DESIGN DOC

Object Oriented Software Design 2016-2017

* Valentina Cecchini 227719
* Stefano Valentini 227718
* Davide Micarelli 236829
* Luca Di Gregorio 229334

Indice

Requirements

Requisiti Funzionali (FR):

* **Profiling utente**

Ogni utente deve avere la possibilità di registrarsi al portale fornendo alcune informazioni base e di conseguenza di effettuare il **LogIn/LogOut**.

Ogni utente avrà a disposizione un’area riservata (profilo) in cui potrà visionare i dettagli relativi al suo account (ed eventualmente modificarli) e l’elenco dei trofei ricevuti con la timeline dei livelli conquistati e il bilancio dei propri punti esperienza.

* **Giocare**

L’utente può scegliere tra un catalogo di giochi e giocare, l’accesso al gioco è ristretto agli utenti registrati e loggati.

* **Votare/Recensire**

L’utente ha la possibilità di attribuire un voto e di dare una recensione a un gioco.

* **Guadagnare punti esperienza**

L’utente riceve punti esperienza giocando, alla fine di ogni sessione di gioco l’utente riceve una determinata quantità di punti esperienza.

* **Collezionare trofei**

Al raggiungimento di determinate soglie di punti esperienza l’utente sale di livello e riceve il trofeo associato.

* Gli utenti appartenenti al gruppo dei **moderatori** possono promuovere/retrocedere gli altri utenti (attraverso aggiunta/rimozione di punti esperienza), possono inoltre rimuovere recensioni.

Possono svolgere le attività degli utenti base.

* Gli utenti appartenenti al gruppo degli **amministratori** possono rendere altri utenti base **moderatori** e viceversa, possono inoltre accedere al backend e da qui gestire l’itero sistema.

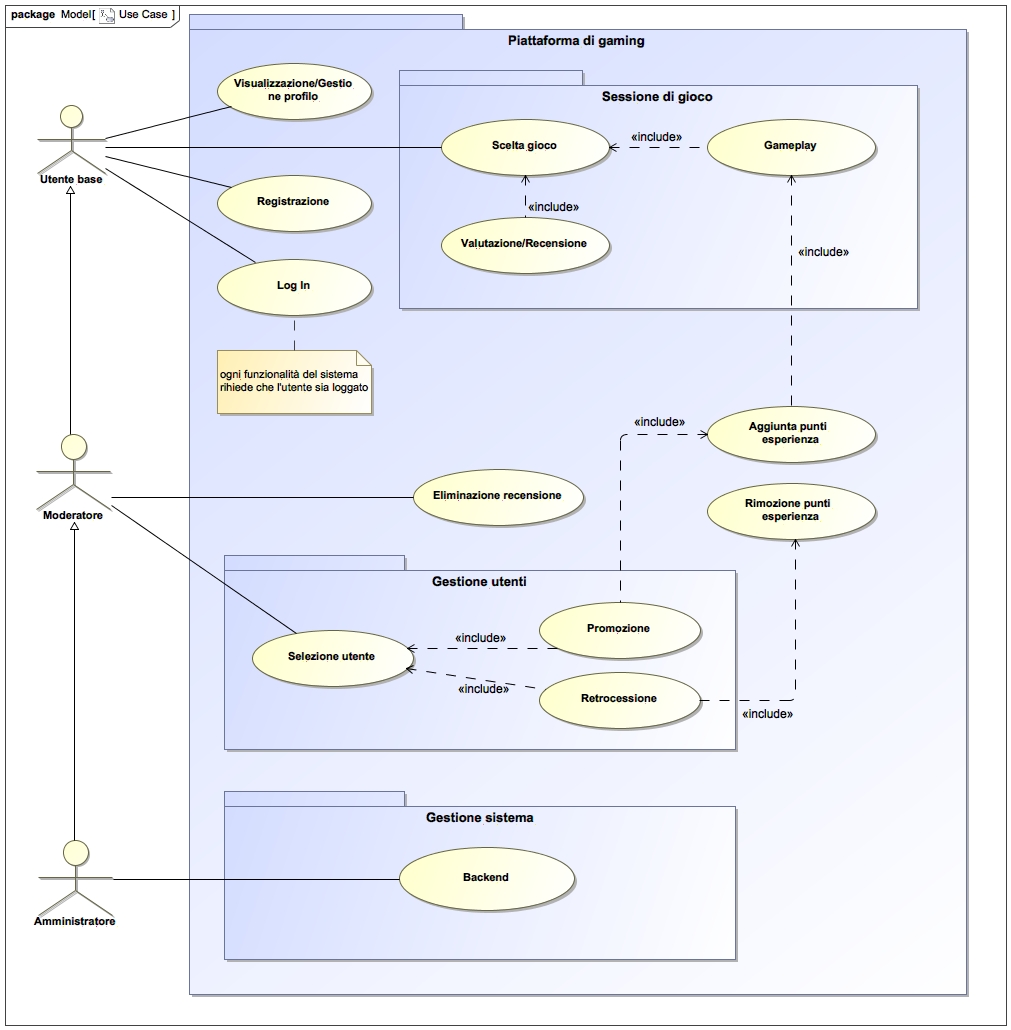
Possono svolgere le attività dei **moderatori.**

