“Object Oriented Software Design”
Course
a.a. 2016-2017

Team Members			
Name & Surname	Matriculation Number	E-mail address	Anno
Stefano Corsetti	*227288*	*s.corsetti@hotmail.it*	*3°anno*
Luca D'Orazio	*227635*	*lucaadorazio@gmail.com*	*3°anno*
Tommaso Di Salle	*236202*	*tommasodisalle@gmail.com*	*3°anno*
Eugenio Mancini	*230024*	*Emancini1992@libero.it*	*3°anno*

INDICE

Documento dei requisiti

System Design

Software Design

A. Documento dei requisiti

A.1 Requisiti Funzionali

1. **Registrazione** di un utente alla piattaforma (vedi scenario 0-1)
   Un utente avrà la possibilità di registrarsi alla piattaforma di gaming e usufruire dei servizi offerti.

2. **Visualizzare profilo**
   Qualsiasi utente loggato potrà visualizzare il proprio profilo.

3. **Visualizzare giochi**
   Qualsiasi utente può visualizzare la lista dei giochi presenti nella piattaforma.

4. **Votare gioco**
   Un utente loggato può esprimere un voto sul gioco.

5. **Recensione gioco**
   Un utente loggato può inserire una recensione.

6. **Sessione di gioco**
   Un utente loggato può effettuare una sessione di gioco e guadagnare punti esperienza.

7. **Collezionare trofei**
   Un utente al raggiungimento di un determinato punteggio può accedere al livello successivo e guadagnare il trofeo associato a quel livello.

8. G**estione recensioni**
   Un moderatore può accedere alla lista delle recensioni inserite ma non ancora approvate e decidere di approvarla oppure eliminarla.

9. **Gestione utente**
   Un moderatore può far diventare un utente un moderatore e far diventare un moderatore un utente base.

A.2 Requisiti Non Funzionali

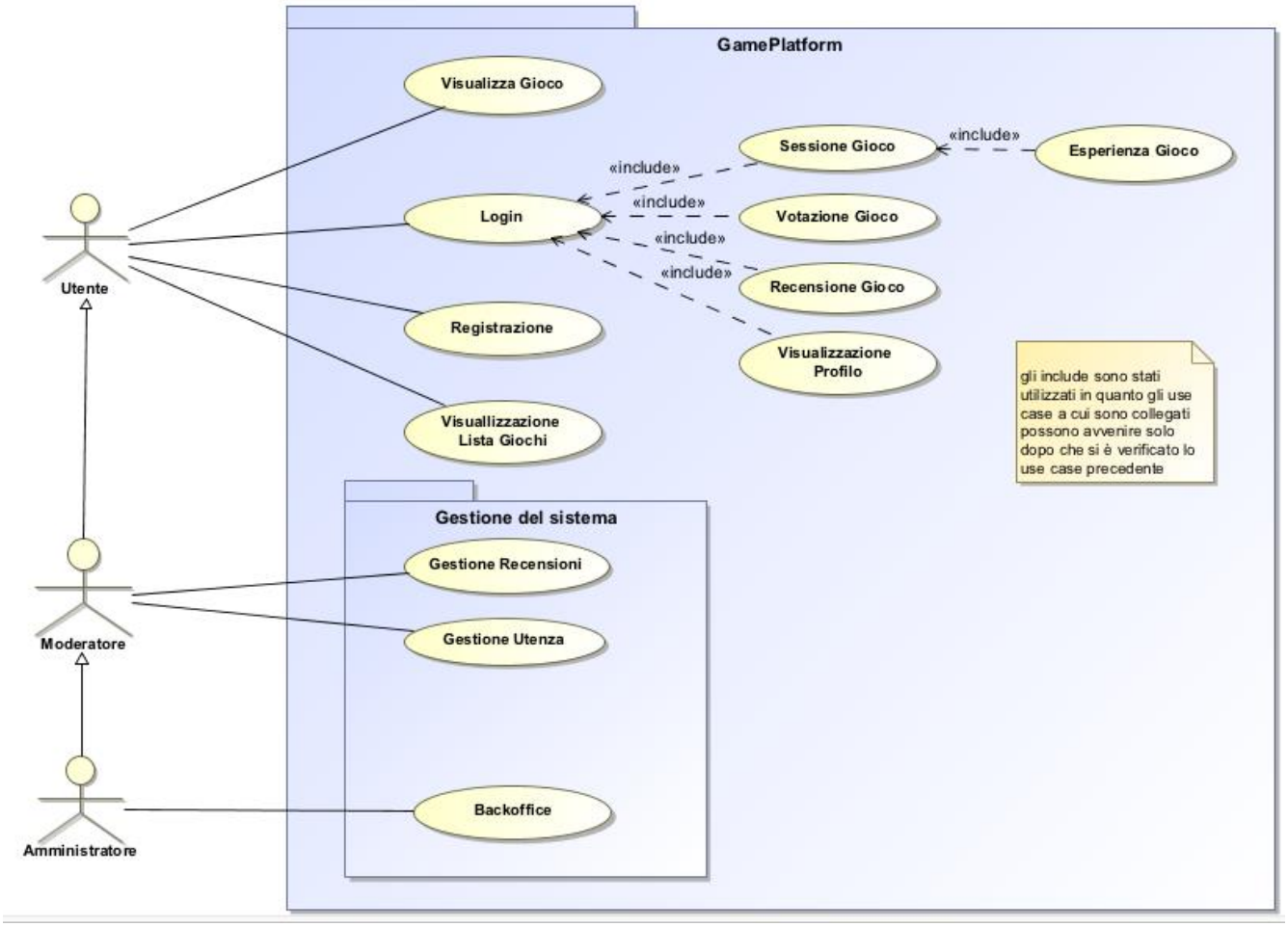
1. **Usability**: Il sistema, in particolare la UI, deve essere intuitiva e facile da utilizzare.

2. **Performance**: Il sistema deve permettere all'utente di svolgere le funzioni in modo efficiente e veloce.

3. **Maintainability**: Il sistema deve essere facilmente mantenibile, aperto a migliorie e aggiunte di nuove funzionalità.

4. **Aviability**: La piattaforma dovrà essere sempre disponibile e garantire in ogni momento tutte le funzioni desiderate dall'utente.

A.6 Use Case



A.7 Descrizione - Use Case (identificazione attori)

Il primo attore individuato nel sistema è **l'Utente** che ha diritto di accesso alle funzionalità di login/registrazione, visualizzazione giochi, visualizzazione profilo, sessione gioco, votazione, recensione.

Il secondo attore identificato è il **Moderatore** che ha funzionalità di un utente più le seguenti funzionalità:
- gestione recensioni: approvazione/eliminazione di una recensione.
-gestione utenza: Upgrade/Downgrade di un utente a moderatore e da moderatore a utente.

L'ultimo attore identificato è **l'Amministratore** con tutte le funzioni del sistema con accesso al BackOffice (pannello di gestione del sistema) dove potrà effettuare le operazioni CRUD sulle entità presenti.

A.8 Descrizione - Use Case

Nome Use-Case	**Login/Registrazione**
Attori partecipanti	Utente - Moderatore - Amministratore
Descrizione	Permette ad un utente/Moderatore/Amministratore registrato di accedere al sistema o ad un utente non registrato di registrarsi.
Evento scatenante	Click su relativo pulsante
Usi	Login/Registrazione effettuati con successo o errore

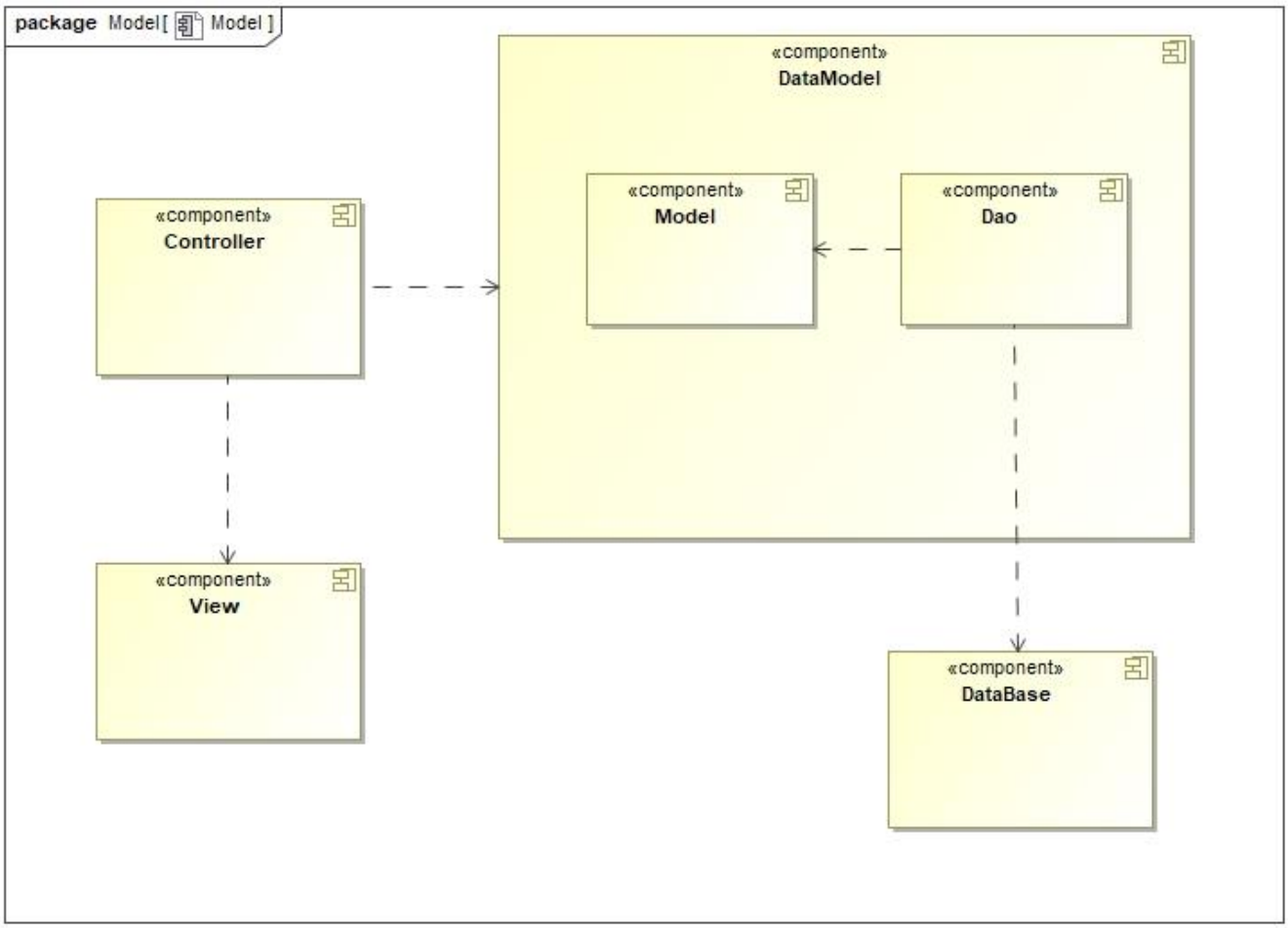
Nome Use-Case	Votazione/Recensione
Attori partecipanti	Utente
Descrizione	Permette a utenti registrati nel sistema di valutare/votare il gioco da 0 a 5 punti nonché di poter scrivere delle recensioni sulla propria esperienza (aspettando l'approvazione)
Evento scatenante	Click su uno dei 5 bottoni per la votazione e "Scrivi recensione" per la recensione

