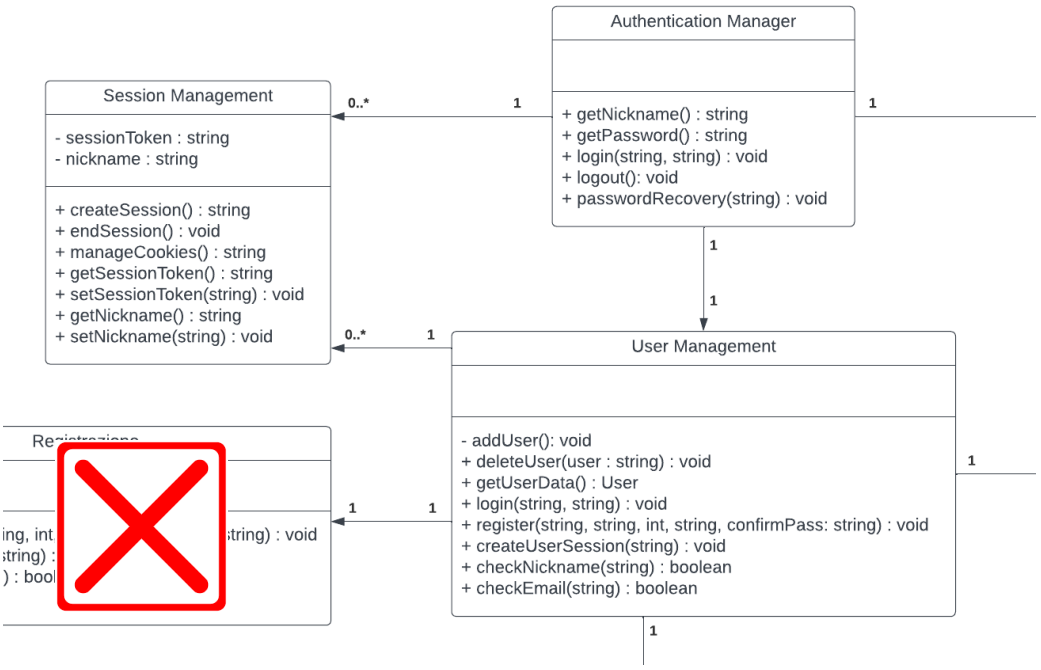
Diagramma delle Classi

Di seguito viene presentato il diagramma delle classi che costituisce l'architettura fondamentale del sistema su cui si basa l'implementazione. Nelle prossime sezioni, vengono fornite descrizioni dettagliate delle diverse componenti del diagramma, offrendo una panoramica chiara della struttura e delle relazioni all'interno del sistema.

Gestione autenticazione

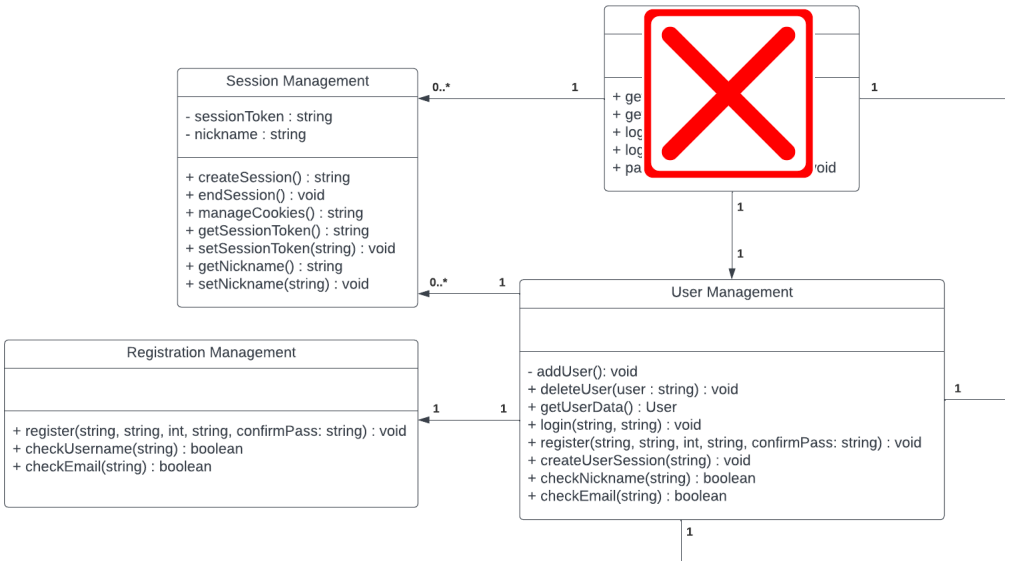
Dal diagramma dei componenti è stata identificata la classe *AutenticationManagement*. Questa classe si interfaccia con la sezione di sistema “Gestione dati utente”, gestita dalla classe *UserManagement*. Questa classe, dedicata all’autenticazione, controlla la correttezza delle credenziali inserite a livello formale ed evita che non vengano lasciati campi vuoti o che vengano inseriti caratteri non ammessi. Successivamente comunica le credenziali ricevute alla sezione Gestione Dati Utente, la quale controlla la correttezza dei dati interrogando il database. Se le credenziali sono corrette restituisce alla classe autenticazione un esito positivo, generando un token per la sessione dell’utente e l’autenticazione si conclude con successo. In caso contrario la classe invia un messaggio di errore da far visualizzare all’utente.



//VA TAGLIATA FUORI LA PARTE REGISTRAZIONE

Gestione registrazione

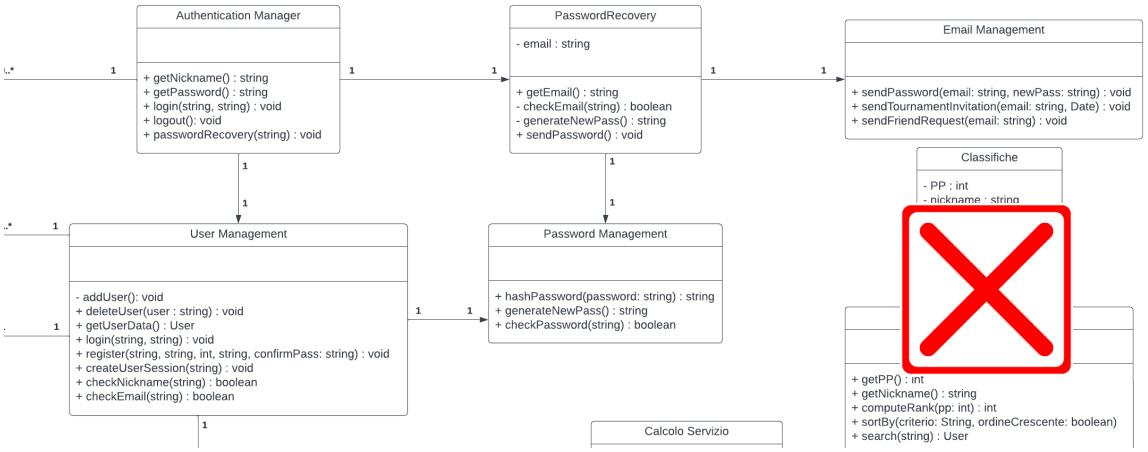
Dal diagramma dei componenti è stata identificata la classe *RegistrationManagement*, che si interfaccia con la sezione di sistema di “Gestione dati utente”, gestita dalla classe *UserManagement*. *RegistrationManagement* controlla la correttezza dei dati inseriti secondo i vari canoni specificati nel punto RNF 3.1, 3.2, 3.3 dell’analisi dei requisiti. Successivamente comunica i dati ricevuti alla sezione “*Gestione Dati Utente*”, la quale procede con l'inserimento nel database e la generazione di un codice hash associato alla password. Se l’inserimento va a buon fine, viene restituito alla classe di registrazione un esito positivo, rimandando l’utente alla pagina di login. In caso contrario, la classe invia un messaggio di errore da far visualizzare all’utente.



