



**Document:** *PartFinder\_Architettura*

**Revision:** *0.1*



**UNIVERSITY  
OF TRENTO**

Dipartimento di Ingegneria e Scienza  
dell'Informazione

**Progetto:**

## PartFinder

**Titolo del documento:**

## Documento di Architettura

### Document Info

Doc. Name	<i>D3-PartFinder_Architettura</i>	Doc. Number	D3
Description	Il documento include diagrammi delle classi e codice in OCL		

# Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto PartFinder usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definito il diagramma delle classi.

## 1. Diagramma delle classi

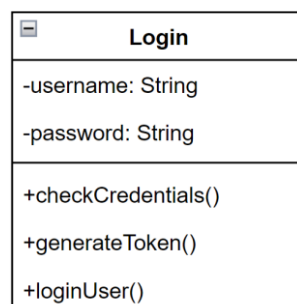
Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto PartFinder. Tutte le classi individuate sono caratterizzate da un nome, una lista di attributi che identificano i dati gestiti dalla classe e una lista di metodi che definiscono le operazioni previste all'interno della classe. Ogni classe può essere anche associata ad altre classi e, tramite questa associazione, è possibile fornire informazioni su come le classi si relazionano tra loro.

Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.

### 1.1. Gestione autenticazione

Dal diagramma di contesto e dei componenti viene individuata la classe *Account* che rappresenta l'account che un utente crea per poter accedere alle funzionalità del sistema PartFinder.

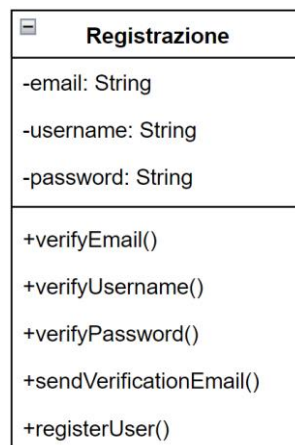
Per poter accedere al proprio account l'utente deve inserire le proprie credenziali. La memorizzazione delle credenziali viene gestita da un sistema subordinato, quindi la classe *Login* si occuperà dell'inserimento delle credenziali e dell'invio di esse al sistema subordinato, con conseguente risposta. In caso di risposta positiva, viene generato un Token di autenticazione.



**Figura 1. Classe per gestione login**

## 1.2. Registrazione account

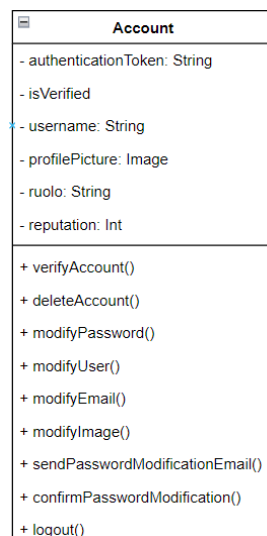
La classe registrazione gestisce il caso in cui l'utente non abbia ancora creato un account. La classe si occupa dell'inserimento delle credenziali e della verifica della corretta formattazione e dell'aderimento agli standard di sicurezza. Una volta effettuata la verifica verrà richiesto al sistema subordinato incaricato della gestione e memorizzazione degli account l'inserimento di uno nuovo con le credenziali specificate. All'utente verrà inviata un'email contenente un link per la verifica dell'account.



**Figura 2. Classe per registrazione Account**

## 1.3. Pagina gestione Account

La classe *Account* prevede tutti i metodi per la gestione di un account creato da un utente, come la modifica dell'email, dello username, dell'immagine di profilo e della password. In quest'ultimo caso, per aggiungere un livello di sicurezza, viene richiesta un'ulteriore conferma via email.

