Ruota della fortuna

PROGETTO INTERDISCIPLINARE B

ENEI STEFANO
NONALI ANDREA
SCOLARI GIANLUCA
UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

Sommario

- Svolgimento, strumenti utilizzati e indicazioni 2
  - Svolgimento
  - Strumenti utilizzati
  - Indicazioni utili
- Analisi dei requisiti 3
  - Requisiti del sistema
  - Possibili azioni degli attori
- Progettazione del Database 4
  - Analisi del documento 4
  - Progettazione concettuale 4
  - Progettazione logica 5
  - Schema logico 5
  - Traduzione 5-6
- Codifica delle classi utili 8
- Codifica del Database 9
- Progettazione del modulo ServerRDF 10
- Progettazione delle classi per l'interazione degli utenti con il server 12
- Progettazione server di gioco 18
- Progettazione e codifica delle interfacce utente per il gioco 22
- Fase di codifica e design patterns 24-26

Svolgimento progetto, strumenti utilizzati e indicazioni utili

- **Svolgimento del progetto**
  Il progetto è stato sviluppato, per la maggior parte in collaborazione con tutti i membri, mentre in minor parte venivano assegnati dei compiti da svolgere liberamente e da caricare su un server remoto tramite il software Git, al fine di ottimizzare lo scambio di informazioni e il flusso di lavoro.

- **Strumenti utilizzati**
  - JDK 13
  - Maven 3.6.3
  - Intellij-Idea Comunity Edition
  - PostgreSQL
  - Server SMTP Google
  - Repository Git su GitLab

- **Indicazioni utili**
  - È possibile, seguendo i passaggi indicati sul file README.pdf, generare la documentazione javadoc contente commenti sulle classi e sui metodi.

  - I class diagram, contenuti nel documento, potrebbero non indicare tutti i metodi esistenti del sistema, per questo motivo generando la javadoc è possibile avere una visione più completa del sistema.

Analisi dei requisiti

➢ REQUISITI DEL SISTEMA

La piattaforma di gioco Ruota della Fortuna (RdF) dovrà essere utilizzata da utenti con ruoli istinti:

- Il concorrente di gioco, che organizza e partecipa a delle partite;
- L'osservatore, che monitora l'andamento di una partita in corso osservando le mosse dei concorrenti;
- L'amministratore, che gestisce le frasi misteriose utilizzabili in gioco.

La piattaforma è composta da:

- Un modulo serverRdF, che fornisce servizi di back-end;
- Un modulo DBRdF impiegato dal server per salvare i dati del sistema;
- Un modulo playerRdF, che fornisce servizi designati per concorrenti e osservatori;
- Un modulo adminRdF, che fornisce servizi di gestione della piattaforma RdF per utenti amministratori.

➢ POSSIBILI AZIONI DEGLI UTENTI

L'utente admin, che usa il modulo adminRdF, deve poter svolgere le seguenti operazioni:

- Iscriversi o accedere al sistema con un interfaccia che si chiude dopo 10 minuti di inattività;
- visualizzare le partite organizzate per la quale non si sono ancora completate le iscrizioni dei concorrenti;
- visualizzare le partite in corso di svolgimento;
- richiedere di partecipare, come osservatore, ad una partita in corso di svolgimento, o per la quale non sono ancora completate le iscrizioni dei concorrenti;
- abbandonare, da osservatore, una partita per la quale sono ancora state chiuse le iscrizioni, oppure che è in corso di svolgimento;
- modificare i dati del proprio profilo, e richiedere per email una nuova password;
- gestire il catalogo di frasi misteriose utilizzabili nelle manche di gioco e ricevere un email in caso di mancanza di frasi ;
- analizzare statistiche di utilizzo della piattaforma di gioco.

L'utente comune, che usa il modulo playerRdF, deve poter svolgere le seguenti operazioni:

- Iscriversi o accedere al sistema con un interfaccia che si chiude dopo 10 minuti di inattività;
- organizzare una nuova partita;
- visualizzare le partite organizzate per la quale non si sono ancora chiuse le iscrizioni dei concorrenti;
- visualizzare le partite in corso di svolgimento;
- richiedere la partecipazione, come concorrente, ad una partita organizzata da altri utenti per la quale non sono ancora state chiuse le iscrizioni (dei concorrenti);
- richiedere la partecipazione, come osservatore, ad una partita per la quale non sono ancora chiuse le iscrizioni (dei concorrenti), o che è in corso di svolgimento;
- abbandonare, da concorrente, o da osservatore, una partita per la quale non sono ancora state chiuse le iscrizioni, oppure che è in corso di svolgimento;
- modificare i dati del proprio profilo, e richiedere per email una nuova password.

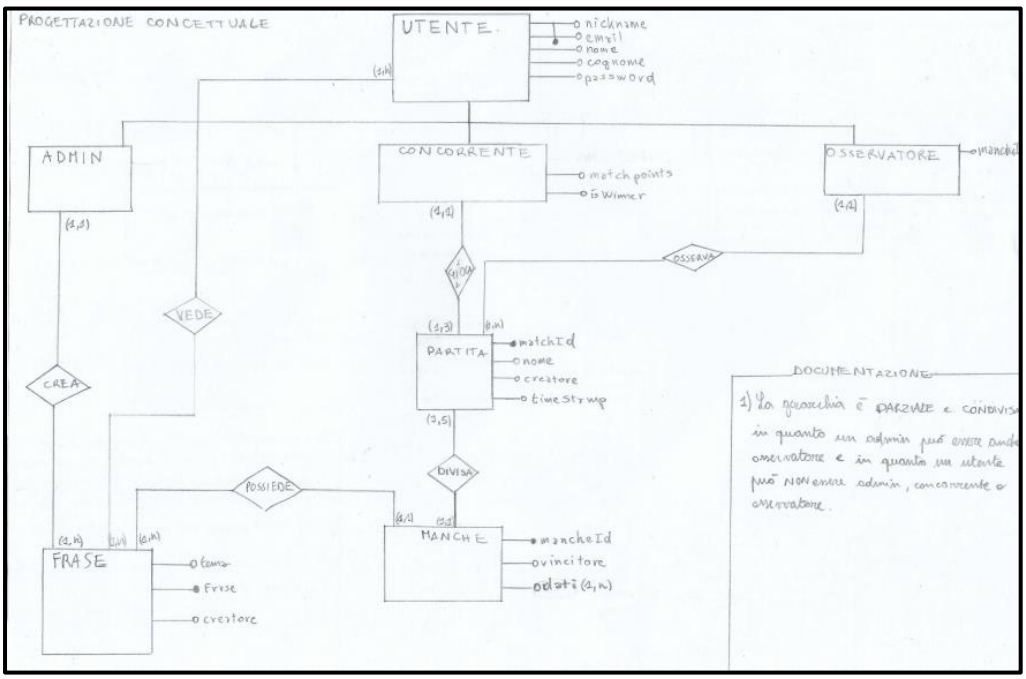
- analizzare le statistiche di utilizzo della piattaforma di gioco

Progettazione del Database

1. **Come primo passo di sviluppo è stato prodotto un apposito documento di analisi dei requisiti:**

> **Analisi dei requisiti**
>
> Si vuole realizzare un database per la gestione del gioco "Ruota della Fortuna" che soddisfi i seguenti requisiti:
>
> Gli utenti possono essere di vario tipo: Admin, Concorrente, Osservatore, Utente ordinario. Essi sono differenziati da un nickname e un email mai utilizzati da nessun altro, nome, cognome e password. Un osservatore e un concorrente possono osservare o giocare una partita mentre un admin può solamente osservare oppure gestire il catalogo delle frasi.
>
> Il database deve contenere delle frasei aggiunte dagli admin che sono contraddistinte da un tema e da una frase di massimo 60 caratteri. Ogni manche possiederà una frase e i concorrenti ed osservatori presenti ad una manche non potranno successivamente giocare una partita con la stessa frase vista.
>
> Il database deve tenere traccia di tutte le partite create e deve distinguerle tramite un identificativo unico un nome un creatore e un timestamp di creazione. Ogni partita sarà divisa in massimo cinque manche contraddistinte anch'esse da un identificativo unico, da un vincitore e da dei dati come ad esempio mossa fatta, punti acquisiti e punti persi. I dati raccolti su partita e manche saranno utilizzati per mostrare delle statistiche ai giocatori.

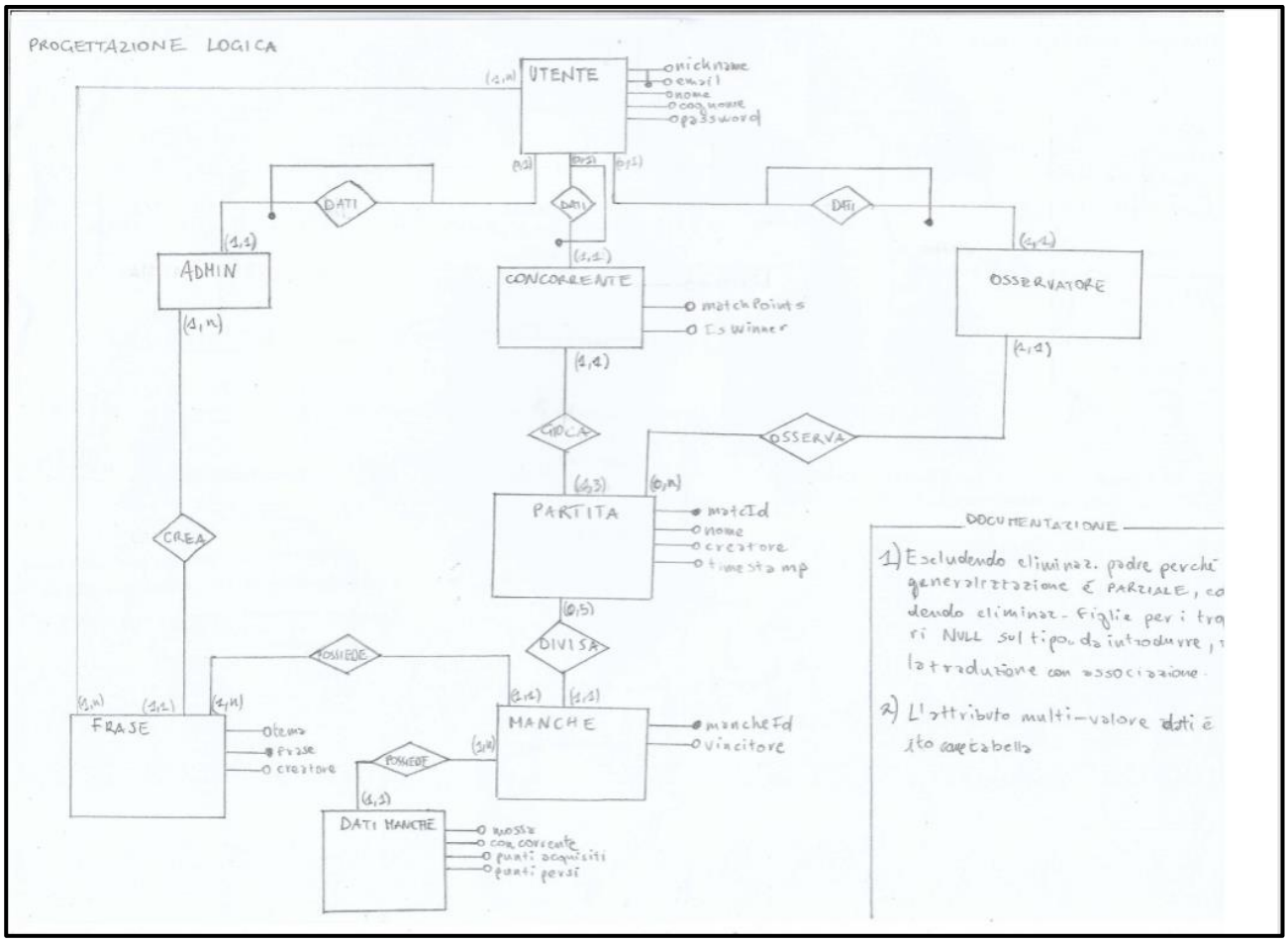
2. **Da questo documento è stato derivato lo schema di <u>progettazione concettuale</u>:**