// TOGLIERE LA TABELLA MEZZA TAGLIATA SOPRA A DX

Recupero password

Dal diagramma dei componenti è stata identificata la classe *PasswordRecovery*, che si interfaccia con le sezioni di sistema “Gestione dati utente”, rappresentata dalla classe *UserManagement*, e “Gestione Mail”, individuato con la classe *EmailManagement*. Questa classe dedicata al recupero password viene utilizzata per generare in autonomia una nuova password da mandare poi all’utente che l’ha richiesta. La classe fornisce la nuova password e l’email a *EmailManagement*, la quale si occupa di fornire la nuova credenziale di accesso all’utente. Inoltre la fornisce anche a *UserManagement* tramite *PasswordManagement*, che a sua volta aggiorna i dati presenti sul database.



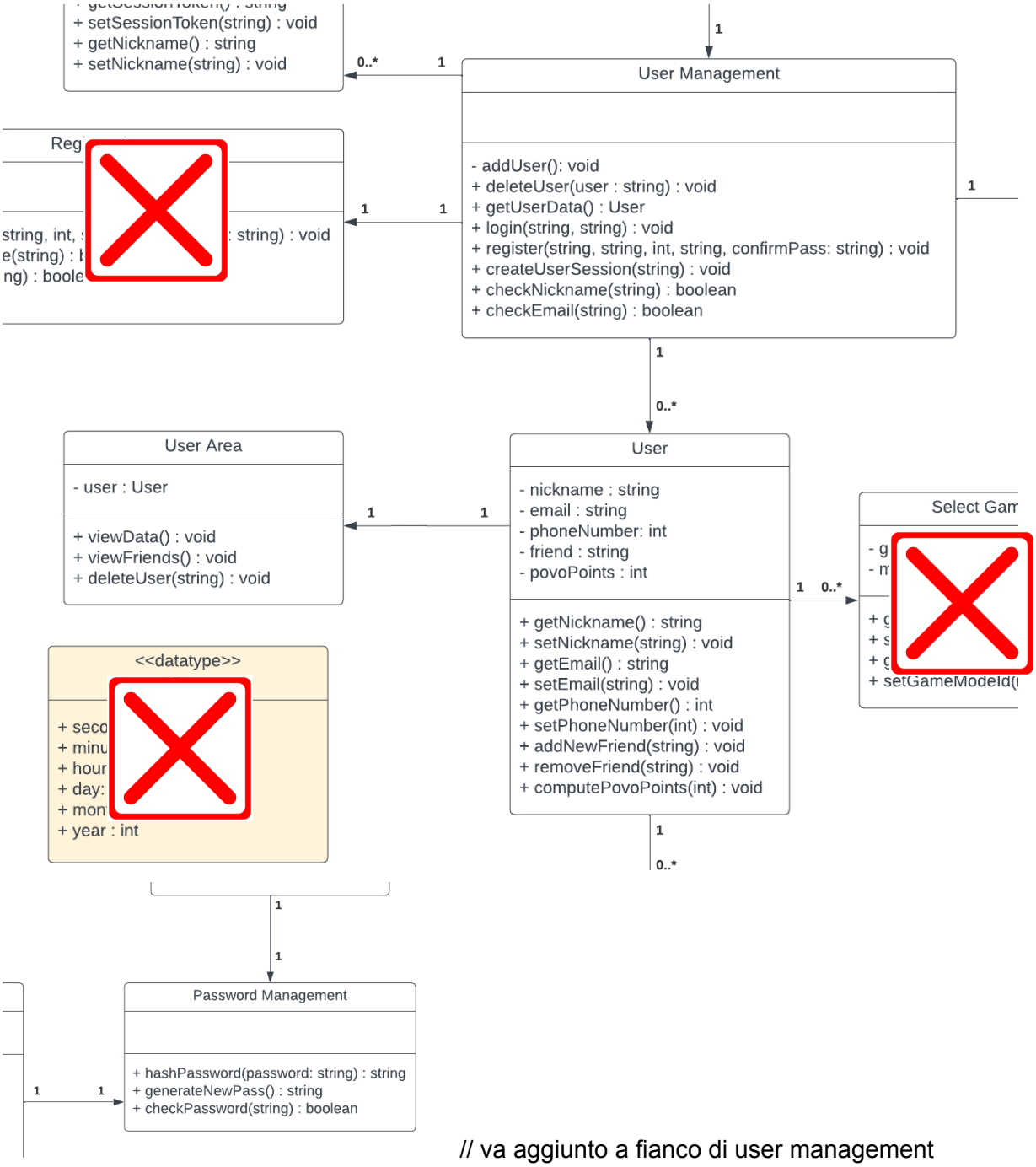
// TOGLIERE CLASSIFIHCE E LE ALTRE CLASSI CHE NON SERVONO

Gestione profilo utente

Dal diagramma dei componenti è stata identificata la classe *UserArea*, che si interfaccia con la sezione del sistema “Gestione Dati Utente” e con la sezione dedicata alla modifica password “Interfaccia Modifica Password”. Tramite questa gestione, l’utente può compiere diverse azioni relative al proprio profilo, quali visualizzare e modificarne i dati o eliminare il proprio profilo

Nel caso in cui si desideri modificare la password, l’utente viene indirizzato verso l’interfaccia dedicata, gestita dalla classe *PasswordManagement*, che gli permetterà di effettuare la modifica di quest’ultima con gli opportuni controlli.

