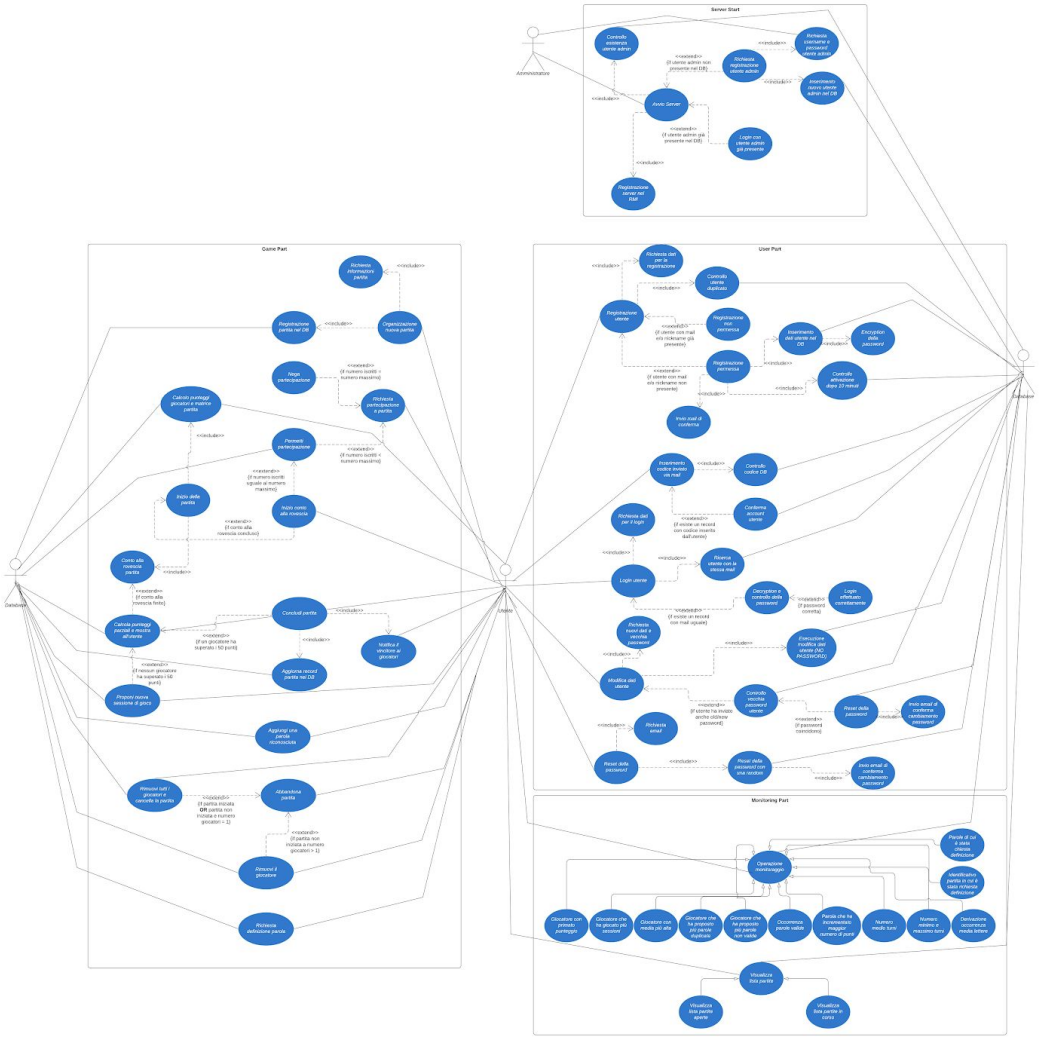
Introduzione
Il sistema software da realizzare rappresenta un applicativo distribuito client/server che implementa il gioco “Il Paroliere” in modalità multigiocatore.

Il software permette, per ogni giocatore, di creare un account utente, di autenticare il giocatore, e di modificare le informazioni associate ad esso.

Il software permette ad ogni giocatore di creare una partita, di visualizzare le partite aperte in corso, ed accedere alle partite aperte (che non hanno ancora raggiunto il limite massimo di giocatori).
Dopo che si raggiunge il numero massimo di utenti, parte un timer di 30 che precede l’inizio della prima (delle n sessioni) della medesima partita.
Ogni sessione dura 3 minuti, nei quali ogni giocatore deve proporre le parole che trova nella matrice di gioco (una griglia di lettere 4x4).
Allo scadere dei 3 minuti, verranno analizzate le parole proposte da ogni giocatore, e verrà assegnato un punteggio se valide.
Dopo aver completato l’analisi, si controllano i punteggi complessivi dei giocatori. Se un giocatore ha raggiunto i 50 punti, viene eletto vincitore e la partita si conclude; altrimenti si propone una nuova sessione.

Il software, per ogni giocatore, permette di visualizzare informazioni analitiche riportate nelle specifiche di progetto indicate.

Casi D’Uso
Lo use case diagram è descritto nel diagramma seguente:



Rappresentazione Tabellare di alcuni Use Case

Avvio Server	Il seguente use case è relazionato con l'attore **Amministratore** e rappresenta la situazione in cui l'amministratore avvia il server. Questo use case include **Controllo esistenza utente admin** e Registrazione server nel RMI. Inoltre, estende **Richiesta registrazione utente admin**, quando non esiste un utente admin nel DB; e **Login con utente admin già presente**, quando esiste un utente admin nel DB
Controllo esistenza utente admin	Questo use case si occupa di eseguire una query nel DB per controllare se esiste già un utente admin
Richiesta registrazione utente admin	Nel caso in cui non esistesse già un utente admin, viene eseguito il flusso che si occupa

	della registrazione di un nuovo utente admin. Questo use case include **Richiesta username e password utente admin e Inserimento nuovo utente admin nel DB**
Richiesta username e password utente admin	Questo use case si occupa di richiedere all'attore Amministratore di inserire username e password dell'utente admin che deve essere creato
Inserimento nuovo utente admin nel DB	Questo use case si occupa di inserire nel database il record relativo al nuovo utente admin
Login con utente admin già presente	Questo use case viene esteso nel momento in cui esiste già un utente admin nel database. In questo caso viene eseguito il login con l'utente admin presente
Registrazione server nel RMI	Dopo aver effettuato l'operazione di login, l'oggetto che rappresenta il server viene inserito nel registro RMI

Richiesta partecipazione a partita	Questo use case è relazionato con il giocatore e rappresenta la situazione in cui un giocatore decide di entrare in una partita. Questo use case estende **Nega partecipazione** e **Permetti partecipazione**
Nega partecipazione	Questo use case viene esteso nel momento in cui il numero di giocatori nella partita è uguale al numero massimo, e quindi il giocatore che fa richiesta di ingresso nella partita non può entrare
Permetti partecipazione	Questo use case viene esteso nel momento in cui il numero di giocatori è minore del numero massimo di giocatori. Il server andrà ad aggiornare la situazione nel DB a seguito dell'ingresso di un nuovo giocatore. Questo use case estende **Inizio conto alla rovescia**
Inizio conto alla rovescia	Questo use case viene esteso nel momento in cui, a seguito dell'ingresso di un giocatore nella partita, si raggiunge il numero massimo di giocatori. A seguito di questo evento, parte un timer di 30 secondi che precede l'inizio della partita. Questo use case estende **Inizio della partita**

Inizio della partita	Questo use case viene esteso nel momento in cui il timer di 30 secondi finisce e la partita deve iniziare. Questo use case include **Calcolo punteggi giocatori e matrice partita** e **Conto alla rovescia partita**
Calcolo punteggi giocatori e matrice partita	Questo use case calcola, per ogni giocatore della partita, I punteggi e genera la matrice da mostrare ai giocatori
Conto alla rovescia partita	Questo use case rappresenta il timer di 3 minuti che viene inizializzato non appena la partita inizia. Allo scadere del timer si andranno a calcolare I punteggi parziali e si mostreranno le parole proposte dai giocatori durante la sessione di gioco (questo grazie alla estensione dello use case **Calcola punteggi parziali e mostra all'utente**)

Operazione monitoraggio	Questo use case ha come attori il giocatore e il database. Questo perchè rappresenta le varie operazioni di monitoraggio presenti nella piattaforma (in totale sono 12). Ognuna di queste è una specializzazione di questo use case
Operazione con primato punteggio	Questo use case è una specializzazione di **Operazione monitoraggio**. Esegue una query nel DB dove ricerca l'utente che detiene il primato di punteggio raggiunto per sessione e per partita, indicando i rispettivi punteggi, e poi lo mostra al giocatore che ne ha fatto richiesta

Reset della password	Il seguente use case è relazionato con l'attore Utente e rappresenta la situazione in cui l'utente effettua il reset della passsword. Questo use case include Richiesta email, Reset della password con una random e Invio email di conferma cambiamento password.
Richiesta email	Questo use case si occupa di richiedere l'e-mail all'utente.
Reset della password con una random	La password nel DB viene sostituita con una casuale creata dal server.

