DESIGN DOC

Object Oriented Software Design 2016-2017

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

- link al repository: https://github.com/davideyoga/OOSD-Project
- tutte le immagini sono presenti nella cartella doc/images in caso non siano chiaramente visibili all'interno del documento.

Indice

1. Requirements

1.1 Requisiti Funzionali (FR)

1.1.1 Registrazione da parte dell'utente al portale (1):

L'utente base avrà la possibilità di registrarsi alla GamingPlatform, inserendo informazioni basilari da poter riutilizzare in un secondo momento per effettuare il Log In e dare il via alla propria esperienza di gioco. Quando un utente si registra gli verrà attribuito il livello 0 e un quantitativo di exp pari a 0.

1.1.2 Visualizzazione del profilo e delle pagine dei relativi giochi (2):

Ogni utente avrà a disposizione un'area riservata dove sarà possibile visualizzare il proprio profilo con le relative informazioni anagrafiche (con possibilità di modifica) e le informazioni riguardanti i livelli e punti esperienza guadagnati.
Vi saranno quindi diversi grafici che mostreranno l'avanzamento dell'utente tra i vari livelli e due differenti timeline che mostrano le varie sessioni di gioco effettuate e i livelli con i relativi trofei guadagnati. In questo modo l'utente potrà avere sotto controllo, in qualsiasi momento, il suo avanzamento tra i livelli.
Un utente, ogni qual volta effettua un accesso sarà reindirizzato alla home page della GamingPlatform, dove potrà visualizzare l'intero catalogo di giochi disponibile.
Quando seleziona un gioco verrà reindirizzato alla pagina contenente la descrizione del gioco selezionato: qui potrà visualizzare tutte le informazioni relative al gioco, tra cui le istruzioni per giocare, i punti che è possibile guadagnare e una breve descrizione del gioco.
Potrà inoltre trovarvi tutte le recensioni lasciate dagli altri utenti della piattaforma.

1.1.3 Possibilità di votare un gioco o esprimere un giudizio sullo stesso tramite recensione (3):

Ad ogni pagina relativa ad un particolare gioco, vi è un'area riservata a votazioni e recensioni esprimibili dai soli utenti loggati. Ogni utente autenticato, può quindi esprimere una votazione in un range da 1 a 5 (dove al crescere del punteggio, cresce il gradimento) per ogni gioco presente sulla piattaforma e, lasciare una recensione sulla propria esperienza di gioco, che sarà utile ai nuovi gamer per iniziare la loro esperienza.

Tali recensioni sono sottoposte a moderazione tramite cancellazione o modifica, a discrezione del moderatore della GamingPlatform.

1.1.4 Effettuare una sessione di gioco e collezionare punti esperienza ad ogni sessione (4):

Ogni qual volta un utente autenticato vuole effettuare una sessione di gioco accede, tramite l'homepage, alla pagina del gioco scelto e seguendo le istruzioni inizierà la sua sessione di gioco. Ad ogni sessione di gioco, l'utente guadagnerà un certo quantitativo di punti esperienza che gli verranno immediatamente notificati tramite un message box e sommati a quelli già posseduti. Inoltre la sessione verrà registrata nella relativa timeline sul proprio profilo. Se nel guadagnare i punti esperienza l'utente sale di livello, gli verrà notificato tramite il message box e sul suo profilo sarà immediatamente disponibile il trofeo guadagnato e, tutte le timeline e i grafici saranno aggiornati con il relativo livello raggiunto.

1.1.5 Collezionare trofei relativamente al livello conquistato (5):
L'utente, effettuando diverse sessioni di gioco potrà guadagnare punti esperienza che gli permetteranno di salire di livello e di guadagnare trofei.
Un livello viene superato quando si raggiunge la soglia fissata per accedere a quello successivo.
Inoltre ogni volta che viene raggiunto un nuovo livello, l'utente riceve il trofeo associato.

1.1.6 (6) Gli utenti appartenenti al gruppo dei moderatori possono promuovere/retrocedere gli altri utenti (attraverso aggiunta/rimozione di punti esperienza) e possono inoltre rimuovere recensioni. Possono svolgere le attività degli utenti base.

1.1.7 (7) Gli utenti appartenenti al gruppo degli amministratori possono rendere altri utenti base moderatori e viceversa, possono inoltre accedere al backoffice e da qui gestire l'intero sistema. Possono svolgere anche le attività dei moderatori.

1.2 Requisiti Non Funzionali (NFR)

1.2.1 Availability: Il sistema dovrà poter garantire in qualsiasi momento tutte le sue funzionalità.

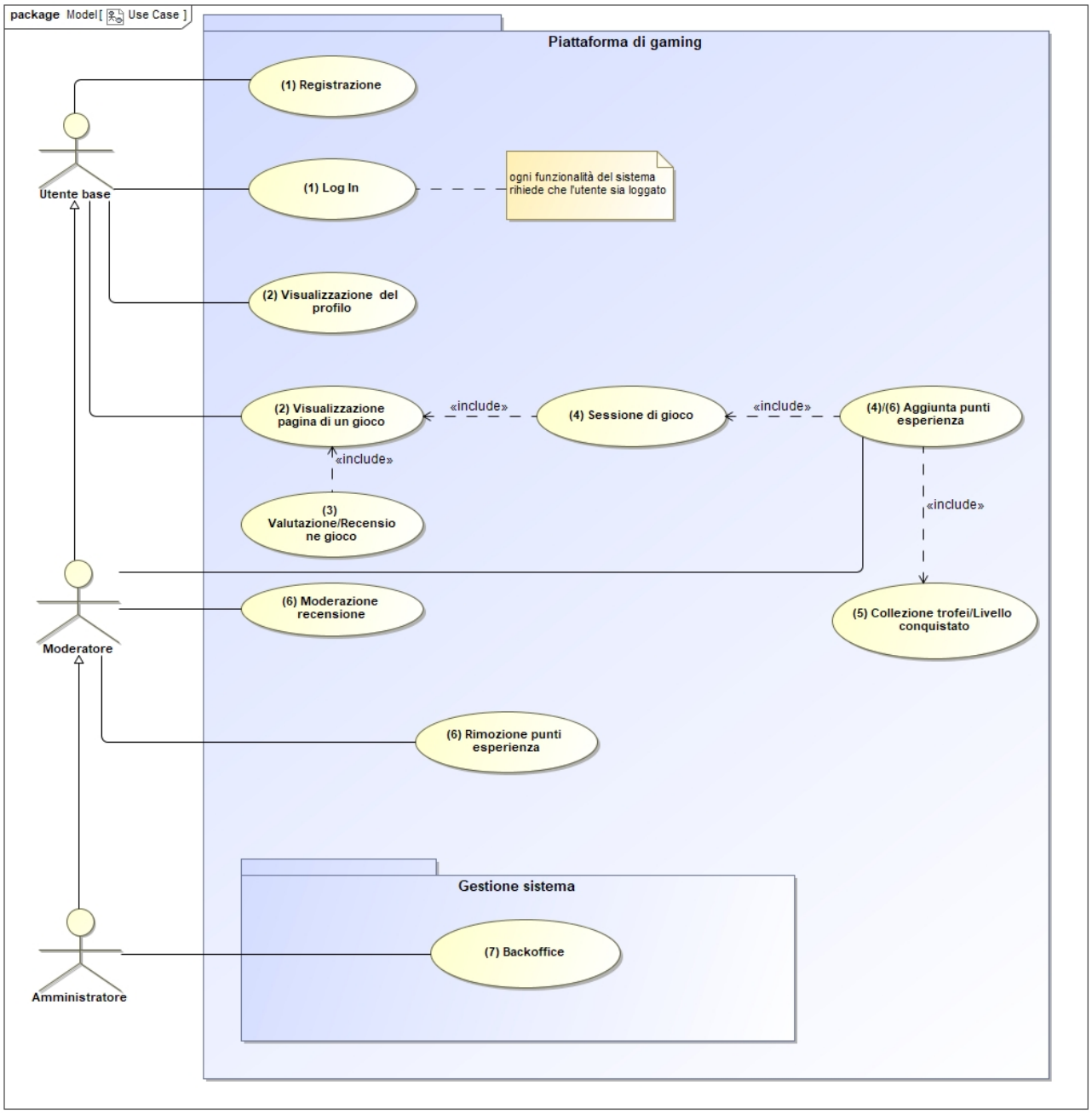
1.2.2 Usability: essendo un sistema di gioco il sistema dovrà essere di facile utilizzo e abbastanza immediato.