Come già indicato nella documentazione di questo diagramma la gerarchia è:

- ➔ **<u>Parziale,</u>** in quanto un admin può essere anche osservatore .
- ➔ **<u>Condivisa,</u>** in quanto un utente può non essere osservatore, concorrente o admin.

3. **Di seguito abbiamo sviluppato lo schema di <u>progettazione logica</u> a:**



Le scelte effettuate in questa fase sono:

Gerarchia:

- Traduzione tramite associazioni, poiché, essendo la generalizzazione parziale, non è possibile eliminare l'entità padre come non è possibile eliminare le entità figlie poichè si otterrebbero troppi valori *null* finalizzati a discriminare di che tipo è l'utente.

Attributo multi-valore

- L'attributo multi-valore "*DATI MANCHE"* è stato ricostruito come tabella. Esso avrà l'attributo mossa per determinare la mossa, quello concorrente per determinare il concorrente e quelli punti acquisiti e punti persi per determinare i punti guadagnati o persi tramite le mosse.

4. **Infine è stato tradotto lo schema logico:**

UTENTE (nickname, email ,nome, cognome, password)

ADMIN (nickname[UTENTE])

CONCORRENTE (nickname[UTENTE], matchId[PARTITA], mathcpoints, isWinner)

OSSERVATORE (nickname, matchId[PARTITA], mancheId[MANCHE].)

PARTITA (matchId, nome, creatore[UTENTE], timestamp)

FRASE (frase. tema, creatore[ADMIN])

MANCHE (mancheId, matchId[PARTITA], frase[FRASE], vincitore)

DATI_**MANCHE** (mancheId[MANCHE], mossa, concorrente, punti_acquisiti, punti_persi)

FRASI_VISTE (nickname[UTENTE], frase[FRASE])

*Aggiunto in fase di codifica al fine di ricostruire le informazioni per query di dati statistici

➢ **Traduzione associazioni:**

- **Utente -> Admin**, **Utente -> Concorrente**, **Utente -> Osservatore**:
  Il valore *nickname* è stato tradotto con una chiave esterna dalle tabelle Admin, Concorrente, Osservatore sulla tabella Utente per evitare eventuali valori null nella tabella Utente, in quanto uno o più utenti possono non essere Admin, Concorrente o Osservatore.

- **Concorrente -> Partita, Osservatore -> Partita**:
  Il valore *mancheId* è stato tradotto con chiave esterna da Concorrente e Osservatore sulla tabella Partita al fine di evitare di ripetere la tabella Partita per ogni concorrente o osservatore che si aggiunge.

- **Osservatore -> Manche** aggiunta in fase di codifica.
  Valore *mancheId* tradotto con chiave esterna dalla tabella Osservatore sulla tabella Manche al fine di evitare di ripetere la tabella Manche per ogni osservatore che si aggiunge.

- **Partita -> Utente**
  Valore creatore tradotto come come chiave esterna dalla tabella Utente alla tabella Partita

- **Frase -> Admin**
  Valore creatore tradotto come chiave esterna da frase su Admin per evitare di ripetere la tabella admin per ogni frase che viene creata da un admin.

- **Manche -> Partita**
  Valore *mancheId* tradotto come chiave esterna da Manche su Partita per evitare di ripetere la tabella Partita per ogni Manche che viene creata.

- **Manche -> Frase**
  Valore Frase tradotto come chiave esterna da Manche su Frase per evitare di ripetere la tabella Frase per ogni frase che viene usata in una manche.

- **Dati_Manche -> Manche**
  Valore *mancheId* tradotto come chiave esterna da Dati Manche su Manche per evitare di ripetere la tabella manche per ogni dato che viene creato da una manche.

➢ **Creazione nuova tabella:**

- **Frasi_viste**
  Tabella creata per tradurre l'associazione binaria molti a molti *"vede"* tra utente e frase. Per fare ciò il valore *nickname* della tabella *frasi_viste* è chiave esterna sul valore *nickname* della tabella *utente,* e il valore *frase* della tabella *frasi_viste* è chiave esterna sul valore *frase* della tabella *frase*.

➢ **Altre considerazioni:**

- L'attributo ***email*** è UNIQUE in quanto la mail deve essere differente per ogni utente.
- L'attributo ***isWinner*** è stato tradotto come attributo booleano.
- L'attributo ***timestamp*** è stato tradotto come valore sql timestamp without time zone.
- Il valore ***matchId*** viene calcolato a partire da 0 ed incrementato ad ogni partita aggiunta,
- Il valore ***mancheId*** viene calcolato concatenando il valore del matchId con il numero di manche in corso (da 0 a 4).

Codifica delle classi utili

1. **Classe CSVFileManager**
   Questa classe sfrutta la dipendenza dal plugin commons-csv del repository di maven e si occupa di gestire l'inserimento di file CSV che verrà effettuata dall'interfaccia di gioco dell'admin.

2. **Classe OutTimeExecLog e OutTimeExecSub**
   Queste due classi estendono la classe TimerTask di java.util. Nel metodo run(), dopo 10 minuti di inattività dell'utente, chiudono l'interfaccia grafica di login o iscrizione come richiesto.

3. **Classe SendMail**
   Questa classe utilizza il server SMTP di Google e rimodella il codice fornito su elearning. Implementa i seguenti metodi:
   - *requestRegistration* →Utenti che richiedono la registrazione.
   - *requestResetPassword* → Utenti che richiedono una nuova password perchè hanno dimenticato la precedente.
   - *adviseAdminUpdatePhraseDb* → Admin che devono essere avvisati di aggiornare il catalogo delle frasi.

4. **Classe TableCreator**
   Questa classe esegue le query richieste per la creazione del database sul server.
   Utilizza la clausola *"if not exists"* per ogni tabella, evitando errori in caso le tabelle siano già esistenti.

5. **Classe phraseLayoutManager**
   Classe che crea un pannello con un bottone per ogni lettera della frase. Inserisce il carattere '-' per ogni spazio vuoto della frase. All'inizio tutte le lettere sono nascoste e permette di rivelare se viene trovata un'occorrenza.

6. **Classe Encryption**
   Classe che genera un hash tramite la tecnica MD5 e una chiave statica con lo scopo di criptare le password salvate nel database.