Invio email di conferma cambiamento password	Viene inviata un'e-mail all'utente con la nuova password generata in modo casuale.

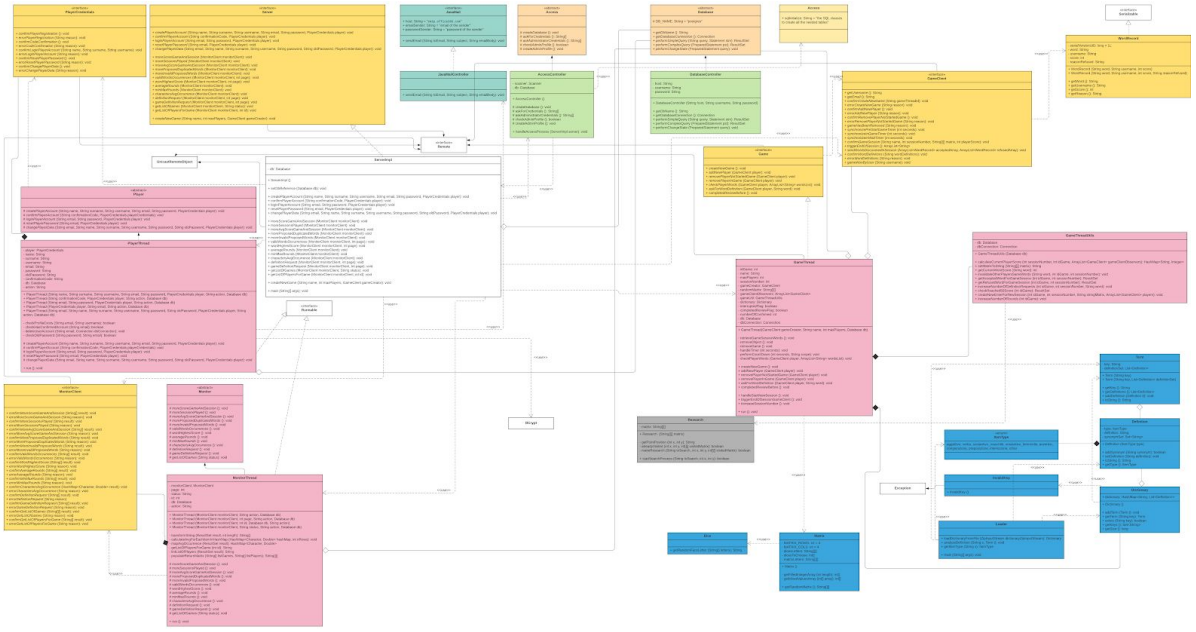
Inserimento codice inviato via mail	Questo use case è relazionato con l'attore Utente e rappresenta la situazione in cui un giocatore sta terminando la fase di creazione di un nuovo account, dove gli viene chiesto di inserire un codice di conferma inviatogli via mail. Questo use case estende Conferma account utente e include Controllo codice DB
Controllo codice DB	Va a controllare che esista un codice identico a quello inserito dall'utente.
Conferma account utente	Questo use case viene esteso nel momento in cui esiste un record identico con il codice inserito dall'utente. Va a cancellare il codice nel DB e setta il campo is_confirmed a true (nel DB).

Requisiti non Funzionali
Il software (client e server) se eseguito sullo stesso o differenti dispositivi appartenenti alla stessa rete, presenta una latenza media estremamente bassa (il comportamento responsive è immediato).
Il software non è stato testato su dispositivi appartenenti a reti differenti.

Il ciclo di vita di una singola partita (costituita da n sessioni di gioco) è interamente gestito da un unico oggetto remoto. Le varie chiamate di metodo fatte a questo oggetto, sono fatte da altri metodi appartenenti all'oggetto (data l'impossibilità di implementare una soluzione asincrona). Questo comporta, per partite molto lunghe, cioè con elevate sessioni, un record di attivazione abbastanza dispendioso in termini di memoria occupata per l'oggetto in questione.

Si è preferito concentrarsi maggiormente sull'aspetto logico/funzionale dell'applicativo, prestando minore attenzione agli aspetti riguardanti l'interfaccia grafica e dell'esperienza utente. Per questo motivo, l'interfaccia grafica è basica, ma allo stesso tempo funzionale.

Analysis Object Model Server



Per gli use case relativi alla parte Server Start, sono coinvolti I seguenti oggetti con I seguenti ruoli/relazioni. ServerImpl è l'oggetto su cui viene eseguito il metodo main() che da inizio a tutto il processo. La classe di questo oggetto implementa l'interfaccia Server, ed estende la classe UnicastRemoteObject. Questo oggetto usa un oggetto di classe AccessController per gestire la parte di login/registrazione utente admin. La classe dell'oggetto AccessController estende la classe astratta Access. Questo oggetto, userà la classe Access e il parametro pubblico sqlInitialize, per creare le tabelle nel database.
Dopo aver eseguito la parte di start del server, l'oggetto AccessController richiama il metodo setDbReference() su ServerImpl per settare il riferimento all'oggetto DatabaseController creato.

Per gli use case relativi alla parte User Part, sono coinvolti I seguenti oggetti con I seguenti ruoli/relazioni. ServerImpl, quando riceve una richiesta su uno dei metodi remoti relativi a questa parte, crea un'istanza (thread) di PlayerThread. La classe PlayerThread estende la classe astratta Player, che usa/sfrutta I metodi dell'interfaccia PlayerCredentials (che estende l'interfaccia Remote). Durante l'esecuzione dei vari metodi presenti sugli oggetti di tipo PlayerThread, questi oggetti usano I metodi dell'interfaccia PlayerCredentials per inviare al client la risposta a seguito della sua richiesta. Gli oggetti/thread PlayerThread, durante le loro operazioni, usano la classe JavaMailController (che implementa l'interfaccia JavaMail) per inviare mail agli utenti (nelle operazioni di registrazione, cambio dati, e reset password). Questi oggetti usano anche la classe Bcrypt nel momento in cui le password vengono memorizzate nel DB (per criptarle ed evitare che vengano salvate in chiaro).

Per gli use case relativi alla parte Monitoring Part, sono coinvolti I seguenti oggetti con I seguenti ruoli/relazioni. ServerImpl, quando riceve una richiesta su uno dei metodi remoti relativi a questa parte, crea un'istanza (thread) di MonitorThread. La classe MonitorThread estende la classe astratta Monitor. Durante l'esecuzione dei vari metodi presenti sugli oggetti di tipo MonitorThread, questi oggetti usano I metodi dell'interfaccia MonitorClient (che estende l'interfaccia Remote) per inviare al client la risposta a seguito della sua richiesta.

Per gli use case relativi alla parte Game Part, sono coinvolti I seguenti oggetti con I seguenti ruoli/relazioni. ServerImpl, quando riceve una richiesta di creazione di una nuova partita, crea un nuovo oggetto di classe GameThread, e invoca il metodo run(). Questo metodo si occuperà della creazione della partita. La classe GameThread estende UnicastRemoteObject, e implementa l'interfaccia Game. L'interfaccia Game estende l'interfaccia Remote, e usa l'interfaccia GameClient (I metodi implementati in GameThread useranno I metodi dell'interfaccia GameClient per comunicare con Il client). L'interfaccia GameClient estende l'interfaccia Remote, e usa la classe WordRecord (usata per inviare le parole trovate durante la sessione di gioco ai client. Questa classe implementa l'interfaccia Serializable).
Durante lo svolgimento delle sessioni di gioco, gli oggetti GameThread, useranno la classe Matrix (che a sua volta usa Dice), per generare le matrici di gioco.
Durante lo svolgimento delle sessioni di gioco, gli oggetti GameThread, sfrutteranno Il metodo performCountdown come ausilio per la gestione dei timer (timer per il countdown di partita; timer per il countdown per il controllo delle parole proposte).
Durante lo svolgimento delle sessioni di gioco, gli oggetti GameThread, useranno gli oggetti di classe GameThreadUtils come aiuto per l'esecuzione di alcune operazioni (tra cui quella di ricerca di una parola nella matrice. Per questo, la classe GameThreadUtils, usa gli oggetti di classe Research).Durante lo svolgimento delle sessioni di gioco, gli oggetti GameThread, useranno gli oggetti di classe Dictionary come ausilio per la ricerca delle parole proposte dai giocatori nel dizionario di lingua italiana.