1.2.3 Reliability: il sistema dovrà garantire all'utente le funzionalità offerte in modo affidabile.

1.2.4 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)

1.3 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.



1.3.1 Mapping UseCases – Requisiti

Requisito 1	Use cases: Registrazione, Log In
Requisito 2	Use cases: Visualizzazione del profilo, Visualizzazione pagina di un gioco
Requisito 3	Use cases: Valutazione/Recensione gioco
Requisito 4	Use cases: Sessione di gioco, Aggiunta punti esperienza
Requisito 5	Use cases: Collezione trofei/Livello conquistato
Requisito 6	Use cases: Moderazione recensione, Rimozione/Aggiunta punti esperienza
Requisito 7	Use cases: Backoffice

1.3.2 Descrizione Attori

Utente base: è il principale utilizzatore del sistema e di tutte le funzionalità che offre.
Si interfaccia con le principali funzionalità offerte dal sistema: registrazione, login, visualizzazione del proprio profilo e delle pagine dei singoli giochi e può effettuare valutazioni e recensioni sui giochi.
Tramite quest'ultime può accedere ad ulteriori funzionalità che gli permettono di effettuare una sessione di gioco e di conseguenza guadagnare punti esperienza necessari al raggiungimento di nuovi livelli e alla collezione dei trofei associati.

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 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.

1.3.3 Descrizione Use Cases

Nome	Registrazione, Login
Goal in context	Registrarsi alla gamingplatform. Effettuare successivamente login.
Preconditions	Per una corretta registrazione alla piattaforma, è necessario inserire le informazioni base, richieste nella form “SignUp”
Success	Se l’operazione va a buon fine, l’utente sarà registrato e gli verrà assegnato il livello 0, con punti esperienza 0.
Fail	Se l’operazione dovesse fallire, lo stato dell’intero sistema rimane invariato e all’utente verrá mostrata la causa del fallimento (ad es: Indirizzo email invalido)
Primary Actor	Utente Base, Moderatori, Amministratori
Trigger	Richiesta da parte dell’utente

Step	Action
1	L’utente si interfaccia con la form della registrazione
2	L'uente effettua l'invio dei dati mediante il relativo pulsante di Sign Up/Log in
3	Il sistema elabora i dati ricevuti e, nel caso in cui si tratti di una registrazione crea un nuovo account, altrimenti concede all'utente le autorizzazioni per poter usufruire delle funzionalità del sistema.

Nome	Visualizzazione del profilo, Visualizzazione pagina di un gioco
Goal in context	Accedere alle informazioni del proprio profilo Poter effettuare modifiche Visualizzare le pagine dei giochi disponibili con le relative informazioni, quali: punti exp ottenibili, recensioni, etc.
Preconditions	L’utente deve essere autenticato
Success	L’utente potrà visualizzare la pagina del proprio profilo oppure (a sua scelta) la pagina del gioco a cui è interessato
Fail	L'unica condizione di fallimento è causata dall'utente non loggato, quindi viene respinto alla home page e gli viene notificato l’errore.
Primary Actor	Utente Base, Moderatori, Amministratori
Trigger	Richiesta da parte dell’utente

Step	Action
1	L'utente tenta di accedere alle pagine che desidera visualizzare
2	Se loggato, il sistema mostrerà all'utente le informazioni che egli aveva richiesto.

Nome	Valutazione/Recensione gioco
Goal in context	Poter votare un gioco (in un range da 1 a 5). Effettuare una recensione del gioco
Preconditions	L'utente deve essere autenticato
Success	Se l'operazione va a buon fine, l'utente potrà effettuare la votazione del gioco, ed eventualmente scrivere una recensione
Fail	Se l'operazione dovesse fallire, lo stato dell'intero sistema rimane invariato e l'utente viene reindirizzato alla pagina di autenticazione
Primary Actor	Utente Base, Moderatori, Amministratori
Trigger	Richiesta da parte dell'utente

Step	Action
1	L'utente si interfaccia con la pagina della visualizzazione gioco
2	L'utente, arrivato alla pagina del gioco, può effettuare una votazione e/o scrivere una recensione ed effettua un submit affinché tale recensione venga pubblicata e poi controllata del moderatore che eventualmente potrà rimuoverla.
3	I dati inseriti dall'utente vengono memorizzati sul database

Nome	Sessione di gioco, Aggiunta punti esperienza
Goal in context	Poter effettuare una sessione di gioco. Guadagnare punti esperienza.
Preconditions	L'utente deve essere autenticato
Success	Se l'operazione va a buon fine, l'utente potrá effettuare la sessione del gioco, e maturare punti esperienza per ogni sessione effettuata
Fail	Se l'operazione dovesse fallire, lo stato dell'intero sistema rimane invariato e l'utente viene notificato dell'errore.
Primary Actor	Utente Base, Moderatori, Amministratori
Trigger	Richiesta da parte dell'utente

Step	Action
1	L'utente si interfaccia con la pagina della visualizzazione gioco
2	L'utente inizia la sua sessione di gioco e alla fine guadagna punti
3	I punti guadagnati dall'utente vengono aggiunti ai punti già posseduti e registrati sul profilo dell'utente

Nome	Collezione trofei/Livello conquistato
Goal in context	Permettere all'utente di salire di livello e conquistare nuovi trofei
Preconditions	L'utente deve essere autenticato
Success	Se l'operazione va a buon fine, l'utente potrá verificare il passaggio a livello successivo e il relativo trofeo acquisito
Fail	Se l'operazione dovesse fallire, lo stato dell'intero sistema rimane invariato e l'utente viene notificato dell'errore
Primary Actor	Utente Base, Moderatori, Amministratori
Trigger	Richiesta da parte dell'utente

Step	Action
1	L'utente si interfaccia con la pagina della visualizzazione gioco
2	L'utente viene fatto salire di livello e colleziona il relativo trofeo.
3	Il profilo dell'utente viene aggiornato con il livello raggiunto e il trofeo conquistato.

Nome	Moderazione recensione, Rimozione/Aggiunta punti esperienza
Goal in context	Poter promuovere/retrocedere gli altri utenti, aggiungendo o rimuovendo punti esperienza. Poter modificare/rimuovere una recensione nel caso non rispetti le regole del sistema o abbia contenuti non appropriati.
Preconditions	L'utente deve essere autenticato, deve avere le autorizzazioni necessarie.
Success	Se l'operazione va a buon fine il moderatore effettua l'aggiunta o rimozione di punti esperienza a un utente e rimuove/modifica una recensione.
Fail	Se l'operazione dovesse fallire, lo stato dell'intero sistema rimane invariato e l'utente viene notificato dell'errore
Primary Actor	Moderatori, Amministratori
Trigger	Richiesta da parte dell'utente