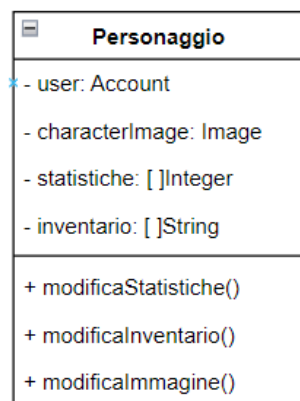


**Figura 3. Classi per gestione Account**

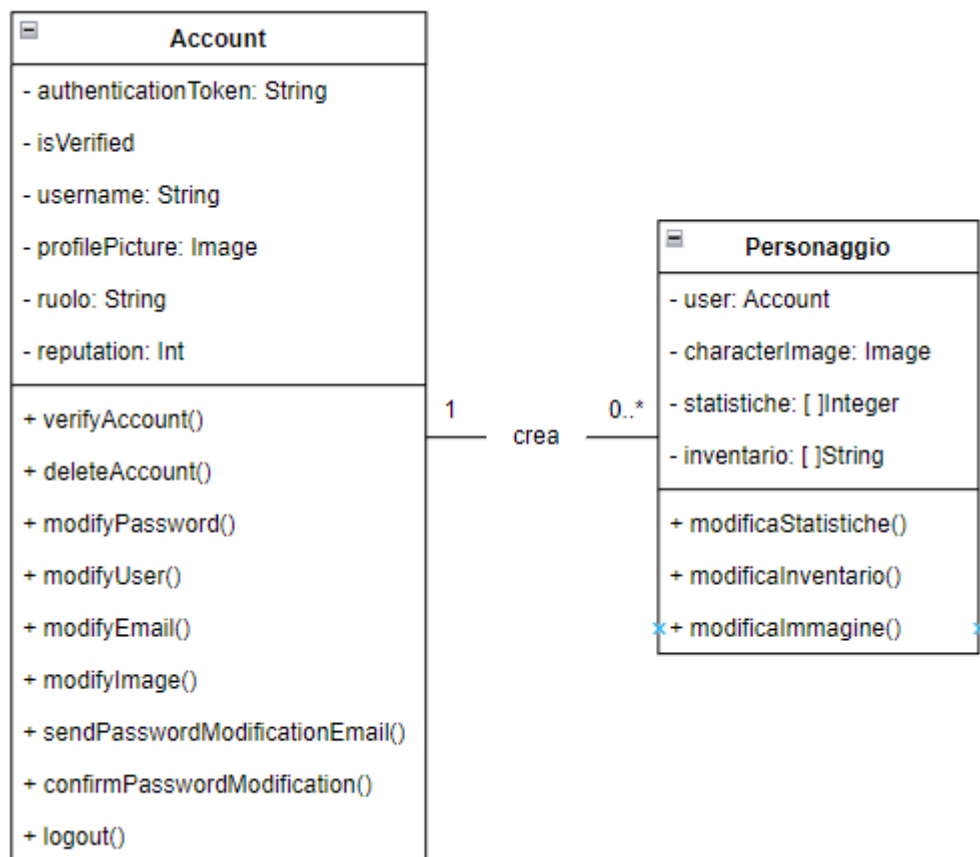
### 1.4. Pagina personaggi

Nel sistema PartFinder, ogni utente per partecipare a un gruppo deve creare prima un personaggio. Nella classe personaggio vengono specificate tutte le caratteristiche di un personaggio, come le statistiche, l'inventario e l'immagine.

La classe personaggio comunica ogni modifica al sistema subordinato MongoDB.

**Figura 4. Classe Personaggio**

Ovviamente, ogni personaggio deve essere collegato a un Account, quindi la relazione si presenta come in Figura 5.

