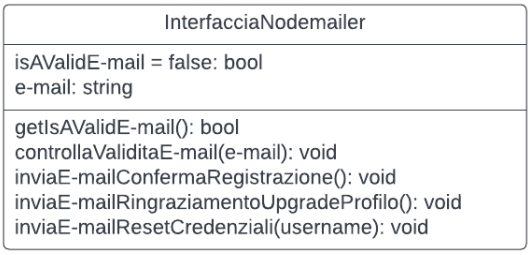

Figura 2: Classe InterfacciaNodemailer

Questa classe sarà quella che verrà utilizzata per verificare la validità di un e-mail, attraverso il metodo controllaValiditàE-mail(e-mail). Tale metodo andrà quindi a settare le variabili e-mail e isAValidE-mail, a cui poi sarà possibile accedere. Oltre a ciò questa classe ha il compito di inviare tutte le e-mail che l'applicazione necessita di spedire.

1.3 reCAPTCHA

Analizzando il diagramma delle componenti del progetto JGuesser è possibile individuare la componente Validatore reCAPTCHA, che si interfaccerà con il corrispettivo sistema esterno. Questa componente è stata trasformata nella seguente classe:

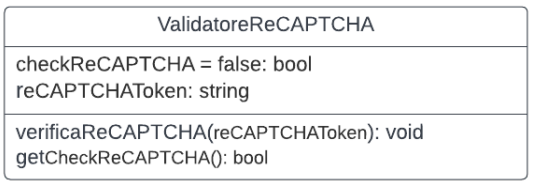

Figura 3: Classe ValidatoreReCAPTCHA

Tale classe si occuperà di ricevere i reCAPTCHAToken e di verificare attraverso il metodo verificaReCAPTCHA() la corretta esecuzione del test reCAPTCHA. Questo metodo andrà quindi a settare le variabili checkReCAPTCHA e reCAPTCHAToken, a cui sarà possibile accedere.

1.4 Registrazione e autenticazione

Osservando il diagramma delle componenti del progetto JGuesser è possibile individuare tre componenti:

- Gestore registrazione: era quella componente che si occupava di gestire la registrazione di un determinato utente all'interno del sistema. Si è deciso di creare per questa componente una classe chiamata Registrazione, che permetterà all'utente di registrarsi;
- Autenticazione: era quella che si occupava di gestire il login con le sole credenziali e si è deciso di creare una classe per tale componente;
- Autenticazione con Google: era quella che si interfacciava con il sistema esterno "Google Identity Service". Data la sua semplicità si è deciso in fase di progettazione del diagramma delle classi di far collassare tutte le funzionalità di quella classe, nella classe autenticazione, che gestisce anche il login con credenziali.

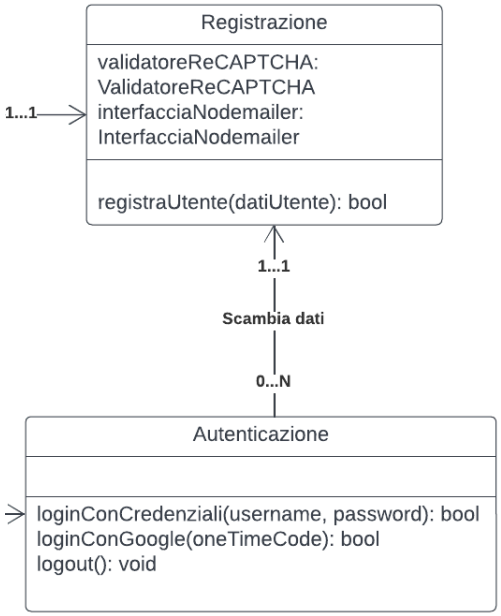

Figura 4: Classi per la registrazione e autenticazione

È stata inoltre identificata un associazione tra le due classi. Questo perchè nel caso in cui si effettui l'autenticazione utilizzando un account google, il sistema ha la necessità di registrare l'utente all'interno del database dell'applicazione, nel caso in cui non fosse gia stato fatto precedentemente. Per questo motivo la classe Autenticazione scambierà una serie d'informazioni (e-mail, token ecc...) con la classe Registrazione per registrare l'utente.