Analysis Object Model Client
(Diagramma UML del client nella cartella documentazione/client/class_diagram_client.pdf. Non qui a causa della grandezza)

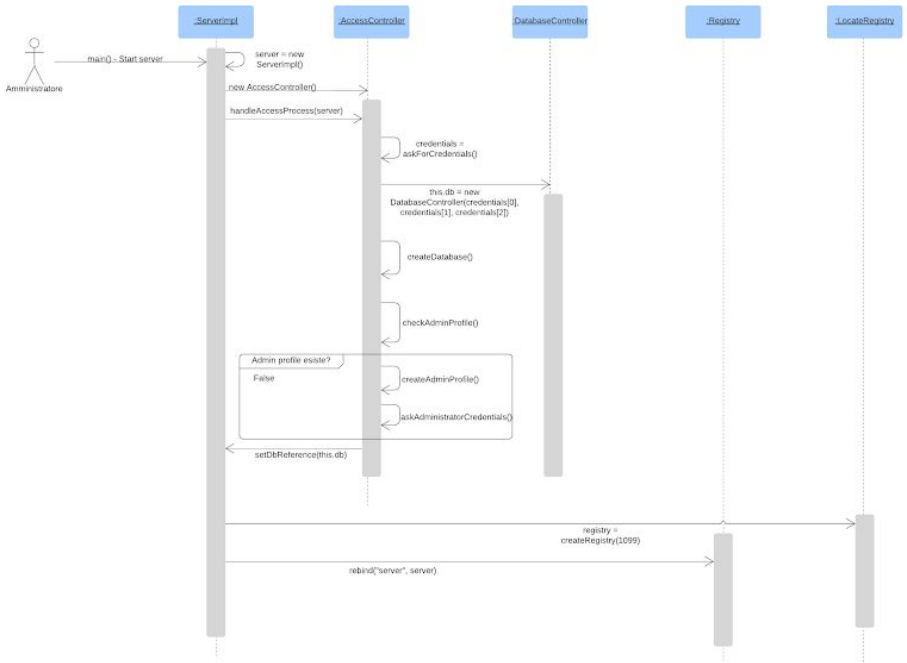
La classe Main è la classe principale che invoca il proprio metodo main quando si avvia l'applicativo (client ovviamente).
La classe astratta GraphicComponent rappresenta un elemento grafico (come un Button o come un Container per organizzare l'ordine di visualizzazione dei vari elementi UI).
Tutti gli elementi che estendono GraphicComponent sono elementi UI.
Quelle classi (come LoginUtente, UserRegistration ecc.) che implementano elementi UI (sia che estendano o meno la classe GraphicComponent), sono frame che hanno il compito di interfacciare l'utente con il server.
La classe LoginUtente permette al giocatore di effettuare il login (tramite email e password). Inoltre tramite questa classe si possono raggiungere le sezioni per la creazione di un nuovo account e per il reset della password.
La classe UserRegistration permette di creare un nuovo account (se l'utente non ha già effettuato il login), altrimenti permetterà di cambiare le informazioni relative al proprio account. Nel caso l'utente non sia loggato, UserRegistration permettà di tornare alla pagina di login, altrimenti alla pagina di Home.
La classe ConfirmCode riguarda il secondo step della fase di registrazione. In questo frame verrà richiesto di inserire il codice inviato via email. Una volta inserito ed inviato, l'utente verrà rimandato alla pagina di Home se il codice è corretto, altrimenti rimarrà sullo stesso frame. Un altro pulsante permette di tornare alla schermata di login.
La classe ResetPassword permette di effettuare il reset della password inserendo l'e-mail del proprio account. Un button permetterà di tornare alla schermata di login.

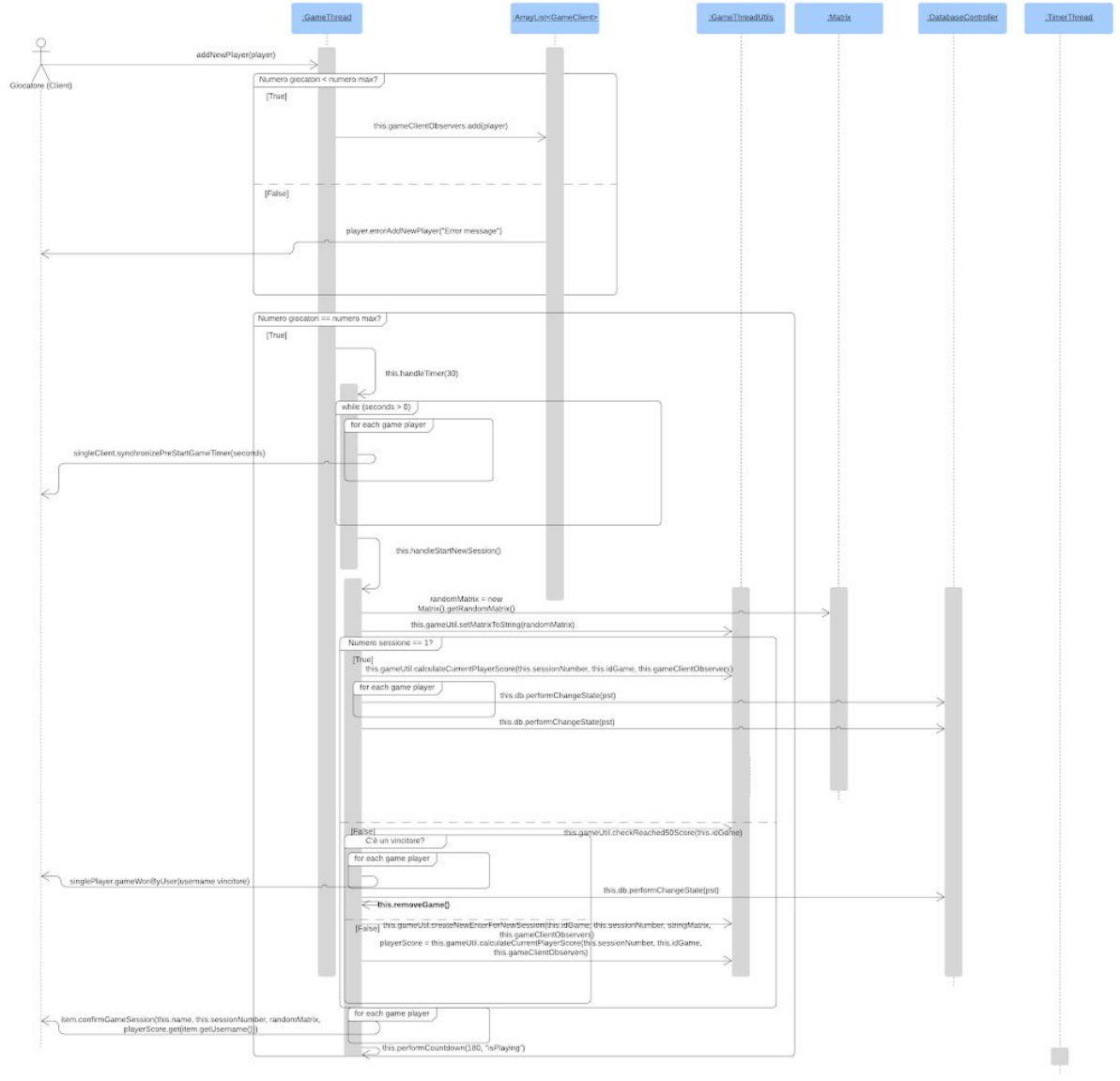
La classe Home permette di cambiare schermata in base al button cliccato. Si possono raggiungere le seguenti classi:
- CreateNewGame
- ListGames
- Analytics
- UserRegistration (per la fase di modifica dei dati)