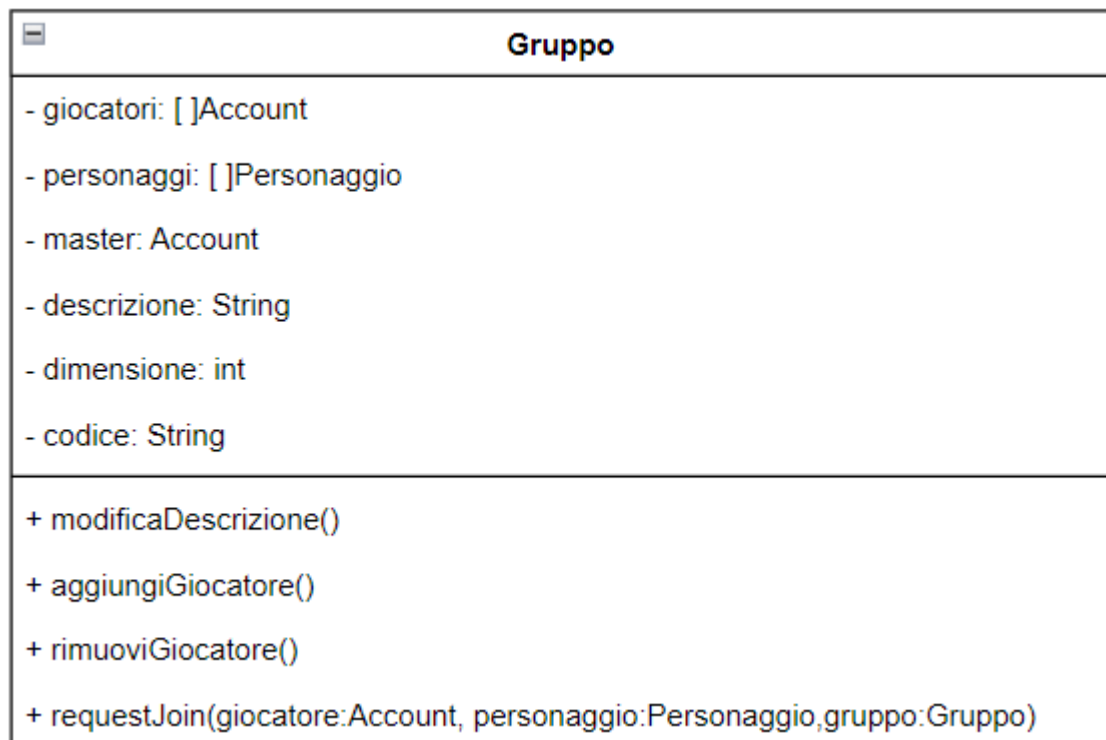


**Figura 5. Relazione tra classe Account e Personaggio**

## 1.5. Gestione dei gruppi

Il sistema PartFinder si basa sulla partecipazione di utenti a gruppi creati da altri utenti, da qui la necessità di creare una classe *Gruppo* per la gestione.

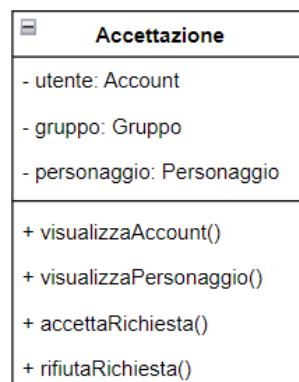
Per ogni gruppo, alla creazione, bisogna specificare la Descrizione e la dimensione del gruppo. Il creatore viene nominato il Master del gruppo. Per ogni gruppo viene generato in automatico un codice identificativo. Tutte queste informazioni vengono comunicate al sistema subordinato MongoDB che si occupa della memorizzazione e gestione delle informazioni.