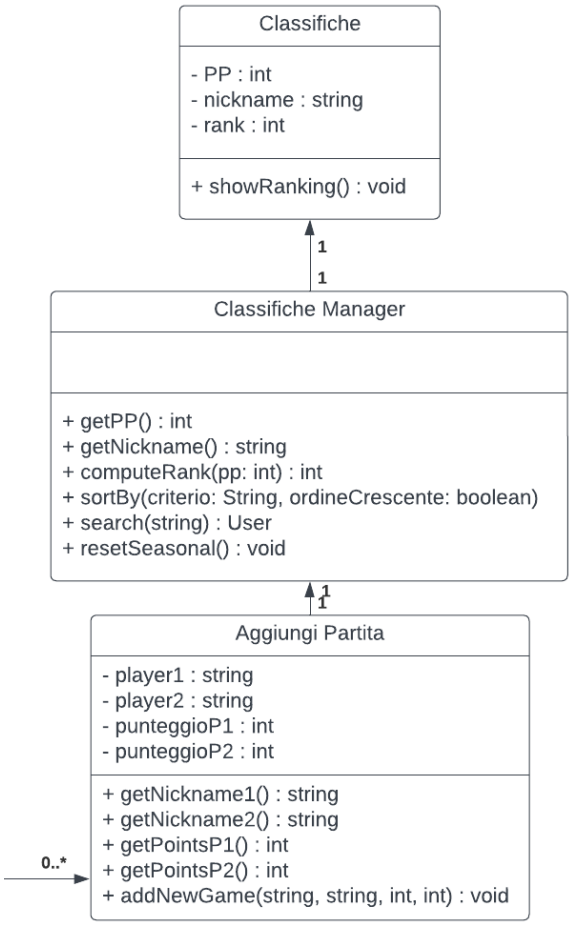
Per tutte le altre azioni invece la classe comunicherà l’operazione da eseguire alla classe *UserManagement*, che ne restituirà l’esito. Si precisa che per la modifica dei dati dell’utente, la classe invierà i nuovi dati per aggiornare quelli attuali.



// va aggiunto a fianco di user management

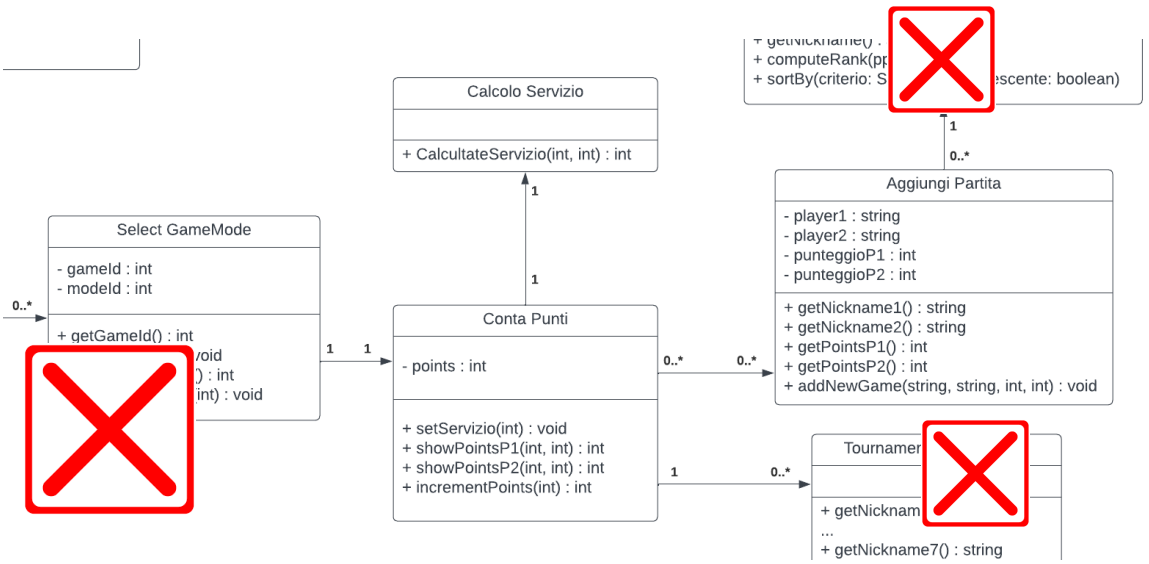
GESTIONE CLASSIFICHE

Dal diagramma dei componenti, è stata individuata la classe *Classifiche*, gestita dalla classe *ClassificheManager*. Quest'ultima è responsabile di ricevere i dati della partita di un giocatore e di passarli alla classe "Classifiche" per consentirne la visualizzazione. Tramite la classe *AggiungiPartita*, l'utente ha la possibilità di aggiungere una partita classificata al sistema. Ciò influisce sulle classifiche e attiva di conseguenza *ClassificheManager*, il quale preleva i dati e modifica le classifiche di conseguenza.
Nel caso di una partita non classificata o libera, *ClassificheManager* ha la sola funzione di prelevare i dati senza eseguire calcoli per i cambiamenti di posizione o Punti di Prestigio (PP) nelle classifiche.



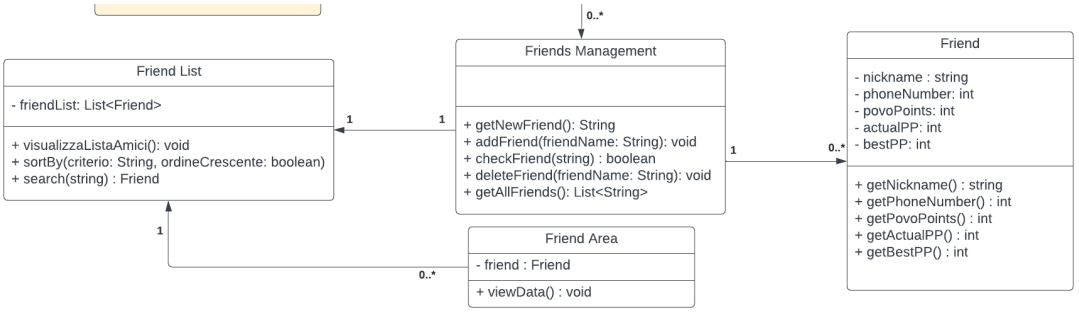
GESTIONE INTERFACCIA CONTEGGIO PUNTI

Dal diagramma dei componenti, è emersa l'identificazione della classe *ContaPunti*, accessibile dal menù principale della web-app. Questa classe varia a seconda dei valori passati dalla classe *SelectGameMode*, che rappresenta la modalità di gioco (11 o 21 punti).
Successivamente, la classe ContaPunti interagisce con la classe *CalcoloServizio*, la quale restituisce un valore per impostare il "servizio". Al termine di una partita classificata, la classe *ContaPunti* può interfacciarsi con la classe *AggiungiPartita*, fornendo il punteggio dei due giocatori.
Questo flusso di interazione tra le classi consente una gestione dinamica del processo di conteggio dei punti in base alla modalità di gioco selezionata, nonché l'aggiornamento dei risultati e delle statistiche attraverso l'interazione con la classe *AggiungiPartita*.