1.5 Modifica e reset credenziali

Analizzando il diagramma delle componenti del progetto JGuesser è possibile individuare la componente Gestore credenziali, che si occupava di gestire la modifica e il reset delle credenziali dell'utente. In fase di progettazione del diagramma delle classi si è scelto di spezzare questa componente in due classi distinte:

- ResetCredenziali: si occupa di gestire il solo reset delle credenziali dell'utente, nel caso in cui questo si fosse dimenticato l'username o la password. In questa classe sono presenti:
  - un istanza della classe InterfacciaNodemailer: serve ad inviare l'e-mail di reset delle credenziali.
  - un istanza della classe ValidatoreReCAPTCHA: serve a verificare che l'utente abbia cliccato il reCAPTCHA.
  - il metodo resettaCredenziali: resetterà le credenziali dell'utente e invierà l'e-mail all'utente con le nuove credenziali (utilizzando interfacciaNodemailer).
- ModificatoreCredenziali: si occupa di gestire la sola modifica dell'e-mail e la password dell'utente nel caso in cui questo sia capace di effettuare il login. In questa classe sono presenti:
  - un istanza della classe Utente: verrà popolata con i dati dell'utente che verranno mostrati a video, nella propria area personale.
  - una serie di attributi che servono a modificare l'e-mail o la password.
  - un istanza della classe InterfacciaNodemailer: serve a controllare la validità di un e-mail.
  - il metodo richiestaDatiUtente: serve a popolare l'istanza datiUtenteCorrente. Questo viene fatto effettuando un interrogazione al DB.
  - il metodo effettuaModificaE-mail: serve a modificare l'e-mail dell'utente. Restituisce 'false' nel caso in cui l'e-mail inserita non è valida.
  - il metodo effettuaModificaPassword: serve a effettuare la modifica della password. Restituisce 'false' nel caso in cui la procedura per modifcare la password non vada a buon fine.

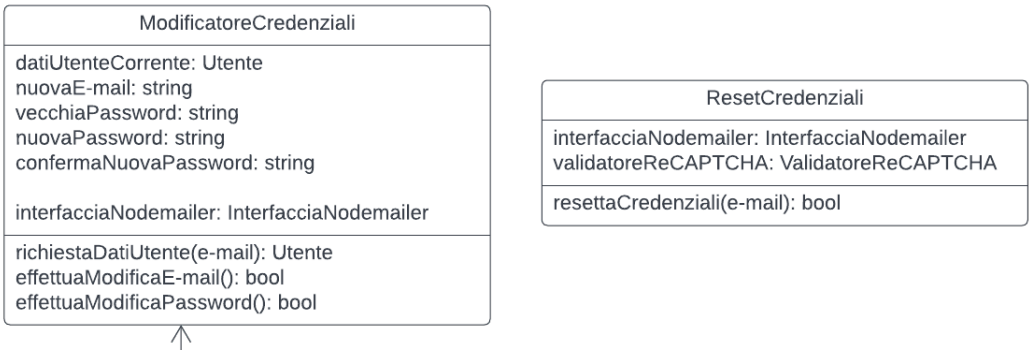

Figura 5: Classi per gestire le credenziali

1.6 Sistema di upgrade profilo

Osservando il diagramma delle componenti del progetto JGuesser è possibile individuare le componenti:

- Gestore pagamenti: si occupava di gestire i pagamenti interagendo con il sistema esterno Paypal;
- Gestore upgrade profilo: si occupava di gestire la sola modifica dei privilegi di un utente (un update al database) e l'invio di un e-mail. Data la semplicità di questa componente, in fase di progettazione si è deciso di far collassare entrambe le componenti in un unica classe chiamata GestoreUpgradePremium.