**Figura 6. Classe Gruppo**

## 1.6. Gestione accettazione

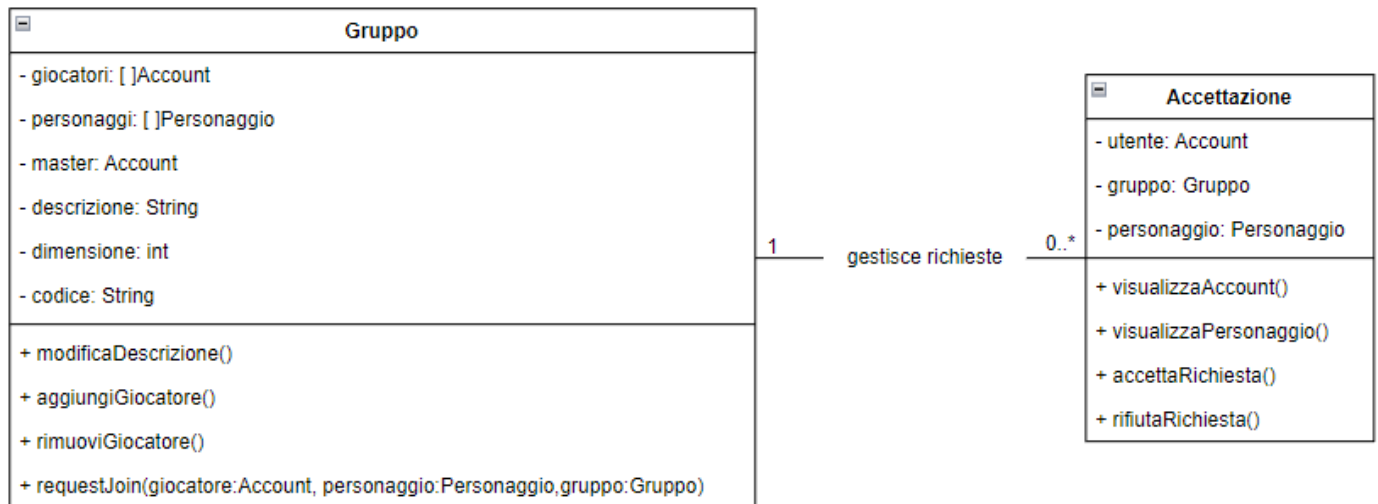
Per potersi unire a un gruppo un giocatore deve inviare attraverso il sistema PartFinder una richiesta al Master del gruppo. Per la gestione di queste richieste è stato individuato nel diagramma dei componenti un componente separato. Nel diagramma delle classi viene creata una classe denominata *Accettazione*.



**Figura 7. Classe Accettazione**

La classe accettazione ha in sé i metodi con cui il Master può gestire ogni richiesta di unione, dalla visualizzazione del profilo del mittente, alla visualizzazione del personaggio con cui vuole partecipare alla vera e propria accettazione(o rifiuto) della richiesta.

La classe *Accettazione* è collegata alla classe *Gruppo* come mostrato in Figura 8.



**Figura 8. Relazione tra la classe Gruppo e Accettazione**

## 1.7. Gestione delle sessioni

Nel sistema PartFinder è previsto che, una volta uniti a un gruppo, i giocatori partecipino alle partite(o "sessioni") di questo.

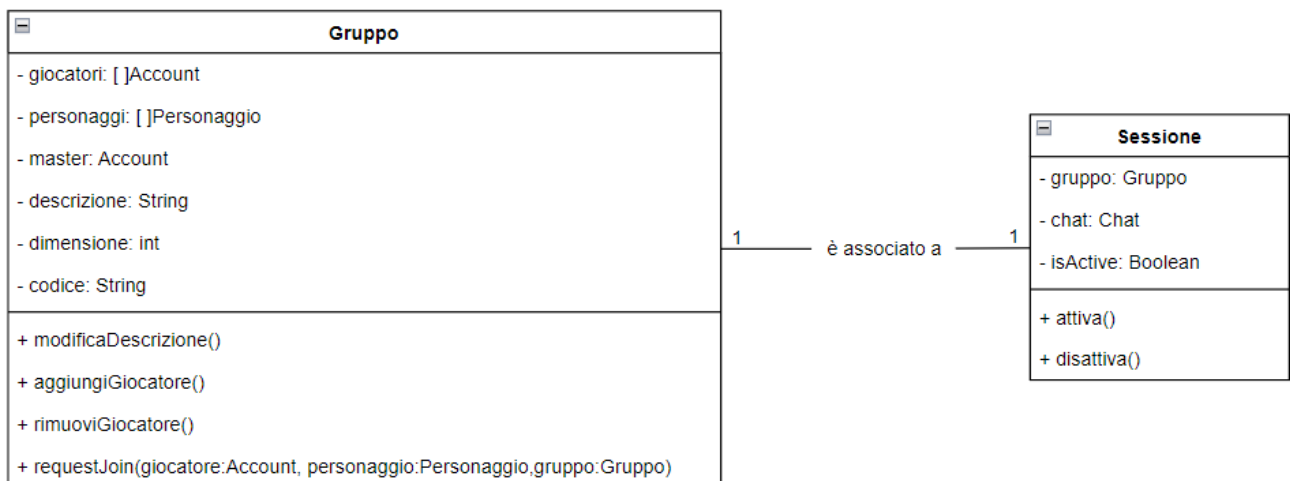
Viene quindi creata la classe Sessione a partire dall'omonimo componente individuato nel diagramma dei componenti.