Nome Use-Case	Esperienza Gioco
Attori partecipanti	Utente
Descrizione	Permette a utenti registrati nel sistema la funzione di visualizzare le proprie esperienze e trofei vinti
Evento scatenante	Bottone "Profilo"

*BackOffice: Pannello gestione di sistema per Amministratore e Moderatore

B. System Design

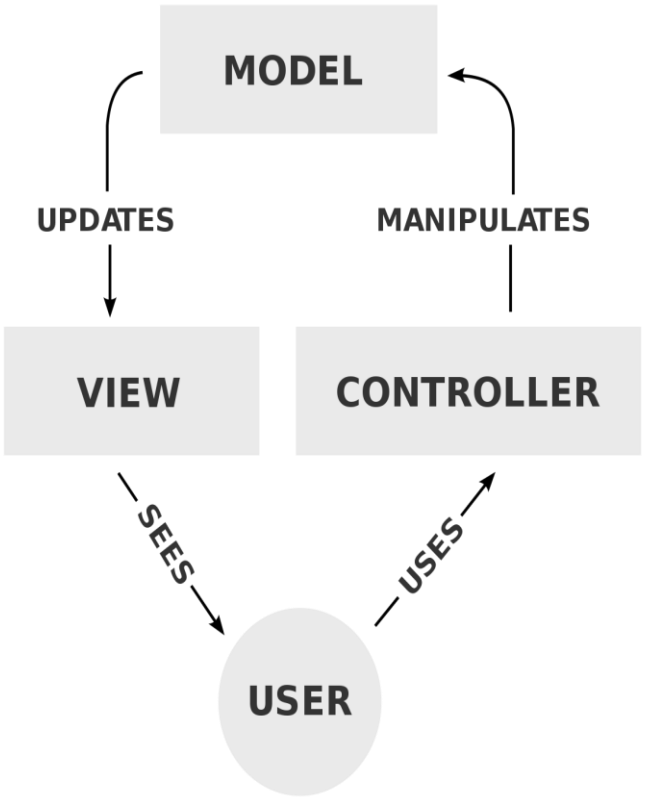
B.1 Component Diagram - Modello architettura software



B.2 descrizione architettura

Il team ha scelto di sviluppare una applicazione web e per pattern architetturale la scelta è ricaduta sul pattern **MVC.**

MVC



Il flusso normale di una applicazione web MVC consiste in una serie di passaggi:

1. un utente richiede di visualizzare una pagina;

2. un Controller riceve tale richiesta e tramite i model , recupera i dati necessari ,li organizza e li spedisce alla view;

3. La View utilizza questi dati per presentare la pagina web finale all' utente.

Il pattern MVC ci permette di separare i modelli di dominio, di presentazione e delle azioni in tre componenti separate Model, View e Controller.

Le componenti sono quindi DataModel, View ,Controller e Database.

View: Definisce ciò che viene presentato all' utente, e permette di inviare gli user-input ai Controller .Nella piattaforma sarà costituita da pagine html rese dinamiche dal template engine **FreeMarker.**

Controller: Definisce il comportamento dell' applicazione mappando azioni dello User a manipolazione dei Model. Nella nostra applicazione saranno tutte Servlet.

DataBase: Si occuperà della persistenza dei dati. Nello specifico il team ha scelto di utilizzare il database MySQL.

DataModel: Contiene i dati, le entità, e i meccanismi per accedere a essi.
nello specifico:
1) Model: sono tutti classi POJO ovvero classi con i soli metodi get/set per la manipolazione dei campi;

2) DAO: i metodi per inserire/recuperare i dati.

Per garantire maggiore astrazione il team ha deciso di utilizzare un altro pattern: il **DAO.**

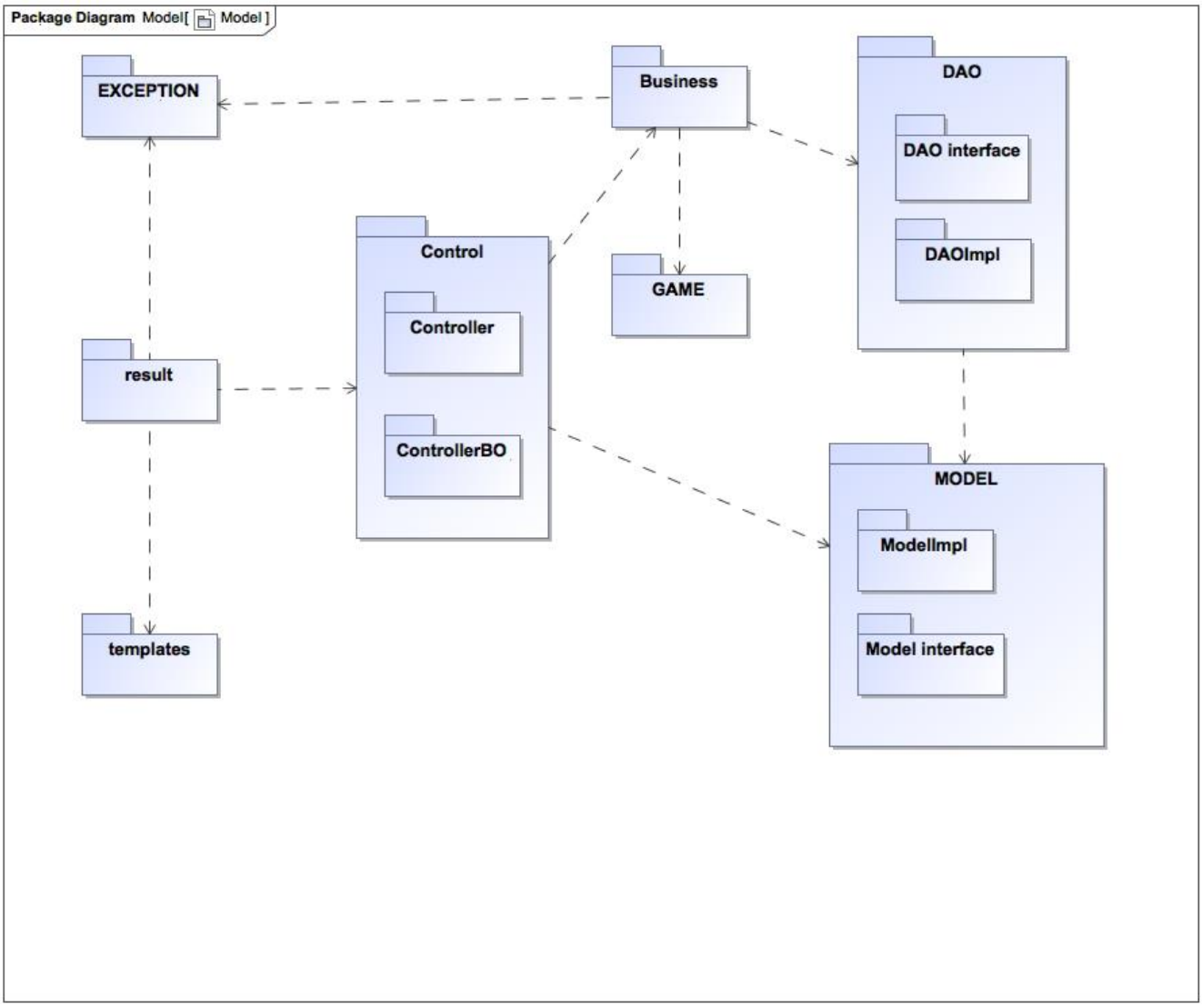
Il **DAO** (Data Access Object) è un pattern che ci permette di separare le due parti dell'applicazione che non possono conoscere nulla tra loro. Questo pattern viene utilizzata per separare i controller dalle classi che implementano i dettagli realizzativi della base di dati.

Questo pattern è costituito da due parti :

BaseDao : interfaccia usata per gli accessi alla base di dati contente i metodi *loadAll()*, *create(Object object), load(Object key), store(Object object), remove(Object key) ,getConnection()* e *closeDbConnection()* usati per inserire, caricare, aggiornare, cancellare.

ConcreteDao : Classe concreta che implementa l'interfaccia BaseDao (per ulteriori dettagli si veda package diagram).

B.3 Package Diagram



Il team cercando di massimizzare il riuso, l'anticipazione di nuovi requisiti, le modifiche ai requisiti esistenti e la progettazione del sistema in modo che possa evolvere conseguentemente, ha identificato i seguenti package:

- **Controller**: contiene tutte le Servlet relative alle singole pagine del sistema. E' stato suddiviso ulteriormente in: Controller e ControllerBO. Esso utilizza Model, Business, Result e Exception.

- **DAO:** contiene tutte le classi e le interfacce per la realizzazione del DAO, suddiviso in:
  - *nomeClasse:*interfacce che definiscono le operazioni effettuabili per entità
  - *nomeClasse*Impl: classi che implementano le interfacce e realizzano in modo concreto le operazioni, usando il Model.

- **Exception**: package adibito alla gestione degli errori.