Requisiti Non Funzionali (NFR):

* **Availability:** Il sistema dovrà poter garantire in qualsiasi momento tutte le sue funzionalità.
* **Usability:** essendo un sistema di gioco il sistema dovrà essere di facile utilizzo e abbastanza immediato.
* **Reliability:** il sistema dovrà garantire all’utente le funzionalità offerte in modo affidabile.
* **Security:** il sistema dovrà garantire un livello di sicurezza adeguato (utenti base non dovranno avere possibilità di accedere a funzionalità riservate a moderatori o amministratori)

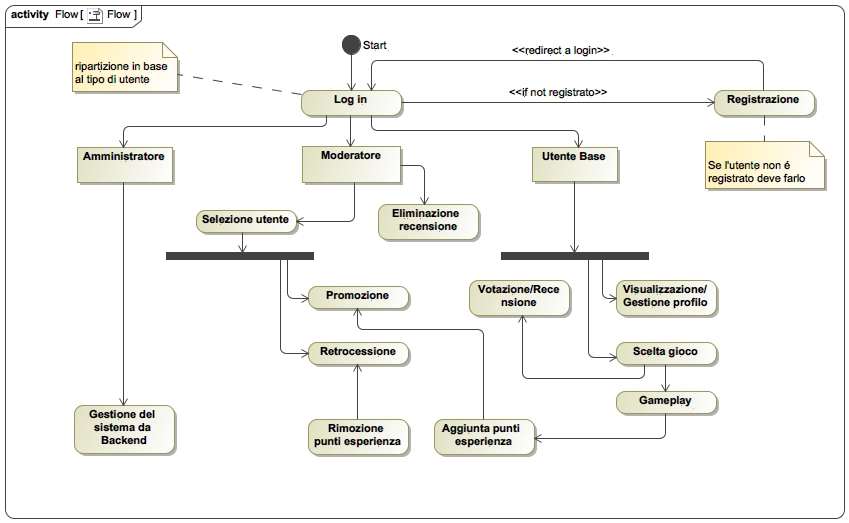
Use Case

Di seguito è riportato lo Use Case diagram che mostra le funzionalità del sistema e come i vari tipi di utenti vi interagiscono.



Activity Diagram

Di seguito è riportato l’activity diagram che mostra l’ordine con cui i vari attori interagiscono con gli use case.



Descrizione Attori

**Utente base:** è il principale utilizzatore del sistema, appartiene al gruppo con il minor numero di autorizzazioni, il principale Use Case con cui si interfaccia è **Gameplay** e di conseguenza può guadagnare punti esperienza e salire di livello ottenendo così trofei, può inoltre aggiungere una recensione/punteggio a un gioco e visualizzare/modificare il proprio profilo.