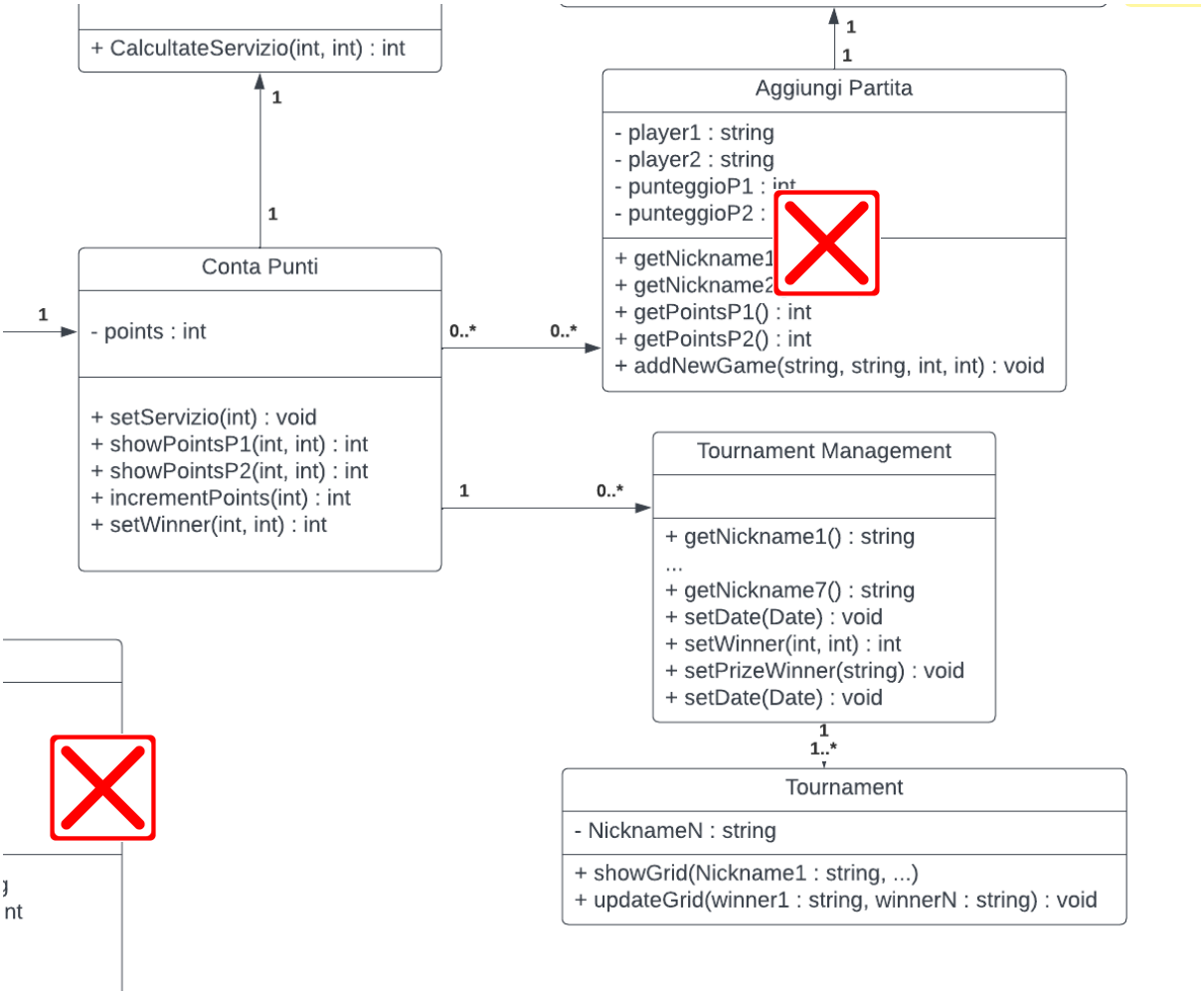
GESTIONE AMICIZIE

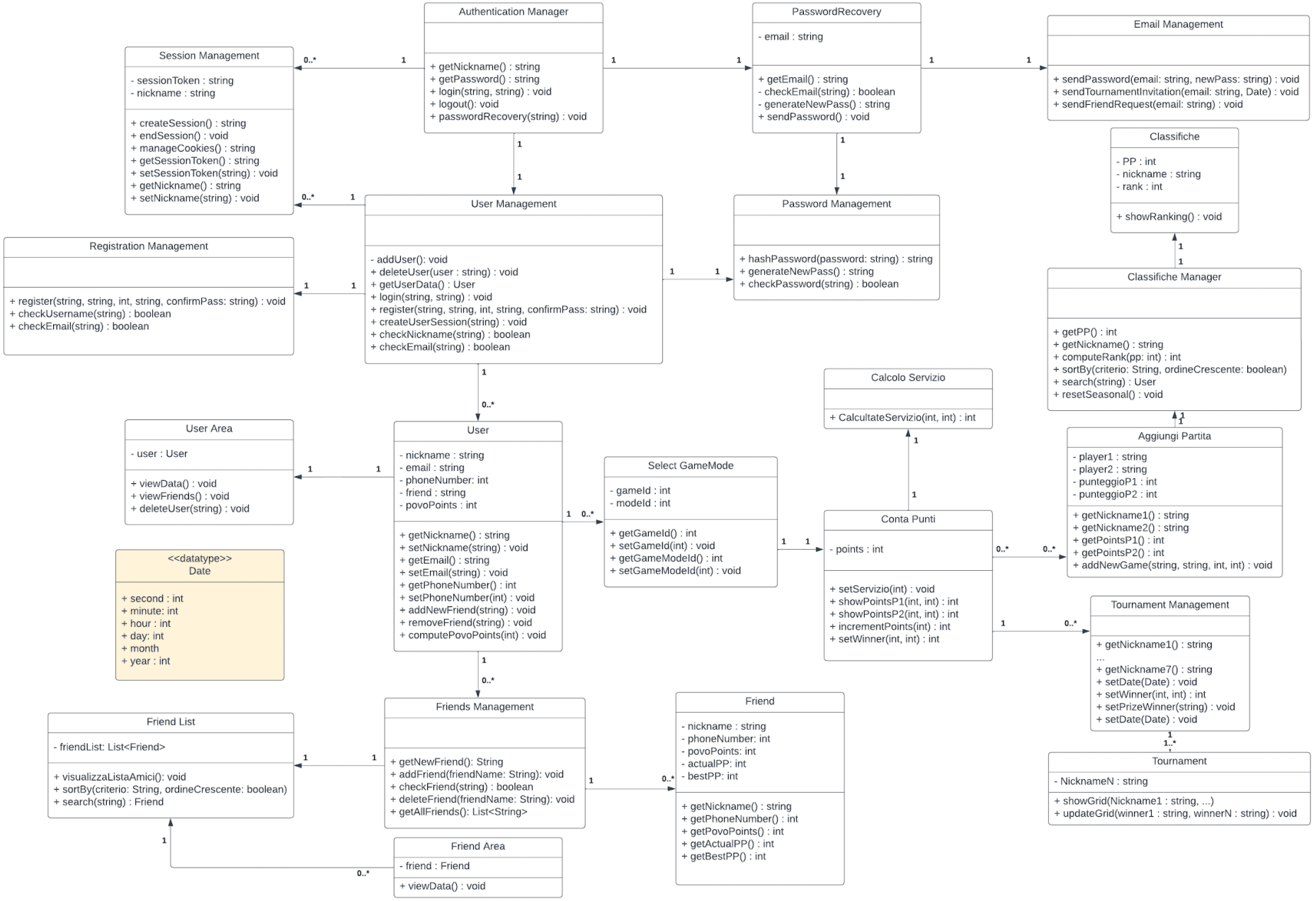
Dall'analisi del diagramma dei componenti, è stata individuata la classe *FriendsManager* che interagisce con le classi *Friend* e *FriendList*. *FriendsManager* è responsabile di gestire l'aggiunta e la rimozione di un utente dalla lista degli amici, interagendo con la classe *Friend* che raccoglie tutti i dati relativi a un amico. Successivamente, questi dati vengono passati alla classe *FriendList* che si occupa di gestire l'insieme completo di amici.
Per quanto riguarda la visualizzazione dei dati degli amici, "FriendList" si appoggia alla classe *FriendArea*, che presenta all'utente le informazioni dettagliate riguardanti i propri amici.
In sintesi, il flusso di interazione coinvolge *FriendsManager* per la gestione dell'elenco degli amici, *Friend* per la raccolta dei dati specifici di un amico e *FriendList* per la gestione complessiva della lista.