**Figura 9. Classe Sessione**

All'interno della classe Sessione sono presenti due metodi utilizzabili dal Master per attivare e disattivare la sessione. Attivare la sessione significa dare inizio alla partita, e questo sarà notificato ai giocatori facenti parte del gruppo quando accedono al sistema PartFinder.

Ogni sessione è associata a un solo gruppo e viceversa, come mostrato in Figura 10.



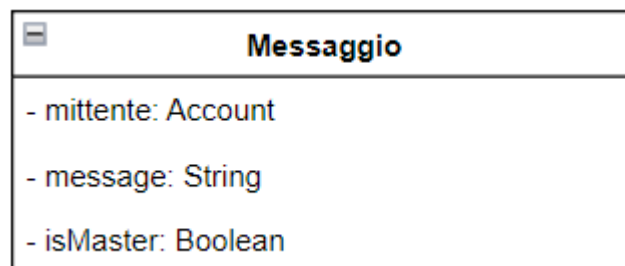
**Figura 10. Relazione tra Gruppo e Sessione**

## 1.8. Gestione della chat

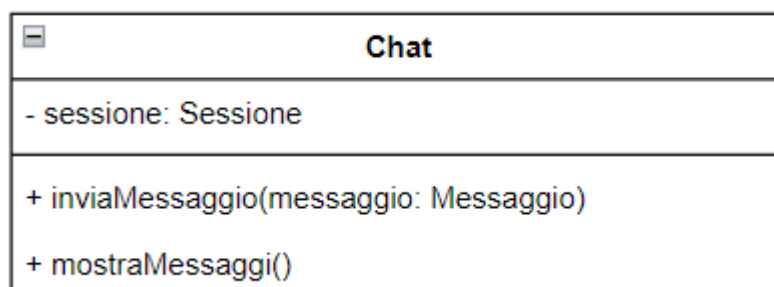
Nel sistema PartFinder, in ogni sessione è presente un sistema di chat testuale per permettere ai giocatori di comunicare.

Dato che nel diagramma di contesto è stato individuato il sistema subordinato Chat, viene creata una classe apposita per comunicare con tale sistema.

Dato che la comunicazione nel sistema è basata sullo scambio di messaggi composti da più parti, viene anche creata la classe *Messaggio*, così da poter tenere traccia del mittente del messaggio, del corpo del messaggio e se il mittente è il Master del gruppo(nel caso il suo messaggio apparirà evidenziato nella chat)



**Figura 11. Classe Messaggio**

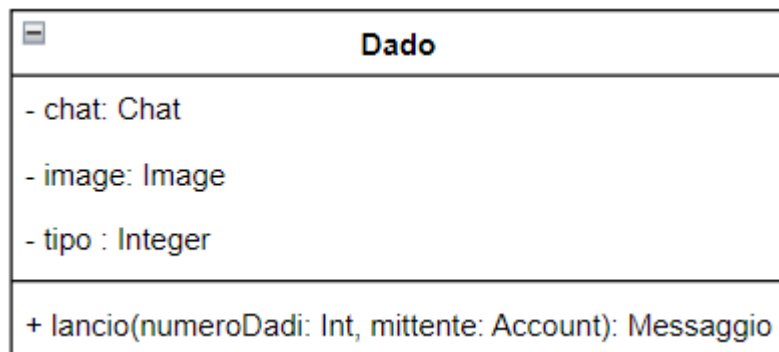


**Figura 12. Classe Chat**



## 1.9. Tool di simulazione dadi

Durante una sessione, a un giocatore può essere richiesto di fare un lancio di dadi. Per questo viene creata la classe Dado, che permette di simulare il lancio di un dado del tipo desiderato (a cui è associata anche la rispettiva immagine). Il risultato del lancio sarà mostrato come *Messaggio* tramite la *Chat* della sessione.