La classe Analytics permette di visualizzare informazioni analitiche relative alle partite giocate e altro. Permette di raggiungere la classe AnalyticsDataPage mostrando un'informazione specifica. Inoltre permette di tornare alla pagina di Home.
La classe AnalyticsDataPage mostra un'informazione fornita dalla classe Analytics.
Permette di tornare alla pagina di Analytics o alla pagina di Home.
La classe CreateNewGame permette di creare un nuovo gioco facendo inserire nome della partita e il numero dei partecipanti (2 – 6). Se la creazione va a buon fine, il metodo specifico di GameClientImpl manda l'utente alla classe WaitingPlayers.
La classe WaitingPlayers rimarrà attiva fino a quando il numero di utenti partecipanti a quella partita non sarà riempito. E' presente un button per tornare alla pagine di Home.
La classe ListGames permette di visualizzare i giochi aperti e in fase di svolgimento (in base al button che si seleziona). Cliccando sul button partecipa di una riga di un gioco aperto, si verrà rimandati alla classe WaitingPlayers. E' presente un button per tornare alla pagine di Home.La classe WaitingStartGame viene mostrata una volta che il numero di partecipanti ad un partita è raggiunto e verrà mostrato un countdown di 30 secondi.
La classe GamePlay viene mostrata una volta che il countdown in WaitingStartGame è terminato.
L'utente potrà giocare al gioco effettivo, con un countdown di 180 secondi. E' presente un button per tornare alla pagine di Home.
La classe WordsAnalysis verrà mostrata una volta che il countdown in GamePlay sarà terminato. In questa classe verranno mostrate le parole trovate da tutti gli utenti. Un button permeterrà di terminare l'analisi rimandando l'utente nella pagina di attesa WaitingPlayers, un altro button permeterrà di tornare alla Home e quello rimanente permetterà di avviare la classe SingleWordAnalysis.
La classe SingleWordAnalysis verrà mostrata dalla classe Analysis per visualizzare le informazioni relative ad una singola parola, con un countdown di 180 secondi. Sarà presente un button per richiedere la definizione della parola, e un altro per chiudere questa finestra (e visualizzare solo la pagina di SingleWordAnalysis).
Una vola terminata la fase di analisi, se un giocatore avrà raggiunto almeno 50, verrà mostrata la pagina di GameWinner che mostrerà il giocatore vincente e permetterà di tornare alla pagina di Home, altrimenti verrà iniziata una nuova partita rimandando l'utente in GamePlay.
Le classi che terminano con "Impl", sono classi che verranno passate come parametro al server che a sua volta le utilizzerà per comunicare con in client. Per poterle inviare come parametro al server, queste classi estendono UnicastRemoteObject.
La classe SingleGame rappresenta il modello di una partita.
La classe Validation fornisce dei metodi per la validazione di un campo input.
Verifica che il campo non sia vuoto e fornisce una funzione per validare un'e-mail.
La classe FrameHandler permette di gestire i vari frame presenti nell'applicazione.
Invece di lasciare la gestione dei frame separati (ovvero ogni classe si gestisce il proprio frame), questa classe centralizza il comportamento di visualizzazione e rimozione di ogni frame, facendo in modo che solo due frame possano essere aperti nello stesso momento

(un frame possiede una priorità maggiore e l’altro una priorità secondaria). In questo modo sono stati risolti bug runtime riguardanti i frame, non riproducibili.

Dynamic Model Server


:ServerImpl
:AccessController
:DatabaseController
:Registry
:LocateRegistry
Amministratore
main() - Start server
server = new ServerImpl()
new AccessController()
handleAccessProcess(server)
credentials = askForCredentials()
this.db = new DatabaseController(credentials[0], credentials[1], credentials[2])
createDatabase()
checkAdminProfile()
Admin profile esiste?
False
createAdminProfile()
askAdministratorCredentials()
setDbReference(this.db)
registry = createRegistry(1099)
rebind("server", server)


:GameThread
ArrayList<GameClient>
:GameThreadUtils
:Matrix
:DatabaseController
:TimerThread
Giocatore (Client)
addNewPlayer(player)
Numero giocatori < numero max?
[True]
this.gameClientObservers.add(player)
[False]
player.errorAddNewPlayer("Error message")
Numero giocatori == numero max?
[True]
this.handleTimer(30)
while (seconds > 0)
for each game player
singleClient.synchronizePreStartGameTimer(seconds)
this.handleStartNewSession()
randomMatrix = new Matrix().getRandomMatrix()
this.gameUtil.setMatrixToString(randomMatrix)
Numero sessione == 1?
[True]
this.gameUtil.calculateCurrentPlayerScore(this.sessionNumber, this.idGame, this.gameClientObservers)
for each game player
this.db.performChangeState(pst)
this.db.performChangeState(pst)
[False]
this.gameUtil.checkReached50Score(this.idGame)
C'è un vincitore?
for each game player
singlePlayer.gameWonByUser(username vincitore)
this.db.performChangeState(pst)
this.removeGame()
[False]
this.gameUtil.createNewEnterForNewSession(this.idGame, this.sessionNumber, stringMatrix, this.gameClientObservers)
playerScore = this.gameUtil.calculateCurrentPlayerScore(this.sessionNumber, this.idGame, this.gameClientObservers)
for each game player
item.confirmGameSession(this.name, this.sessionNumber, randomMatrix, playerScore.get(item.getUsername()))
this.performCountdown(180, "isPlaying")

Sequence Diagram – Start Server

Il flusso logico parte a seguito dell'avvio del server da parte dell'amministratore. Viene invocato il metodo main() della classe ServerImpl. Dopo aver gestito la parte sulla sicurezza (SecurityManager), il metodo crea una nuova istanza della classe ServerImpl, e il riferimento lo salva in una variabile nominata server. A questo punto il metodo crea un'istanza della classe AccessController e, tramite il riferimento all'oggetto creato, invoca il metodo handleAccessProcess(server) (che si occuperà del processo di registrazione/login utente admin).

L'attenzione passa ora sull'oggetto AccessController e, in particolare, sul metodo handleAccessProcess. Qui viene richiamato il metodo askForCredentials() che permetterà all'utente amministratore di inserire le credenziali del database (host, username, password); il metodo restituirà un array di stringhe di lunghezza 3.

A questo punto viene creata una istanza della classe DatabaseController (tipo Database) passando le credenziali inserite dall'amministratore.

Successivamente si invoca il metodo createDatabase(), che andrà a creare le varie tabelle necessarie all'esecuzione del software. Se il database le contiene già, questa operazione viene saltata.

Successivamente viene invocato il metodo checkAdminProfile(), che andrà ad eseguire una query sul DB controllando se esistono record nella tabella administrator. Se esistono tornerà True, altrimenti False. Se non esiste nessun utente amministratore, viene invocato il metodo createAdminProfile(). Questo metodo, dopo aver richiesto username e password del nuovo utente amministratore, inserisce un un nuovo record nel DB.

Dopodichè viene chiamato il metodo setDbReference(this.db) che permetterà di salvare l'oggetto che funge da tramite verso il DB nell'istanza di ServerImpl (per usi futuri).

A questo punto l'istanza di ServerImpl crea l'RMI registry chiamando il metodo createRegistry(1099) e, utilizzando il nuovo riferimento del registro RMI, inserisce il riferimento dell'oggetto di tipo ServerImpl.

Sequence Diagram – Add new player to the game

(QUESTA è LA LOGICA CHE VIENE SEMPRE ESEGUITA QUANDO UNA PARTITA INIZIA)

Il flusso logico parte a seguito della richiesta del giocatore (client) di essere aggiunto ad una partita tramite l'invocazione di addNewPlayer(player) su un oggetto di tipo GameThread.

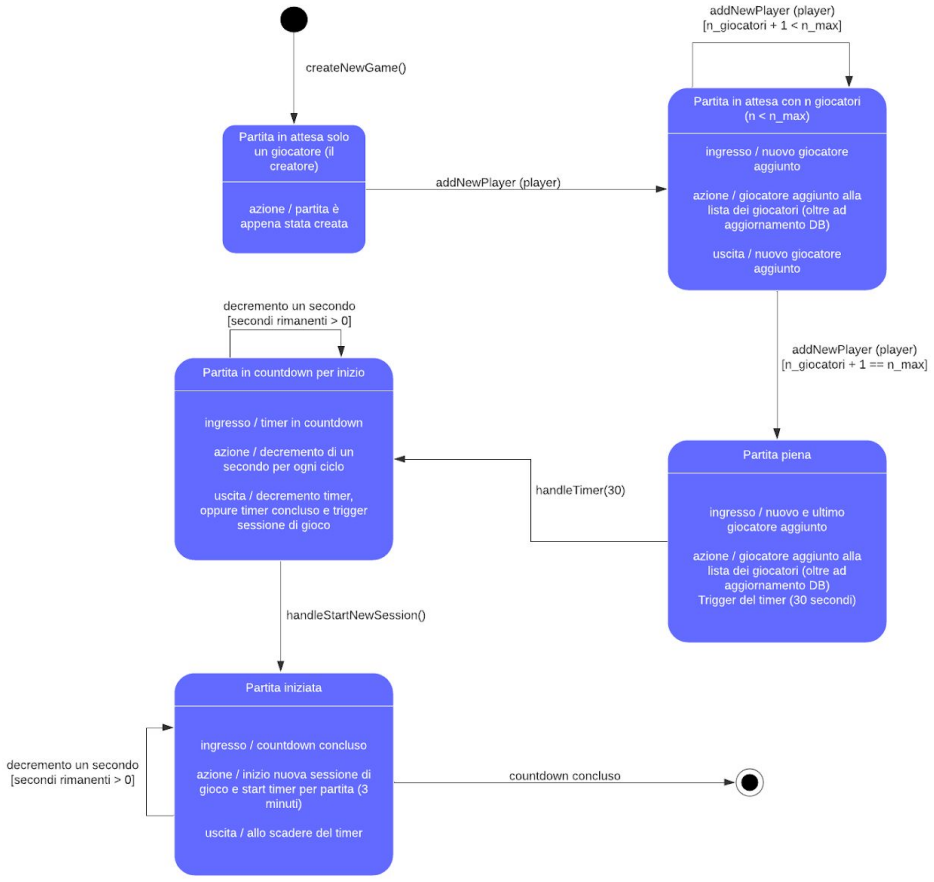
Se il numero di giocatori è minore del numero massimo, l'utente viene aggiunto alla lista dei giocatori (ArrayList<GameClient>) tramite this.gameClientObservers.add(player); altrimenti viene inviato al client un errore tramite player.errorAddNewPlayer("Error message").