GESTIONE TORNEI

Dal diagramma dei componenti è stata identificata la classe *Tournament Management* che ha lo scopo di gestire la funzionalità torneo della web-app. La classe interagisce con *Conta Punti* per decretare il vincitore di una partita e interagisce con la classe *Tournament* per lo svolgimento vero e proprio del torneo. In particolare *Tournament* permette all'utente la visualizzazione del tabellone e aggiorna quest'ultimo dopo ogni partita.

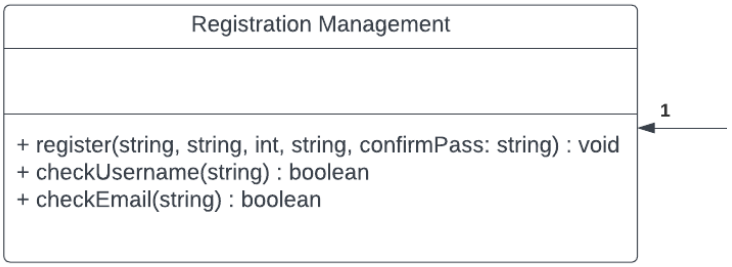



Authentication Manager
+ getNickname() : string
+ getPassword() : string
+ login(string, string) : void
+ logout(): void
+ passwordRecovery(string) : void
Session Management
- sessionToken : string
- nickname : string
+ createSession() : string
+ endSession() : void
+ manageCookies() : string
+ getSessionToken() : string
+ setSessionToken(string) : void
+ getNickname() : string
+ setNickname(string) : void
PasswordRecovery
- email : string
+ getEmail() : string
- checkEmail(string) : boolean
- generateNewPass() : string
+ sendPassword() : void
Email Management
+ sendPassword(email: string, newPass: string) : void
+ sendTournamentInvitation(email: string, Date) : void
+ sendFriendRequest(email: string) : void
User Management
- addUser(): void
+ deleteUser(user : string) : void
+ getUserData() : User
+ login(string, string) : void
+ register(string, string, int, string, confirmPass: string) : void
+ createUserSession(string) : void
+ checkNickname(string) : boolean
+ checkEmail(string) : boolean
Password Management
+ hashPassword(password: string) : string
+ generateNewPass() : string
+ checkPassword(string) : boolean
Registration Management
+ register(string, string, int, string, confirmPass: string) : void
+ checkUsername(string) : boolean
+ checkEmail(string) : boolean
Classifiche
- PP : int
- nickname : string
- rank : int
+ showRanking() : void
Classifiche Manager
+ getPP() : int
+ getNickname() : string
+ computeRank(pp: int) : int
+ sortBy(criterio: String, ordineCrescente: boolean)
+ search(string) : User
+ resetSeasonal() : void
Calcolo Servizio
+ CalcultateServizio(int, int) : int
User Area
- user : User
+ viewData() : void
+ viewFriends() : void
+ deleteUser(string) : void
User
- nickname : string
- email : string
- phoneNumber: int
- friend : string
- povoPoints : int
+ getNickname() : string
+ setNickname(string) : void
+ getEmail() : string
+ setEmail(string) : void
+ getPhoneNumber() : int
+ setPhoneNumber(int) : void
+ addNewFriend(string) : void
+ removeFriend(string) : void
+ computePovoPoints(int) : void
Select GameMode
- gameId : int
- modeId : int
+ getGameId() : int
+ setGameId(int) : void
+ getGameModeId() : int
+ setGameModeId(int) : void
Conta Punti
- points : int
+ setServizio(int) : void
+ showPointsP1(int, int) : int
+ showPointsP2(int, int) : int
+ incrementPoints(int) : int
+ setWinner(int, int) : int
Aggiungi Partita
- player1 : string
- player2 : string
- punteggioP1 : int
- punteggioP2 : int
+ getNickname1() : string
+ getNickname2() : string
+ getPointsP1() : int
+ getPointsP2() : int
+ addNewGame(string, string, int, int) : void
<<datatype>>
Date
+ second : int
+ minute: int
+ hour : int
+ day: int
+ month
+ year : int
Tournament Management
+ getNickname1() : string
...
+ getNickname7() : string
+ setDate(Date) : void
+ setWinner(int, int) : int
+ setPrizeWinner(string) : void
+ setDate(Date) : void
Tournament
- NicknameN : string
+ showGrid(Nickname1 : string, ...)
+ updateGrid(winner1 : string, winnerN : string) : void
Friend List
- friendList: List<Friend>
+ visualizzaListaAmici(): void
+ sortBy(criterio: String, ordineCrescente: boolean)
+ search(string) : Friend
Friends Management
+ getNewFriend(): String
+ addFriend(friendName: String): void
+ checkFriend(string) : boolean
+ deleteFriend(friendName: String): void
+ getAllFriends(): List<String>
Friend
- nickname : string
- phoneNumber: int
- povoPoints: int
- actualPP: int
- bestPP: int
+ getNickname() : string
+ getPhoneNumber() : int
+ getPovoPoints() : int
+ getActualPP() : int
+ getBestPP() : int
Friend Area
- friend : Friend
+ viewData() : void

CODICE IN OBJECT CONSTRAINT LANGUAGE

Gestione registrazione

Le seguenti condizioni si applicano sulla classe *RegistrationManagement*.