**Moderatore:** è il tipo di utente che ha il compito di monitorare e gestire l’attività degli utenti base, può attribuire/rimuovere punti esperienza ad altri utenti e può eliminare recensioni che non violano il regolamento del portale (ha inoltre accesso a tutte le funzionalità offerte agli utenti base).

**Amministratore**: è la figura con il più alto livello di accesso, ha la capacità di conferire e revocare autorizzazioni ad altri utenti (sia base che moderatori), ha inoltre accesso al backend e a tutte le funzionalità riservate agli utenti base e ai moderatori.

**Ogni funzionalità** del sistema può essere utilizzata **solo dopo essersi autenticati**.

Descrizione Use Cases

|  |  |
| --- | --- |
| Nome | **Registrazione** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | Permette agli utenti di registrarsi al portale |
| Trigger | L’attore fa click sull’apposito pulsante |
| End condition/Conseguenze | Registrazione avvenuta con successo o meno, l’utente può ora procedere al Log In |

|  |  |
| --- | --- |
| Nome | **Log in** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | Permette agli utenti di accedere al portale, ogni utente per usare qualsiasi funzionalità deve essere loggato |
| Trigger | L’attore fa click sull’apposito pulsante |
| End condition/Conseguenze | L’utente accede con successo o meno al portale |

|  |  |
| --- | --- |
| Nome | **Visualizzazione/Gestione profilo** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | Permette agli utenti di visualizzare il proprio profilo con i loro dati e modificarli |
| Trigger | L’attore fa click sull’apposito pulsante |
| End condition/Conseguenze | Modifica avvenuta con successo o meno |

|  |  |
| --- | --- |
| Nome | **Scelta gioco** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | Permette ad un utente di scegliere un gioco tra quelli disponibili nel catalogo |
| Trigger | Accesso dell'utente nel catalogo giochi |
| End condition/Conseguenze | / |

|  |  |
| --- | --- |
| Nome | **Gameplay** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | L'utente interagisce con gioco scelto |
| Trigger | Scelta di un gioco da parte di un utente nel catalogo giochi e di conseguenza click sull’apposito pulsante |
| End condition/Conseguenze | L’utente chiude la sessione di gioco/Il gioco termina, l’utente riceve i punti esperienza |

|  |  |
| --- | --- |
| Nome | **Valutazione/Recensione** |
| Attori partecipanti | Utenti base, moderatori, amministratori |
| Descrizione | Permette all'utente scrive una recensione/assegna una valutazione al gioco |
| Trigger | Click da parte dell’utente sull’apposito pulsante |
| End condition/Conseguenze | Recensione correttamente registrata nel sistema |

|  |  |
| --- | --- |
| Nome | **Selezione utente** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette all’attore di scegliere e selezionare un utente |
| Trigger | Click da parte dell’attore sull’identificativo dell’utente scelto |
| End condition/Conseguenze | / |

|  |  |
| --- | --- |
| Nome | **Eliminazione recensione** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette all’attore di eliminare una recensione |
| Trigger | Click da parte dell’attore sull’apposito pulsante |
| End condition/Conseguenze | Recensione correttamente rimossa dal sistema |

|  |  |
| --- | --- |
| Nome | **Aggiunta punti esperienza** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette di attribuire una certa quantità di punti esperienza ad un utente |
| Trigger | Click da parte dell’attore sull’apposito pulsante/L’utente completa una sessione di gioco |
| End condition/Conseguenze | I punti vengono aggiunti con successo/l’utente sale di livello (se i punti sono sufficienti) |

|  |  |
| --- | --- |
| Nome | **Rimozione punti esperienza** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette all’attore di sottrarre una certa quantità di punti esperienza ad un altro utente |
| Trigger | Click da parte dell’attore sull’apposito pulsante |
| End condition/Conseguenze | I punti vengono sottratti con successo/l’utente viene retrocesso al livello precedente (se i punti rimossi sono sufficienti) |