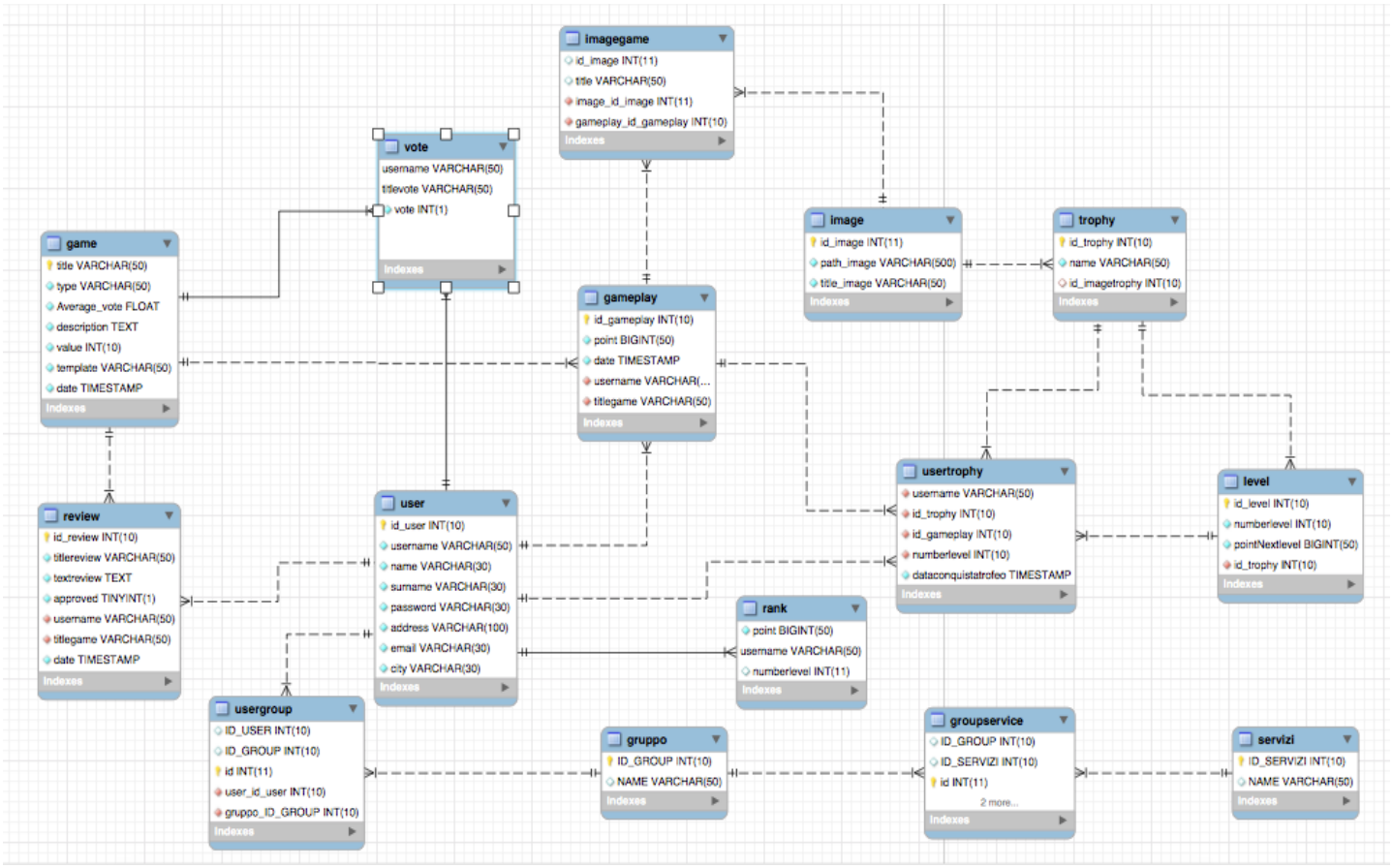
- **Model**: tutte le classi rappresentanti le entità gestite dal sistema.

- **Result**: package contente classi che popolano il template e lo restituiscono allo user.

- **Game**: package contentente le classi per la gestione dell'assegnazione dell'esperienza.

- **Business:** Ulteriore Layer di separazione tra Controller e DAO.

B.4 Modello ER



C. Software/Object Design

C.1 Class Diagram

Il team, ha deciso di descrivere non le classi singolarmente (facilmente deducibili dal class diagram di seguito allegato), ma analizzarle a livello di package, dato che le funzionalità sono più o meno le stesse.

- **Control (Controller / ControllerBO)**

In **Controller** troviamo le seguenti classi:

- *GamePlay*: Servlet che gestisce la sessione di gioco.
- *Home*: Servlet che gestisce la pagina iniziale dell'applicazione dove viene mostrata la lista dei giochi presenti nella piattaforma.
- *Logout*: Servlet che gestisce la terminazione di una sessione.
- *MyProfile*: Servlet che gestisce la pagina dei profili personali dove sarà possibile vederele info personali la timeline, la current balance e la lista dei trofei conquistati.
- *PaginaGioco*: Servlet che mostra una descrizione del gioco selezionato, la possibilità di votare e/o recensire tale gioco.
- *Ranking*: Servlet che gestisce la visualizzazione della classifica generale.
- *Register*: Servlet che gestisce la pagina di registrazione.

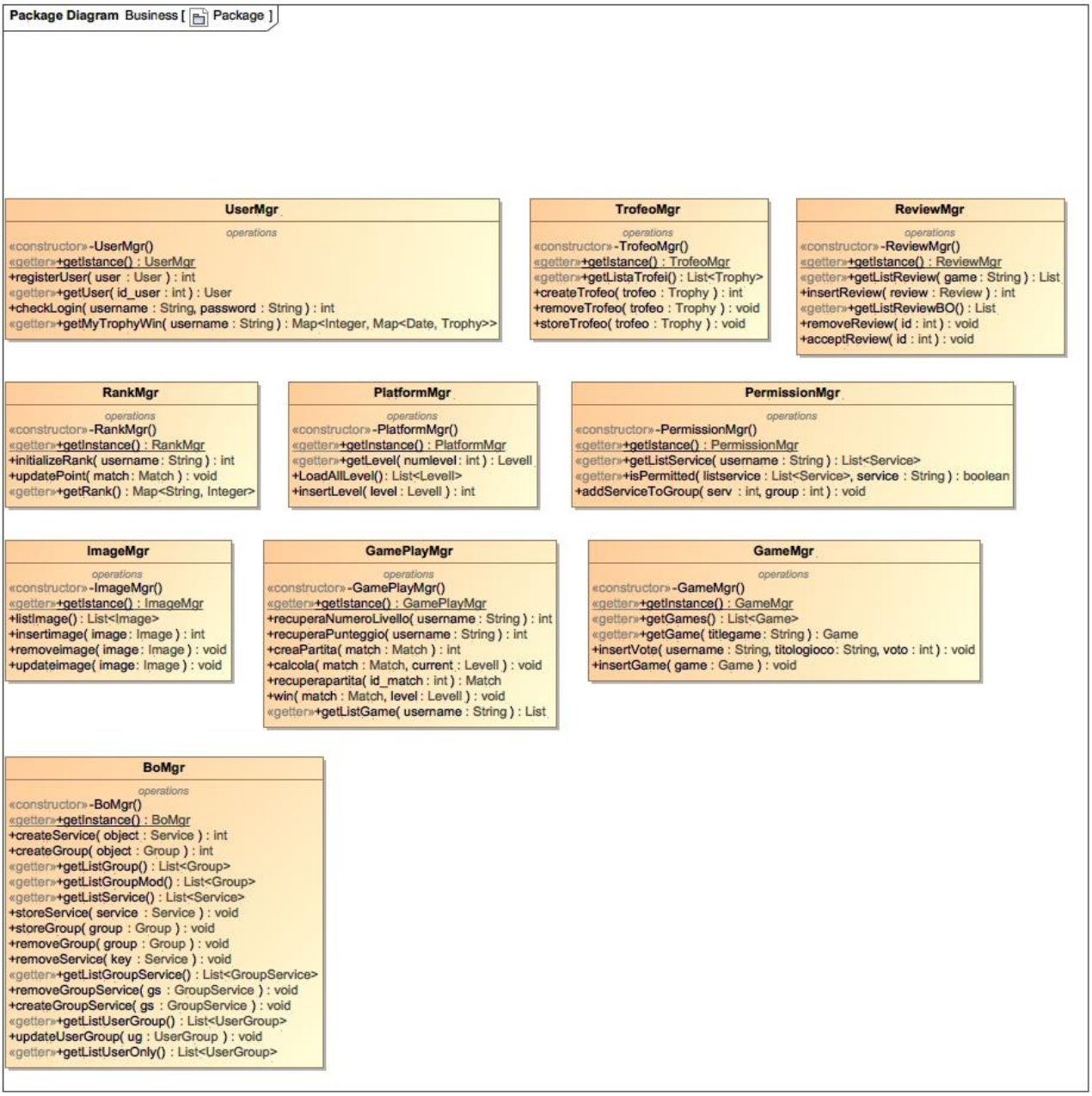
- **ControllerBO**

-In ControllerBO troviamo:

BO, BOgroup, BOgroupeservice, BOlivello, BOreview, BOservizi, BOgame, BOimmagine, BOtrofeo, BOuserpermission.

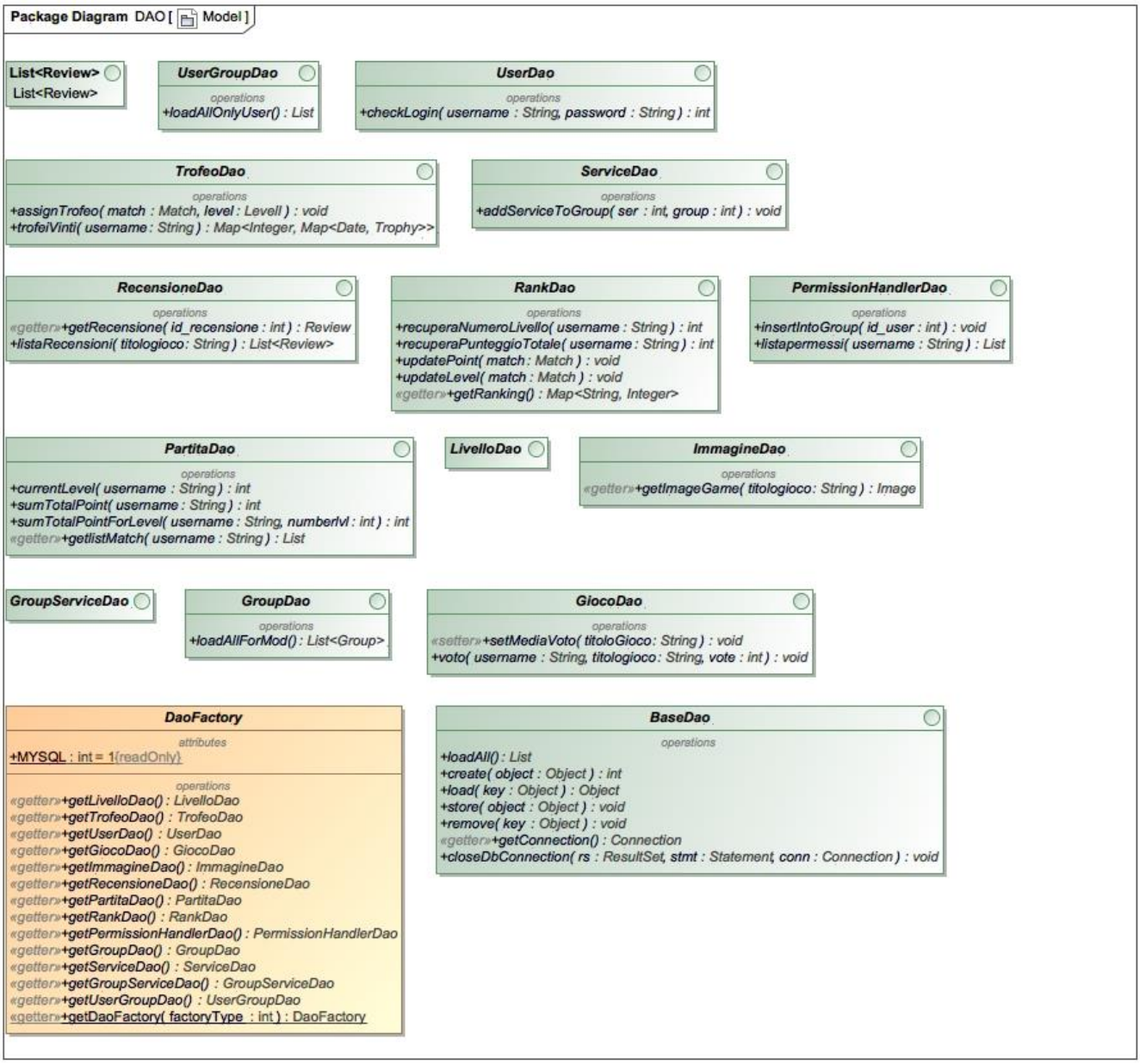
Qui ogni classe è adibita alle operazioni CRUD del relativo modello. Le classi BOgroup E BOreview sono accessibili solo a moderatori e admin. Tutte le altre solo a l'ADMIN.