**Figura 13. Classe Dado**

Ogni tool di dadi è associato alla rispettiva *Chat* della sessione, come mostrato in Figura 13.



**Figura 13. Relazione tra la classe Dado e la classe Chat**

## 1.10. Diagramma delle classi complessivo

Il diagramma delle classi complessivo del sistema PartFinder viene mostrato in Figura 14.

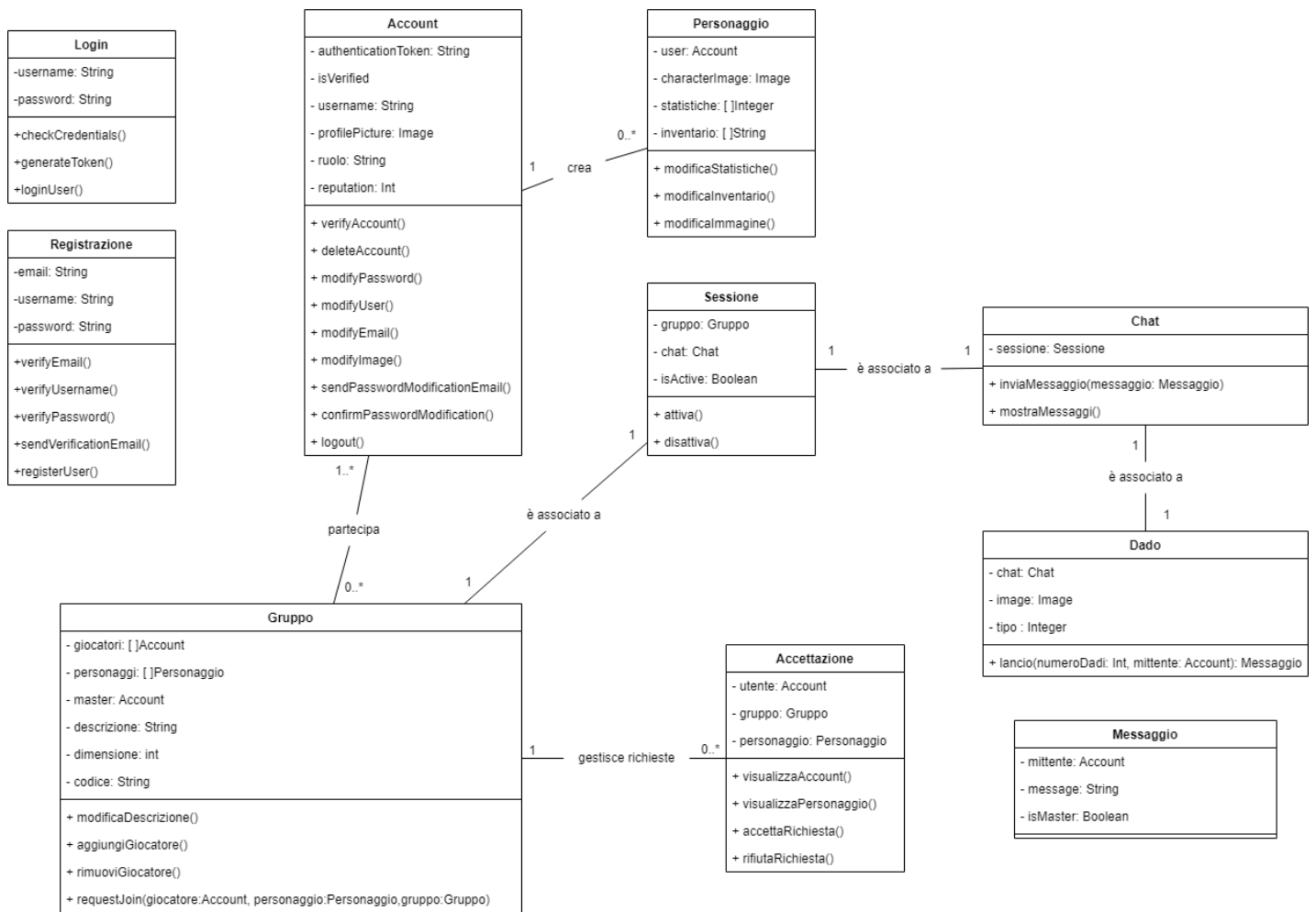
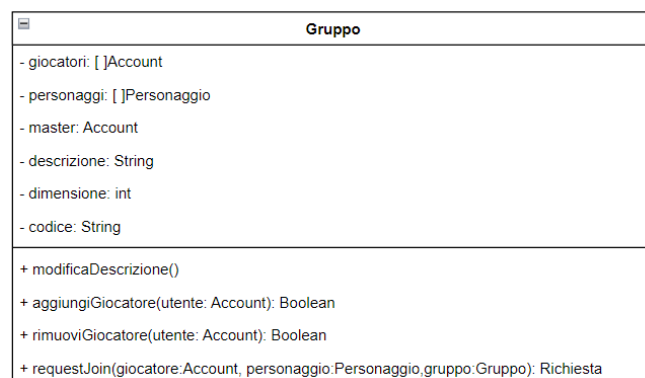