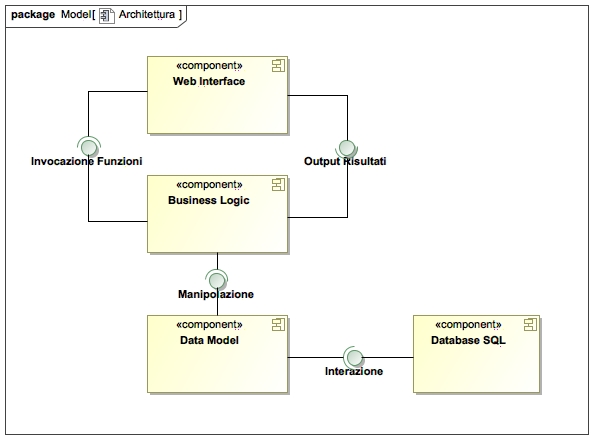
|  |  |
| --- | --- |
| Nome | **Promozione** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette all’attore di promuovere l’utente |
| Trigger | Intervento da parte dell’attore |
| End condition/Conseguenze | L’utente viene promosso al livello successivo |

|  |  |
| --- | --- |
| Nome | **Retrocessione** |
| Attori partecipanti | Moderatori, amministratori |
| Descrizione | Permette all’attore di retrocedere l’utente |
| Trigger | Intervento da parte dell’attore |
| End condition/Conseguenze | L’utente viene retrocesso al livello precedente |

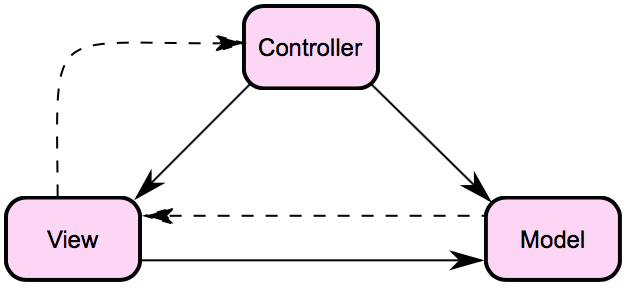
|  |  |
| --- | --- |
| Nome | **Backend** |
| Attori partecipanti | Amministratori |
| Descrizione | Per backend si intende tutte le funzionalità riservate all’amministratore: visualizzazione/aggiunta/modifica/rimozione di utenti/giochi/livelli (trofei)/gruppi/servizi |
| Trigger | / |
| End condition/Conseguenze | / |

System Design

Modello Architetturale



Descrizione Modello Architetturale

Il **pattern architetturale** di riferimento è il pattern **MVC (Model View Controller)**, è un pattern architetturale particolarmente indicato per la realizzazione di sistemi che hanno una interfaccia utente.

Permette di suddividere il sistema in 3 macro componenti interconnesse che permettono di rappresentare le funzionalità al loro interno in modo indipendente così da favorire riuso del codice e parallelizzazione dello sviluppo.

**Perché MVC?**

Il pattern architetturale **MVC** permette una divisione pulita della business logic dai dati e dalla presentation logic.

Questo ci permette di sviluppare un sistema modulare e facilmente espandibile (in ottica futura).

In poche parole ci permette:

* **Sviluppo parallelo:** ci permette una elevata divisione dei compiti e quindi un lavoro parallelizzato
* **Low coupling**: abbiamo sottosistemi (componenti) indipendenti che possono essere quindi modificati senza impattare sul resto del sistema.
* **High cohesion:** le classi nei vari sottosistemi eseguono operazioni simili e comunicano tra di loro attraverso diverse associazioni.
* **Facile manutenibilità**
* **Riuso del codice**

**Web Interface**: si è scelto di utilizzare un approccio web a livello di presentazione in quanto il contesto d’uso è particolarmente adatto. L’utente interagirà quindi con il sistema attraverso elementi html, l’iterazione in particolare è guidata da servlet che attraverso chiamate post e get permetteranno all’utente di intervenire sul sistema e di ottenere informazioni in output. Per la realizzazione della parte relativa alla presentazione si farà uso del motore di templating **FreeMarker.**

**FreeMarker** è una libreria che aiuta nella separazione delle logiche per quanto riguarda la presentazione web, in poche parole permette di astrarre la struttura e “l’estetica” del livello presentazione dai dati e dalla logica che vi è dietro.