- **Business**

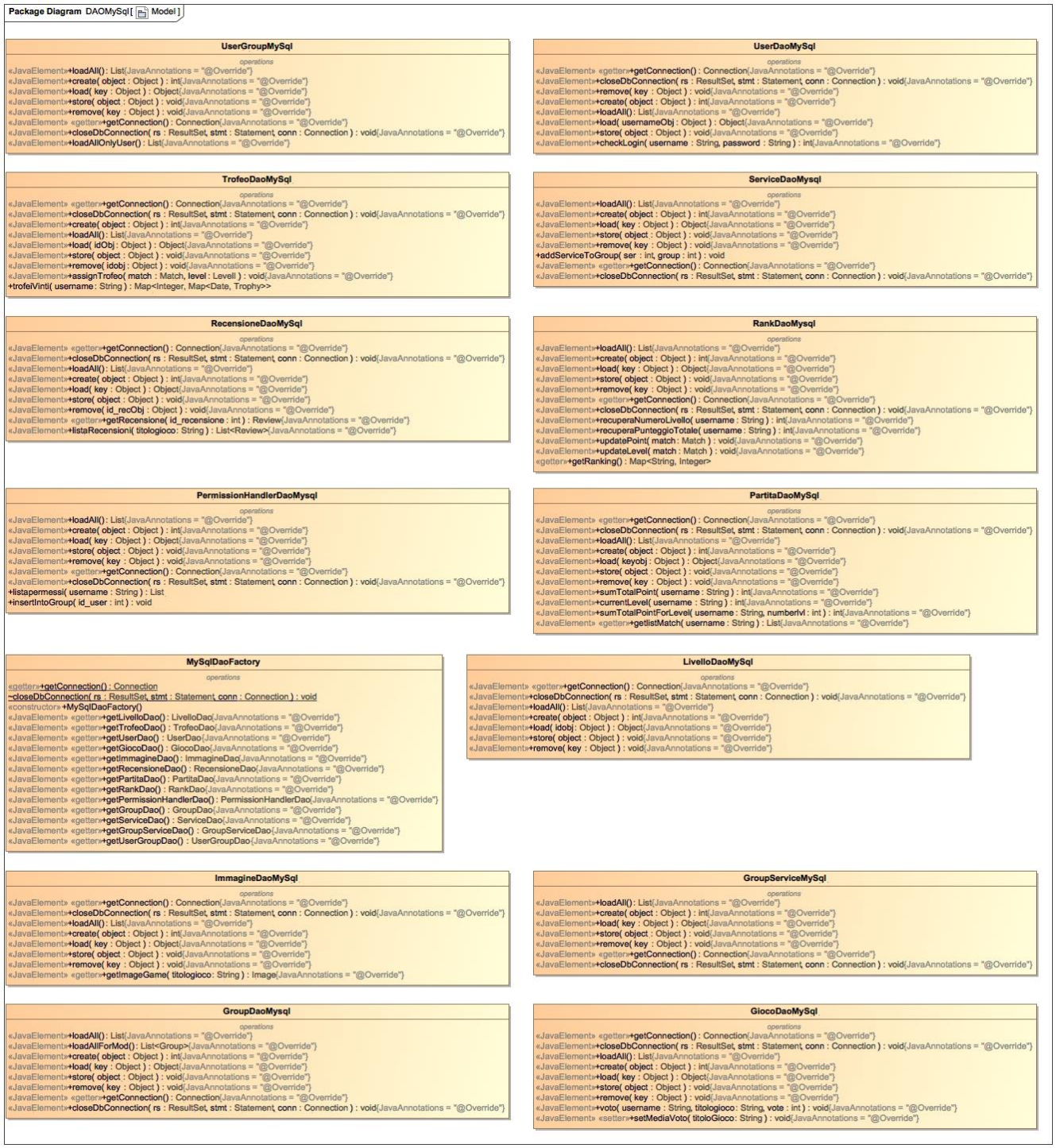


Questo package contiene le seguenti classi:
-BOMgr, GameMgr, GamePlayMgr, ImageMgr, PermissionMgr, PlatformMgr, RankMgr, ReviewMgr, TrofeoMgr, UserMgr. Ogni classe di questo package è cosi strutturata:
ha un solo campo privato *dbfactory* di tipo DAOfactory. Un costruttore che inizializza tale campo e un metodo statico *getIstance* che ritorna un istanza della classe.
Ogni classe di questo package ha metodi relativi a metodi della relativa classe del package DAO.