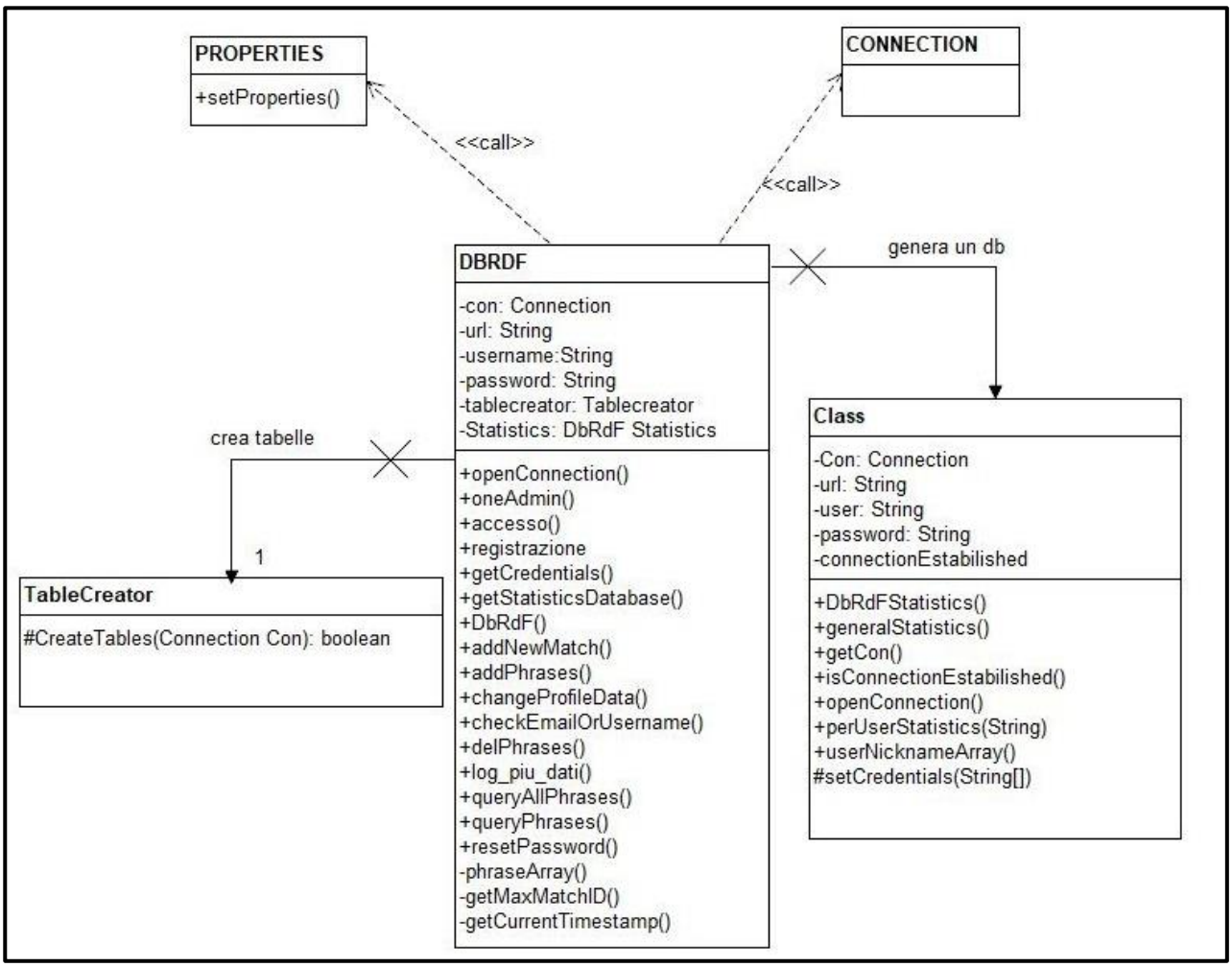
Codifica del Database

La classe DbRdF implementa tutti i metodi richiamati da quest'ultima per la gestione delle interfacce degli utenti.

I principali metodi sono:

- inserire e cercare utenti.
- modifcare i dati di un utente eccetto l'email.
- inserire, cercare ed,eliminare frasi.
- aggiungere, cercare ed eliminare nuove partite.
- gestire le query di statistica

➢ **Class Diagram**



1. **DbRdFStatistics (Class)**:
   La classe DbRdf ne tiene un istanza per andare ad eseguire tutte le query di statistiche specificate nel documento fornito. Le query si possono trovare nella cartella del progetto nel file StaticsQuery.

2. **TableCreator:**
   Specificata nel paragrafo di classi utili (Pagina 7)

Progettazione del modulo ServerRDF

➢ Gestione della concorrenza

La gestione della concorrenza si basa su dei concetti fondamentali:

1. **Client**
   Ogni client non deve poter svolgere più operazioni contemporaneamente, ma più client possono svolgere operazioni contemporanee se queste non vanno in conflitto.

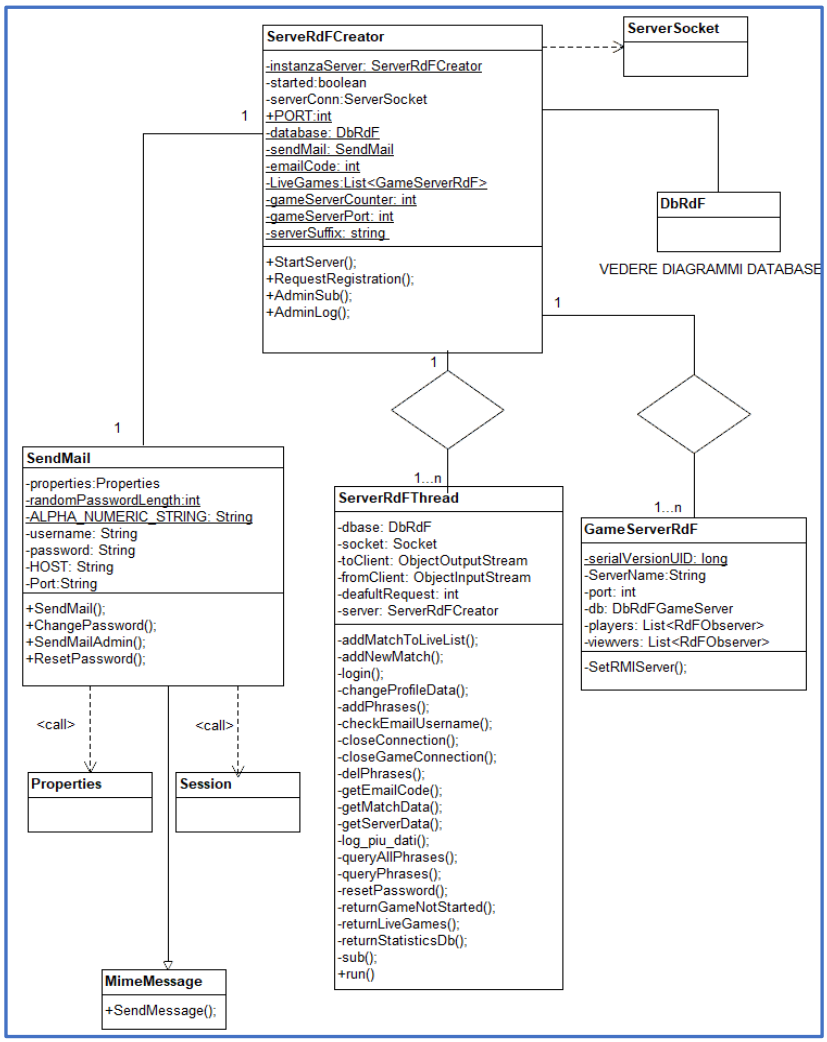
2. **Database**
   Il database PostgreSQL gode delle proprietà *ACID* (atomicità, consistenza, isolamento e persistenza) le quali, come visto nel corso di basi di dati, gestiscono le richieste concorrenti al database, sollevando il codice dalla responsabilità.

3. **Server**
   Il server, che contiene una lista di riferimenti ai server di gioco remoto, deve gestire le richieste per ottenere queste istanze in modo concorrente.

Seguendo questi punti, la tecnica di concorrenza adottata è stata quella di affidare a tutti i client connessi al server un serverThread: un thread che gestisce autonomamente un solo client, il quale però non è dotato di metodi synchronized. Questo perché ogni client sarà provvisto di un'istanza di Proxy, ovvero un proxy che invece possiede tutti i metodi synchronized.
In questo modo ogni client può fare una richiesta alla volta al proprio server thread tramite il proxy, ma più client possono fare richieste contemporanee al proprio server thread, il quale inoltrerà le richieste al server o al database. Questi ultimi avranno tutti i metodi synchronized, per evitare che richieste contemporanee inoltrate da client diversi possano andare a violare l'integrità dei dati.
Nonostante il punto 2, quindi, per DbRdf è necessario dichiarare i metodi synchronized. Infatti i metodi di DbRdF, prima di eseguire la query, vanno ad operare su strutture dati che potrebbero perdere d'integrità se un metodo viene chiamato più volte prima della terminazione.
DbRdFGameServer, il database utilizzato dal server di gioco, ha i metodi dichiarati synchronized ma non vincolati dalle azioni di DbRdF: i possibili confilitti, tuttavia, sono risolti dal punto 2.
Infine, per il punto 3, il server ha dichiarato synchronized tutti i metodi che vanno a chiedere di ottenere il nome o altre informazioni sui server di gioco.

Con questa soluzione se due client pongono una richiesta contemporanea ciascuno al proprio server thread, ad esempio una che domanda servizi posti sul server e una che domanda servizi posti sul database, queste saranno eseguite contemporamente e senza bloccarsi, proprio come richiesto dal punto 1.

➢ **Class Diagram**



1. **ServerRdFCreator:**
   Questa classe sarà la responsabile della creazione di ServerRdF e della gestione di tutte le sue operazioni, tra cui:
   - ➔ Richiesta di registrazione di un admin nel caso non sia presente
   - ➔ Richiesta di accesso a DbRdF, per permettere di creare le tabelle del database del sistema
   - ➔ Inizializzazione del server, dopo la registrazione di un admin, tramite la classe ServerSocket, che attenderà connessioni dei client sulla porta 8888.
   - ➔ Tramite la classe DbRdF costruzione istanza database e apertura della connessione al tramite il metodo *openConnection()*
   - ➔ Creazione di un serverThread ogniqualvolta un client richieda la connessione.

2. **ServerThread:**
   Ogni volta che un client richiederà una connessione al server, essà verrà gestita unicamente da un ServerThread a cui ServerRdFCreator passarà l'istanza del database e il socket di gestione della connessione.

3. **GameServerRdF:** Ogni volta che una partita è iniziata ServerRdFCreator salva in un ArrayList l'istanza della partita.

Progettazione delle classi per l'interazione degli utenti con il Server

Ogni utente è stato dotato di un menù per gestire l'interazione con il server. Per determinare i principali componenti di un menù è stato prodotto un diagramma delle classi.

Il menù ha diverse componenti in base al tipo di utente che lo possiede:

Admin, la cui interfaccia permette di:

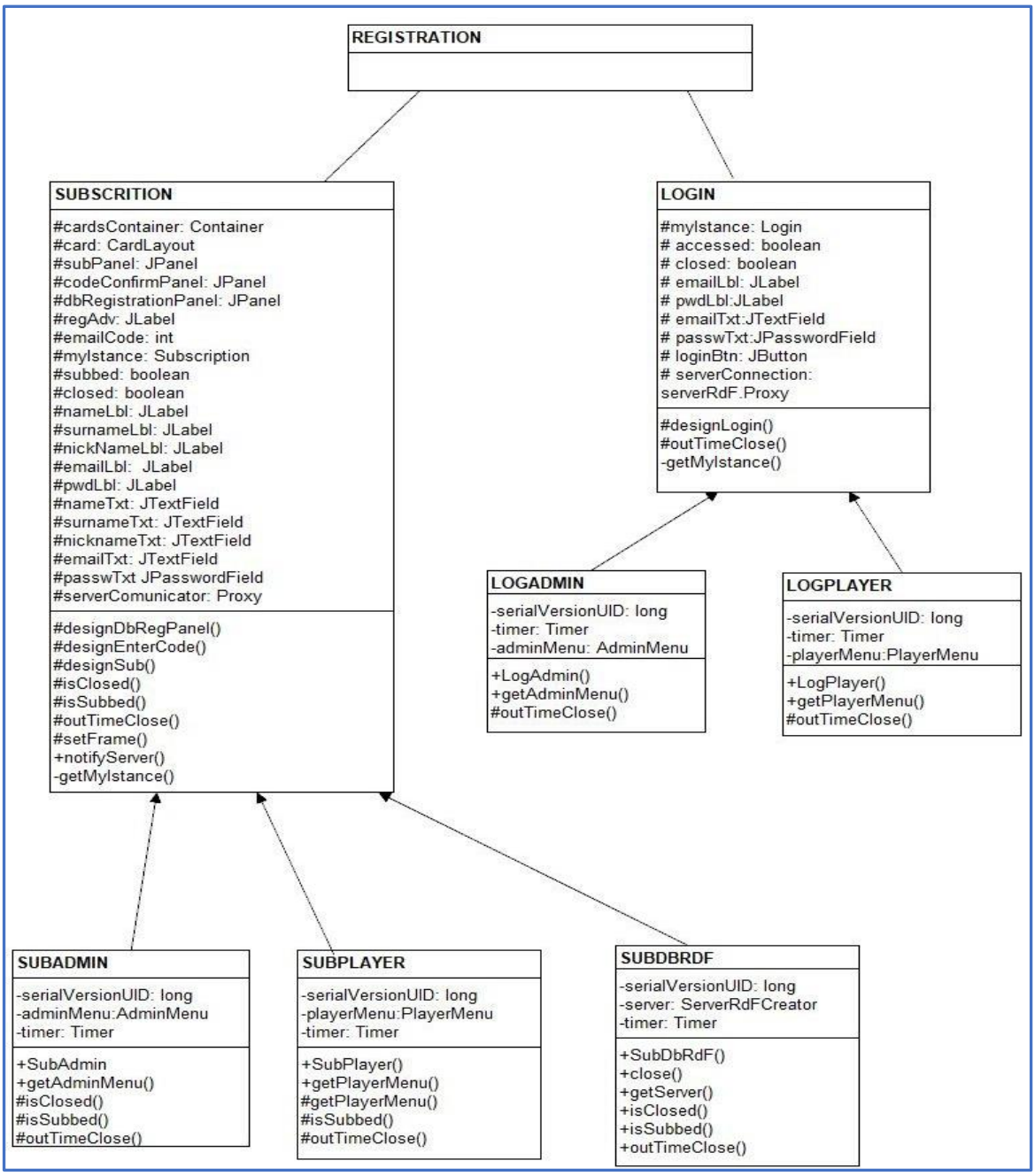
- Gestire il proprio profilo cambiando password, nome, cognome e nickname;
- Richiedere una nuova password per email;
- Partecipare da osservatore a una partita;
- Visualizzare le statistiche generali e quelle relative ad un utente;
- Gestire il catalogo delle frasi, aggiungendole ed eliminandole.

Player, la cui interfaccia permette di:

- Gestire il proprio profilo cambiando password, nome, cognome e nickname.
- Richiedere una nuova password per email;
- Partecipare da osservatore o concorrente ad una partita.
- Visualizzare le statistiche generali e relativi ad un utente.

Ogni utente, al momento della connessione con il server verrà stato dotato di un'istanza della classe Proxy. Questa classe, come già anticipato, solleverà l'interfaccia dell'utente dalla comunicazione con il server.

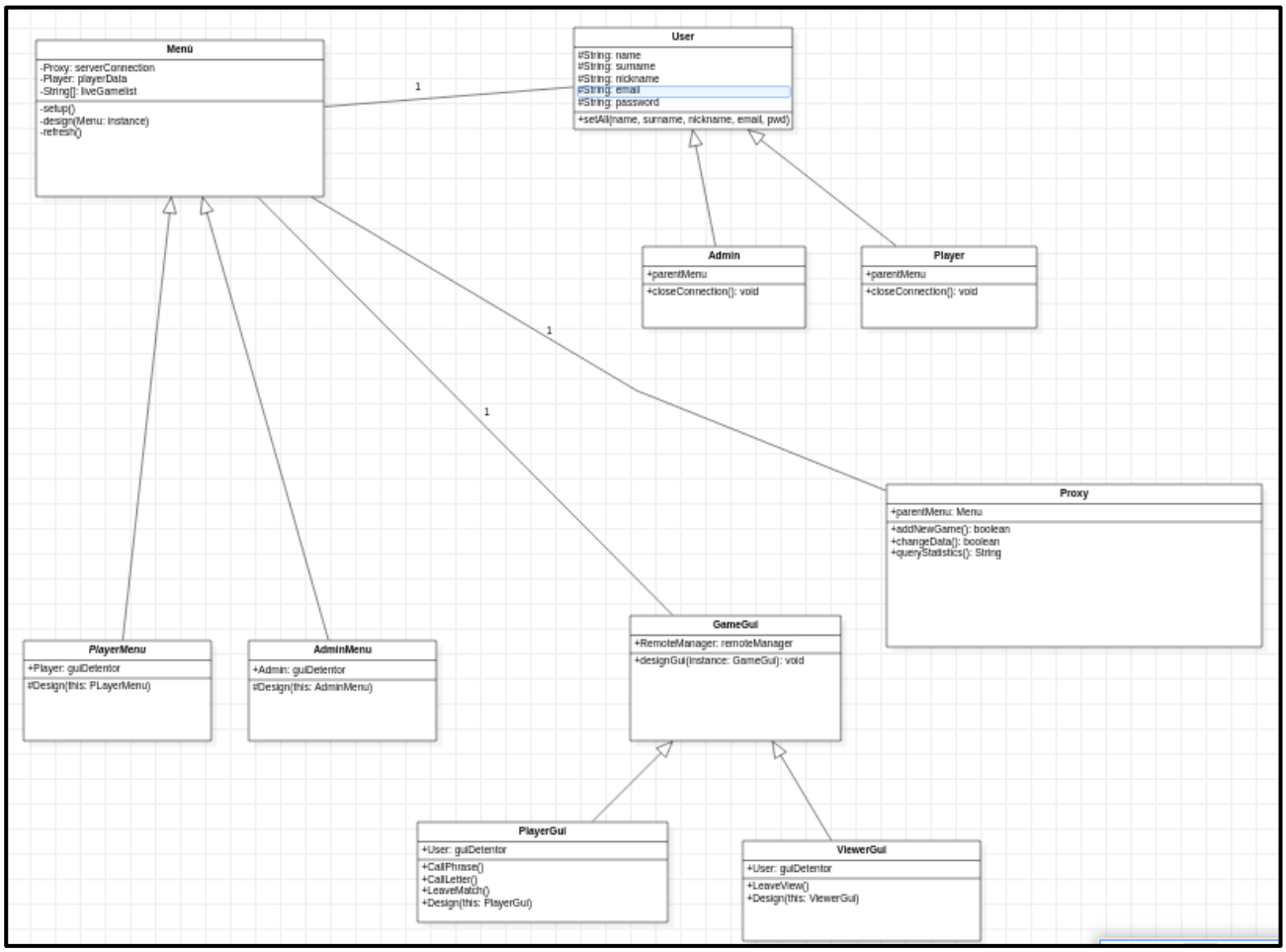
- **Class Diagram: Login e iscrizione utente**



Le classi *subscription* e *login* sono astratte e il loro metodo *design()* varia i componenti dell'interfaccia in base al tipo di utente che la istanzia.

Queste classi permettono all'utente di: registrarsi o di effettuare il login e, per gli utenti registrati, richiedere una nuova password generata in modo casuale da un insieme di caratteri, tramite email.

➢ **Class Diagram: Menù utente**



1. **Proxy:**
   Implementa il pattern proxy e svolge tutte le funzioni di comunicazione con il server per sollevare da questo compito l'interfaccia dell'utente.

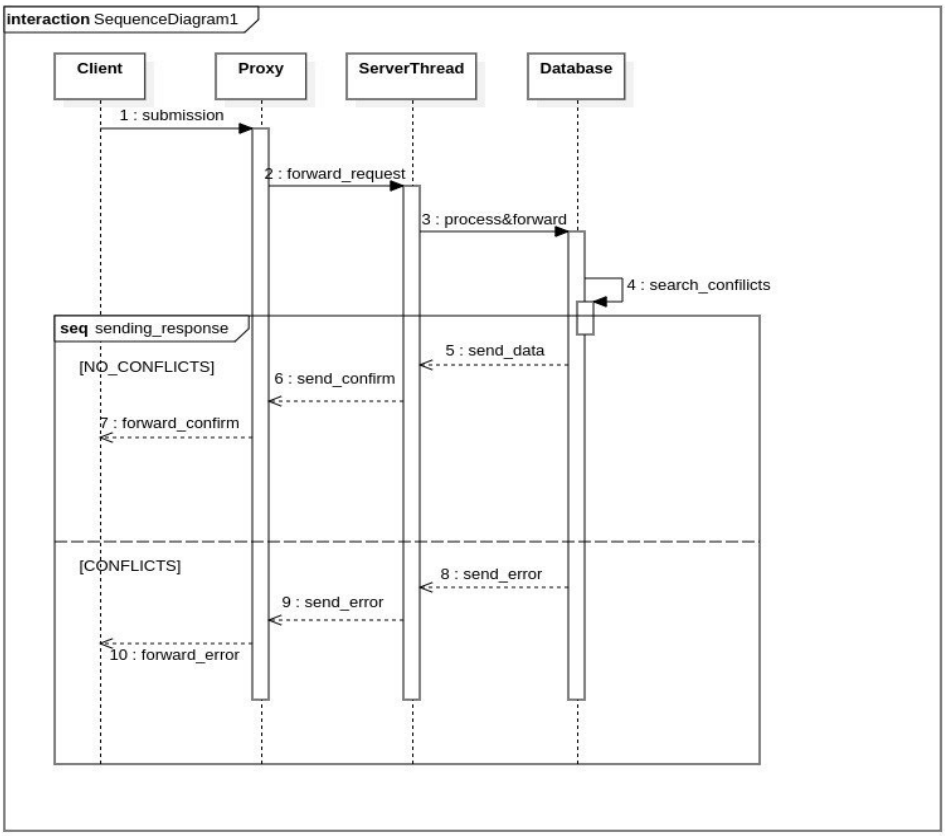
2. **GameGui:**
   Classe che gestisce l'interfaccia di gioco nel caso in cui un utente vada, tramite l'interfaccia del menu, a iniziare una nuova partita, aggiungersi ad una partita in corso oppure osservarne una.

3. **Menù:**
   Classe che gestisce il Menù degli utenti. Cambia i suoi componenti in base al tipo di utente: se l' utente è un admin il bottone per gestire le partite sarà sostituito con quello per gestire le frasi.