Questo permette un grande livello di modularità, in quanto renderà possibile in un futuro la sostituzione della template web senza aver bisogno di modificare il resto del sistema.

**Business Logic:** contiene le classi e i metodi atti alla gestione del sistema, le quali si occupano della manipolazione dei dati.

Contiene inoltre il codice relativo all’implementazione delle servlet per la comunicazione con l’interfaccia web.

**Data Model:** è un “contenitore” di classi e oggetti che sono manipolati dalla business logic (controller) e che vengono poi immagazzinati nel database.

Contiene inoltre l’infrastruttura di interfacce **DAO** che permette un’interazione modulare col database e di conseguenza l’astrazione del livello dati da quello controller.

**Database SQL:** l’architettura si conclude con il database che implementa la persistenza dei dati e che quindi contiene fisicamente i dati relativi al sistema.

Design Pattern Utilizzati

Per quanto riguarda il **Data Model** e la sua interazione con il **database** si è scelto di utilizzare il design pattern **DAO (Data Access Object).**

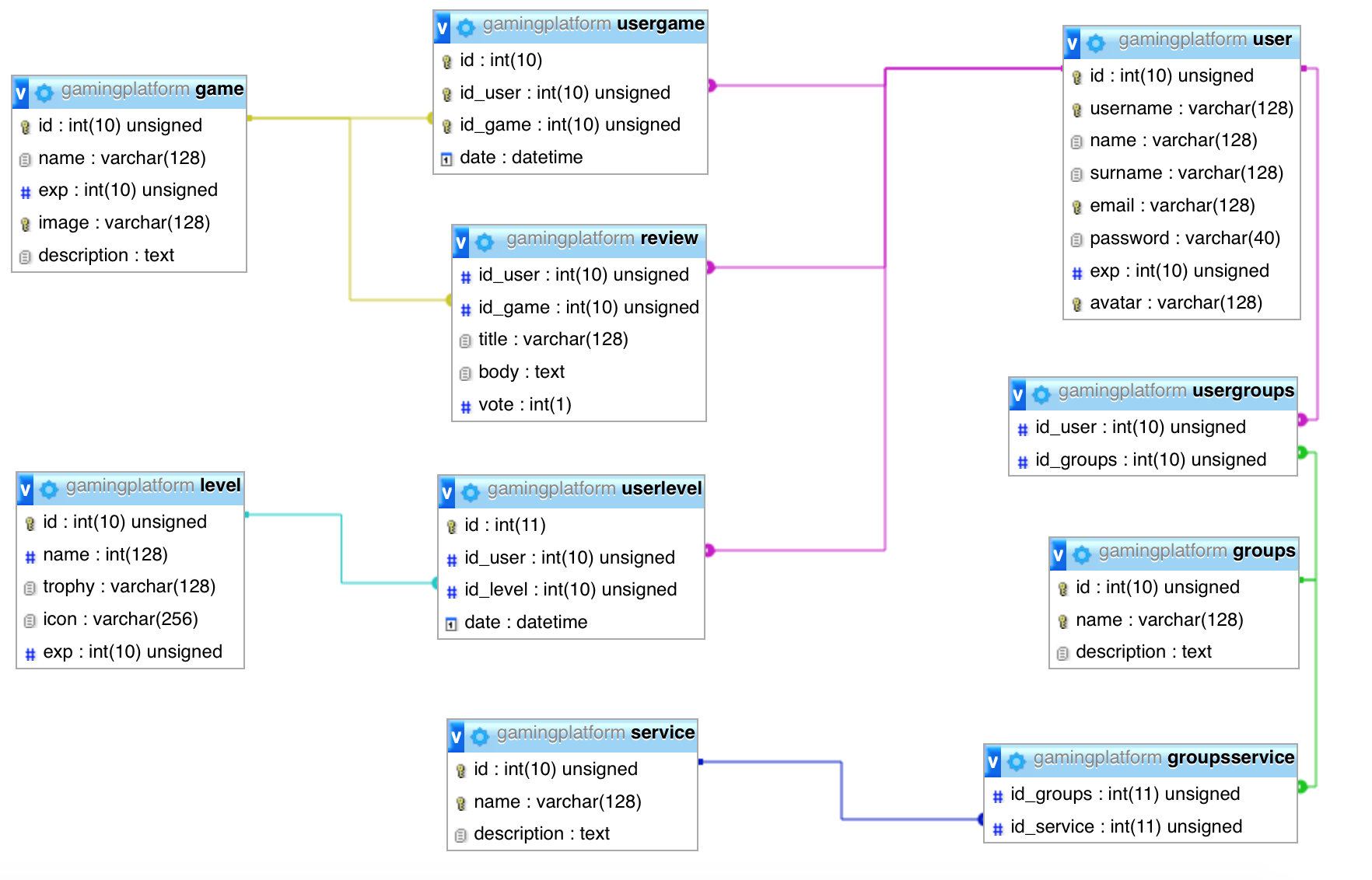
Il **DAO** è un design pattern che fornisce un’interfaccia astratta ad alcuni tipi di database (nel nostro caso un database relazionale SQL), mappando le chiamate della business logic sul modello di persistenza dati.