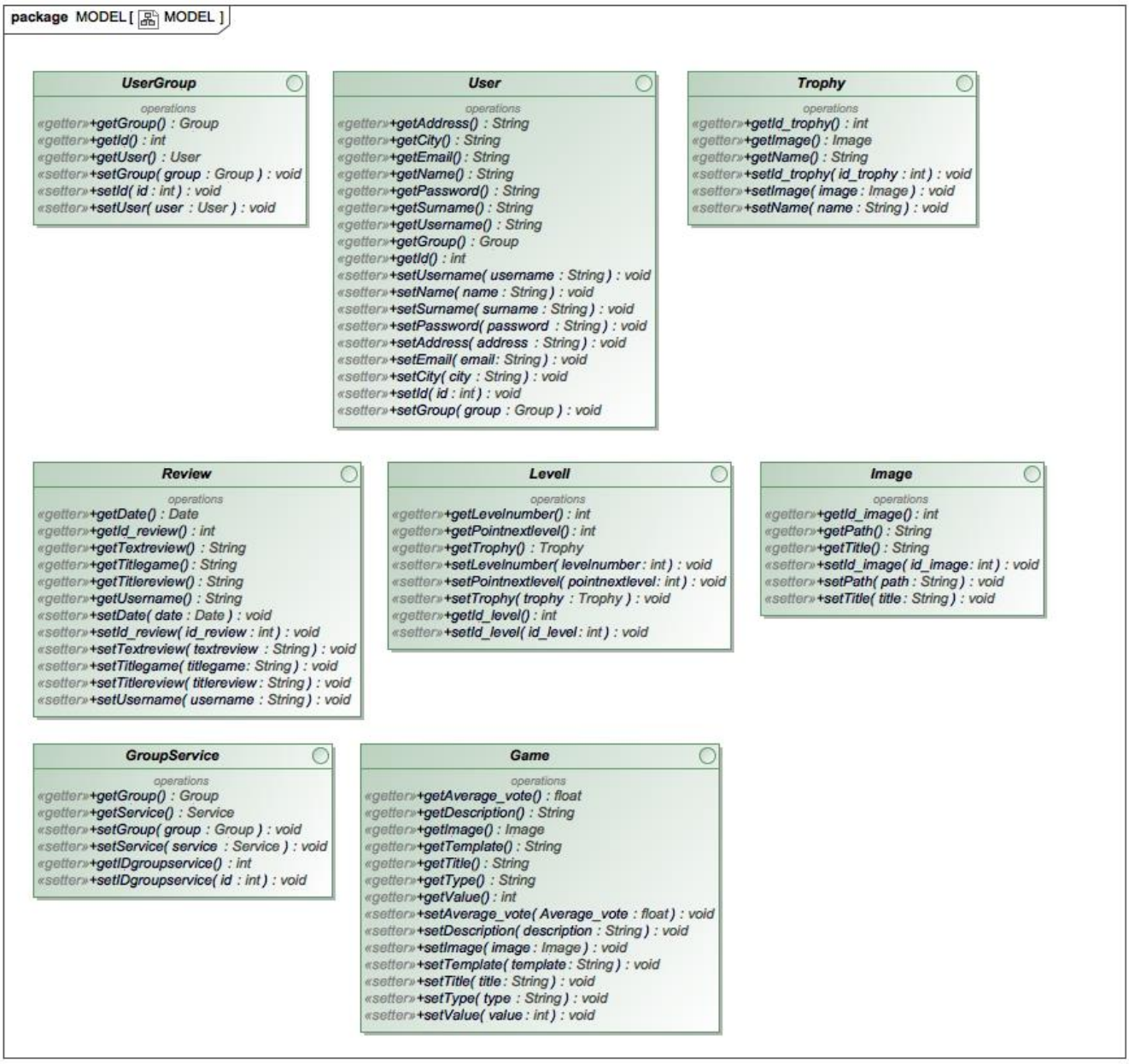
- **DAO (DAO / DAOMySql)**



- **DAOMySql**


Package Diagram DAOMySql [Model]
UserGroupMySql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAllOnlyUser() : List{JavaAnnotations = "@Override"}
UserDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+load(usernameObj : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+checkLogin(username : String, password : String) : int{JavaAnnotations = "@Override"}
TrofeoDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+load(idObj : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(idobj : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+assignTrofeo(match : Match, level : LevelI) : void{JavaAnnotations = "@Override"}
+trofeiVinti(username : String) : Map<Integer, Map<Date, Trophy>>
ServiceDaoMysql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
+addServiceToGroup(ser : int, group : int) : void
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
RecensioneDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(id_recObj : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getRecensione(id_recensione : int) : Review{JavaAnnotations = "@Override"}
«JavaElement»+listaRecensioni(titologioco: String) : List<Review>{JavaAnnotations = "@Override"}
RankDaoMysql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+recuperaNumeroLivello(username : String) : int{JavaAnnotations = "@Override"}
«JavaElement»+recuperaPunteggioTotale(username : String) : int{JavaAnnotations = "@Override"}
«JavaElement»+updatePoint(match : Match) : void{JavaAnnotations = "@Override"}
«JavaElement»+updateLevel(match : Match) : void{JavaAnnotations = "@Override"}
«getter»+getRanking() : Map<String, Integer>
PermissionHandlerDaoMysql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
+listapermessi(username : String) : List
+insertIntoGroup(id_user : int) : void
PartitaDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(keyobj : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+sumTotalPoint(username : String) : int{JavaAnnotations = "@Override"}
«JavaElement»+currentLevel(username : String) : int{JavaAnnotations = "@Override"}
«JavaElement»+sumTotalPointForLevel(username : String, numberlvl : int) : int{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getlistMatch(username : String) : List{JavaAnnotations = "@Override"}
MySqlDaoFactory
operations
«getter»+getConnection() : Connection
-closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void
«constructor»+MySqlDaoFactory()
«JavaElement» «getter»+getLivelloDao() : LivelloDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getTrofeoDao() : TrofeoDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getUserDao() : UserDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getGiocoDao() : GiocoDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getImmagineDao() : ImmagineDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getRecensioneDao() : RecensioneDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getPartitaDao() : PartitaDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getRankDao() : RankDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getPermissionHandlerDao() : PermissionHandlerDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getGroupDao() : GroupDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getServiceDao() : ServiceDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getGroupServiceDao() : GroupServiceDao{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getUserGroupDao() : UserGroupDao{JavaAnnotations = "@Override"}
LivelloDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(idobj : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
ImmagineDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getImageGame(titologioco: String) : Image{JavaAnnotations = "@Override"}
GroupServiceMySql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
GroupDaoMysql
operations
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+loadAllForMod() : List<Group>{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
GiocoDaoMySql
operations
«JavaElement» «getter»+getConnection() : Connection{JavaAnnotations = "@Override"}
«JavaElement»+closeDbConnection(rs : ResultSet, stmt : Statement, conn : Connection) : void{JavaAnnotations = "@Override"}
«JavaElement»+loadAll() : List{JavaAnnotations = "@Override"}
«JavaElement»+create(object : Object) : int{JavaAnnotations = "@Override"}
«JavaElement»+load(key : Object) : Object{JavaAnnotations = "@Override"}
«JavaElement»+store(object : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+remove(key : Object) : void{JavaAnnotations = "@Override"}
«JavaElement»+voto(username : String, titologioco: String, vote : int) : void{JavaAnnotations = "@Override"}
«JavaElement» «setter»+setMediaVoto(titoloGioco: String) : void{JavaAnnotations = "@Override"}

- **Model (Model / ModelImpl)**



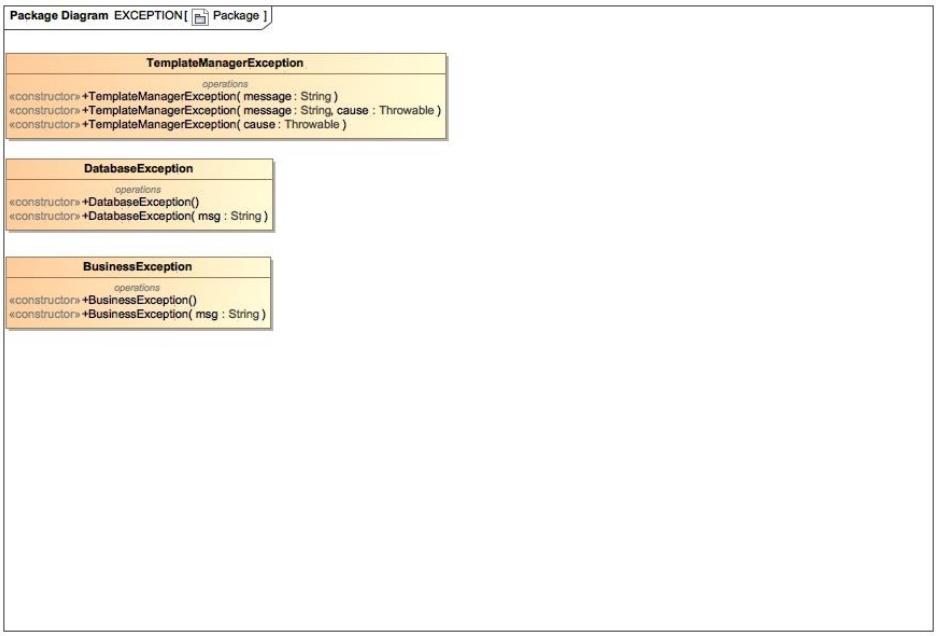
- **ModelImpl**

Questo package è stato suddiviso in: Model e ModelImpl.

Ogni classe del package Model contiene le interfacce con i metodi *get()* e *set().*

Mentre ModelImpl è il package che contiene le classi concrete. Ogni classe corrisponde ad una tabella nel database.

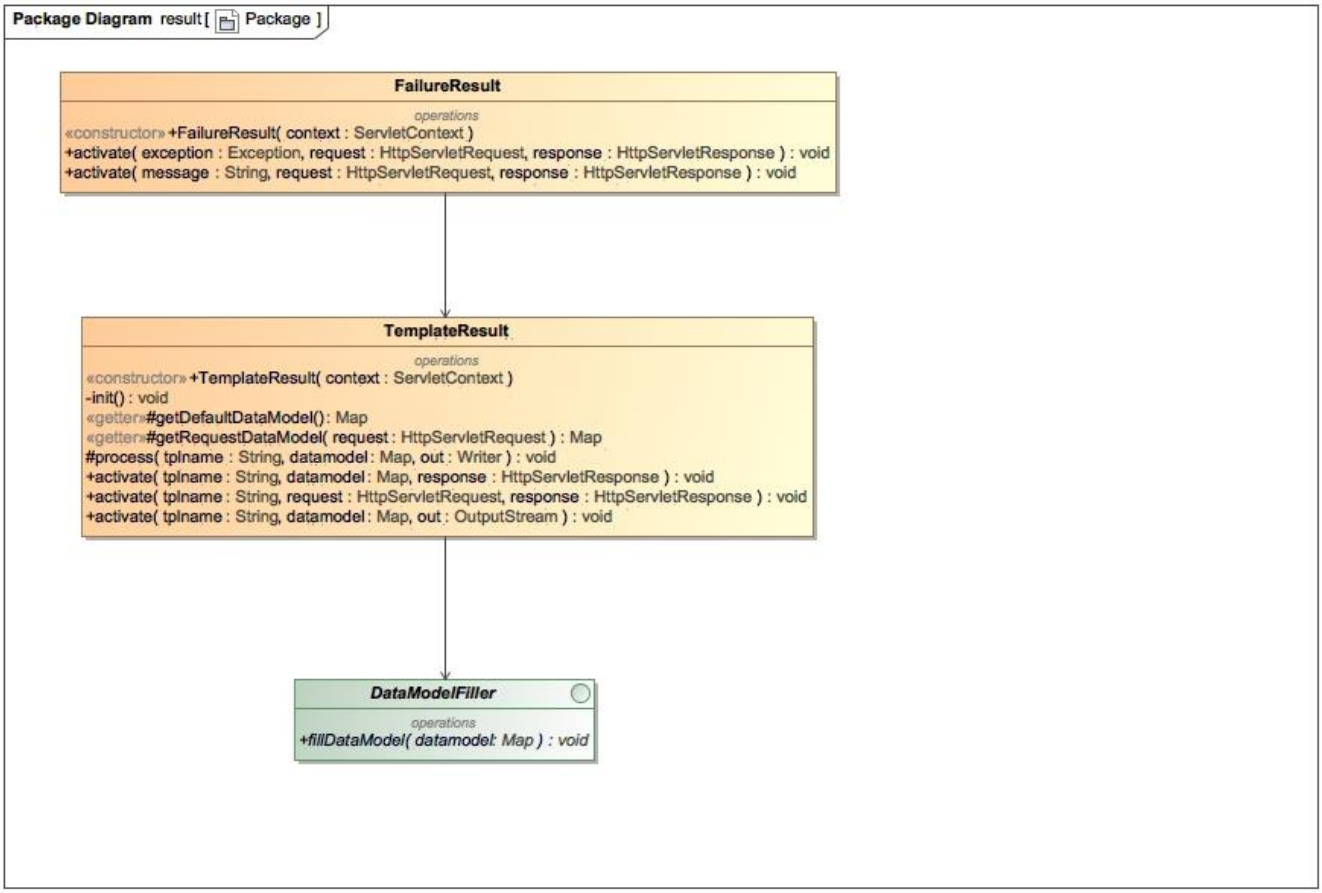
- **Exception**



Contiene tre classi che estendono Exception, denominate: *BusinessException, DataBaseException e TemplateManagerException*. Esse hanno il compito di gestire le eccezioni del relativo package.

BusinessException per le eccezioni di Business, DataBaseException per le eccezioni del Dao e infine TemplateManagerException per le eccezioni del package result.