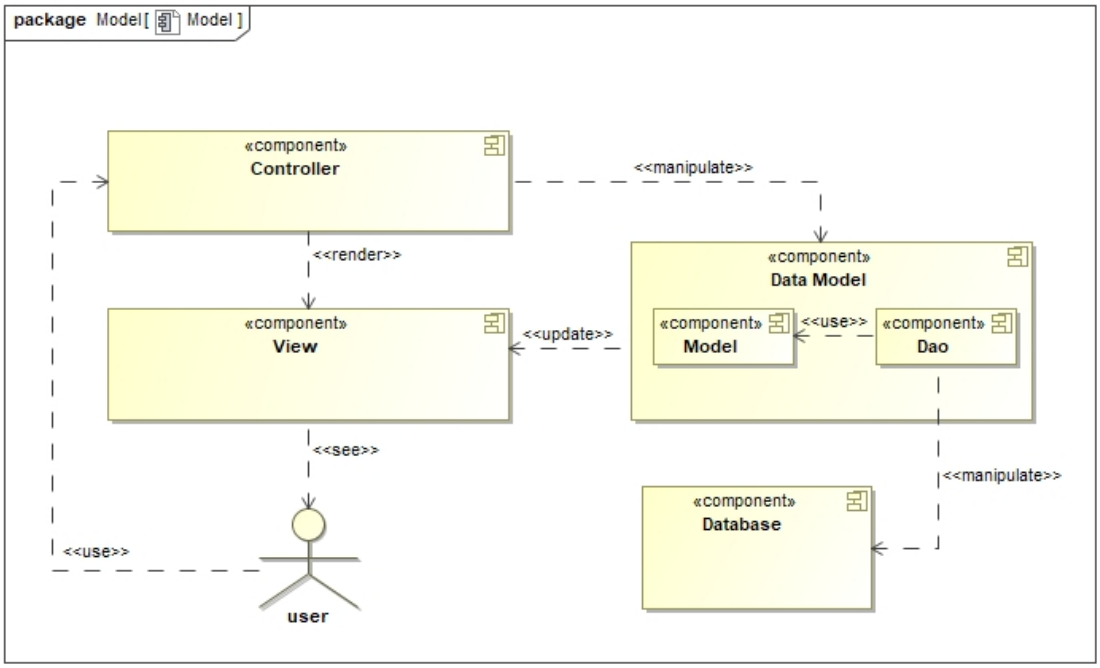
Step	Action
1	Il moderatore si interfaccia con la pagina della visualizzazione dei profili utente. Il moderatore si interfaccia con le pagine dei giochi
2	Vengono sottratti o aggiunti punti esperienza all'utente La recensione viene rimossa/modificata.

Nome	Backoffice
Goal in context	Gestione del sistema (CRUD operations, e amministrazione)
Preconditions	L'utente deve essere autenticato e avere le autorizzazioni necessarie.
Success	Se l'operazione va a buon fine, l'utente potrà effettuare (a seconda della scelta effettuata): 1) Le operazioni consentite ad un utente base 2) Le operazioni consentite ad un moderatore 3) Aggiunta o rimozione di moderatori 4) Gestione del sistema
Fail	Se l'operazione dovesse fallire, lo stato dell'intero sistema rimane invariato e l'utente viene notificato dell'errore
Primary Actor	Amministratori
Trigger	Richiesta da parte dell'utente

Step	Action
1	L'Amministratore si interfaccia con la pagina relativa all'operazione scelta (vedi punti sopra 1,2,3,4)
2	L'Amministratore effettua le operazioni
3	Le modifiche vengono salvate sul database

2. System Design

2.1 Modello Architetturale



2.2 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.

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

View: 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 del template web senza aver bisogno di modificare il resto del sistema.

Controller: 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: l'architettura si conclude con il database relazionale mysql che implementa la persistenza dei dati e che quindi contiene fisicamente i dati relativi al sistema.

2.3 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.

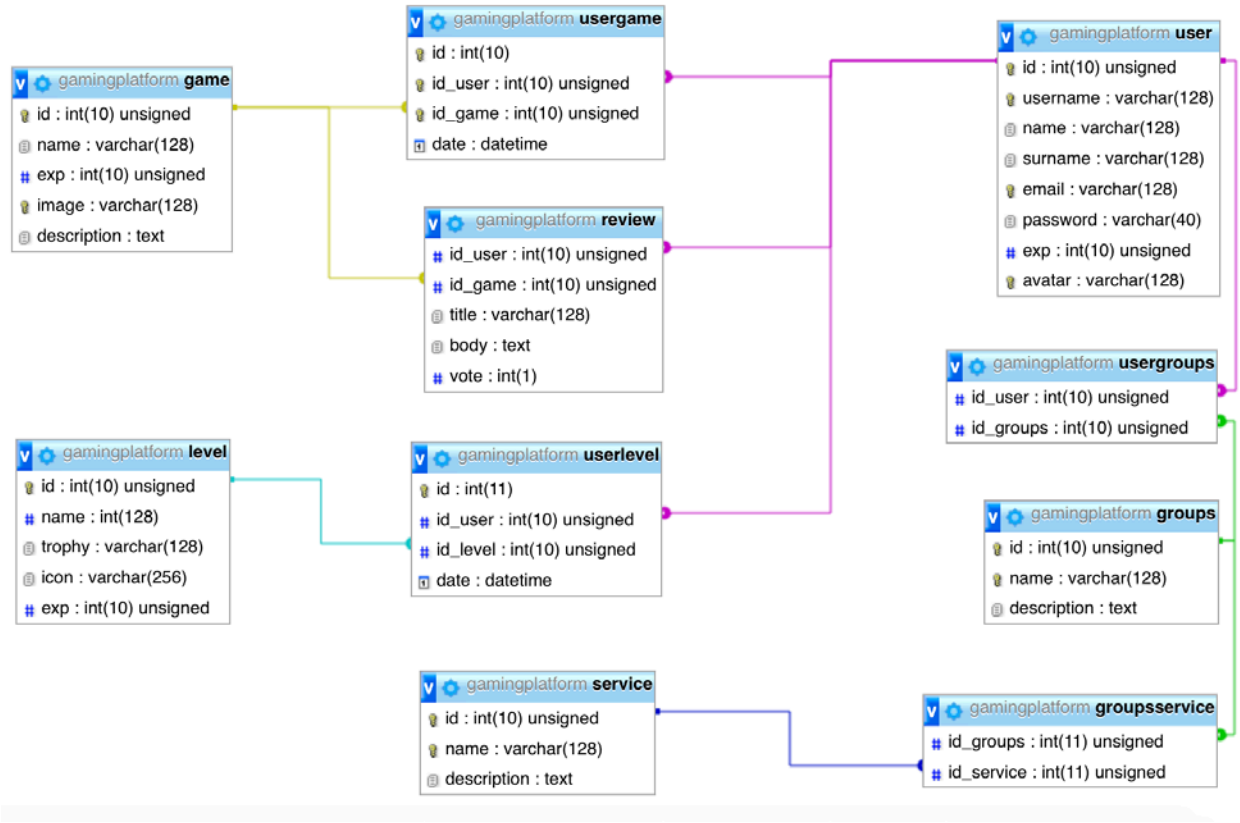
Il vantaggio di usare DAO consiste nel costruire una separazione semplice e rigorosa tra parti del sistema che ci si aspetta evolveranno nel tempo e che possono, ma non dovrebbero "conoscersi" (tutti i dettagli dell'immagazzinamento dati sono nascosti al resto del sistema), è quindi particolarmente adatto 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.

Un altro design pattern utilizzato è il singleton.
Il singleton è un design pattern creazionale che ha lo scopo di assicurarsi che venga creata una sola ed unica istanza di una classe.

Nel dettaglio il design pattern è stato utilizzato per la gestione della configurazione di freemarker.
Come suggerito dalla documentazione è preferibile creare un'unica istanza della classe atta alla configurazione di freemarker in modo da non sprecare risorse di sistema.

Nello specifico il design pattern è stato implementato utilizzando Enum (disponibile da Java 5).
Enum è thread-safe (ovvero si comporta in modo corretto in situazioni di multithreading).
Di seguito il codice della classe che implementa il design pattern singleton (descritta più in dettaglio nella sezione 3 del documento).

2.4 Schema ER



Per l'implementazione della persistenza dati si è deciso di utilizzare un database SQL mysql.

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 delle sessioni di gioco.
- **review:** contiene i dati relativi alle recensioni/voti che un utente ha espresso su un certo gioco

Di un **gioco** è necessario salvare il nome, i punti esperienza guadagnabili ad ogni sessione, una breve descrizione e l'immagine di copertina.
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 tale 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 uno o più gruppi (in realtà, ai fini del progetto, i gruppi utili sono due: moderatore e amministratore e ogni utente può appartenere ad uno solo di questi gruppi, ma con questo approccio, si potranno, in ottica futura, aggiungere altri gruppi e permettere agli utenti di far parte di più gruppi); non è necessario creare un gruppo a parte per gli utenti base in quanto sono la maggioranza e non potranno comunque accedere a nessuna funzionalità del backoffice.
Ogni gruppo potrà accedere ad uno o più servizi, che sono riservati a quel particolare gruppo.