Se, a seguito dell'aggiunta dell'utente, si è raggiunto il numero di giocatori massimo, viene invocato il timer di 30 secondi tramite this.handleTimer(30). In questi 30 secondi, per ogni secondo che passa viene notificato al giocatore il tempo rimanente all'inizio della partita tramite singleClient.synchronizePreStartGameTimer(seconds).

Allo scadere del timer la partita deve iniziare, e viene chiamato il metodo this.handleStartNewSession(). Qui si andrà a generare la matrice di gioco (e la sua rappresentazione sottoforma di stringa), e poi si controlla il numero di sessione corrente. Se è la prima sessione di gioco, si calcolano I punteggi dei giocatori (saranno 0) e si vanno a settare le informazioni (compresa la matrice di quella sessione) nel DB.

(QUESTA è LA LOGICA CHE VIENE ESEGUITA QUANDO LA SESSIONE DA GIOCARE NON é LA PRIMA)

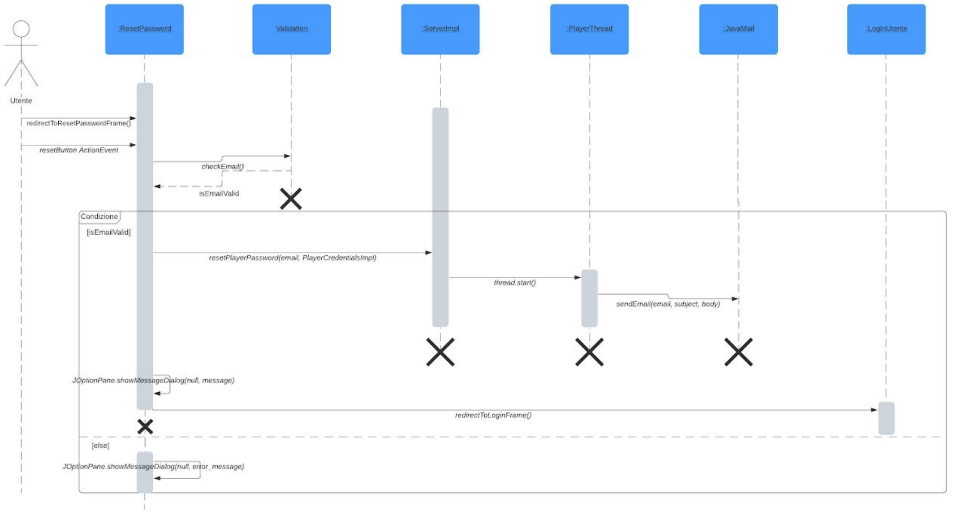
Se invece non è la prima sessione di gioco ma una successiva, si controlla se c'è un vincitore con il metodo this.gameUtil.checkReached50Score(this.idGame). Se c'è, si notifica a tutti I giocatoritramite il metodo singlePlayer.gameWonByUser(username vincitore), si aggiorna lo stato della partita nel DB, ed infine si elimina l'oggetto remoto responsabile della partita tramite this.removeGame().
Se invece non c'è un vincitore, si creano delle nuove sessioni di gioco per ogni utente nel DB tramite il metodo this.gameUtil.createNewEnterForNewSession(this.idGame, this.sessionNumber, stringMatrix, this.gameClientObservers). Poi si calcolano I punteggi dei giocatori con il metodo this.gameUtil.calculateCurrentPlayerScore(this.sessionNumber, this.idGame, this.gameClientObservers). Infine, si notifica l'inizio di una nuova sessione per ogni giocatore con il metodo item.confirmGameSession(this.name, this.sessionNumber, randomMatrix, playerScore.get(item.getUsername())). Dopo questa operazione, si invoca un nuovo timer di 3 minuti che scandirà il tempo della partita.

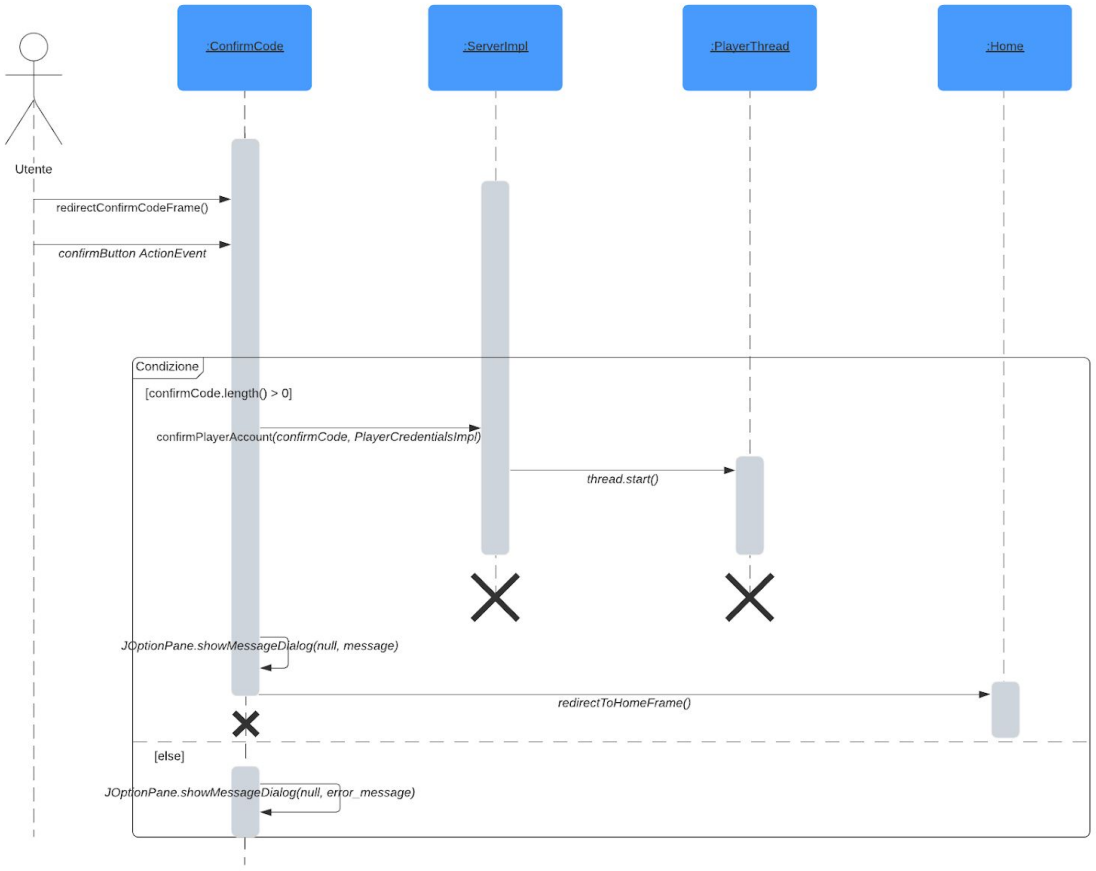


State Machine - Add New Player (and trigger a session)
Lo stato iniziale si raggiunge nel momento in cui un giocatore crea una nuova partita, la partita è nello stato Partita in attesa un solo giocatore.
La prima volta che un giocatore (diverso dal creatore, che è già dentro) entra nella partita, lo stato passa In Partita in attesa con n giocatori (n < n_max).