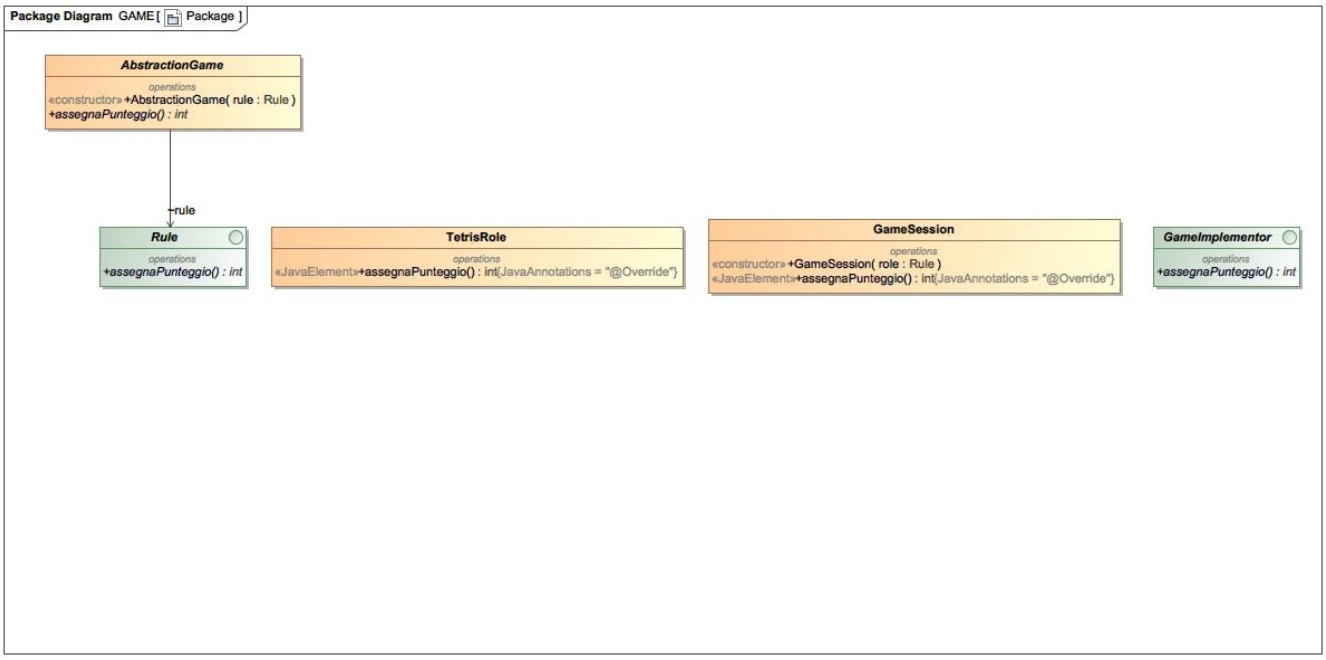
- **Result**



Package contenente le classi per la visualizzazione del template in output:

- *TemplateResult* serve per caricare il template desiderato.
- *FailureResult* classe atta alla visualizzazione degli errori.

- **Game**



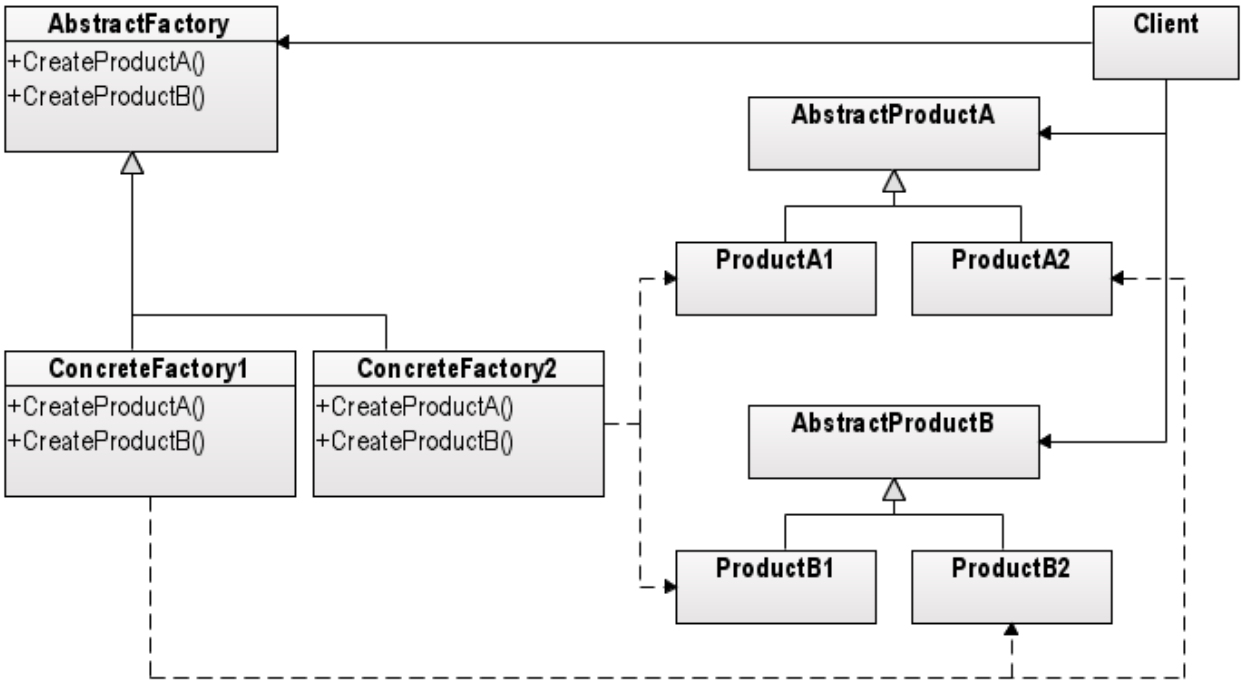
Questo package contiene le classi che specificano le regole di assegnazione dell'esperienza dei giochi.

C.4 Descrizione dettagli design pattern scelti

ABSTRACT FACTORY

Il pattern AbstractFactory può essere utlizzato quando:

- un sistema deve essere indipendente dalle modalità di creazione, composizione e rappresentazione dei suoi prodotti.
- si vuole fornire una libreria di classi, rivelando soltanto le loro interfacce e non le implementazioni.
- Consente di cambiare in modo semplice la famiglia di prodotti utilizzata.



Legenda schema:

1) AbstractFactory consiste in un'interfaccia per le operazioni di oggetti prodotto astratti.
   Tale classe, chiamata DaoFactory, conterrà un metodo astratto per ogni classe del Model e restituira il relativo AbstractProduct e un metodo statico getDaoFactory che restituira la ConcreteFactory scelta.

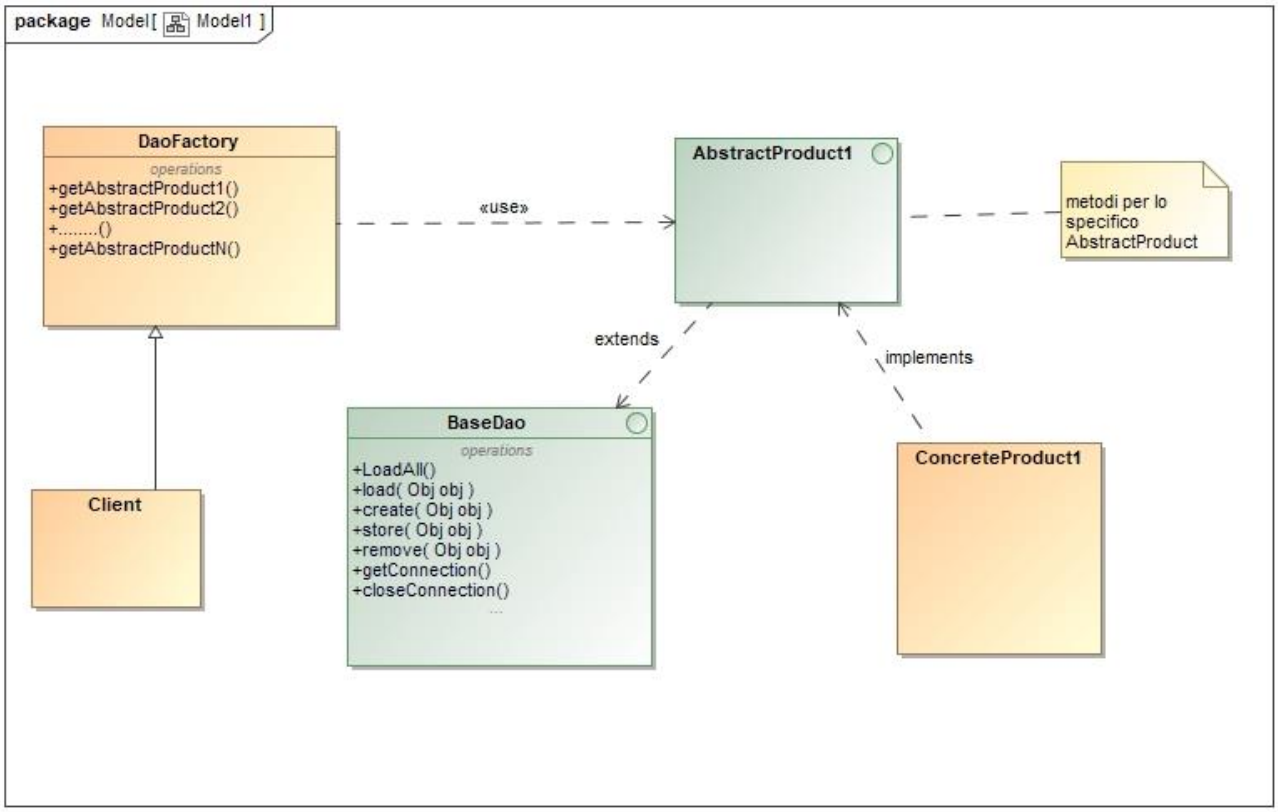
2) ConcreteFactory implementa le operazioni di creazione degli oggetti prodotto concreti.

3) AbstractProduct dichiara un'interfaccia per una tipologia di oggetti prodotto. Ogni AbstractProduct viene chiamato NameProduct+Dao.

4) ConcreteProduct definisce un oggetto prodotto che dovrà essere creato dalla corrispondente factory concreta e implementa l'interfaccia AbstractProduct Ogni ConcreteProduct viene chiamato NameProduct+DaoMySql .

5) Client utilizza solo le interfacce dichiarate da AbstractFactory e AbstractProduct.

Raggruppando i due pattern DAO e ABSTRACT FACTORY otteniamo il seguente diagramma.



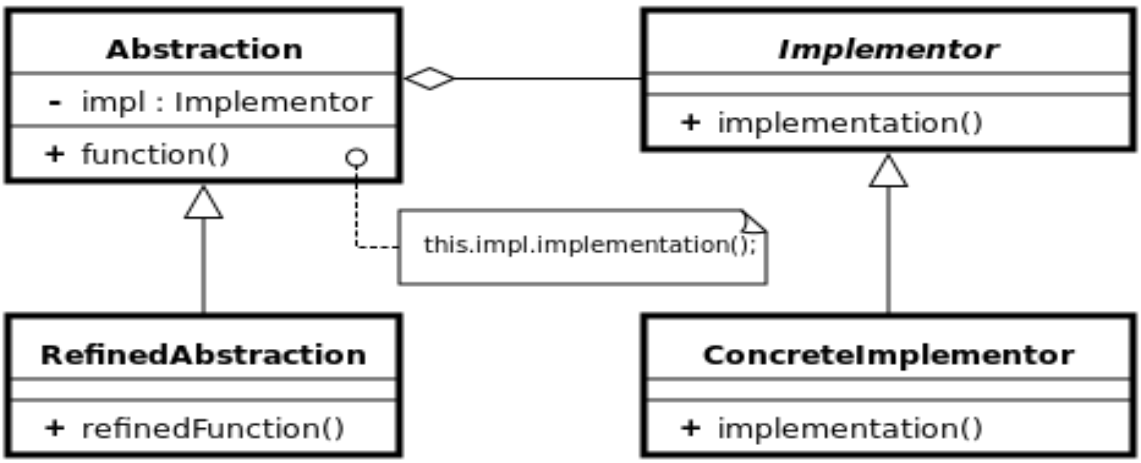
Così facendo se dovessimo cambiare la base di dati, l'interfaccia che il modello DAO fornisce resta invariata e verrebbero modificate le sole classi concrete contenti i dettagli implementativi, cioè i vari ConcreteProduct.