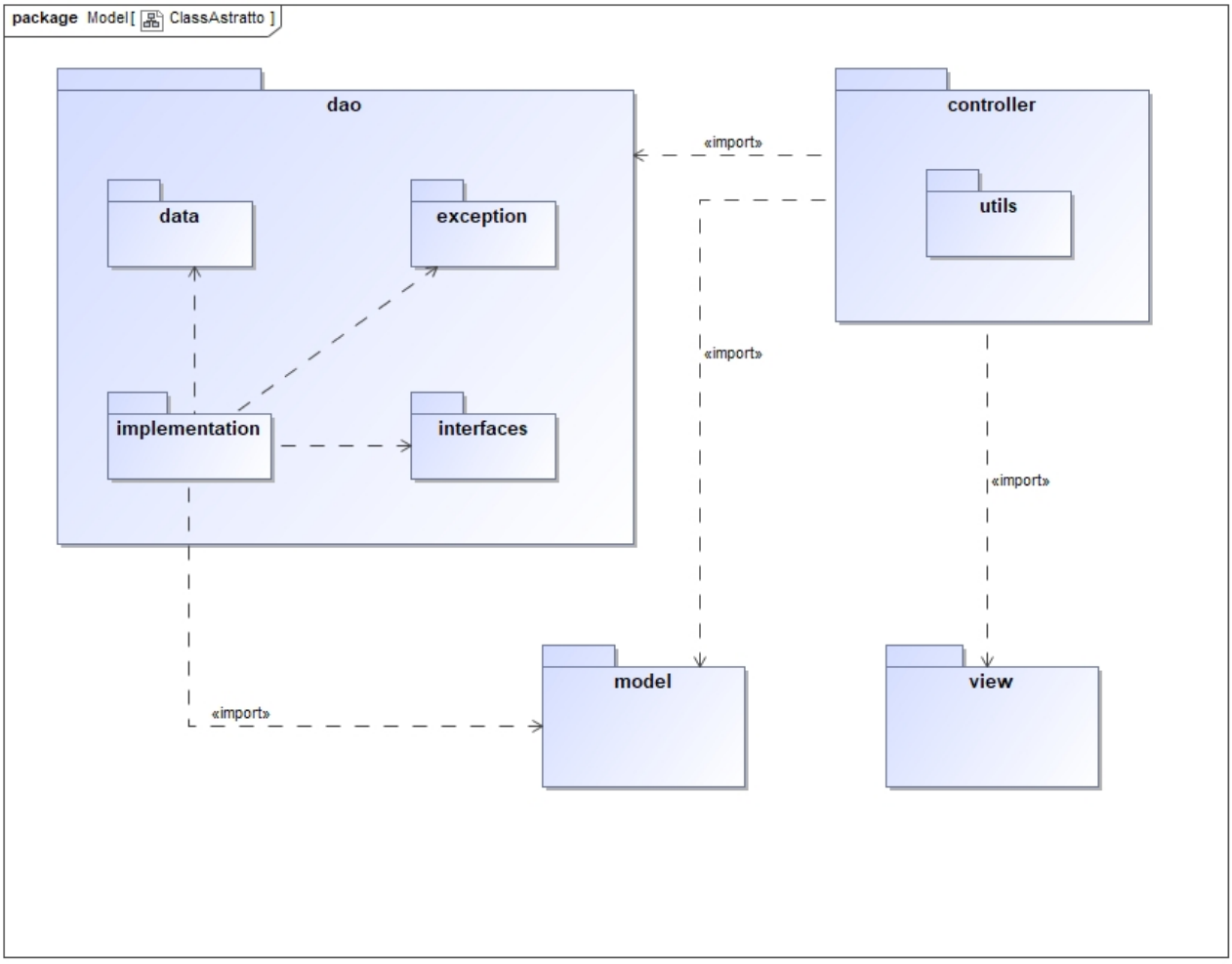
Il gruppo è memorizzato con un id che lo identifica, un nome e una descrizione che esplicita il ruolo degli utenti appartenenti a quel particolare gruppo.
Il servizio è salvato anch'esso con un id, un nome e una descrizione che spiega cosa permette di fare quel particolare servizio.

Per la gestione delle immagini (avatar utente/immagine copertina del gioco/logo del trofeo) si è deciso di immagazzinare nel db solo il nome/path al file immagine sul file system che ospita l'applicativo e non il file immagine nella sua interezza.
Questo ci permette di mantenere il db più leggero e ce ne facilita la gestione anche in ottica di interfaccia web.

3. Software Object Design

3.1 Software Design Diagrams

3.1.1 Package Diagram



Rispecchiando il pattern architetturale MVC abbiamo i seguenti packages:

- View: contiene le classi per la configurazione e l'utilizzo di freemarker (i file relativi alla web interfacce risiedono in una cartella a parte).

- Controller: contiene le servlet ed il sottopackage utils che contiene classi utility relative alla sicurezza dell'applicativo, gestione dei file ecc.

- Model: contiene la rappresentazione delle entità del sistema con i setter e getter (cercando di rispecchiare quanto presente sul database 1:1).

- Dao: contiene classi ed interfacce che realizzano il design pattern data access object, in particolare:

    - Data: sottopackage che contiene classi e interfacce che gestiscono e realizzano la connessione al database attraverso connection pooling.
      Il connection pooling è un sistema di cache che permette di mantenere aperte delle connessioni in modo da renderle subito disponibili nel caso in cui ve ne sia richiesta, senza il bisogno di inizializzare e chiudere la connessione ad ogni richiesta.
      Il sistema di connection pooling viene gestito in modo trasparentemente al programmatore, che dovrà occuparsi solo di chiedere ed in seguito di eliminare un riferimento a tale risorsa, in modo da renderla immediatamente disponibile per un successivo utilizzo.
      In java questo sistema, comunemente definito come pool di connessioni, viene definito nell'interfaccia Datasource.
      Nel nostro sistema l'interfaccia che definisce il funzionamento di tali meccanismi è la classe DaoData (nel package gamingplatform.dao.data) ed implementata nella classe DaoDataMySQLImpl (stesso package).
      Tale classe, una volta istanziata tramite costruttore, richiede al gestore delle connessioni (nel nostro caso l'application server Tomcat) un puntatore al Datasource.
      Da tale Datasource nel metodo init viene estratto un oggetto Connection, che una volta utilizzato viene chiuso nel metodo destroy.
      Ogni classe che va ad interagire col database va ad estendere DaoDataMySQLImpl per avere accesso ad una connessione.

    - Exception: contiene l'implementazione di eccezioni personalizzate.

    - Interfaces: contiene le interfacce del dao che descrivono la struttura ed il funzionamento delle classi che comunicano con il db e realizzano la persistenza dati.

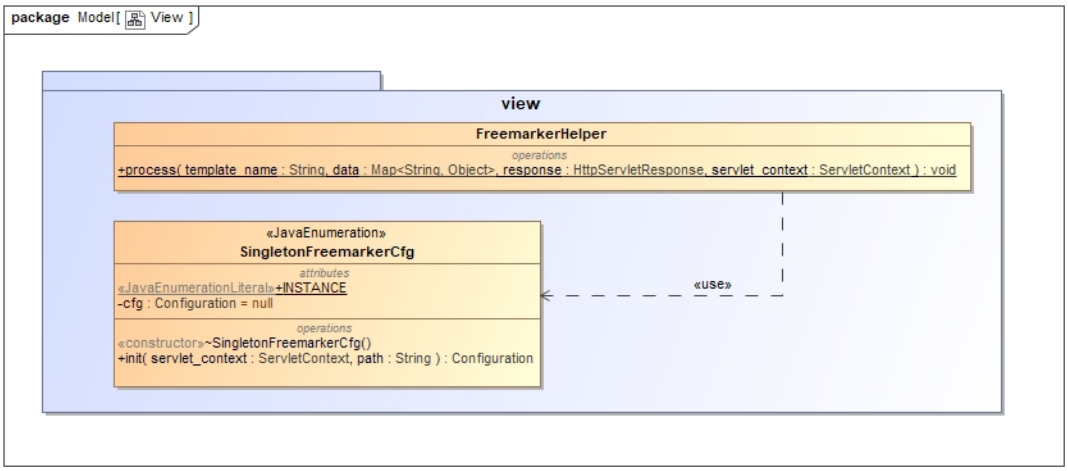
    - Implementation: contiene il cuore del dao, le classi (implementano le interfacce in interfaces) che appoggiandosi alle entità definite nel model interrogano e manipolano i dati sul database, queste sono le uniche classi che hanno la capacità di istanziare oggetti del model.