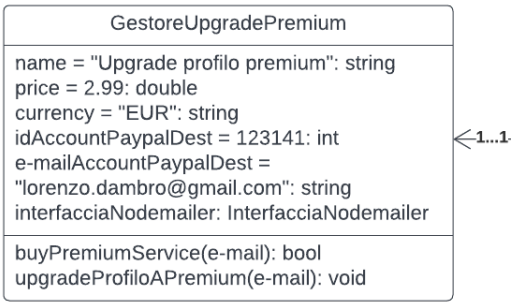

Figura 6: Classe per upgradare il profilo

Questa classe contiene come attributi tutti i dati necessari per effettuare la transazione e un istanza della classe InterfacciaNodemailer, che sarà utilizzata dal metodo upgradeProfiloAPremium per inviare un e-mail di ringranziamento per aver upgradato il proprio profilo.
Oltre a ciò sono presenti i metodi:

- buyPremiumService: effettua il pagamento interagendo con il relativo sistema esterno.
- upgradeProfiloAPremium: aggiorna i permessi del profilo a premium ed invia l'e-mail di ringraziamento.

1.7 Classifica e statistiche utente

Analizzando il diagramma delle componenti del progetto JGuesser è possibile individuare la componente Gestore dati utente. Questa componente si occupava di gestire i dati da mostrare in classifica e le statistiche del singolo utente. Date queste due differenze in fase di proggettazione si optato per spezzare questa componente in due classi:

- StatisticheUtente: si occupa di contenere tutti i dati relativi alle sole statistiche di un utente. Le statistiche di un singolo utente verranno richieste al database utilizzando il metodo richiediStatisticheUtente.
- Classifica: si occupa di contenere tutte le statistiche di tutti gli utenti che verranno visualizzati nella classifica. I dati degli utenti che verranno visualizzati in

classifica possono essere richiesti tramite il metodo richiediDatiClassifica. I dati del singolo utente che si vuole filtrare nella classifica, verranno richiesti con il metodo richiediStatisticheUtenteSpecifico.

Figura 7: Classe per la classifica

1.8 QuizMaker, Quiz e Simbolo alfabeto

Analizzando il diagramma delle componenti si può evincere la componente QuizMaker. Questa componente si interfaccia con il tipo di dato quiz, che a sua volta richiede i simboli dei vari alfabeti, per questo oltre alla classe QuizMaker sono state anche aggiunte le risorse Quiz e SimboloAlfabeto.

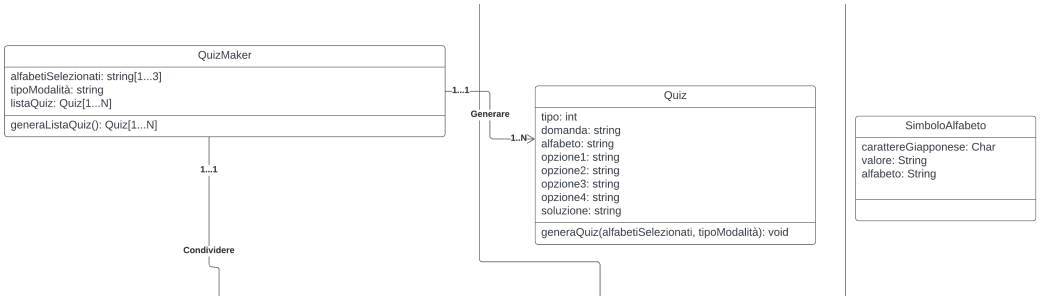

Figura 8: Gestione creazione quiz

La classe QuizMaker si occupa di generare una lista di quiz in base alle informazioni che riceve dal Front-End riguardo le scelte dell'utente, di conseguenza abbiamo la relazione 'generare' tra QuizMaker e Quiz. QuizMaker si occuperà anche di inviare i quiz al Front-End stesso per essere visualizzati e per essere a loro volta inoltrati verso la classe CorrettoreQuiz.

1.9 Correttore Quiz e Interprete Disegni

Analizzando il diagramma delle componenti si possono evincere le componenti:

- Sistema interpretazione disegno: si occupa di inviare il disegno dell'utente fatto nel quiz di tipo 4, al sistema esterno Nyckel per essere interpretato, riceve indietro una stringa che inoltra nuovamente a paginaQuiz. Questa componente è stata tradotta nella classe InterpreteDisegno, che si interfaccia all'API di Nyckel.