4. **User:**
   Struttura dati per gestire gli utenti. Un utente può essere admin o player

Al fine di chiarire maggiormente la codifica dei metodi che permettono di gestire la comunicazione tra utente e server sono stati prodotti dei diagrammi di tipo *sequence*.

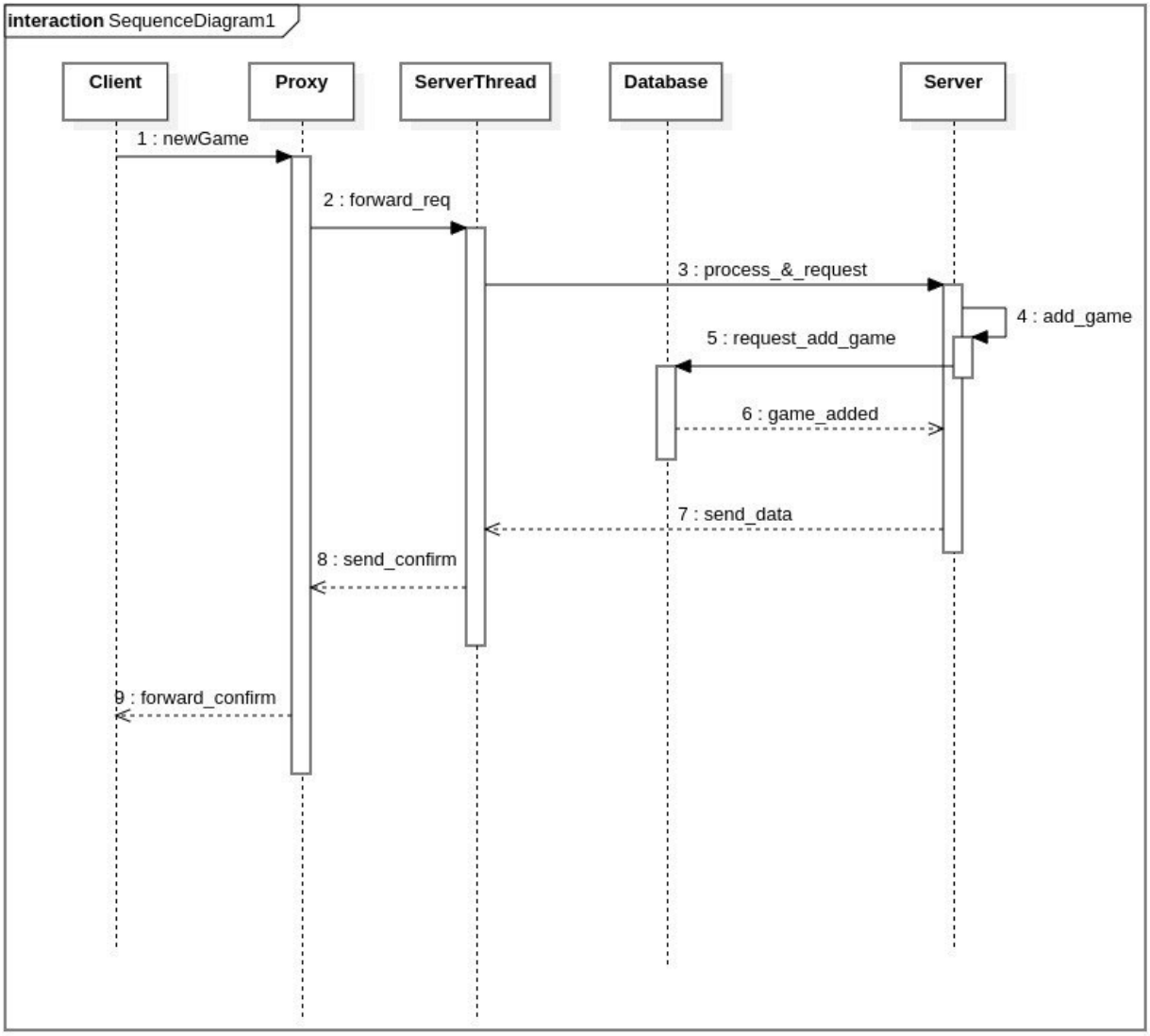
- **Sequence Diagram: registrazione**



In questo diagramma è stato definito il protocollo di comunicazione tra utente e server per la registrazione.

Come si può notare l'utente invia la richiesta di registrazione tramite il proprio Proxy, il quale inoltrerà la richiesta al ServerThread adibito a gestire il client; quest'ultimo effetturà la query tramite DbRdF per controllare se l'utente è presente nel sistema. A questo punto la risposta verrà inoltrata dal proxy all'utente e la risposta verrà visualizzata ad interfaccia.

- **Sequence Diagram: creazione partita**



Similmente al diagramma precedente, in questo viene mostrata la sequenza di creazione di una partita.

Come si può notare il client richiede tramite interfaccia grafica al proxy di creare una nuova partita. Il proxy inoltrerà la richiesta al ServerThread adibito a gestire il client e quest'ultimo invierà la richiesta alla classe ServerRdFCreator di aggiungere il gioco alla propria lista di partite attualmente disponibili; il server inoltre aggiungerà la nuova partita al database e invierà al ServerThread i dati della nuova partita, il quale a sua volta li inoltrerà tramite proxy all'utente che ha richiesto la creazione, per permettere all'interfaccia di gioco di inizializzare il suo manager remoto.

Progettazione del server di gioco

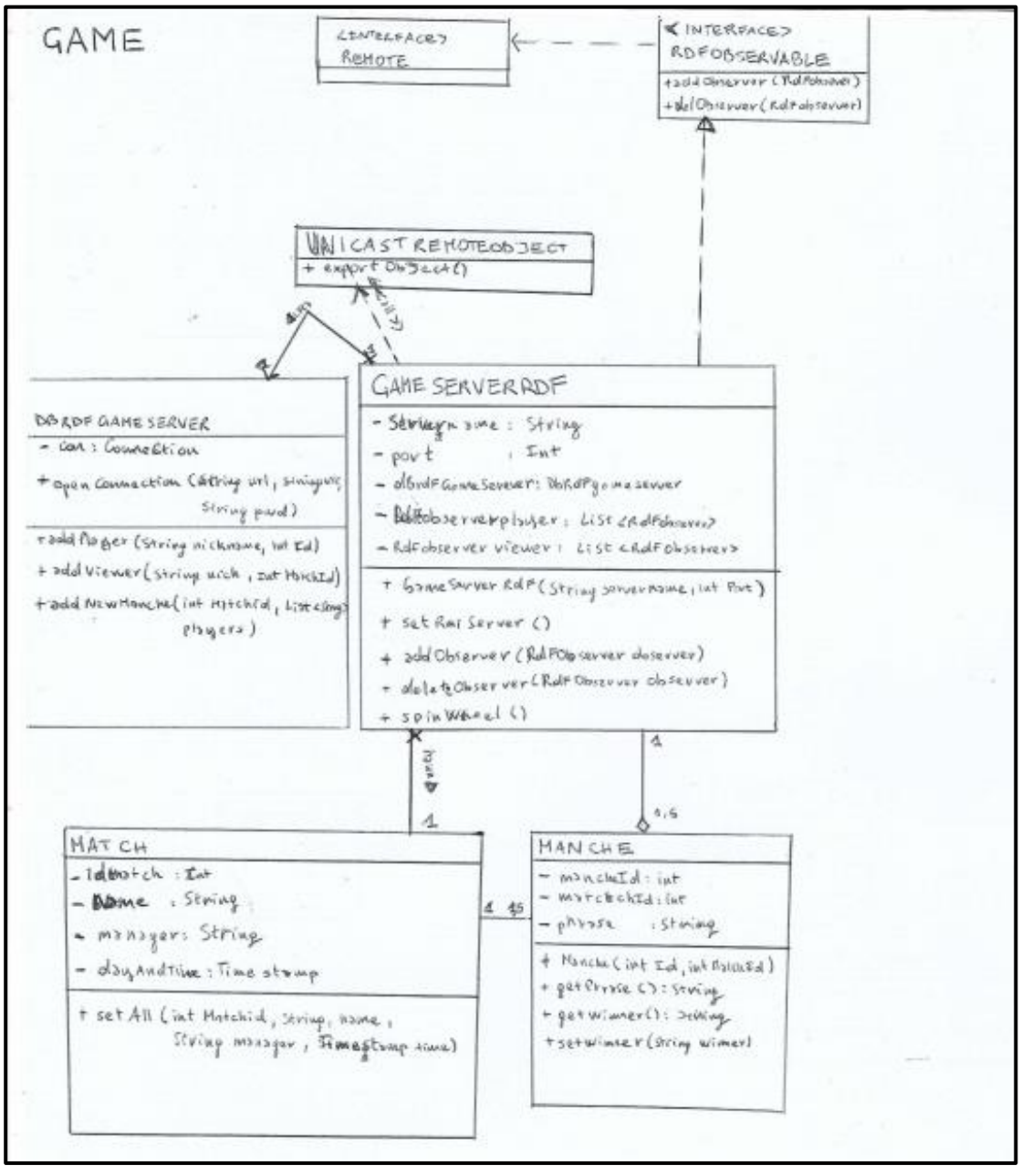
Prima di produrre i documenti è stata concordata l'idea di utilizzare la tecnologia Remote Method Invocation (RMI), fornita da Java, al fine di sollevare la stesura del codice di gioco dalle procedure di comunicazione per concentrarsi principalmente sulla codifica dei metodi di gioco.

Un' altra idea concordata precedentemente alla progettazione è stata quella di dividere i compiti di utente e server: il server deve poter gestire l'intera logica di gioco, mentre il client deve poter chiamare i metodi del server tramite interfaccia senza mai poter prendere decisioni sulla scelta delle frasi, dei turni, dei punteggi o dei modifcatori assegnabili.

Infine, è stato concordato l'utilizzo del pattern observer. (Vedere pagina 26).

A questo punto è stato prodotto il diagramma delle classi del package di gestione del gioco.