- Model: contiene le classi che rappresentano le entità sul db.

Di seguito una rappresentazione diagrammatica del contenuto dei vari packages.

3.1.2 Class Diagrams

3.1.2.1 gamingplatform.view

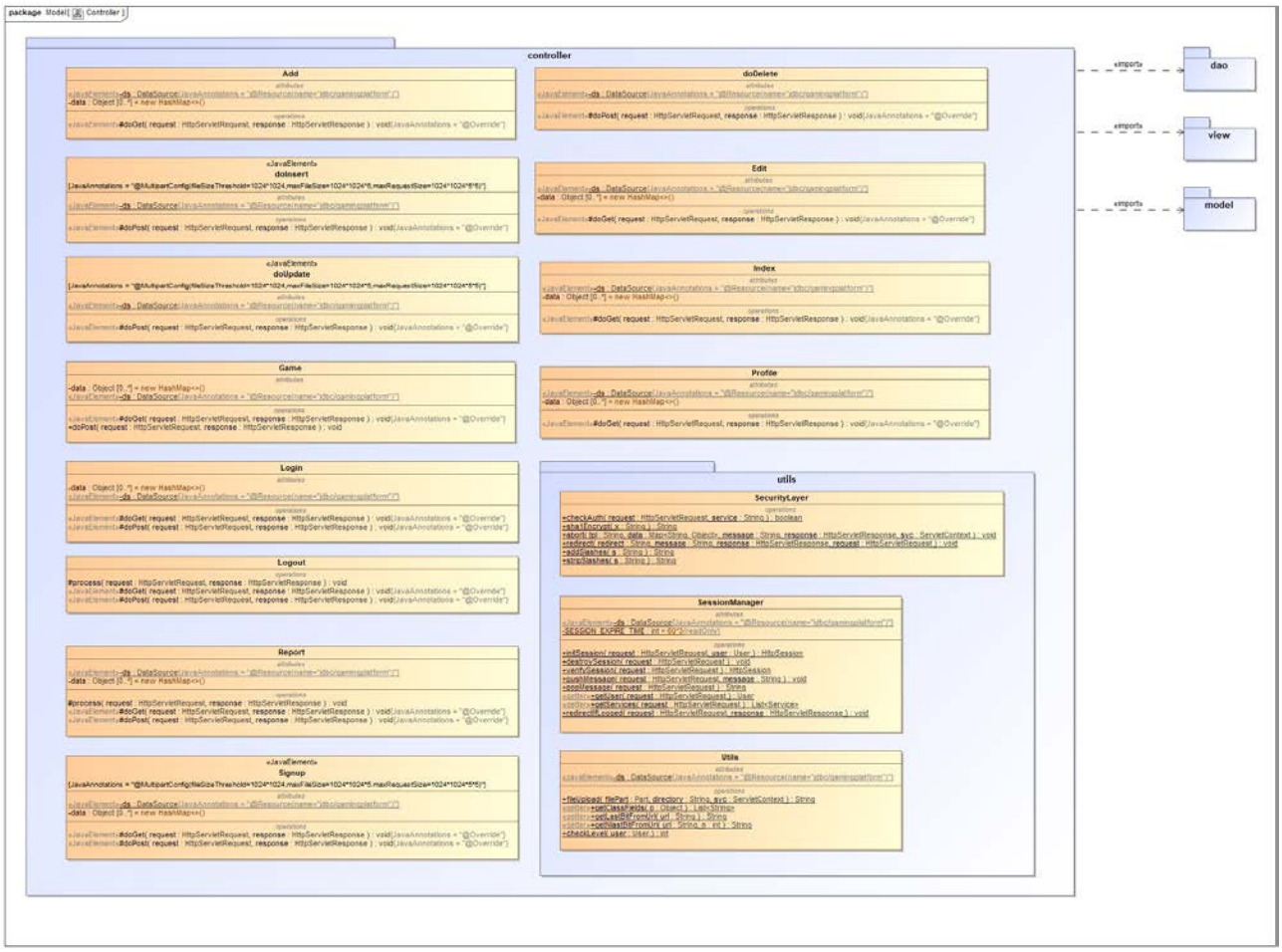


- gamingplatform.view.FreemarkerHelper: contiene il metodo process che data una configurazione dalla classe singleton SingletonFreemarkerCfg processa appunto la template iniettando i dati caricati su una Map.

- gamingplatform.view.SingletonFreemarkerCfg: è la classe singleton che gestisce la configurazione di freemarker.

  Come specificato in precedenza la classe singleton è stata realizzata utilizzando Enum.
  Enum è uno speciale di tipo di dato che vincola una variabile ad assumere solo un determinato set di valori.
  Nel nostro caso utilizziamo questa proprietà per creare una classe enum con un metodo init che restituisce l'oggetto configuration di freemarker istanziandolo se non esiste.
  Questo ci permette di avere sempre solo un'unica istanza attiva nel sistema.
  Nel dettaglio, nella prima chiamata al metodo init, l'oggetto configuration sarà a null, init in caso configuration sia null procede all'istanziazione e lo restituisce al chiamante.
  Da questo momento in poi l'oggetto configuration esiste e non è null, quindi ogni successiva chiamata a init restituirà sempre lo stesso oggetto.

3.1.2.2 gamingplatform.controller



- gamingplatform.controller.Add: servlet che risponde solo a richieste GET che ha il compito di presentare l'interfaccia web relativa all'inserimento di una generica entità nel db, nel dettaglio recupera la struttura della tabella del db e dinamicamente genera la form di inserimento.

- gamingplatform.controller.Edit: analoga alla Add ma genera il form di modifica (ed eliminazione).

- gamingplatform.controller.doDelete: servlet che risponde solo a chiamate Ajax che, disambiguando in base al parametro ricevuto sulla URL elimina (o meglio delega al dao) una data entità dal database.
  L'accesso in generale alle servlet di CRUD è ristretto agli aventi diritto.

- gamingplatform.controller.doInsert: analoga alla doDelete, effettua l'inserimento di entità nel db.

- gamingplatform.controller.doUpdate: analoga alla doDelete e alla doInsert, si occupa di effettuare l'update delle entità sul database.

- gamingplatform.controller.Game: servlet che si occupa di presentare l'interfaccia relativa alla pagina di un dato gioco, ha inoltre il compito, una volta recuperati i dati relativi al gioco dal dao di calcolare la media dei voti.
  In caso di chiamata POST (ovvero l'utente ha effettuato una giocata) calcola l'exp da aggiungere all'utente, se loggato, e torna un messaggio popup all'interfaccia che indica se l'utente ha semplicemente guadagnato exp ed in caso quanta, oppure se ha raggiunto un nuovo livello.

- gamingplatform.controller.Index: servlet che presenta l'interfaccia relativa alla homepage dell'applicativo, nel dettaglio recupera l'elenco dei giochi dal database (attraverso il dao) e li presenta all'utente.

- gamingplatform.controller.Login: servlet che presenta l'interfaccia relativa al login dell'utente, in chiamata POST raccoglie i dati ricevuti e procede all'autenticazione se i dati sono corretti, altrimenti notifica l'utente dell'errore.