- Pagina quiz: riceve dalla componente quizMaker i vari quiz, li fa visualizzare in sequenza all'utente, stabilisce se le risposte di quest'ultimo sono corrette e, nel caso di una sfida online, ordina al Gestore sessione online di far terminare la sfida. la parte di questa componente che si occupa solamente di riceve il quiz,

e di verificare la risposta dell'utente è stata tradotta nel diagramma delle classi nella classe CorrettoreQuiz.

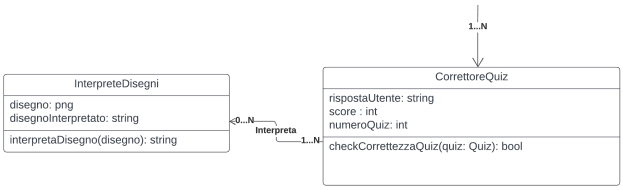

Figura 9: Correttore quiz

Inoltre è stata identificata una relazione tra le due classi. Questo perchè nel caso il Correttore quiz debba correggere la risposta ad un quiz di tipo 4 avrà bisogno di interpretare il disegno fornito dall'utente tramite l'Interprete disegno.

1.10 Sfida Giornaliera

Si è ritenuto oppurtuno scomporre la componente Pagina quiz del diagramma delle componenti in più classi, una di queste è la classe SfidaGiornaliera.

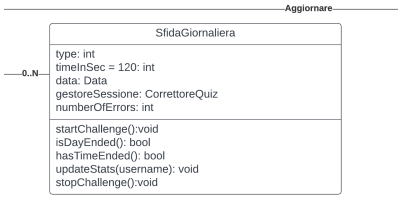

Figura 10: Classe sfida giornaliera

Questa componente si occupa di correggere le risposte ai quiz della sfida giornaliera di un singolo utente, e di aggiornare le sue statistiche una volta finita la sfida. Per questa ragione la classe istanzia un oggetto di tipo CorrettoreQuiz. L'operazione isDayEnded si occupa di verificare che non sia stata superata la data collegata alla sfida giornaliera che sta visualizzando, in tal caso segna la sfida come persa a priori e aggiorna le statistiche degli utenti di conseguenza.

1.11 Sistema di ricerca giocatori, gestione giocatori e inviti

Analizzando il diagramma delle componenti si possono trovare le seguenti componenti:

- Sistema di ricerca giocatori: si occupa di distinguere il tipo di sfida online l'utente vuole fare (casuale o sfidando un giocatore specifico). Se viene inserito uno username verifica la sua validità interfacciandosi con il Database. Nel diagramma delle classi è stato tradotto nella classe omonima. Quest'ultima si occupa anche di istanziare il GestoreGiocatoriDisponibi o il GestoreInivitiSfida a seconda del tipo di ricerca viene effettuata.

- Gestore invio inviti di sfida: si occupa di inviare una richiesta di invio invito, questa componente è stata trasformata nella classe GestoreInvitiSfida.
- Notifica di sfida: si occupa di inviare l'invito alla sfida all'utente e di inoltrare la risposta di quest'ultimo alla componente pagina multiplayer, questa componente è stata inglobata dalla classe GestoreInvitiSfida che nel suo complesso si occupa di inviare una richiesta di invito al Front-End e di ricevere la risposta dell'utente, nel caso essa sia positiva inoltra verso la classe GestioneSessioneMultiplayer gli username dei due utenti.
- Gestore giocatori disponibii: si occupa di aggiornare lo stato dell'utente, e di scegliere un utente casuale tra quelli disponibili per iniziare la sessione di sfida, nel diagramma delle classi diventa la classe GestoreGiocatoriDisponibili che si interfaccia all'API di mongoDB.