Sessione di gioco

Il team ha deciso di slegare completamente l'assegnazione dei punteggi dalla piattaforma dall' assegnazione dei punteggi del gioco. Per ottenere questo si è scelto di utilizzare il pattern **Bridge**.

L'uso di questo pattern ha le seguenti conseguenze:

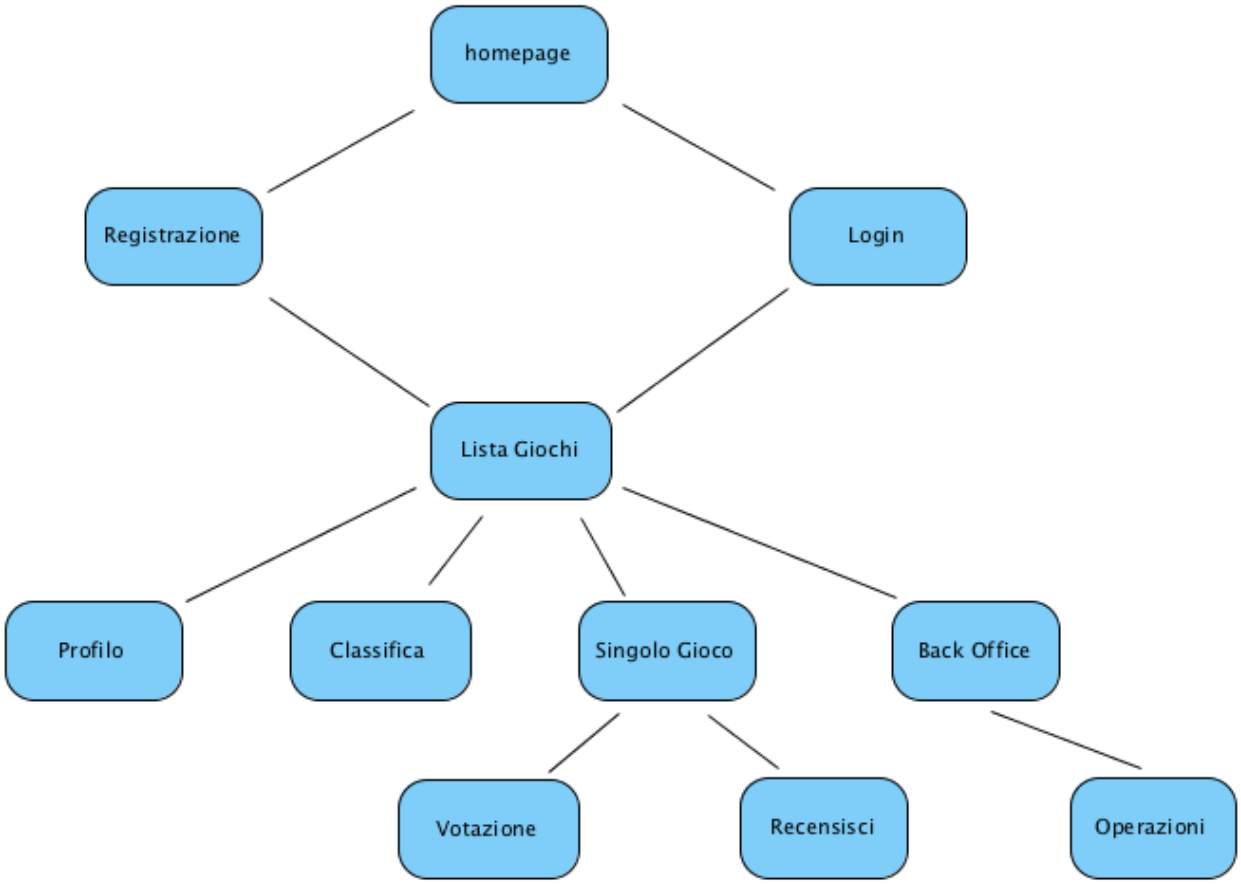
- Disaccoppiamento tra interfaccia e implementazione.
- Maggiore estendibilità.
- Mascheramento dei dettagli dell'implementazione ai Client.



L'unico metodo definito in Abstraction che deve essere implementato da ogni ConcreteImplementor è assegnaPunteggio(Object bj) che ritorna un intero (ovvero la exp guadagnata).

In questo modo ad ogni gioco può essere associato l'algoritmo più opportuno di assegnazione del punteggio.

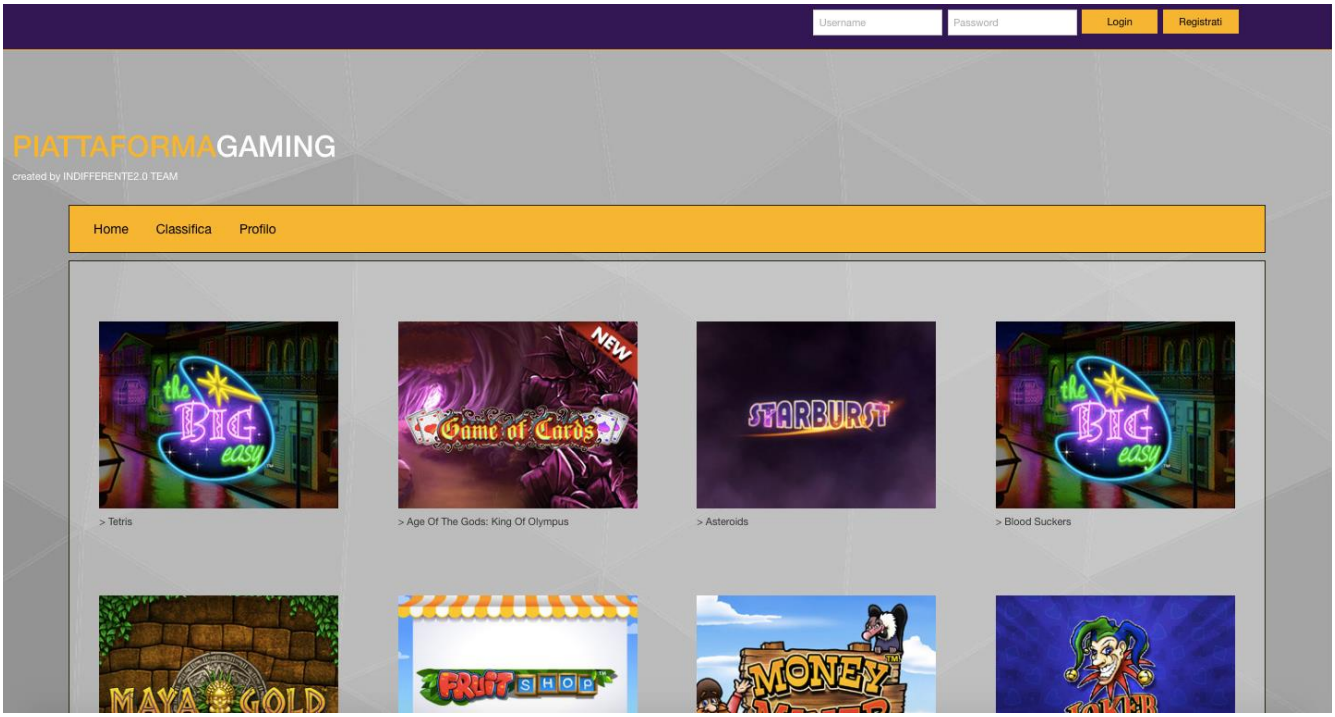
C.2 Site Map



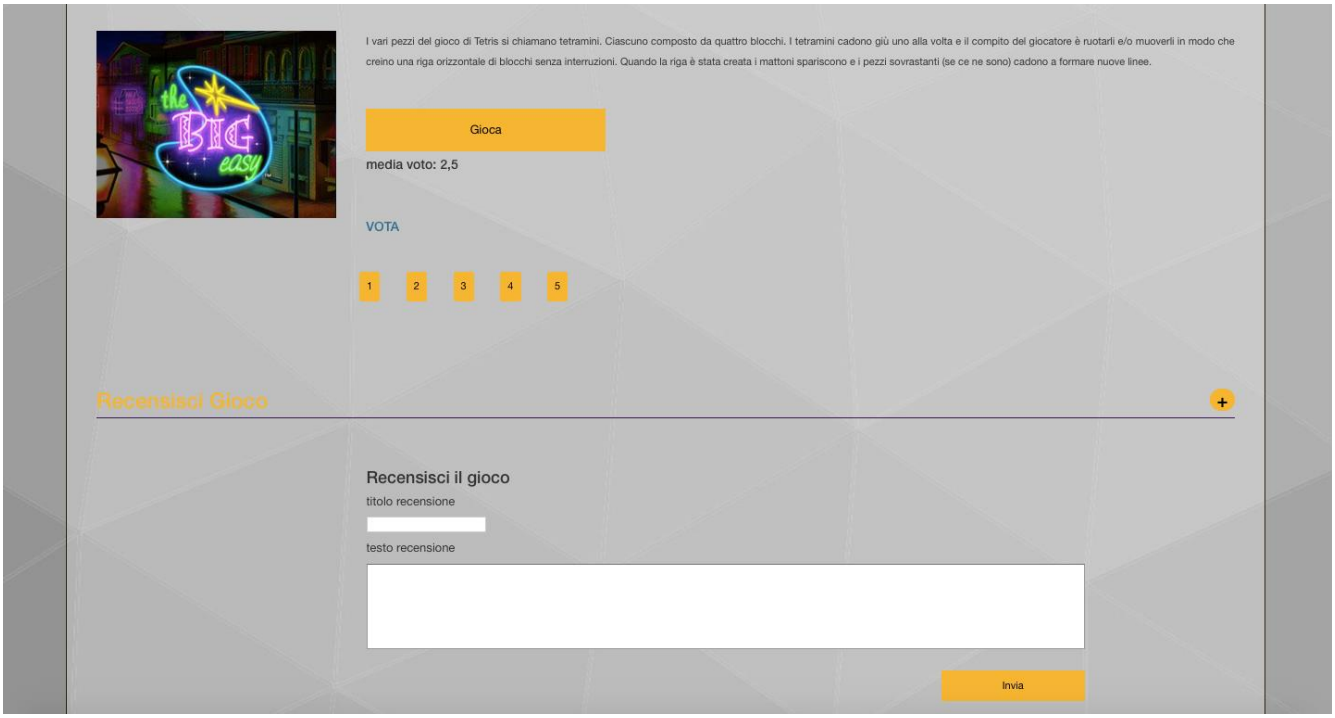
Nel seguente grafico viene mostrata la sequenza delle azioni che l'utente può compiere interagendo con il nostro sistema. Nel caso in cui l'utente sia registrato come Admin o Moderatore, avrà la possibilità di accedere al pannello di gestione del sistema(Backoffice) con le relative operazioni. Nel caso in cui l'utente in questione è un utente "normale", il sistema non da la possibilità a quest'ultimo di accedere al BackOffice.