- **Class Diagram**



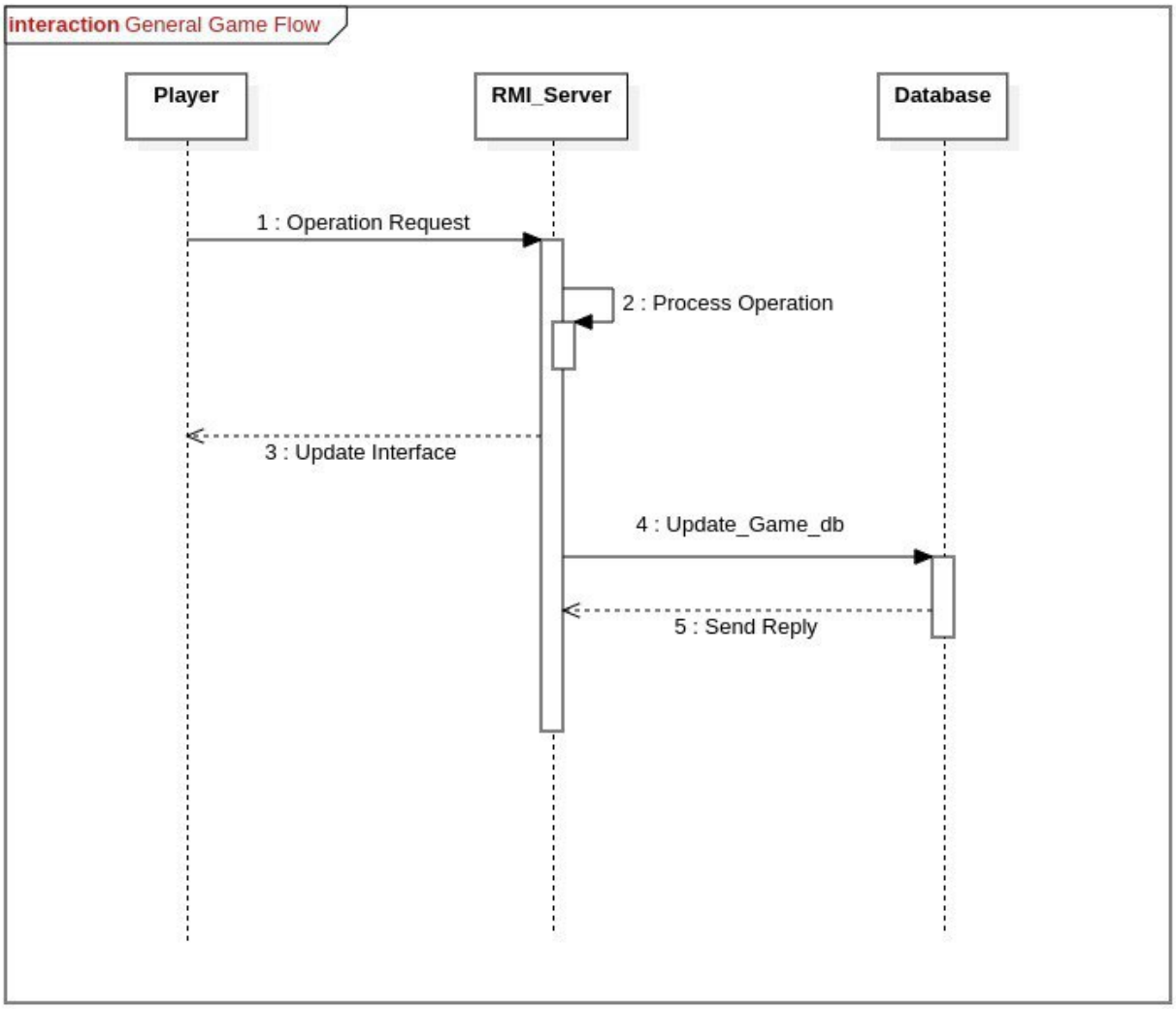
1. **GameServerRdF**:
   Classe principale di gestione della logica di un match, implementa la controparte osservabile del pattern observer tramite l'interfaccia observable da noi creata. Essa contiene tutti i metodi che un utente può chiamare sul server. GameServerRdF dovrà offrire varie funzionalità e metodi tra cui le principali:

   - *adviseAdminUpdate*() --> avvisa gli admin tramite mail in caso le frasi disponibili siano terminate.
   - *addObserver(isPlayer, instance)* → aggiunge un giocatore o osservatore alla partita e lo aggiorna dopo ogni cambiamento di stato di essa
   - *setTurn*() → metodo che sceglie ciclicamente il turno se nello stesso match o in maniera random se si inizia una nuova manche
   - *setTimer*() → metodo usato per fra partire o fermare il timer del giocatore con il turno.
   - *spinWheel*() → metodo usato dal giocatore per girare la ruota. Sceglierà un elemento random dall'array di modificatori.
   - *callLetter*() → metodo usato dai giocatori per chiamare una lettera consonante, controllando occorenze e avvisando osservatori e giocatori.
   - *callVocal*() → metodo usato dai giocatori per chiamare una lettera vocale, controllando occorrenze e avvisando osservatori e giocatori.
   - *callPhrase*() → metodo usato dai giocatori per indovinare una frase, controllando correttezza risposta e avvisando osservatori e giocatori.
   - *useJolly*() → permette al giocatore di usare un jolly nel caso ne abbia uno o più da parte.
   - *winMatch*() → La classe salva tutti i punti relativi ad una manche ed alla partita di un utente e determina alla fine della partita il vincitore, avvisando osservatori e giocatori.
2. **DbRdFGameServer**:
   Questa classe è l'istanza di un database utilizzato dal server di gioco per varie operazioni di gestione e storage dei dati, tra cui:
   - Aggiunta di un giocatore o osservatore ad una partita nel database.
   - Aggiunta della correlazione tra giocatori e frasi viste per ogni frase che appare ad un giocatore in una manche per evitare che uno stesso giocatore giochi due volte con stessa frase
   - Aggiunta di una nuova manche in una partita e storage di TUTTE le mosse effettuate dagli utenti al fine di poter fornire dati statistici.
   - Ricerca delle mail degli admin in caso serva avvisarli di aggiornare il catalogo delle frasi.
3. **Match:**
   Struttura dati utilizzata per modellare un match.
4. **Manche:**
   Struttura dati utilizzata per modellare una manche.

(Ricordiamo che questa documentazione può non essere completa e che eventuali metodi aggiunti sono stati inseriti nella javadoc, generabile leggendo le istruzioni fornite nel documento README.md)

Al fine di comprendere meglio la comunicazione tra utente e server in fase di gioco, è stato prodotto un diagramma che descrive la comunicazione generale delle informazioni:

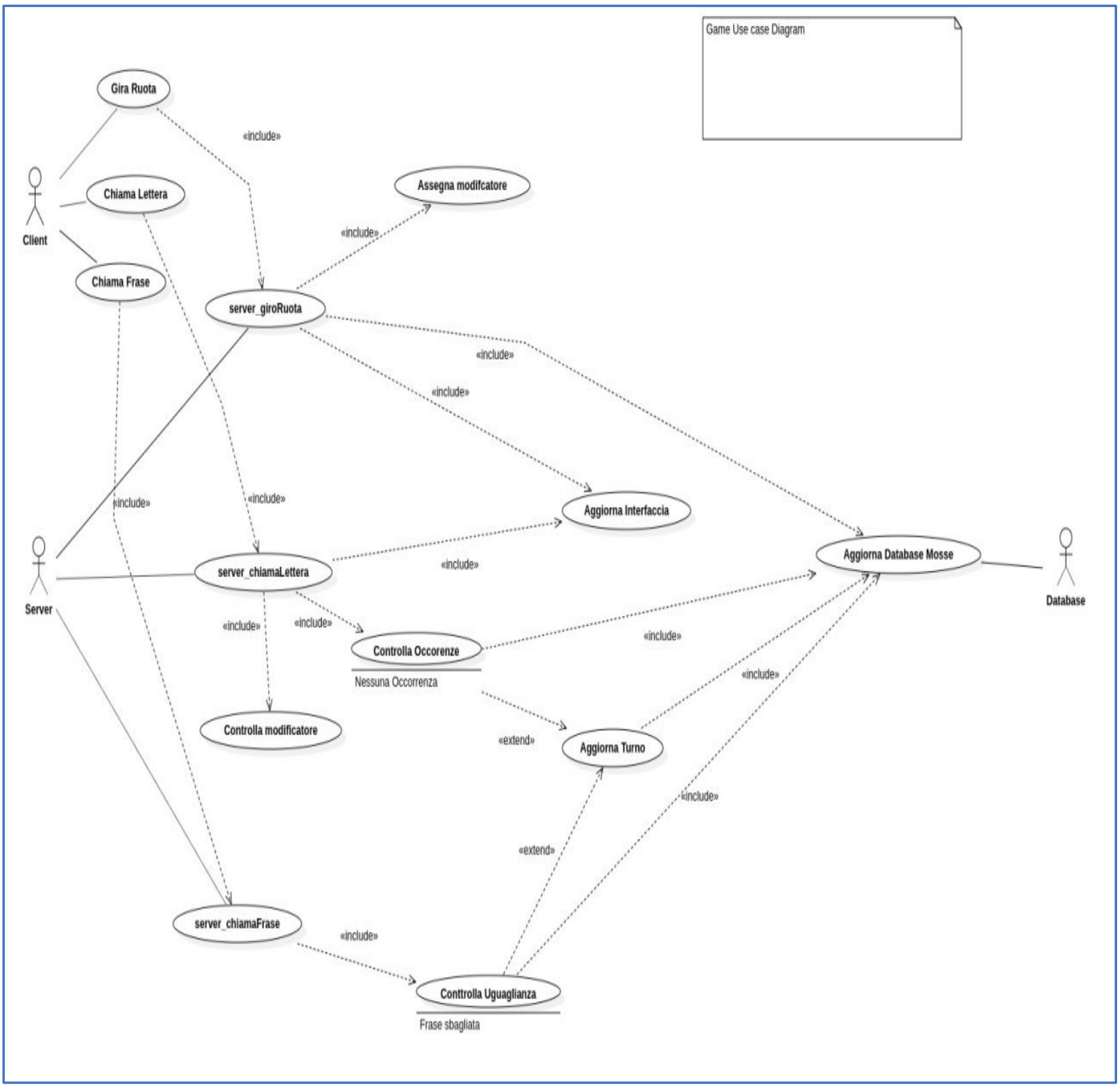
- **Sequence Diagram: comunicazione player-server di gioco**



Come illustrato, il client richiede un'operazione al server tramite RMI. Il server RMI (GameServerRdF) andrà a processare l'operazione e di seguito svolgerà due operazioni: aggiornamento dell'interfaccia dei giocatori e degli osservatori e salvataggio nel database dell'operazione richiesta dall'utente.