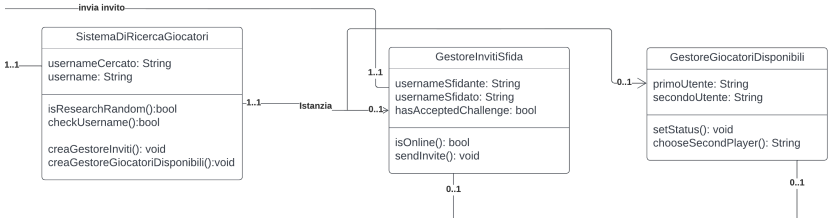

Figura 11: Gestore ricerca, inviti e disponibilità

Sono state individuate anche delle relazioni tra le tre classi, infatti il Sistema di ricerca giocatori è in relazione sia con GestoreGiocatoriDisponibili che con GestoreInvitiSfida attraverso la relazione 'istanziare'.

1.12 Gestore sessione Multiplayer

Nel diagramma delle componenti si trovano la classe pagina Multiplayer, quest'ultima si occupa di far visualizzare ai due utenti i quiz, riceve le loro risposte e stabilire i vincitori. Questa componente è stata tradotta nella classe GestoreSessioneMultiplayer per quanto riguarda il suo ruolo nel Back-End, ossia correggere i quiz, e tramite l'API di mongoDB aggioranre il punteggio dei due giocatori e le loro statistiche in base all'esito della sfida. QUesta classe istanzia due oggetti di tipo CorrettoreQuiz, uno per utente. L'operazione stopSession si occupa di far terminare la sfida online una volta che uno dei due giocatori avrà risposto a tutti i quiz della sfida.