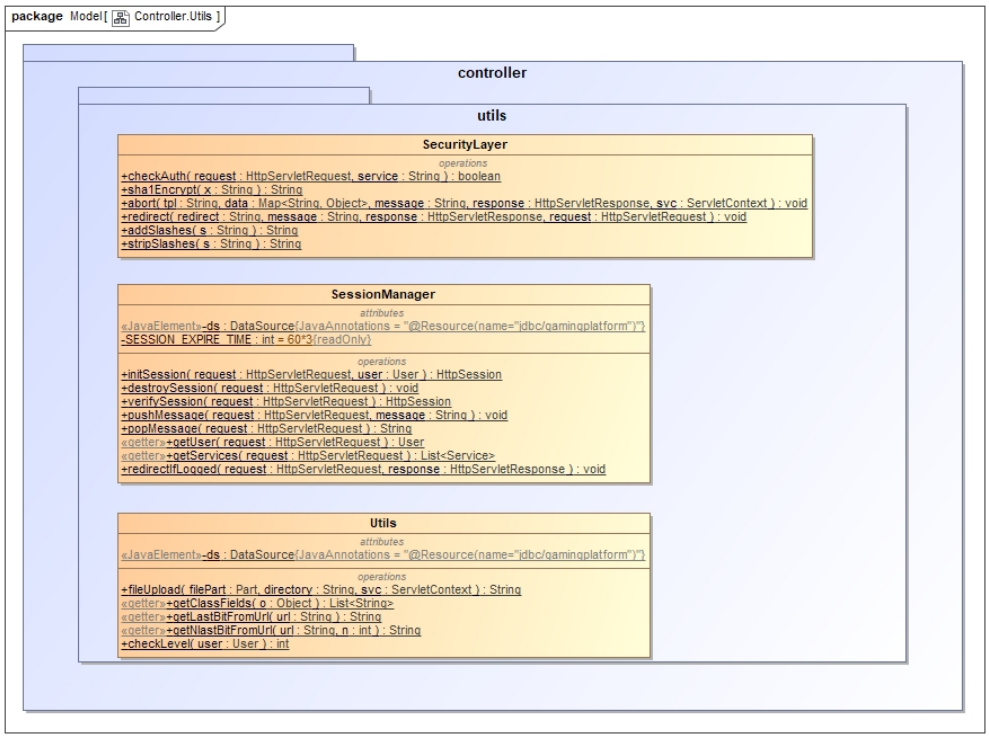
- gamingplatform.controller.Logout: servlet che si occupa di effettuare il logout dell'utente distruggendo la sessione a lui associata.

- gamingplatform.controller.Profile: servlet che presenta all'utente i dati relativi al suo profilo, tra cui i grafici relativi agli ultimi 5 livelli raggiunti/ultime 5 giocate effettuate e l'elenco completo di tutte le giocate e di tutti i livelli/trofei guadagnati.

- gamingplatform.controller.Report: servlet del backoffice che presenta agli utenti autorizzati i report delle entità sul database (es. elenco di tutti gli utenti registrati).
  Nel dettaglio, in base alla tabella d'interesse, recupera la sua struttura e l'elenco delle tuple.

- gamingplatform.controller.Signup: mostra l'interfaccia relativa alla registrazione di un nuovo utente.
  In caso di chiamata POST delega al dao l'inserimento del nuovo utente sul database.

3.1.2.2.1 gamingplatform.controller.utils

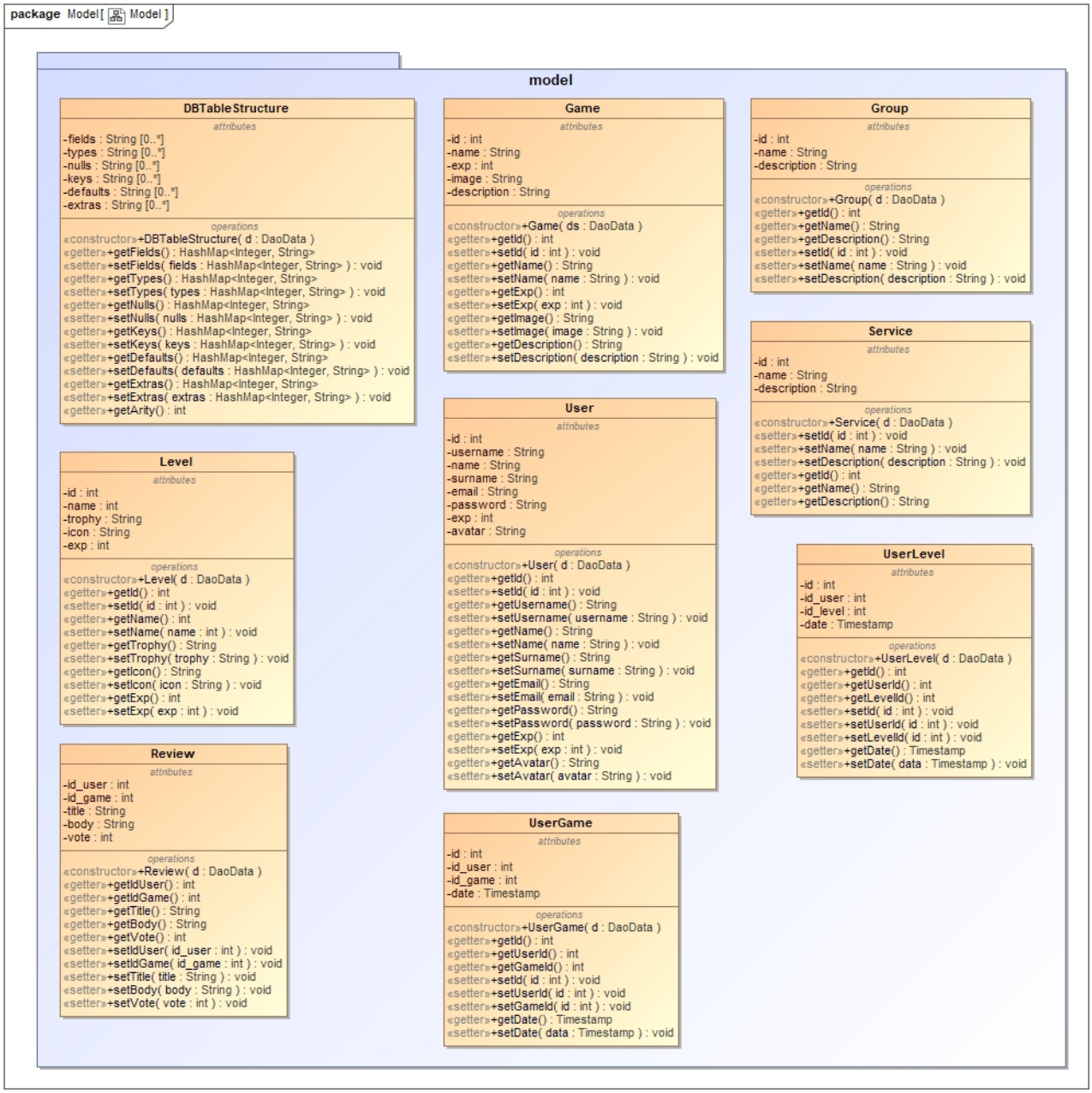


- gamingplatform.controller.utils.SecurityLayer: classe che contiene i metodi atti alla sicurezza del sistema, nel dettaglio contiene tutti i metodi relativi al sanitizing dei dati provenienti dalle form, cifratura della password in sha1 ecc.

- gamingplatform.controller.utils.SessionManager: contiene i metodi che gestiscono le sessioni, ovvero creazione, verifica, eliminazione.
  Contiene inoltre i metodi necessari all'iniezione di messaggi popup nell'interfaccia web, nel dettaglio quando una servlet ha necessità di lasciare un messaggio all'utente prima di effettuare redirect o comunque di chiudere la connessione fa il push del messaggio in sessione; nella servlet di arrivo la prima operazione sarà quella di pop, così, se c'è, il messaggio viene passato alla map di freemarker che lo manipola e lo presenta all'utente.

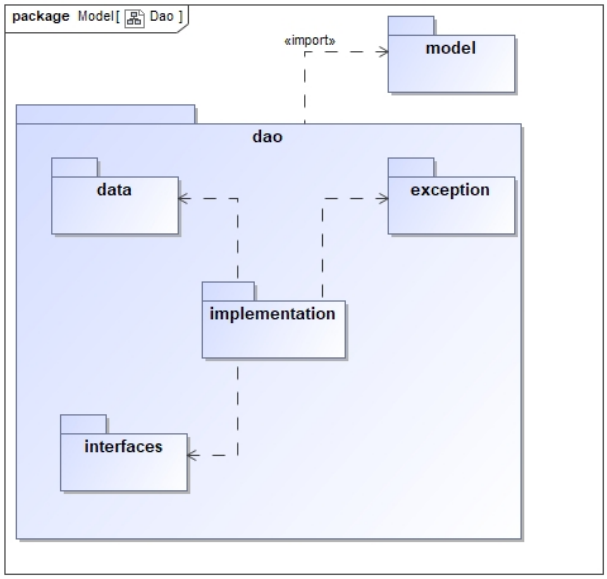
- gamingplatform.controller.utils.Utils: contiene metodi utility non classificati, tra ciu il metodo relativo all'upload dei file (avatar utente e immagine copertina dei giochi), i metodi che recuperano le informazioni dalla URL e il metodo che ha il compito di verificare il livello di un utente:
  Questo metodo prende in input un User e controlla che il livello a cui si trova attualmente l'utente sia effettivamente quello in cui dovrebbe essere, in caso l'exp fosse tale da richiedere un avanzamento di livello, il metodo calcola i livelli che mancano per raggiungere il livello "giusto" e li assegna all'utente, promuovendolo.
  Viceversa può anche retrocedere l'utente.
  Questo metodo è particolarmente utile a seguito di una massiccia modifica dell'exp da parte di una giocata a un gioco che assegna un generoso quantitativo di exp tale da far salire l'utente di più livelli con una sola giocata oppure a seguito dell'intervento di un moderatore/amministratore.