Tutte le volte che un giocatore entra nella partita e il numero totale di giocatori è comunque < del numero massimo di giocatori, la partita non cambia stato.
Nel momento in cui entra un giocatore e si raggiunge il numero massimo di giocatori, la partita passa in stato Partita piena. A seguito di questo evento e cambio di stato, viene triggerato un timer interno e la partita passa nello stato Partita in countdown per inizio.
Qui si continuerà ad uscire ed entrare nello stesso stato ad ogni decremento del timer finchè il numero di secondi rimanenti è > 0.
Nel momento in cui il countdown finisce, viene invocato l'inizio di una sessione di gioco e la partita inizia.
La partita passa in stato Partita iniziata. Appena entrati in questo ultimo stato, viene inizializzato un nuovo timer (di 3 minuti) che scandirà il passare del tempo. Finchè I secondi rimanenti sono > 0, si continuerà ad uscire e ad entrare in questo ultimo stato. Non appena questo countdown completa, la sessione di gioco si conclude.

Dynamic Model Client





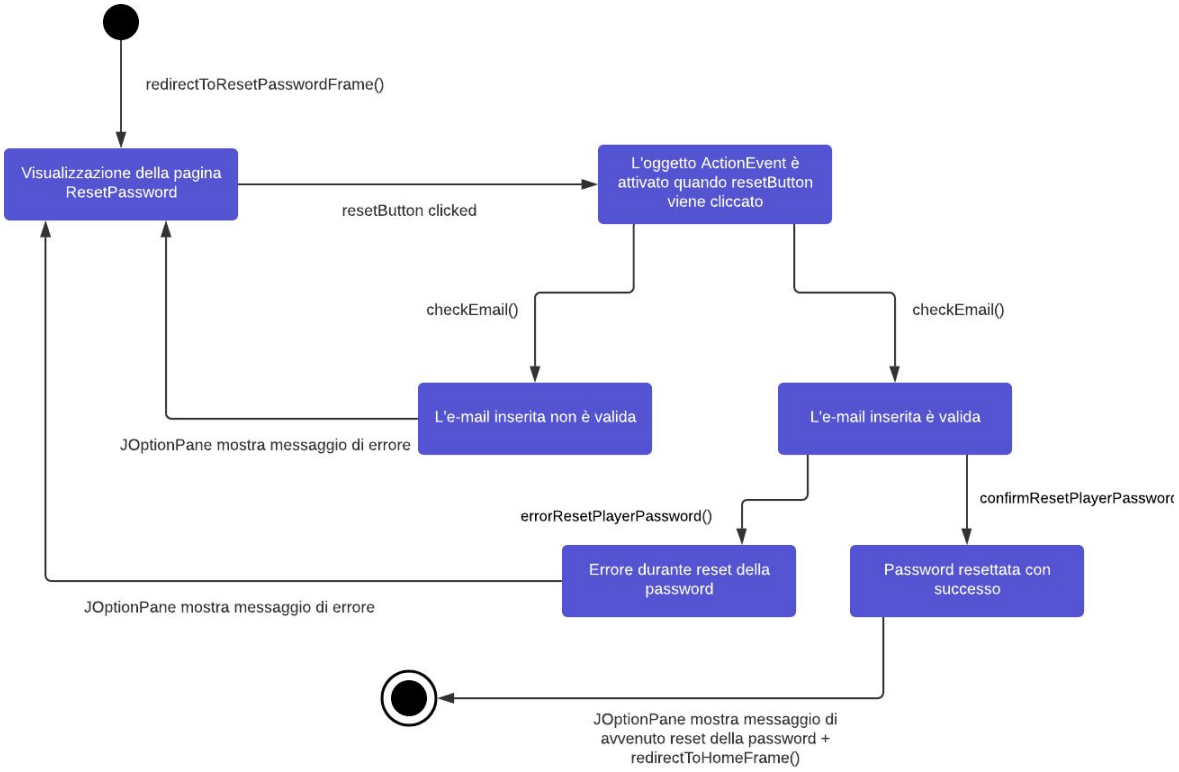
Sequence Diagram - Reset della password

L'utente accede alla pagina di reset della password cliccando sul label password dimenticata, attivando così il metodo redirectToResetPasswordFrame.
Una volta avuto accesso alla pagina e dopo aver inserito l'e-mail, l'utente clicca sul pulsante per resettare la password (resetButton).
Questo triggera l'ActionListener per il button resetButton che va a controllare che l'e-mail inserita sia un'e-mail valida tramite l'utilizzo dei metodi della classe Validation.
Se l'e-mail è valida, viene chiamato il metodo resetPlayerPassword a cui vengono passati come parametri attuali sia l'e-mail che un nuova istanza di PlayerCredentialsImpl con i metodi che il server andarà poi a chiamare in caso di successo o meno.
Il metodo resetPlayerPassword creerà una nuova istanza della classe PlayerThread ed avvierà tale Thread chiamando il metodo start.
A sua volta nel metodo run verrà eseguito il codice che, in base ai parametri passati precedentemente, si occuperà di verificare che l'account esista e di generare una password randomica che andrà a sostituire quella attuale.
Infine, il server invierà un'e-mail, medianta alla classe JavaMail, all'indirizzo di posta elettronica associato a quell'account avendo come contenuto la nuova e-mail settata per permettere all'utente di potersi loggare. Infine l'utente verrà rimandato alla schermata di login.

Nel caso in cui, invece, l'e-mail inserita non fosse corretta, verrà mostrato un alert indicando il problema relativo all'errore avvenuto.

Sequence Diagram - Codice di conferma (in fase di registrazione)

L'utente accede alla pagina di inserimento del codice di conferma, dopo aver inserito ed inviato i dati del proprio account al server durante la fase di registrazione. La transizione nella pagina di inserimento del codice di conferma avviene tramite il metodo redirectConfirmCodeFrame. Una volta che l'utente che ha inserito il codice di conferma e cliccato sul pulsante per inviare il codice di conferma (confirmButton), verrà verificato che il codice inserito non sia una stringa vuota. Nel caso questa condizione venga rispettata, verrà chiamato il metodo remoto confirmPlayerAccount.
Tale metodo creerà una nuova istanza della classe PlayerThread ed avvierà tale Thread chiamando il metodo start.
A sua volta nel metodo run verrà eseguito il codice che, in base ai parametri passati precedentemente, si occuperà di verificare che il codice inserito esista nella tabella, successivamente eliminarlo e settare il campo is_confirmed nel DB a true.
Infine lato client verrà mostrato un alert con un messaggio per avvisare del completamento dell'operazione con successo e si verrà reindirizzati alla pagina di Home tramite il metodo redirectToHomeFrame.
Nel caso in cui, invece, il codice inserito non fosse corretto, verrà mostrato un alert indicando il problema relativo all'errore avvenuto.

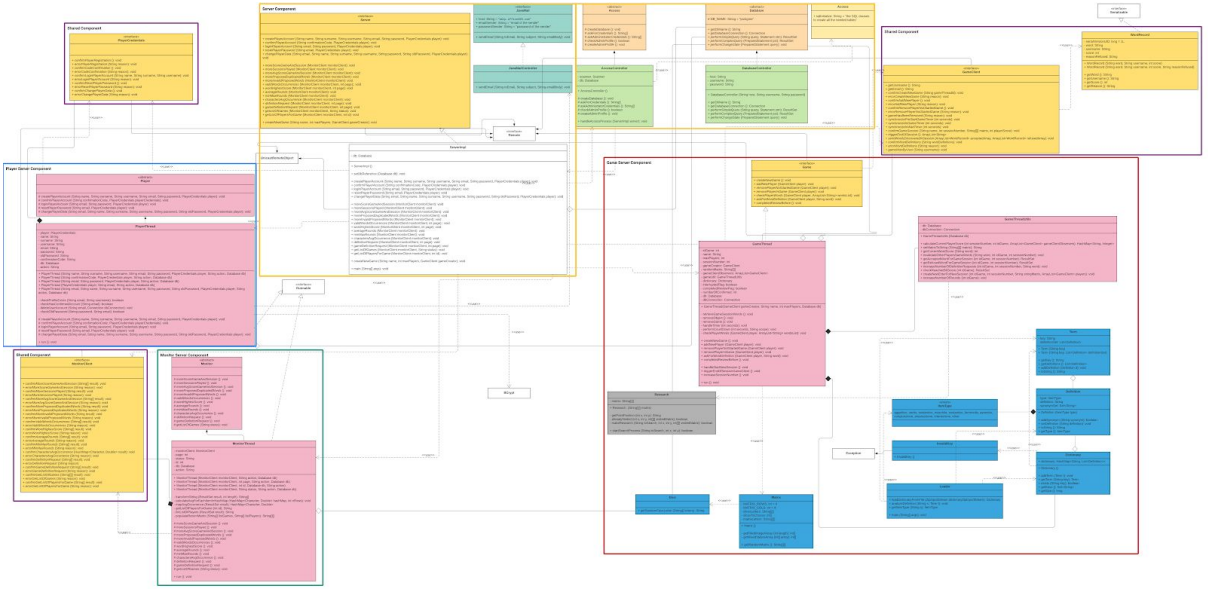


State Machine Diagram - Reset Password

Una volta avuto accesso alla pagina di reset della password, e dopo aver inserito l'e-mail, l'utente clicca sul pulsante per resettare la password (resetButton).
Questo attiva l'ActionListener generando un ActionEvent per il button resetButton che va a

controllare che l'e-mail inserita sia un'e-mail valida tramite il metodo checkEmail().
Se l'e-mail è valida viene avviato il processo di reset della password (chiamando il metodo remoto resetPlayerPassword).
Se il reset della password andrà a buon fine, il server chiamerà la funzione confirmResetPlayerPassword(), mostrerà un alert con un messaggio del corretto reset della password e manderà l'utente nella pagina Home.
Nel caso, invece, in cui l'e-mail sia errata o ci sia un problema durante il reset della password, verrà mostrato un alert mostrando un messaggio relativo all'errore avvenuto.