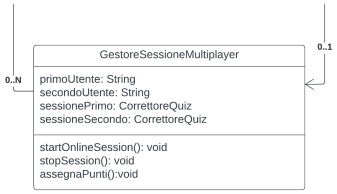

Figura 12: Classe gestione multiplayer

1.13 Diagramma delle classi complessivo

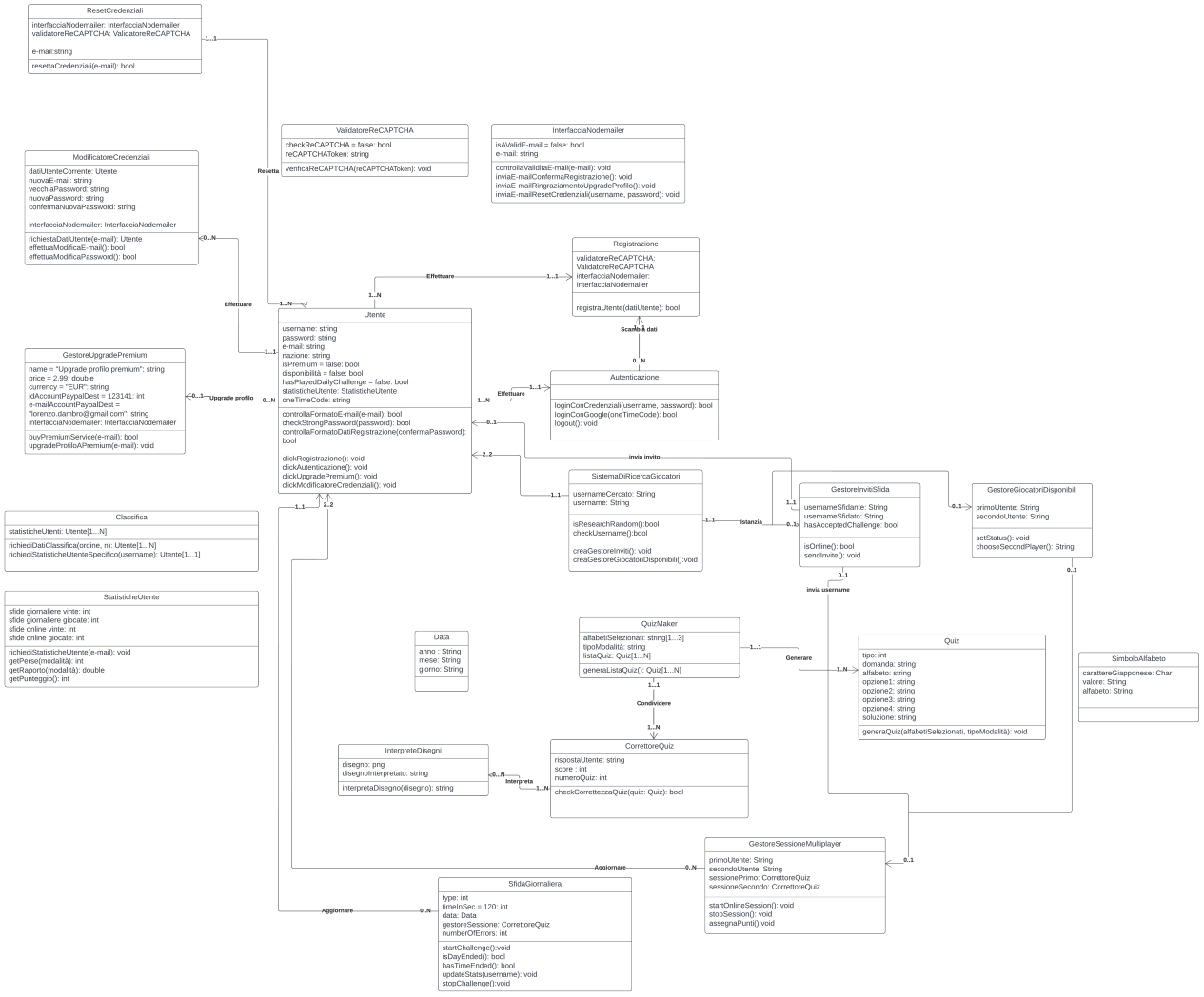

Figura 13: Diagramma classi completo

2 Codice in Object Constraint Language

In questo capitolo sono descritti in maniera formale alcuni vincoli che devono rispettare alcune classi, utilizzando il linguaggio OCL. Sono state prese le singole classi, per i quali sono state individuate delle OCL e per ognuna di esse si è proceduto alla descrizione delle OCL.

2.1 Sistema d'invio e-mail

Nella classe InterfacciaNodemailer sono stati individuati tre metodi che inviano e-mail:

- inviaE-mailConfermaRegistrazione;
- inviaE-mailResetCredenziali;
- inviaE-mailRingraziamentoUpgradeProfilo.

Per inviare un e-mail è necessario, che questa sia valida e quindi devono valere le seguenti pre-condizioni:

context InterfacciaNodemailer::inviaE-mailConfermaRegistrazione()
pre: interfacciaNodemailer.isAValidE-mail=true

context InterfacciaNodemailer::inviaE-mailResetCredenziali(username)
pre: interfacciaNodemailer.isAValidE-mail=true

context InterfacciaNodemailer::inviaE-mailRingraziamentoUpgradeProfilo()
pre: interfacciaNodemailer.isAValidE-mail=true

2.2 Registrazione e autenticazione

Prima di registrarsi devono valere due condizioni:

1. che l'utente abbia premuto il reCAPTCHA;
2. che l'e-mail inserita dall'utente sia valida.

Deve valere quindi la seguente condizione:

context Registrazione::registraUtente(datiUtente)
pre: (interfacciaNodemailer.isAValidE-mail=true) AND
(validatoreReCAPTCHA.checkReCAPTCHA=true)

Inoltre si deve gestire la disponibilità del giocatore per gli inviti online e per questo motivo devono valere:

context Autenticazione::loginConCredenziali(username, password)
post: Utente.disponibilita=true

context Autenticazione::loginConGoogle(oneTimeCode)
post: Utente.disponibilita=true

context Autenticazione::logout()
post: Utente.disponibilita=false

2.3 Reset delle credenziali

Per resettare le proprie credenziali è necessario che l'utente abbia cliccato sul reCAPTCHA, quindi deve valere la condizione:

context ResetCredenziali::resettaCredenziali(e-mail)
pre: validatoreReCAPTCHA.checkReCAPTCHA=true

Quello che questo metodo fa è quello di resettare la password dell'utente, con una casuale. Per questo motivo deve valere la post condizione:

context ModificatoreCredenziali::resettaCredenziali(e-mail)
post: Utente.password=Random(password)

2.4 Modifica delle credenziali

Nella classe modificatore credenziali, sono riportati i due metodi:

- effettuaModificaE-mail;
- effettuaModificaPassword.