3.1.2.3 gamingplatform.model



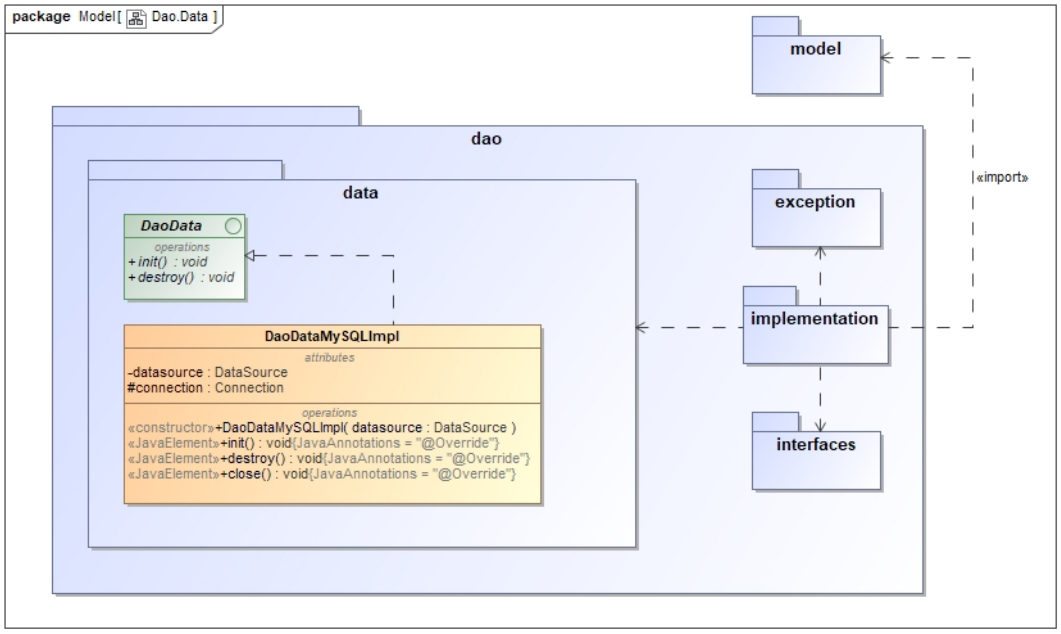
Contiene per quanto possibile la rappresentazione 1:1 delle entità sul database con i set/get, in più contiene una classe DBTableStructure utilizzata per contenere i dati relativi alla struttura (che campi contiene, di che tipo, se una dato campo è nullable, ecc.) di una data tabella sul db.

3.1.2.4 gamingplatform.dao



Come spiegato nel package diagram (sez. 3.1.1) contiene classi ed interfacce che realizzano il design pattern **data access object**.

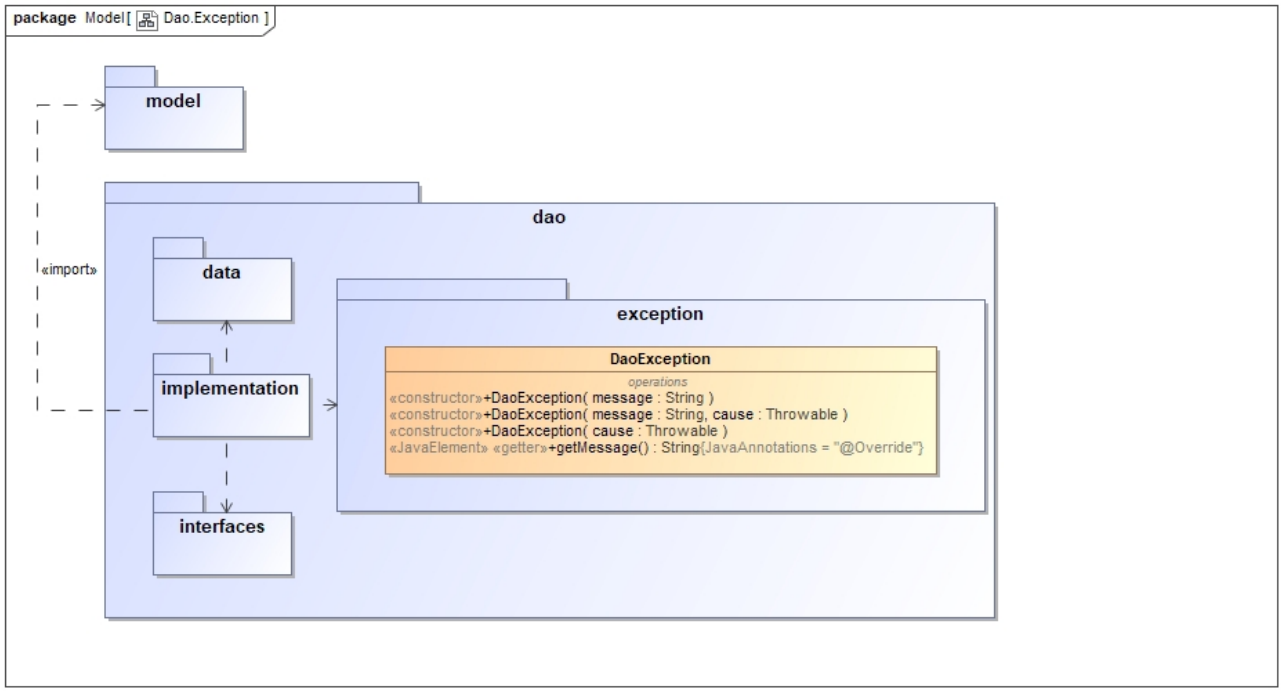
3.1.2.4.1 gamingplatform.dao.data



- **gamingplatform.dao.data.DaoData:** interfaccia che regola la struttura della classe che si occupa della gestione della connessione tramite connection pooling.