Figura 14. Diagramma delle classi del sistema PartFinder

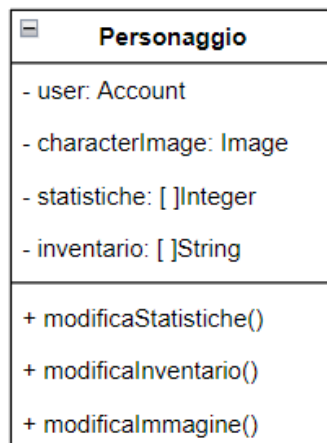
## 2.0 OCL



context::Gruppo inv:  
dimensione>0

context::Gruppo.requestJoin(giocatore:Account,  
personaggio:Personaggio,gruppo:Gruppo)  
pre: giocatore.isVerified = true

Al interno della classe "Gruppo" la variabile "dimensione" deve essere sempre >0  
Per far si che il metodo "requestJoin" di Gruppo è necessario che l'utente che richiede l'accesso sia autenticato




```
context::Personaggio inv:  
self.user.isVerified = true
```

Un personaggio può esistere se e solo se l'utente che lo crea è un utente verificato




```
context::Sessione.attiva()  
pre: self.sessione.isActive = false  
  
context::Sessione.disattiva()  
pre: self.sessione.isActive = true
```

Una sessione può essere attivata solo se precedentemente disattivata. Al contrario, una sessione può essere disattivata solo se precedentemente attivata

 <b>Dado</b>
- chat: Chat - image: Image - tipo : Integer
+ lancio(numeroDadi: Int, mittente: Account): Messaggio

```
Context::Dado.lancio()  
pre: self.chat.sessione.isActive=true
```

Il simulatore può essere utilizzato solo durante una sessione attiva, a differenza della chat, che può essere sempre usata

 <b>Richiesta</b>
- utente: Account - gruppo: Gruppo - personaggio: Personaggio
+ visualizzaAccount() + visualizzaPersonaggio() + accettaRichiesta() + rifiutaRichiesta()

```
context::Richiesta inv:  
(utente.isVerified = true) AND (self.utente.personaggi ->  
includes(self.personaggio))  
  
context::Richiesta.accettaRichiesta()  
post: self.gruppo.aggiungiGiocatore(self.utente) = true
```

Per creare una richiesta l'account dell'utente deve essere verificato e il personaggio inviato deve essere incluso nella sua lista di personaggi (utente.personaggi[]). Inoltre, dopo aver accettato la richiesta tramite "accettaRichiesta", "aggiungiGiocatore()" del gruppo deve ritornare true.