Devono valere le seguenti post-condizioni:

context ModificatoreCredenziali::effettuaModificaE-mail()
post: Utente.e-mail=nuovaE-mail

context ModificatoreCredenziali::effettuaModificaPassword()
post: Utente.password=nuovaPassword

2.5 Upgrade del profilo a premium

Nella classe GestoreUpgradePremium oltre al metodo buyPremiumService(e-mail) è presente il metodo upgradeProfiloAPremium(e-mail), che si occupa di aggiornare il livello di privilegio dell'utente che ha fatto l'upgrade e di inviarli un e-mail di ringraziamento. Per questo metodo deve valere la seguente post-condizione:

context GestoreUpgradePremium::upgradeProfiloAPremium(e-mail)
post: Utente.isPremium=true

2.6 Quiz

Esistono solamente 4 tipi di quiz, quindi deve valere la condizione:

context Quiz inv: type $\geq$ 1 AND type $\leq$ 4

Esistono solo tre alfabeti diversi, di conseguenza vale la condizione:

Context Quiz inv: alfabeto = 'Hiragana' OR alfabeto = 'Katakana' OR alfabeto = 'Kanji'

La soluzione del quiz deve essere una delle opzioni che l'utente può selezionare, di

conseguenza vale:

Context Quiz inv: soluzione = opzione1 OR soluzione = opzione2 OR soluzione = opzione3 OR soluzione = opzione4

Il testo della domanda di un quiz deve essere o il carattere giapponese di uno dei simboli o il suo significato/pronuncia:

Context Quiz inv: self.domanda = SimboloAlfabeto.carattereGiapponese OR SimboloAlfabeto.valore

2.7 QuizMaker

Ogni elemento del vettore alfabetiSelezionati deve essere uno tra gli alfabeti possibili dei quiz, quindi deve valere la condizione:

Context QuizMaker inv: self.alfabetiSelezionati[n] = Quiz.alfabeto

Esistono solo tre tipi di modalità giocabili, di conseguenza vale la seguente condizione:

Context QuizMaker inv: tipoModalita = 'training' OR tipoModalita = 'daily-challenge' OR tipoModalita = 'multiplayer'

2.8 CorrettoreQuiz

La risposta dell'utente deve essere una delle opzioni messa a disposizione dal sistema, di conseguenza vale:

Context CorrettoreQuiz inv: rispostaUtente = Quiz.opzione1 OR rispostaUtente = Quiz.opzione2 OR rispostaUtente = Quiz.opzione3 OR rispostaUtente = Quiz.opzone4

Dopo l'operazione checkCorrettezzaQuiz(quiz), il numero del quiz deve essere necessariamente incrementato:

Context CorrettoreQuiz:: checkCorrettezzaQuiz(quiz)
post: numeroQuiz += 1

Il numero del quiz non può che essere un numero positivo, quindi vale la condizione:

Context CorrettoreQuiz inv: numeroQuiz > 0

2.9 InterpreteDisegni

L'interpretazione del disegno deve essere o un carattere giapponese della risorsa simboloAlfabeto oppure il suo significato/pronuncia:

Context InterpreteDisegni inv: self.disegnoInterpretato = SimboloAlfabeto.valore OR self.disegnoInterpretato = SimboloAlfabeto.carattereGiapponese

2.10 SfidaGiornaliera

Esistono solamente due tipi di sfida giornaliera, quindi vale la condizione:

Context SfidaGiornaliera inv: type = 1 OR type = 2

L'operazione hasTimeEnded() viene eseguita solo ed esclusivamente nella sfida giornaliera di tipo 1:

Context SfidaGiornaliera:: hasTimeEnded()
pre: type = 1

Context SfidaGiornaliera inv: numberOfErrors $\geq$ 0 AND numberOfErrors $\leq$ 3

3 Diagramma delle classi con codice OCL

In questo capitolo è stato riportata un immagine dell'intero diagramma delle classi con anche insieme i codici OCL.