Component Diagram



Lato server ci sono I seguenti componenti:

- **Server Component**: Questo componente contiene le classi e le interfacce che vengono gestite direttamente dal thread server principale, cioè quello che viene lanciato all'avvio del server e che si occupa di gestire tutte le richieste (ad eccezione di quelle fatte nel momento in cui si sta giocando una partita). Sono presenti le classi responsabili per la gestione/comunicazione con il DB, le classi per la gestione del login lato server (con utente amministratore), e le classi che contengono I metodi remoti richiamabili dal client.
- **Player Server Component**: Questo componente contiene le classi e le interfacce che sono responsabili per la gestione degli utenti (players) sulla piattaforma. La classe PlayerThread rappresenta il thread che verrà istanziato ogni volta che sul server (ServerImpl) verrà richiamato uno dei metodi che gestiscono gli utenti (ad esempio il login, piuttosto che la registrazione).
- **Monitor Server Component**: Questo componente contiene le classi e le interfacce che sono responsabili per la gestione delle operazioni di analytics/monitoring sulla piattaforma. La classe MonitorThread rappresenta il thread che verrà istanziato ogni volta che sul server (ServerImpl) verrà richiamato uno dei metodi che gestiscono le operazioni di analytics/monitoring (ad esempio la query che restituisce il giocatore che ha giocato più sessioni).
- **Game Server Component**: Questo componente contiene le classi e le interfacce che sono responsabili per la gestione delle operazioni di gestione della partita sulla

piattaforma. La classe GameThread, utilizzata da ServerImpl, sarà il punto cruciale per la creazione e per tutta la fase di vita di una partita. Per eseguire tutte le operazioni (ad esempio ricerca nella matrice ecc), sfrutterà le altre classsi/interfacce presenti nel Game Server Component.
- **Shared Component**: Questo componente contiene le interfacce e la classe WordRecord che sono utilizzate sia dal client che dal server. Il server le userà per sapere quale metodo richiamare per comunicare con il client; il client implementerà queste interfacce con classi in cui andrà a scrivere Il codice dei metodi per reagire alle chiamate da parte del server.

Gestione Dati - Schema ER
Il database è costituito dalle seguenti entità (che sono state poi tramutate in tabelle come descritto nello schema relazionale in formato SQL allegato in server/documentazione/schema_relazionale.sql).

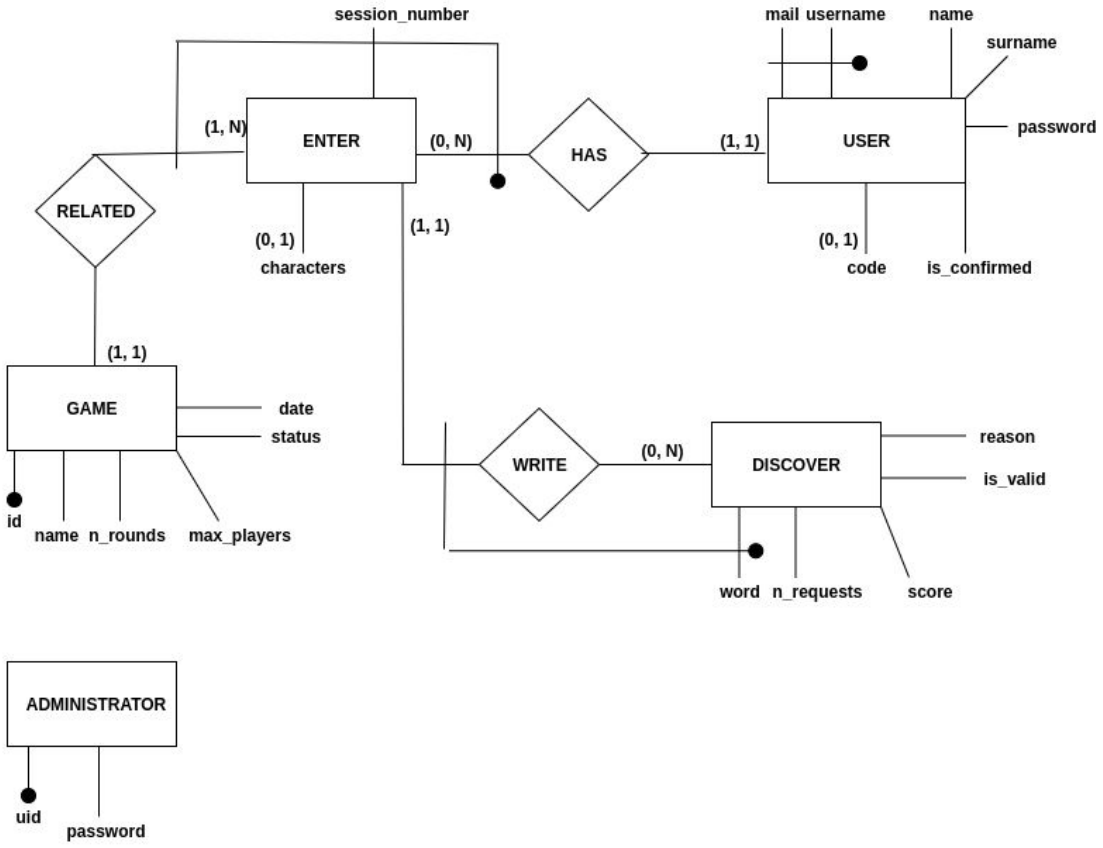
- **ADMINISTRATOR**: Le istanze di questa entità rappresentano gli utenti amministratori del server. All'avvio del server IP è necessario che almeno un utente amministratore esista. In caso contrario, prima che il server inizi a processare eventuali richieste da parte dei client, provvederà a crearne uno chiedendo all'utente uno uid (identificatore univoco per ogni amministratore) e una password da associare all'amministratore. Entrambi gli attributi sono di tipo stringa, ed entrambi sono obbligatori.
- **GAME**: Le istanze di questa entità rappresentano le partite del gioco "Il paroliere" (Le partite, non le sessioni di gioco associate alla partita). Ogni partita è identificata univocamente da un attributo id di tipo intero (autoincrement). Inoltre ogni partita ha come attributi: name (nome della partita) di tipo stringa, n_rounds (numero di round/sessioni associati alla partita) di tipo intero, max_players (il numero massimo di utenti ammissibili in partita) di tipo intero, status (lo stato della partita (Questo attributo può assumere solamente i seguenti valori: open → La partita non è ancora iniziata, nuovi utenti possono accedere; playing → la partita si sta giocando, nessun nuovo utente ammesso; closed → la partita si è conclusa)) di tipo stringa, date (la data e ora di creazione della partita) di tipo datetime.
- **USER**: Le istanze di questa entità rappresentano gli utenti giocatori che si sono registrati sul server del gioco "Il paroliere". Ogni utente è identificato univocamente sia da una mail che da uno username (inoltre questi due attributi sono entrambi univoci all'interno del database). I due attributi che fungono da primary key e unique sono di tipo stringa. Inoltre ogni utente ha come attributi: name (il nome dell'utente) di tipo stringa, surname (il cognome dell'utente) di tipo stringa, password (la password che l'utente dovrà usare insieme alla mail per accedere alla piattaforma) di tipo stringa, is_confirmed (attributo booleano che viene settato a false nel momento in cui l'utente viene creato, e che verrà settato a true solamente quando l'utente confermerà la propria registrazione tramite la mail inviata in fase di registrazione) di tipo boolean, code (codice che viene settato in fase di registrazione e che verrà inviato all'utente via mail. L'utente dovrà verificare la propria registrazione usando questo codice. Una volta confermata l'identità, questo attributo verrà setttato a NULL) di tipo stringa.
- **ENTER**: Le istanze di questa entità rappresentano il fatto che un utente sia entrato in una partita. C'è un'istanza di ENTER per ogni utente/partita per ogni sessione di

gioco. Significa che se la partita ha avuto 3 sessioni di gioco, ci saranno 3 istanze di ENTER relative allo stesso utente e alla stessa partita. Di fatti, le istanze di ENTER sono identificate univocamente dall'id della partita, dalla mail e username dell'utente, e dal numero di sessione (session_number). Inoltre ogni istanza ha l'attributo characters (rappresenta le lettere uscite nella griglia di quella particolare sessione. Dato che la griglia è una 4x4, la stringa in characters sarà lunga 16 → primi 4 caratteri prima riga; prossimi 4 caratteri seconda riga; prossimi 4 caratteri terza riga; ultimi 4 caratteri quarta ed ultima riga).