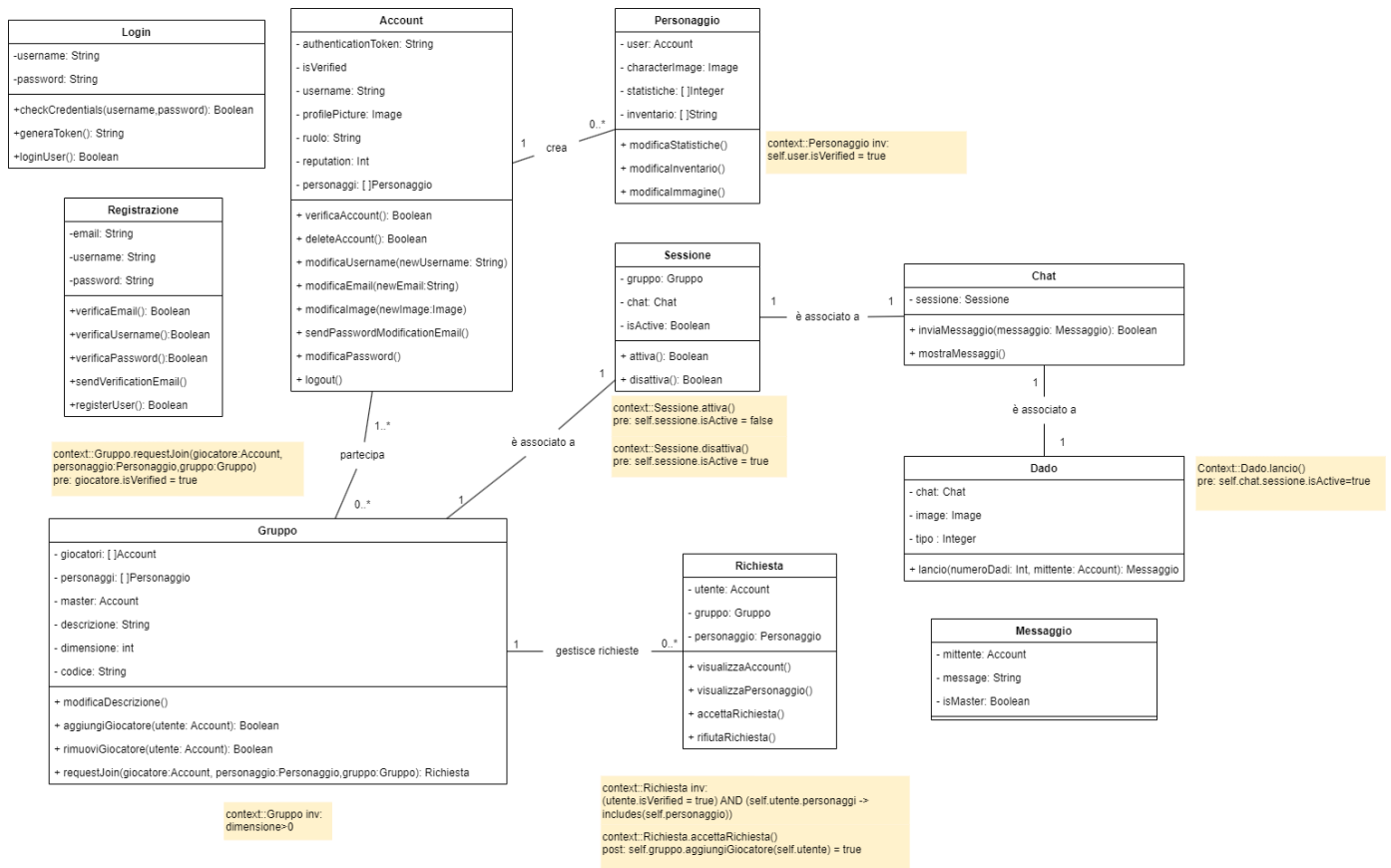


Diagramma totale incluso di OCL