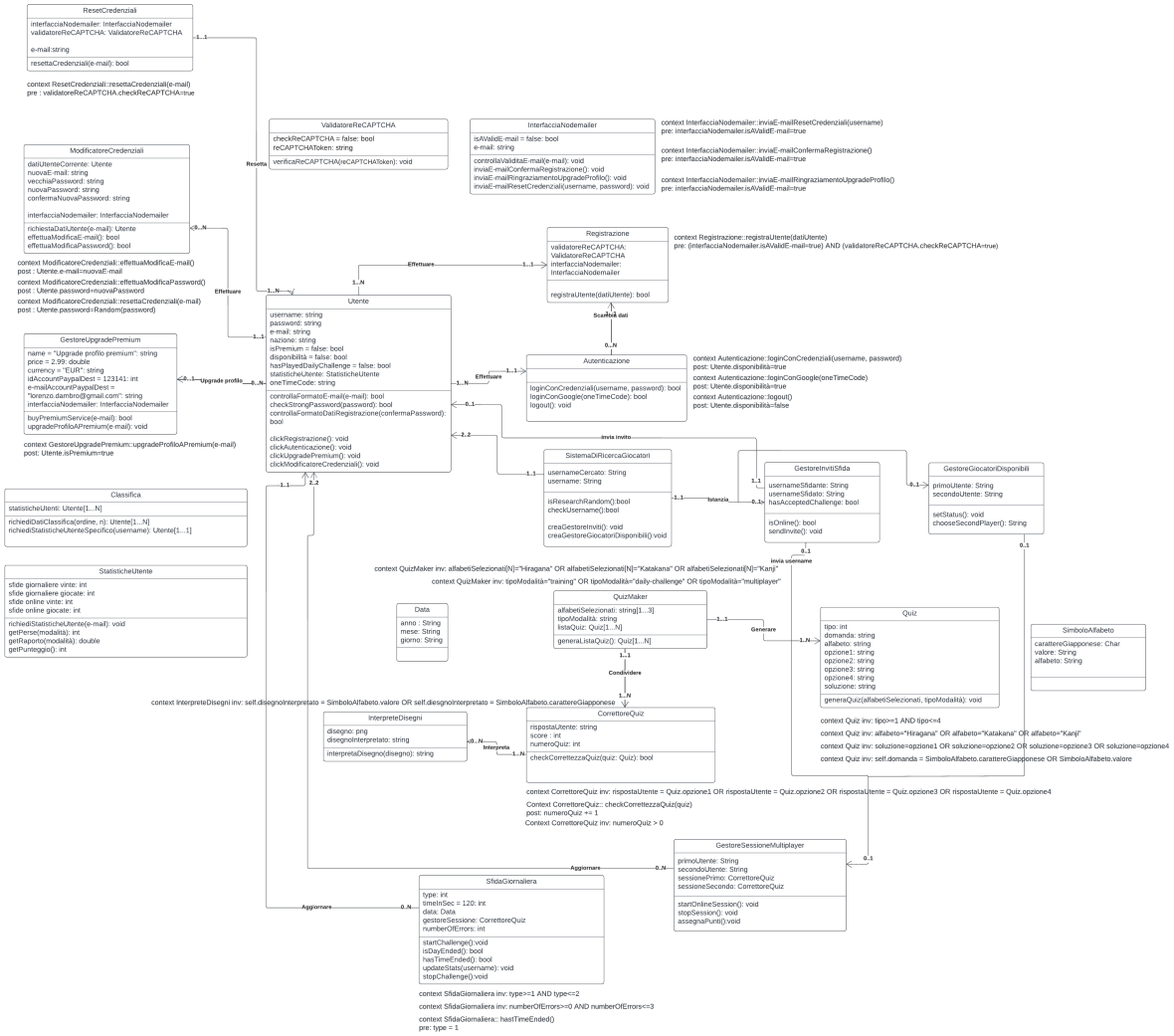

Figura 14: Diagramma delle classi complessivo piu OCL