Al fine di comprendere meglio la gestione e la divisione dei compiti da parte del server e dell'utente è stato definito il seguente diagramma.

- **Use case Diagram: divisione compiti server – user**



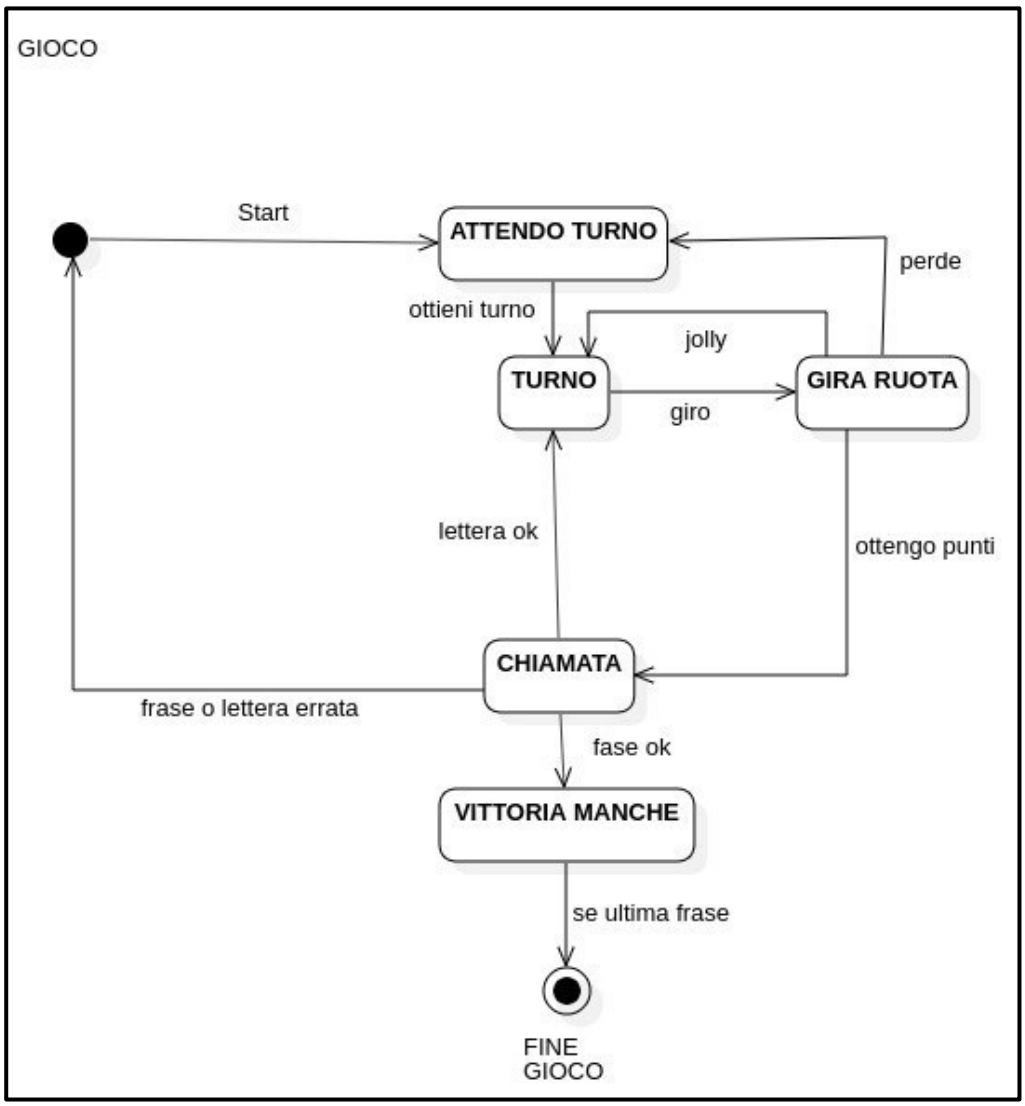
Come mostrato in questo diagramma il client può solo interagire con l'interfaccia per svolgere operazioni di richiesta al server ma non può gestire nessuna fase della logica di gioco che è interamente affidata al server, il quale inoltre salverà ogni operazione svolta dagli utenti nel database.

Progettazione delle interfacce utente per il gioco

Questo capitolo descrive la controparte observer del pattern Observer, utilizzato per gestire l'aggiornamento delle interfacce di gioco.

Al fine di gestire al meglio la gestione dell'interfaccia di gioco e le condizioni con cui far svolgere o meno le operazioni al giocatore è stato prodotto un diagramma di flusso:

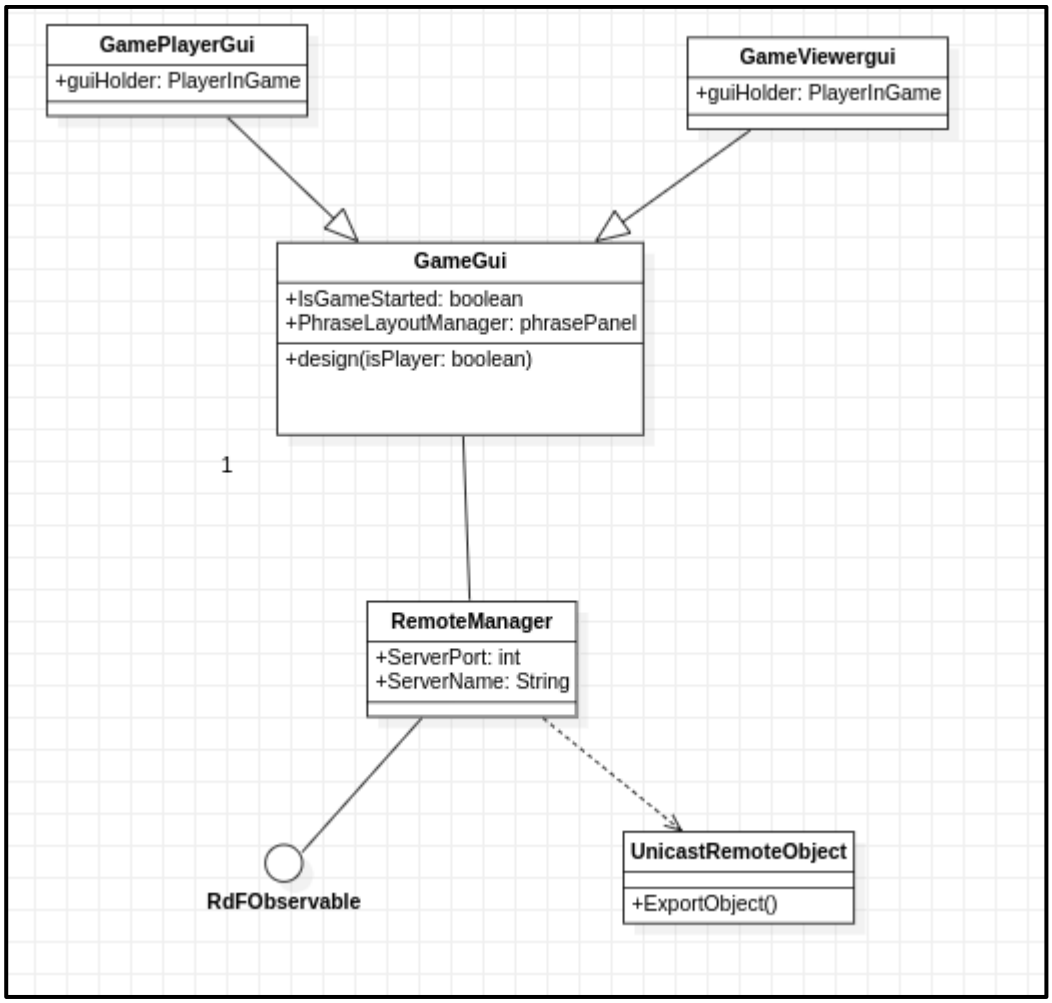
- **Flow Diagram: operazioni utente**



Un giocatore attende il suo turno e non può svolgere nessuna operazione prima che lo ottenga. Durante il suo turno può effettuare una chiamata: se essa non rispetta le condizioni allora perde il turno, altrimenti ottiene i punti guadagnati e può girare nuovamente la ruota; nel caso indovini la frase vince la manche oppure la partita.

Infine, è stato definito un diagramma delle classi per rappresentare le classi che saranno coinvolte.

- **Class Diagram:**



1. **GameGui:**
   Classe preposta al design dell'interfaccia che permette all'utente di giocare la partita e all'osservatore di osservarla. Essa modifica le sue componenti in base al tipo di utente (Giocatore o Osservatore) che ne crea un'istanza.

2. **RemoteManager**:
   Classe utilizzata dall'interfaccia di gioco per gestire la comunicazione con il server remoto. Questa infatti cercherà il server sul registry mediante il nome e la porta forniti in fase di ricerca o creazione della partita da parte di ServerRdF. Essa dovrà svolgere varie operazioni:

   - Aggiungere un giocatore o osservatore alla partita e aggiornarlo dopo ogni cambiamento di stato della partita.
   - Far partire o fermare il timer del giocatore con il turno su richiesta del server.