**Perché DAO?**

Il vantaggio di usare **DAO** consente nel costruire una separazione semplice e rigorosa tra parti del sistema che ci si aspetta evolveranno nel tempo e che possono ma non dovrebbero “conoscersi” (tutti i dettagli dell’immagazzinamento dati sono nascosti al resto del sistema), è quindi particolarmente adatto a sistemi basati su **MVC** e in particolare aderenti al **paradigma OO**.

In poche parole la business logic farà affidamento sempre sui metodi esposti dall’interfaccia DAO, mentre la sua implementazione può cambiare al cambiare del metodo di persistenza adottato.

Schema ER



I tre “protagonisti” del sistema sono gli **utenti,** i **giochi** e i **livelli.**

Di un **utente** viene salvato l’username (nickname da lui scelto, che potranno eventualmente vedere gli altri utenti), il nome, il cognome, l’e-mail, la password, il path (sul filesystem del server) al suo avatar e l’esperienza (cioè i punti accumulati dall’utente), ogni utente è identificato da un id.

Ogni **utente** è relazionato ad un gioco tramite:

* **usergame:** contiene data e ora della sessione in cui un dato utente gioca a un dato gioco
* **review:** contiene i dati relativi alle recensioni/voti che un certo utente ha espresso su un certo gioco

Di un **gioco** è necessario salvare il nome, i punti esperienza assegnati per ogni sessione, una breve descrizione e l’immagine.

Come nel caso degli **utenti**, un **gioco** è identificato tramite un id.

Ogni **utente**, inoltre, ha assegnato un **livello**, in base al punteggio accumulato. **Utente** e **livello** sono, quindi, relazionati tramite **userlevel**, che contiene data e ora di quando un dato utente è avanzato o retrocesso di un livello.

Dell’entità **level** è necessario mantenere in memoria il nome del livello (che in questo caso è rappresentato da un numero), il relativo trofeo da assegnare, un’icona raffigurante il trofeo e il numero di punteggio necessario per poter raggiungere il relativo **livello**.

Si è poi pensato di strutturare la gestione delle diverse categorie di **utente** con il design pattern **utenti-gruppi-servizi**, quindi ogni **utente** potrà appartenere o no ad un **gruppo** (nel nostro caso i gruppi saranno due, moderatore e amministratore (ma possono comunque essere facilmente aggiunti altri gruppi); non è necessario fare un gruppo a parte per gli utenti base in quanto sono la maggioranza e non potranno comunque accedere a nessuna funzionalità del backend. Ogni **gruppo** potrà accedere ad uno o più **servizi**, che sono riservati a quella particolare tipologia di utente.

Il **gruppo** è memorizzato con un id che lo identifica, un nome e una descrizione che esplicita il ruolo degli utenti appartenenti a quel particolare gruppo.

Il **servizio** è salvato anch’esso con un id, un nome e una descrizione che spiega cosa permette di fare quel particolare servizio.