Nel form di registrazione, viene richiesto obbligatoriamente di inserire la password in due campi diversi:
- Un campo in cui viene richiesto l'inserimento della nuova password
- Un campo in cui viene richiesto di riscrivere la stessa password appena scelta,
per confermare che non ci siano errori di battitura nella prima.
Il form, quindi, è completabile solo se la password coincide in entrambi i campi. Ciò viene espresso in OCL attraverso una precondizione scritto nella forma seguente:

```
context RegistrationManagement :: register(string,
string, int, password : string, confirmPass : string)
pre : password = confirmPassword
```

La password deve inoltre rispettare le regole di formattazione imposte nel punto RNF 4: - lunghezza di almeno 8 caratteri
- almeno una lettera minuscola
- almeno una lettera maiuscola
- almeno un numero
- almeno un carattere speciale tra quelli individuati nella tabella ASCII
nell'intervallo [33, 47]
In OCL viene rappresentato mediante la precondizione:

```
context RegistrationManagement :: register(string,
string, int, password : string, confirmPass : string)
pre : password.exists (c | c =
/^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*\W)(?!.*
).{8,16}$/)
```

Nella registrazione la email deve rispettare i seguenti vincoli. In OCL vengono rappresentati mediante la precondizione:

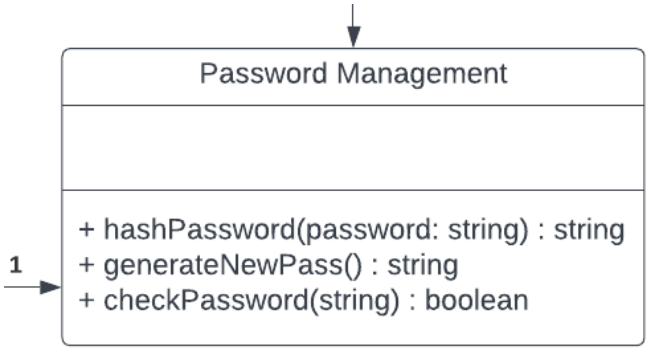
```
context RegistrationManagement :: register(email : string,
string, int, string, string )
pre : email.exists (c | c =
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$)
```

Inoltre, come specificato nel punto RNF3.1, RNF3.2 dei requisiti, username e email devono essere univoci, ovvero non devono essere già associati ad un altro account. Ciò viene espresso in OCL tramite la precondizione seguente:

```
context RegistrationManagement :: register(string,
string, int, password : string, confirmPass : string)
pre : checkUsername(username) = FALSE &
checkEmail(Email) = FALSE
```

Gestione Password

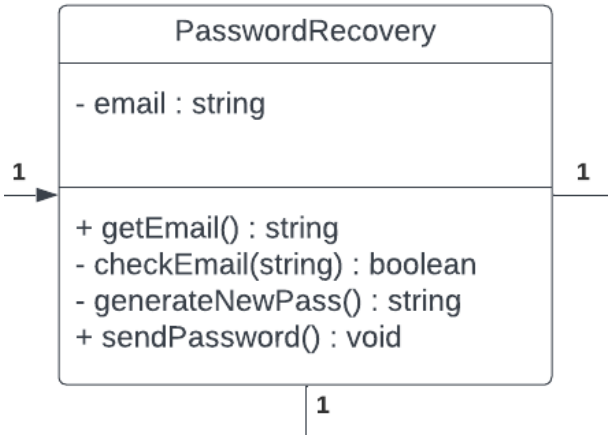
Il vincolo espresso in linguaggio OCL assicura che la chiamata al metodo generateNewPass restituisca una password non vuota, contribuendo a garantire che la generazione di una nuova password produca un valore significativo e utilizzabile.



```
context PasswordManagement ::
generateNewPass() : String
post: not result.isEmpty()
```

Recupero della password

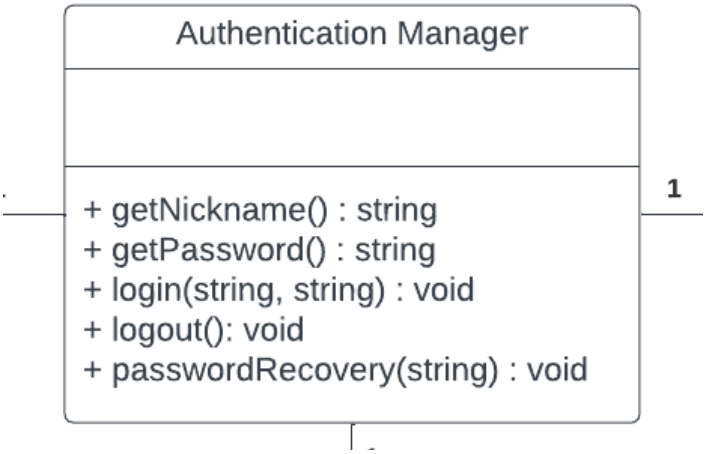
Il recupero della password richiede all'utente l'inserimento della propria e-mail. Per evitare di inviare password a indirizzi e-mail non corretti, la e-mail inserita dall'utente deve essere presente nel database ed essere associata ad un utente.
Ciò viene descritto in OCL dalla precondizione:




```
context passwordRecovery :: sendPassword()
pre : checkEmail(Email) = TRUE
```

Autenticazione

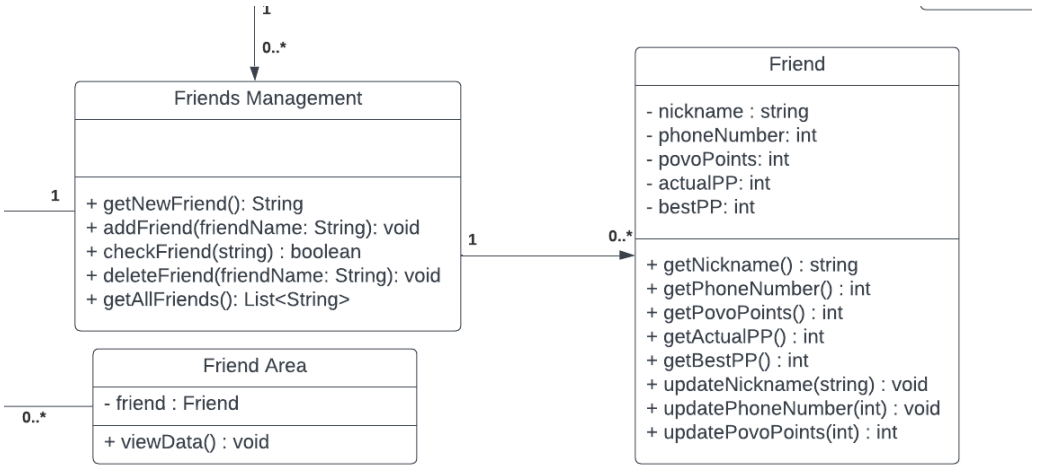
Per garantire che non ci siano ambiguità durante la fase di accesso è espresso in OCL il seguente vincolo per impedire il login se la password recovery è attiva.




```
context AuthenticationManager
inv: self.isPasswordRecoveryRequested('username')
implies not self.isAuthenticated('username')
```