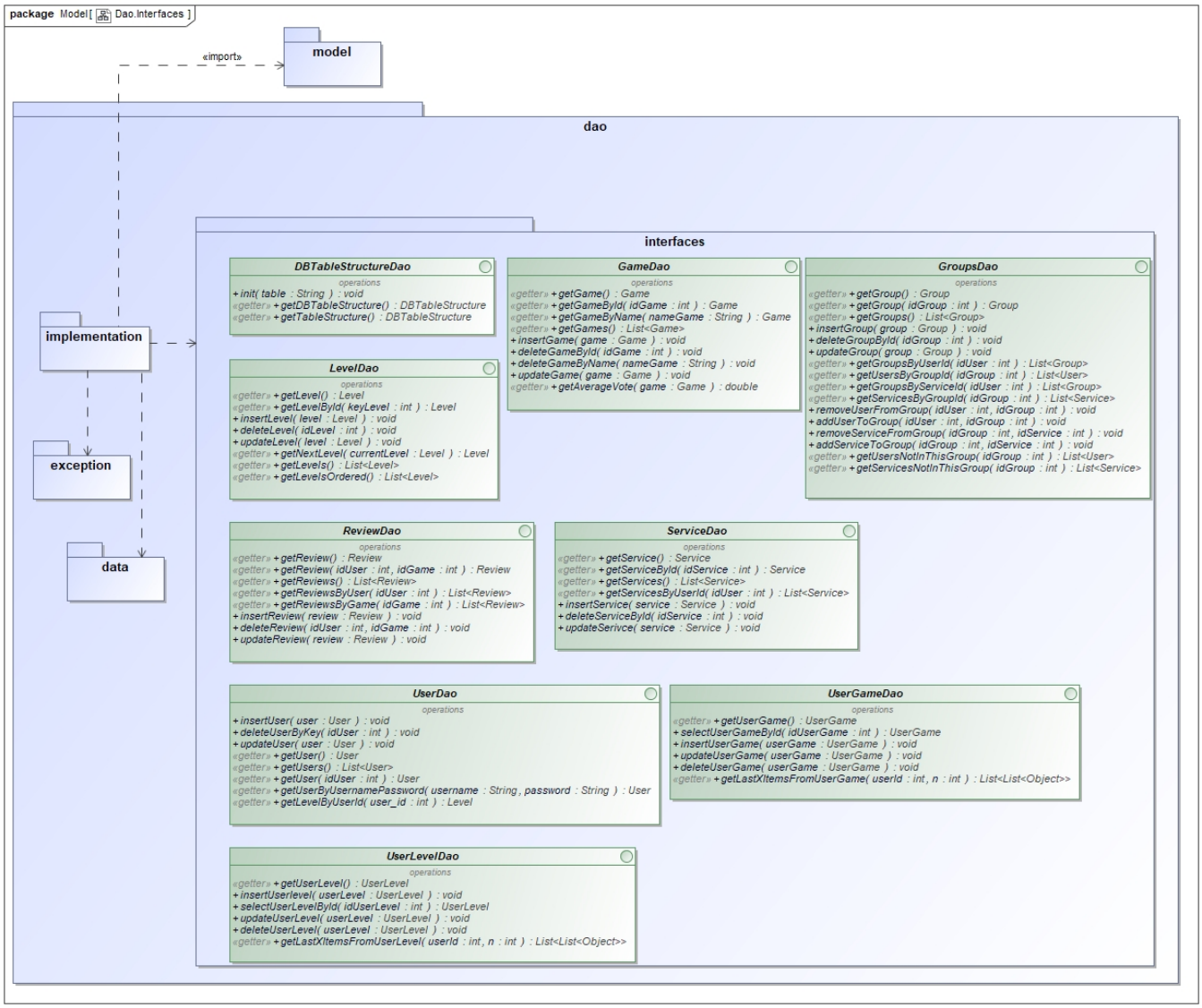
- **gamingplatform.dao.data.DaoDataMySQLImpl:** classe che implementa l'interfaccia DaoData per connessione al database MySQL.

3.1.2.4.2 gamingplatform.dao.exception



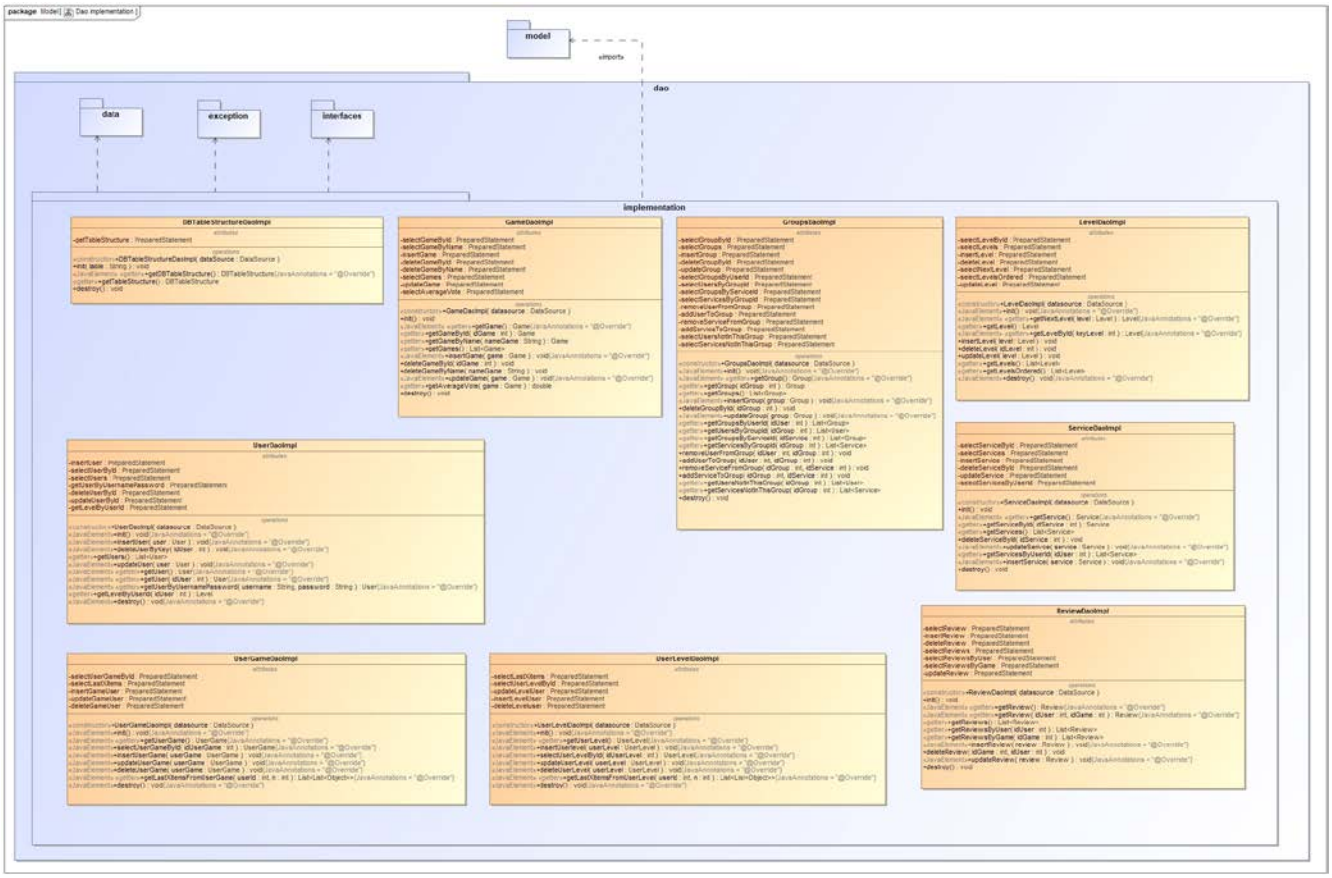
- gamingplatform.dao.exception.DaoException: contiene i metodi per la gestione dell'eccezione DaoException.

3.1.2.4.3 gamingplatform.dao.interfaces



Sottopackage che contiene le interfacce del Dao.
Ogni interfaccia impone la struttura ed il comportamente alle rispettive classi che le implementano per la comunicazione con il db.

3.1.2.4.4 gamingplatform.dao.implementation



Sottopackage che contiene le implementazioni delle interfacce del package **Interfaces.**
Queste classi importano le classi del **model, l'exception,** ed estendono **data.**
Tutte le classi hanno due metodi **init() e destroy()** che rispettivamente inizializzano e distruggono la connessione al db.
È stato deciso di impostarlo in questo modo per poter fare l'init una sola volta ed utilizzare i metodi della classe senza avere la necessità di chiudere la connessione e riaprirla per ogni operazione, ovvero si fa l'init, si eseguono i metodi di interesse e solo alla fine si chiama il destroy.
Inoltre è stato deciso di creare una classe Dao per ogni entità del db invece che un'unica grande classe che contenesse tutti i metodi di tutte le entità per avere una divisione netta nell'organizzazione del software.
Nel senso, se e quando si avrà necessità di apportare delle modifiche queste interesseranno solo una classe e non andranno ad intaccare le altre, inoltre ci ha permesso di lavorare in modo parallelo.
Le classi di questo package sono le **uniche** che hanno possibilità di istanziare oggetti nel **model.**
Ovvero abbiamo bisogno che il controller, se e quando ha bisogno di ottenere un'istanza del model la chieda alla rispettiva classe Dao, questo per separare logicamente il controller dal modello dati e mantenere una certa dinamicità.
Quindi, essendo sempre e solo il Dao che comunica col model possiamo apportare modifiche solo ai metodi interessati del Dao quando cambiamo model senza intaccare il controller.

3.1.2.5 gamingplatform

Class diagram completo dell'intero sistema.
(risorsa disponibile a risoluzione originale in doc/images).