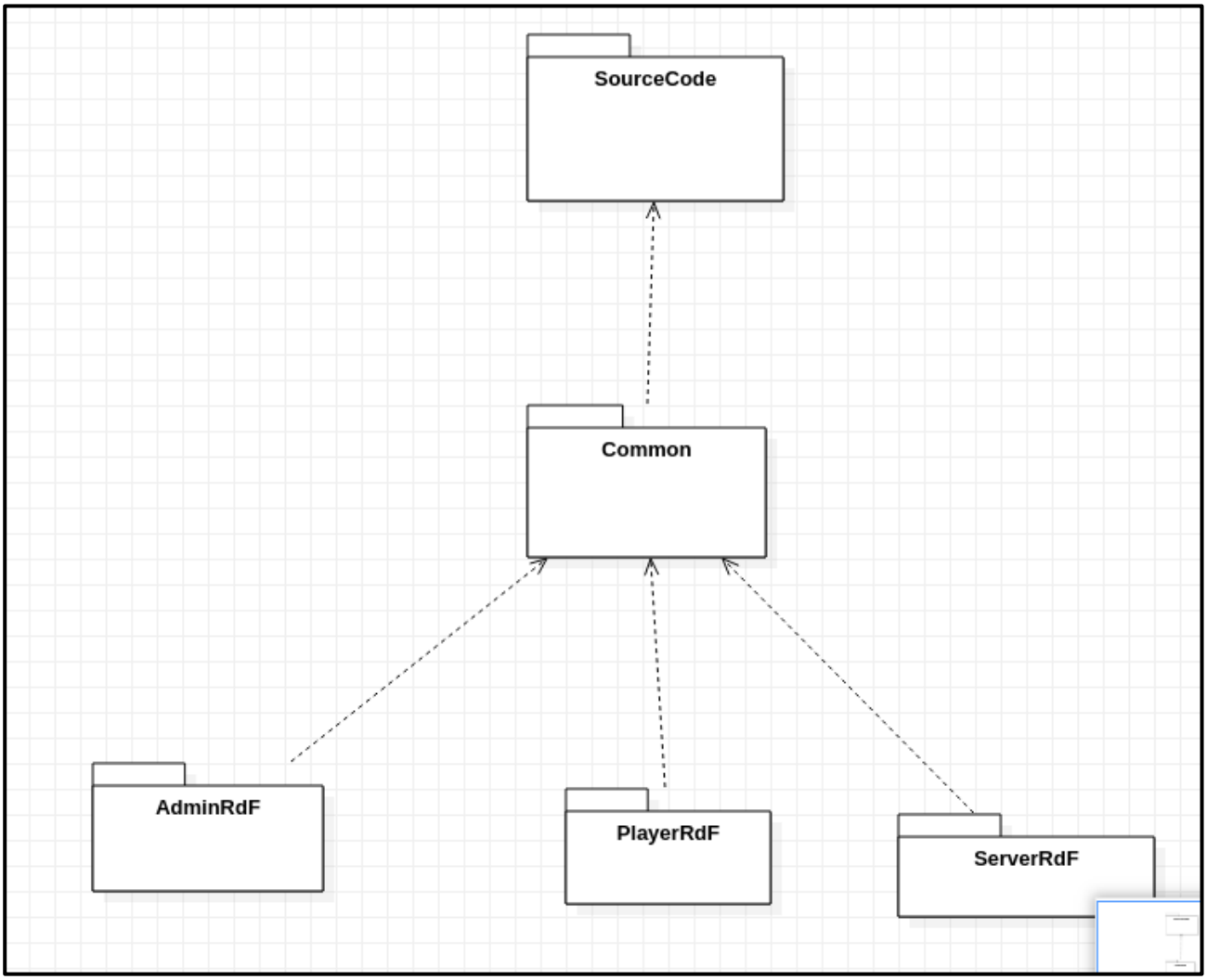
- Richiedere al server di girare la ruota.
- Richiedere al server la possibilità di chiamare una lettera consonante.
- Richiedere al server la possibilità di chiamare una lettera vocale.
- Richiedere al server la possibilità di indovinare una frase.
- Richiedere al server la possibilità di usare un jolly nel caso ne abbia uno o più da parte.

3. **RdFObserver:**
   Interfaccia utilizzata da RemoteManager che concretizza la parte "osservatore" del pattern observer da noi creato (vedere pagina 22). Essa contiene tutti i metodi che il server può chiamare sull'interfaccia dell'utente, tra cui:

   - startMatch() → inizializza l'interfaccia di gioco quando ci sono abbastanza giocatori;
   - getModifier() → il giocatore ottiene un modificatore dal server scelto in maniera random;
   - spinWheel() → permette di girare la ruota;
   - setTimer(boolean init) → permette al server di far partire o fermare il timer di gioco;
   - setJolly() → permette al server di mostrare ad interfaccia i jolly di un giocatore;
   - changePhrase() → permette al server di cambiare frase una volta terminata una manche.
   - endGame() → permette al giocatore di lasciare la partita. Se la partita era già iniziata verrà terminata e verrano avvisati tutti gli altri giocatori e osservatori con il metodo setPrasePanel();
   - setCanCallPhraseOrVocal() → permette ad un utente di chiamare una vocale o una frase solo sotto certe condizioni.

Fase di codifica

- **Struttura Maven del progetto: i moduli**



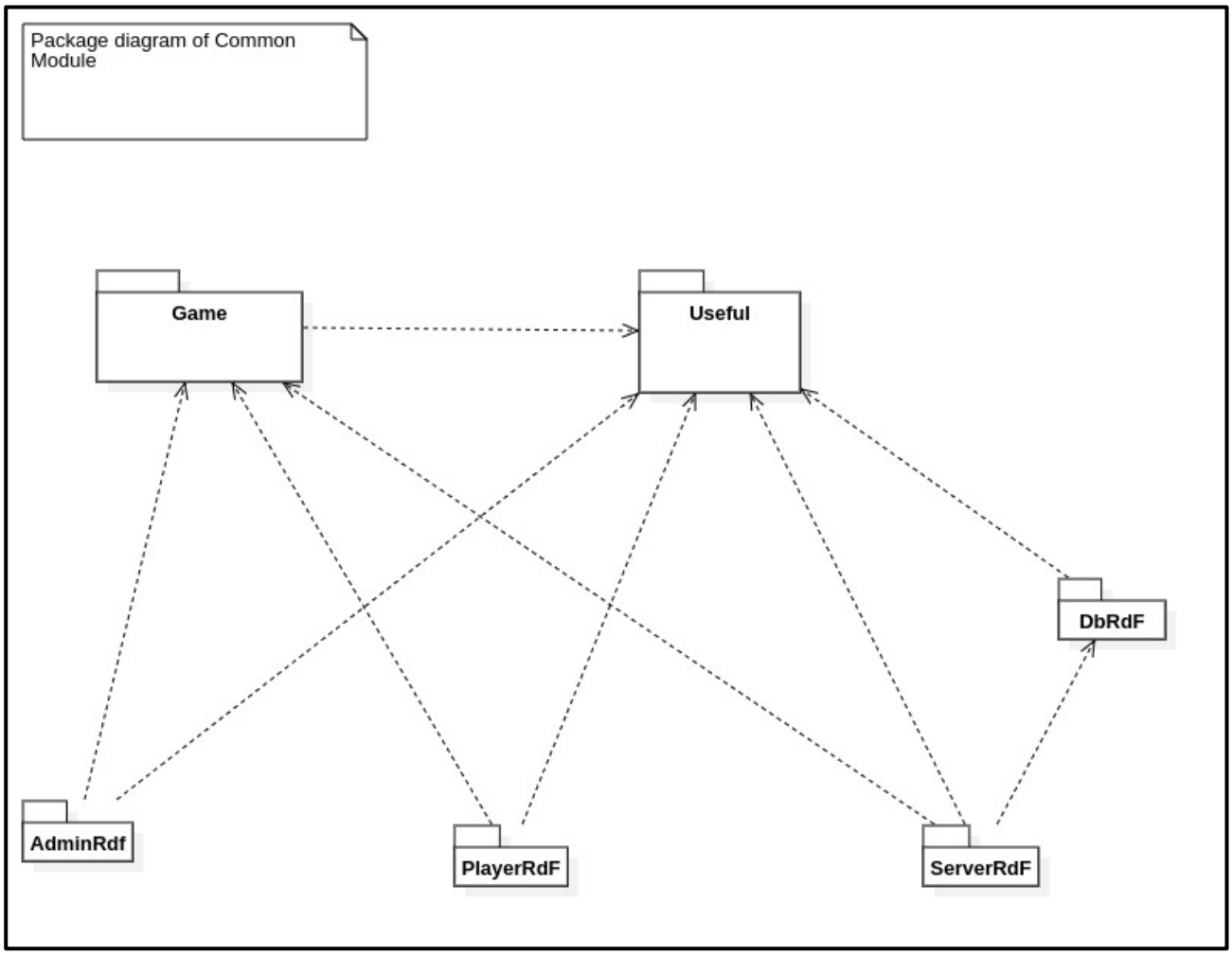
La struttura dei moduli si basa sul pattern parent child:

- Il modulo ***source code*** è il modulo parent. Esso contiene tutti le dipendenze comuni tra i moduli.

- Da esso dipende il modulo child ***common***. Esso contiene le classi per la creazione del sistema, divise semanticamente in package. La decisone di utilizzare questo modulo è finalizzata ad evitare di creare dipendenze cicliche tra moduli.

- I moduli sottostanti a common contengono le classi main che permettono di eseguire il progetto tramite java. Essi infatti, mediante il plugin maven shade, creeranno un file jar contente tutte le dipendenze necessarie alla compilazione.

- **Struttura dei package**



Il package ***useful*** contiene tutte le classi discusse a pagina 8.

Il package ***game*** contiene le tutte le classi finalizzate alla creazione del server di partita, dell'interfaccia di partita e di strutture dati sfruttate da essi.

Il package ***adminRdF*** contiene le classi utilizzate da un admin durante la sua permanenza sulla piattaforma, come l'interfaccia di menù, quella di visione del gioco.

Il package ***playerRdF*** contiene le classi utilizzate da un player durante la sua permanenza sulla piattaforma, come l'interfaccia di menù, quella di visione del gioco e quella di partita.

Il package ***serverRdF*** contiene le classi utilizzate dal server al funzionamento del server, il quale processa le richieste degli utenti.

➢ Design patterns

➔ Proxy

Viene utilizzato dai client in tutte le fasi di comunicazione tra client e server: nel caso della comunicazione durante il gioco è RMI che fornisce un stub per il client e uno skeleton per il server; nel caso della comunicazione tramite interfaccia di menù è stato appositamente codificato da noi.
Il suo obiettivo è quello di sollevare il client dalle operazioni di richiesta al server al fine di rendere il codice più modulare e pulito.

➔ Observer

Dal momento che java ha deprecato l'utilizzo del pattern observer ne è stato sviluppato uno apposito per l'applicazione. Questo pattern supporta l'aggiunta e l'eliminazione di osservatori sul server e metodi di aggiornamento dello stato della partita per tutti gli observer. Il pattern è sfruttato dal server di gioco, il quale è denominato observable. I giocatori e gli osservatori vengono salvati in un ArrayList sul server come observer e il server aggiornerà l'interfaccia di ogni observer in base alle mosse fatte durante una partita.

➔ Singleton

Il pattern è applicato alla classe serverRdfCreator (artefice del funzionamento del server). Il server, per gestire più connessioni, si affiderà a dei thread da lui creati, per questo motivo è inutile avere più di una istanza di esso.