- **DISCOVER**: Le istanze di questa entità rappresentano le parole che un utente, in una specifica sessione di una specifica partita propone. Tutte le parole proposte dall'utente (sia valide che non valide vengono registrate). Ogni istanza è identificata quindi dall'identificatore della partita, dall'identificatore dell'utente, dall'identificatore della sessione di gioco e dalla parola stessa (word di tipo stringa). Questo approccio permette a più utenti di proporre la stessa parola nella medesima sessione della stessa partita. Inoltre ogni istanza ha l seguenti attributi: n_requests (numero di richieste di spiegazione da parte degli utenti) di tipo intero, score (punteggio che questa parola ha fatto ottenere all'utente che l'ha proposta) di tipo intero, is_valid (booleano che rappresenta se la parola è validaoppure no) di tipo boolean, reason (specifica, in caso di parola non valida, la ragione del rifiuto) di tipo stringa.

Il database, inoltre, contiene le seguenti relazioni tra le entità:

- **HAS**: Relazione che collega le entità ENTER e USER. Questa relazione rappresenta il fatto che un utente entra in una partita. Dalle cardinalità capiamo che un utente può essere associato con 0 o n istanze di enter (il che significa che un utente può aver partecipato a sessioni di gioco. Data la struttura del DB, comprendiamo il fatto che un utente può partecipare a più sessioni relative alla stessa partita (grazie al parametro session_number)). Dato che le istanze di enter sono anche identificate dall'utente, una istanza di enter può essere relazionata con uno ed un solo utente.
- **RELATED**: Relazione che collega le entità GAME e ENTER. Questa relazione rappresenta il fatto che una partita ha associate più partecipazioni da parte di utenti. Un'istanza di GAME ha associate A x B numero di istanze di ENTER (A = numero di turni/sessioni di gioco; B = numero di giocatori iscritti alla partita). Un'istanza di una partita ha associata almeno una partecipazione (nel momento in cui tutti lasciano la partita, questa viene cancellata). Dato che le istanze di enter sono anche identificate dalla partita, una istanza di enter può essere relazionata con una ed una sola partita.
- **WRITE**: Relazione che collega le entità ENTER e DISCOVER. Questa relazione rappresenta la registrazione di una parola relativa ad una sessione di gioco (che a sua volta è relativa ad un solo utente e ad una sola partita).



*** Gli attributi per i quali non è presente la cardinalità esplicita, sono obbligatori**

Gestione Dati - Schema Relazionale

Lo schema relazionale del database rispecchia lo schema ER presente nel documento di analisi dello
schema ER. Lo schema relazionale è costituito dalle seguenti tabelle:

- administrator (**uid**, password)
  - uid → primary key
- users (**email, username**, name, surname, password, is_confirmed, code)
  - email, username → primary key e unique
  - is_confirmed → valore di default settato a **false**
- game (**id**, name, max_players, status)
  - id → primary key
  - max_players → vincolo associato per verificare che il valore sia >= 2 e < 7
  - status → vincolo associato per verificare che il valore sia “open” oppure “playing” oppure “closed”
- enter (**id_game**, **email_user**, **username_user**, **session_number**, characters)
  - id_game, email_user, username_user, session_number → primary key
  - email_user, username_use → foreign key. Nel momento in cui una operazione dovesse invalidare la chiave esterna, il DBSM blocca l’operazione (on delete no action)

    - id_game → foreign key. Nel momento in cui una operazione dovesse invalidare la chiave esterna, il DBSM apporta il cambamento/eliminazione a cascata (on delete cascade)
    - session_number → valore di default **1**. Inoltre c'è un vincolo che garantisce che questo valore sia > 0
- discover (**word, <u>id_game, email_user, username_user, session_number_enter</u>**, n_requests, score, is_valid, reason)
    - word, id_game, email_user, username_user, session_number_enter → primary key
    - id_game, email_user, username_user, session_number_enter → foreign key. Nel momento in cui una operazione dovesse invalidare la chiave esterna, il DBSM apporta il cambamento/eliminazione a cascata (on delete cascade)
    - n_requests → valore di default **0**. Inoltre c'è un vincolo che garantisce che questo valore sia >= 0
    - score → C'è un vincolo che garantisce che questo valore sia >= 0
    - reason → Valore di default stringa vuota “”

Componenti Off-the-Shelf e Design Patterns
L'unico componente off the shelf utilizzato lato server è la classe Bcrypt. Questa classe ha permesso di cifrare le password prima di inserirle nel DB per evitare di salvarle in chiaro.
I metodi utilizzati sono I seguenti:

- **hashpw(clear_pw, salt)**: metodo che permette di cifrare una password in chiaro (cioè clear_pw). Il parametro salt è una stringa alfanumerica aggiunta alla password prima che venga cifrata. In questo modo si cerca di avere password cifrate univoche.
- **gensalt()**: Metodo che genera un salt casuale.
- **checkpw(clear_pw, hashed_pw)**: metodo che controlla se una password in chiaro equivale ad una stringa hash.

I design pattern adottati lato server sono I seguenti:

- **Observable/Observer**: In questo caso gli oggetti Observable sono le istanze di GameThread, cioè gli oggetti remoti che gestiscono il ciclo di vita di una partita. Gli Observer, invece, sono I client che partecipano alla partita gestita da uno specifico GameThread. Quando ci sono eventi da notificare agli Observer (cioè ai giocatori), l'oggetto GameThread esegue una chiamata ad un metodo remoto presente sull'Observable (singolarmente nel caso in cui solo un observer debba essere notificato; ciclicamente nel caso la notifica debba essere inviata a tutti gli observer).
- **Thread**: Per evitare che ogni chiamata al server (non relativa ad una partita ma relativa ad operazioni di gestione degli utenti e/o operazioni di analytics) monopolizzasse il server, per ognuna di queste chiamate viene creato un thread (PlayerThread o MonitorThread) che gestirà, in parallelo con gli altri eventuali thread, le richieste di questo tipo. In questa maniera, abbiamo evitato che l'oggetto remoto ServerImpl dovesse richiamare e gestire tutti I flussi associati alle varie operazioni sfruttabili dal client. Saranno I vari thread che, in funzione della richiesta fatta, eseguiranno una certa porzione di codice.

Suddivisione del Lavoro

Dopo aver analizzato a fondo i requisiti dell'applicativo da realizzare, abbiamo lavorato insieme al fine di generare lo **use case diagram** per stabilire quali fossero i casi d'uso del software.
Successivamente, al fine di realizzare un applicativo distribuito, abbiamo definito **tre moduli**:

- **Server**: che contiene e descrive la parte relativa al server (ad esempio la gestione degli oggetti remoti responsabili della partita)
- **Client**: che contiene e descrive la parte relativa al client (ad esempio i vari frame che si occupano di creare un'interfaccia tra l'utente e il server)
- **Shared**: che contiene le interfacce condivise tra client e server.

A questo punto, Simone Paglino si è occupato della progettazione ed implementazione della parte client (e dell'implementazione delle interfacce nel modulo Shared).
Stefano Cascavilla, invece, si è occupato della progettazione e realizzazione della parte server (e della creazione delle interfacce nel modulo Shared).

Per quanto riguarda il database dell'applicazione, abbiamo lavorato insieme per l'analisi dei requisiti, per capire come dovesse essere strutturato. Infine, Stefano Cascavilla, ha provveduto a creare lo schema ER, per poi tradurlo in schema relazionale.

Dopo aver completato le varie parti, entrambi ci siamo organizzati per integrare client e server, e per eseguire testing dell'applicativo completato.