Di seguito alcuni screen:

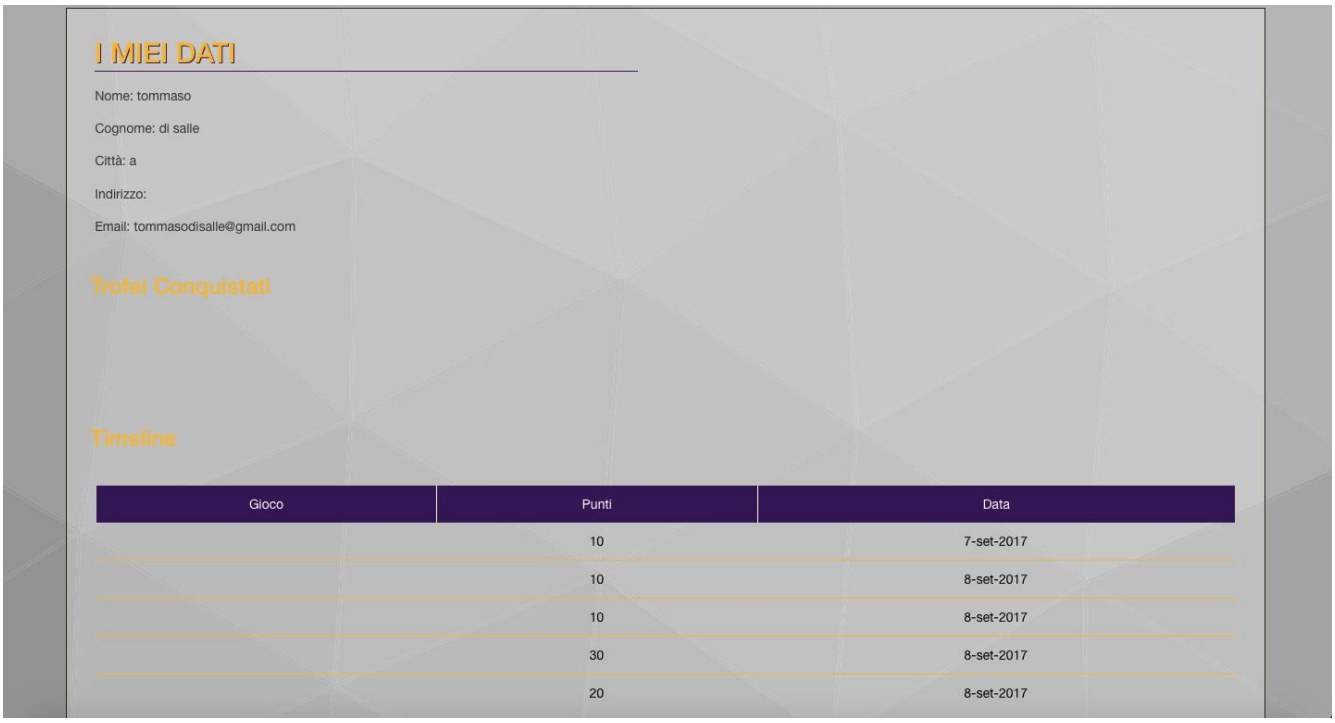
- Home



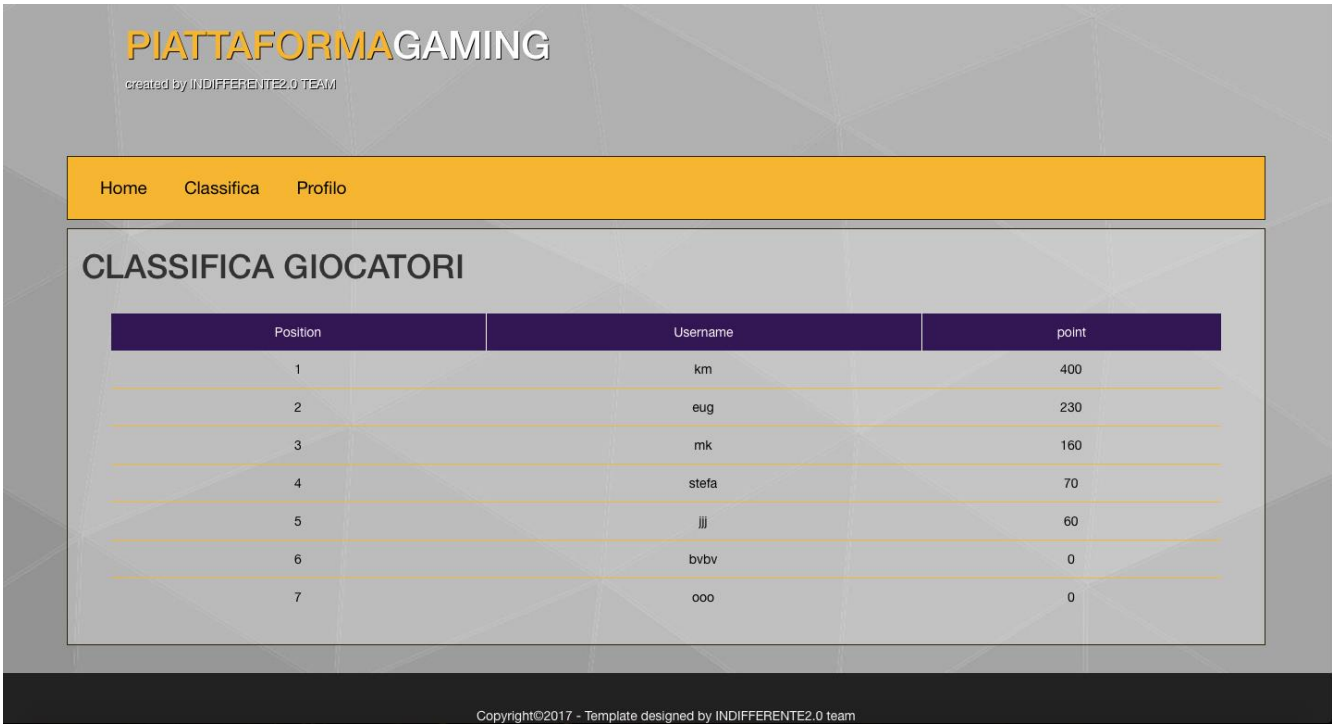
- Pagina Gioco



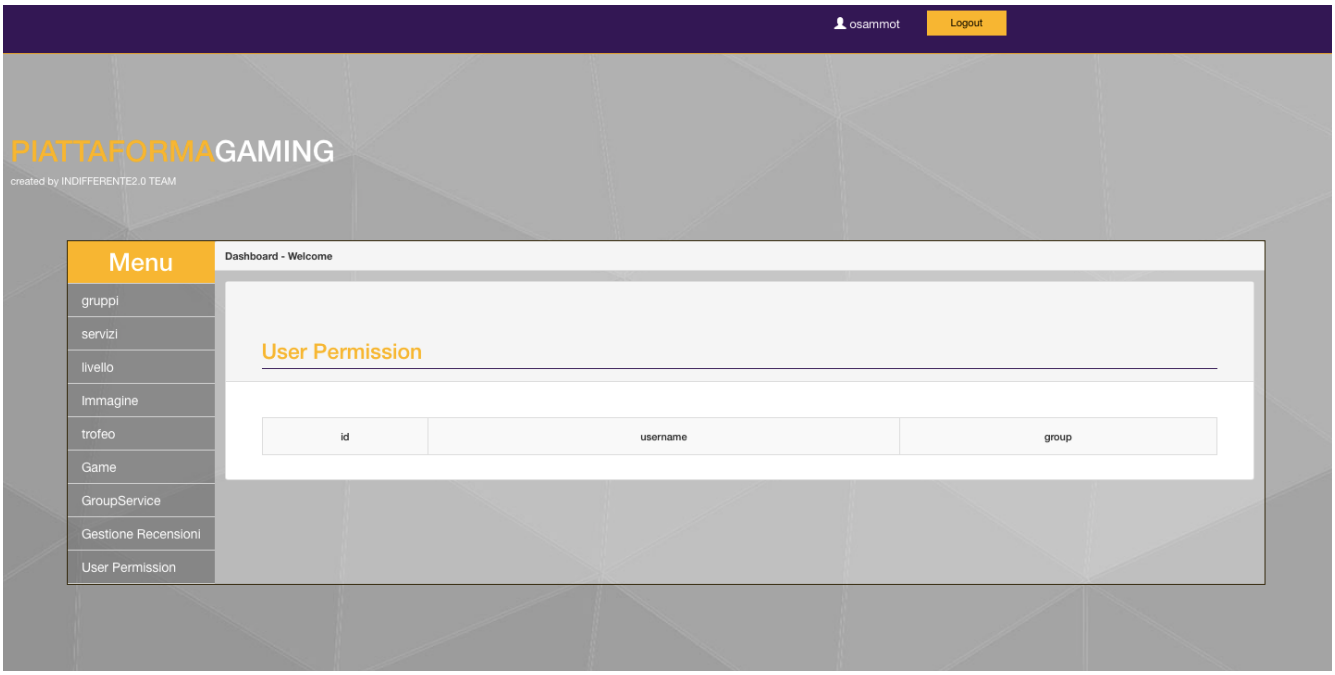
- Profilo utente



- Classifica generale



- Dashboard


osammot
Logout
PIATTAFORMAGAMING
created by INDIFFERENTE2.0 TEAM
Menu
Dashboard - Welcome
gruppi
servizi
livello
Immagine
trofeo
Game
GroupService
Gestione Recensioni
User Permission
User Permission
id
username
group