Amicizie

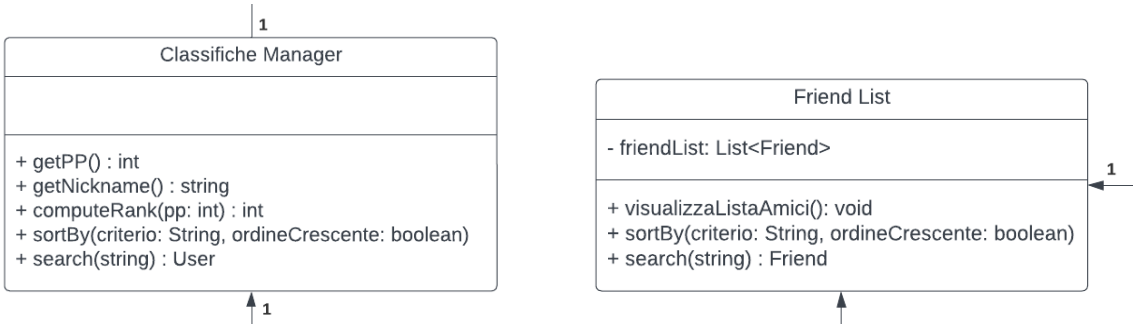
Le amicizie si stringono prendendo un Nickname dal database e inserendolo nell'apposita lista amici. Il sistema per aggiungere/eliminare un amico controlla prima se l'amico è nella lista di amici e poi applica l'azione desiderata. Per evitare di incorrere in problemi dove un amico è già stato aggiunto, oppure un utente non è tra gli amici, vi sono dei vincoli creati in OCL:



```
context FriendList
inv: friends->forAll(f1, f2 | f1 <> f2 implies f1 <>
newFriend)
```

Ordinamento Classifiche e Amici

Le classifiche e la lista di amici sono visualizzabili tramite metodi gestiti da *sortBy* che prende in input il criterio (stringa di caratteri) e seleziona l'ordine crescente/descrescente tramite un parametro boolean.



Le condizioni dell'input sono espresse in OCL attraverso le seguenti precondizioni:

```
context ClassificheManager :: sortBy(string :
criterio, ordineCrescente : boolean)
pre : criterio = "PP" OR criterio =
"Vittorie/Sconfitte" OR criterio = "A-Z"
```

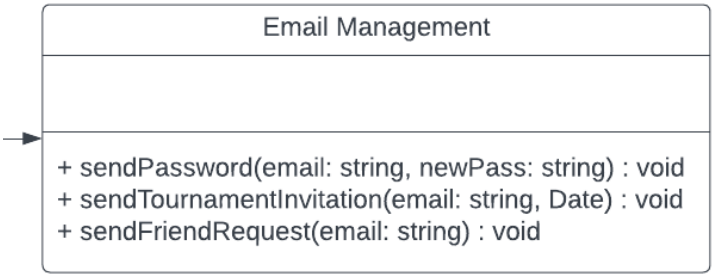
```
context ClassificheManager :: sortBy(string :
criterio, ordineCrescente : boolean)
pre : ordineCrescente = TRUE OR
ordineCrescente = FALSE
```

Per garantire che il reset stagionale avvenga con precisione ogni 6 mesi, e non anzitempo, si è specificato tale vincolo in OCL:

```
context ClassificheManager :: resetSeasonal() : void
inv: self.resetSeasonal() implies
self.getLastSeasonalResetDate().addMonths(6) <=
Date.today()
```

Servizio Email

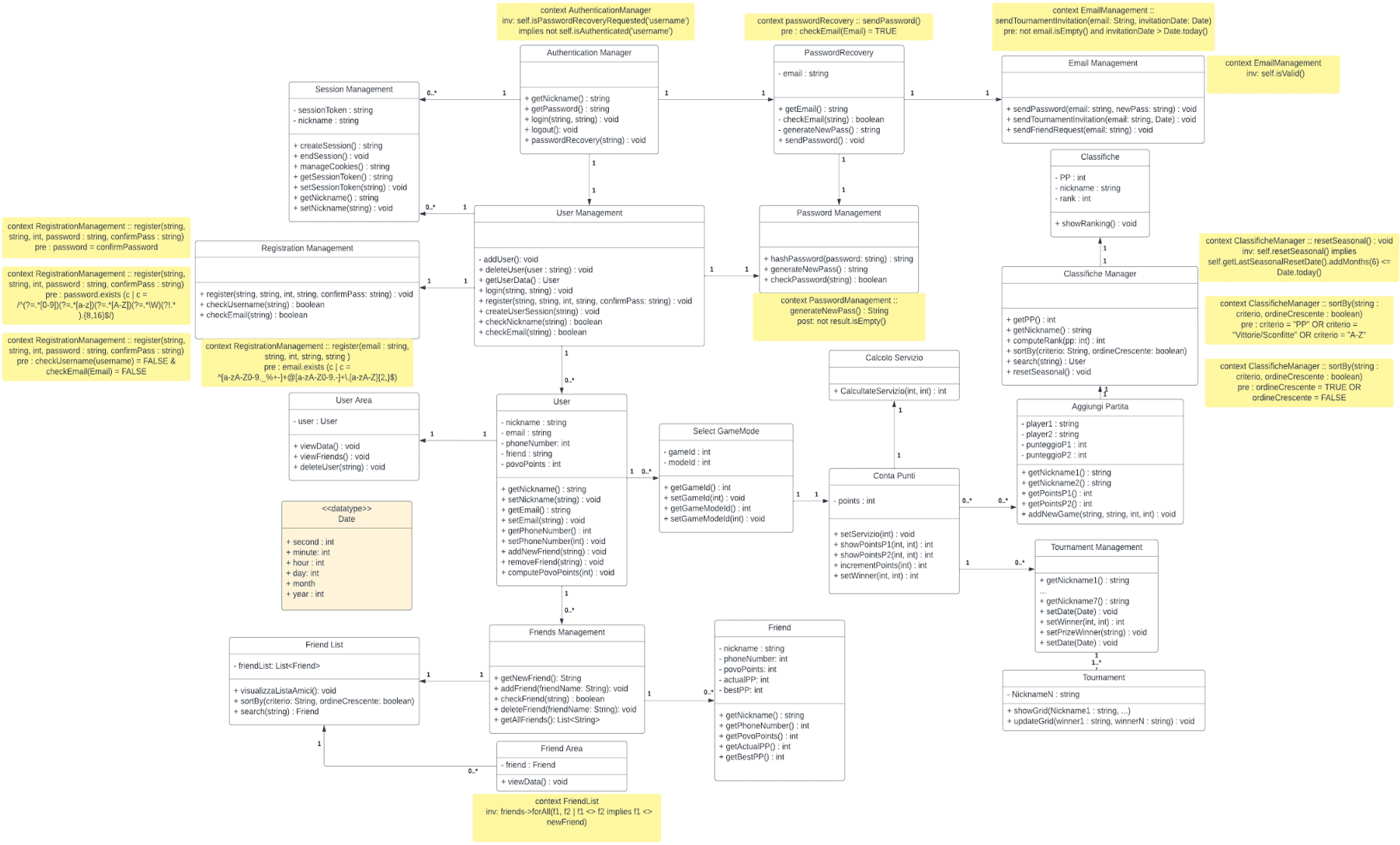
Questo vincolo assicura che la chiamata a *sendTournamentInvitation* riceva un indirizzo email non vuoto e una data per gli inviti/promemoria che sia nel futuro.




```
context EmailManagement ::
sendTournamentInvitation(email: String, invitationDate: Date)
  pre: not email.isEmpty() and invitationDate > Date.today()
```

Inoltre per garantire che il servizio mail sia attivo e che il setup della mail sia avvenuto con successo il seguente vincolo è specificato in OBL:

```
context EmailManagement
  inv: self.isValid()
```


context AuthenticationManager
inv: self.isPasswordRecoveryRequested('username') implies not self.isAuthenticated('username')
context passwordRecovery :: sendPassword()
pre : checkEmail(Email) = TRUE
context EmailManagement :: sendTournamentInvitation(email: String, invitationDate: Date)
pre: not email.isEmpty() and invitationDate > Date.today()
Authentication Manager
+ getNickname() : string
+ getPassword() : string
+ login(string, string) : void
+ logout(): void
+ passwordRecovery(string) : void
PasswordRecovery
- email : string
+ getEmail() : string
- checkEmail(string) : boolean
- generateNewPass() : string
+ sendPassword() : void
Email Management
+ sendPassword(email: string, newPass: string) : void
+ sendTournamentInvitation(email: string, Date) : void
+ sendFriendRequest(email: string) : void
context EmailManagement
inv: self.isValid()
Session Management
- sessionToken : string
- nickname : string
+ createSession() : string
+ endSession() : void
+ manageCookies() : string
+ getSessionToken() : string
+ setSessionToken(string) : void
+ getNickname() : string
+ setNickname(string) : void
User Management
- addUser(): void
+ deleteUser(user : string) : void
+ getUserData() : User
+ login(string, string) : void
+ register(string, string, int, string, confirmPass: string) : void
+ createUserSession(string) : void
+ checkNickname(string) : boolean
+ checkEmail(string) : boolean
Password Management
+ hashPassword(password: string) : string
+ generateNewPass() : string
+ checkPassword(string) : boolean
context PasswordManagement :: generateNewPass() : String
post: not result.isEmpty()
Classifiche
- PP : int
- nickname : string
- rank : int
+ showRanking() : void
context ClassificheManager :: resetSeasonal() : void
inv: self.resetSeasonal() implies self.getLastSeasonalResetDate().addMonths(6) <= Date.today()
Classifiche Manager
+ getPP() : int
+ getNickname() : string
+ computeRank(pp: int) : int
+ sortBy(criterio: String, ordineCrescente: boolean)
+ search(string) : User
+ resetSeasonal() : void
context ClassificheManager :: sortBy(string : criterio, ordineCrescente : boolean)
pre : criterio = "PP" OR criterio = "Vittorie/Sconfitte" OR criterio = "A-Z"
context ClassificheManager :: sortBy(string : criterio, ordineCrescente : boolean)
pre : ordineCrescente = TRUE OR ordineCrescente = FALSE
context RegistrationManagement :: register(string, string, int, password : string, confirmPass : string)
pre : password = confirmPassword
context RegistrationManagement :: register(string, string, int, password : string, confirmPass : string)
pre : password.exists (c | c = /^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*\W)(?!.*).{8,16}$/)
context RegistrationManagement :: register(string, string, int, password : string, confirmPass : string)
pre : checkUsername(username) = FALSE & checkEmail(Email) = FALSE
Registration Management
+ register(string, string, int, string, confirmPass: string) : void
+ checkUsername(string) : boolean
+ checkEmail(string) : boolean
context RegistrationManagement :: register(email : string, string, int, string, string)
pre : email.exists (c | c = ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$)
User Area
- user : User
+ viewData() : void
+ viewFriends() : void
+ deleteUser(string) : void
User
- nickname : string
- email : string
- phoneNumber: int
- friend : string
- povoPoints : int
+ getNickname() : string
+ setNickname(string) : void
+ getEmail() : string
+ setEmail(string) : void
+ getPhoneNumber() : int
+ setPhoneNumber(int) : void
+ addNewFriend(string) : void
+ removeFriend(string) : void
+ computePovoPoints(int) : void
Select GameMode
- gameId : int
- modeId : int
+ getGameId() : int
+ setGameId(int) : void
+ getGameModeId() : int
+ setGameModeId(int) : void
Calcolo Servizio
+ CalcultateServizio(int, int) : int
Conta Punti
- points : int
+ setServizio(int) : void
+ showPointsP1(int, int) : int
+ showPointsP2(int, int) : int
+ incrementPoints(int) : int
+ setWinner(int, int) : int
Aggiungi Partita
- player1 : string
- player2 : string
- punteggioP1 : int
- punteggioP2 : int
+ getNickname1() : string
+ getNickname2() : string
+ getPointsP1() : int
+ getPointsP2() : int
+ addNewGame(string, string, int, int) : void
<<datatype>>
Date
+ second : int
+ minute: int
+ hour : int
+ day: int
+ month
+ year : int
Tournament Management
+ getNickname1() : string
...
+ getNickname7() : string
+ setDate(Date) : void
+ setWinner(int, int) : int
+ setPrizeWinner(string) : void
+ setDate(Date) : void
Tournament
- NicknameN : string
+ showGrid(Nickname1 : string, ...)
+ updateGrid(winner1 : string, winnerN : string) : void
Friend List
- friendList: List<Friend>
+ visualizzaListaAmici(): void
+ sortBy(criterio: String, ordineCrescente: boolean)
+ search(string) : Friend
Friends Management
+ getNewFriend(): String
+ addFriend(friendName: String): void
+ checkFriend(string) : boolean
+ deleteFriend(friendName: String): void
+ getAllFriends(): List<String>
Friend
- nickname : string
- phoneNumber: int
- povoPoints: int
- actualPP: int
- bestPP: int
+ getNickname() : string
+ getPhoneNumber() : int
+ getPovoPoints() : int
+ getActualPP() : int
+ getBestPP() : int
Friend Area
- friend : Friend
+ viewData() : void
context FriendList
inv: friends->forAll(f1, f2 | f1 <> f2 implies f1 <> newFriend)

//TODO: AGGIUNGERE RIFERIMENTI A REQUISITI
//CAPIRE SE MATEJ METTE O NO IL LOGOUT