Object Oriented Programming

Piattaforma di gaming

Marco Amadio - 230406
Marco Angelini - 228613

Requirements

Requisiti funzionali

- **Accesso alla schermata di Login/Registrazione**
  La pagina di Login/Registrazione corrisponderà alla pagina iniziale della nostra piattaforma.

- **Login**
  Funzionalità che permette ad un utente registrato di accedere al sistema.

- **Registrazione**
  Funzionalità che permette ad un utente non registrato di iscriversi al sistema.

- **Visualizzazione elenco giochi**
  Funzionalità che permette di visualizzare l'elenco dei titoli dei giochi della piattaforma.

- **Visualizzazione profilo personale**
  Pagina relativa ad un utente registrato che racchiude le informazioni anagrafiche dell'utente, i punti esperienza accumulati, il livello attuale ed i trofei raggiunti.

- **Visualizzazione dettaglio gioco**
  Pagina relativa ad un singolo gioco in cui un utente registrato può visualizzarne il titolo e una descrizione ed ha la possibilità di votarlo, recensirlo o accedere alla pagina per iniziare una nuova sessione di gioco.

- **Votazione gioco**
  Funzionalità tramite la quale un utente registrato può attribuire una votazione al gioco (da 0 a 5).

- **Recensione gioco**
  Permette ad un utente registrato di recensire un gioco, inserendo un titolo e una descrizione.

- **Effettuare una sessione di gioco**
  Funzionalità tramite la quale un utente può effettuare una nuova sessione di gioco.

- **Collezionare punti esperienza**
  Durante una sessione di gioco l'utente colleziona dei punti esperienza, che gli permetteranno di salire di livello.

- **Collezionare trofei**
  Ogni qual volta che un utente raggiunge un nuovo livello, gli verrà assegnato un nuovo trofeo, che verrà visualizzato nel proprio profilo personale.

- **Accesso alla BackOffice Dashboard**
  Funzionalità riservata agli utenti *Moderatori* ed *Amministratori* del sistema. Permette di accedere alla BackOffice Dashboard.

- **Promozione ruolo**
  Funzionalità riservata agli utenti *Moderatori* ed *Amministratori* del sistema. Permette a questi gruppi di utenza di promuovere un utente al grado successivo.

- **Retrocessione ruolo**
  Funzionalità riservata agli utenti *Moderatori* ed *Amministratori* del sistema. Permette a questi gruppi di utenza di retrocedere un utente al grado precedente.

- **Approvazione recensione**
  Funzionalità riservata agli utenti *Moderatori* ed *Amministratori* del sistema. Permette ai gruppi di utenza citati di approvare una recensione, se la si ritiene opportuna.

- **Eliminazione recensione**
  Funzionalità riservata agli utenti *Moderatori* ed *Amministratori* del sistema. Permette a questi gruppi di utenza di eliminare una recensione se non rispetta gli standard di correttezza.

- **Inserimento gioco**
  Funzionalità riservata agli utenti *Amministratori* del sistema. Permette di inserire un nuovo gioco all'interno della piattaforma.

- **Eliminazione gioco**
  Funzionalità riservata agli utenti *Amministratori* del sistema. Permette di eliminare un gioco dalla piattaforma.

Requisiti non-funzionali

- **Correttezza**
  Il sistema deve comportarsi esattamente secondo quanto previsto dalla specifica dei requisiti funzionali.

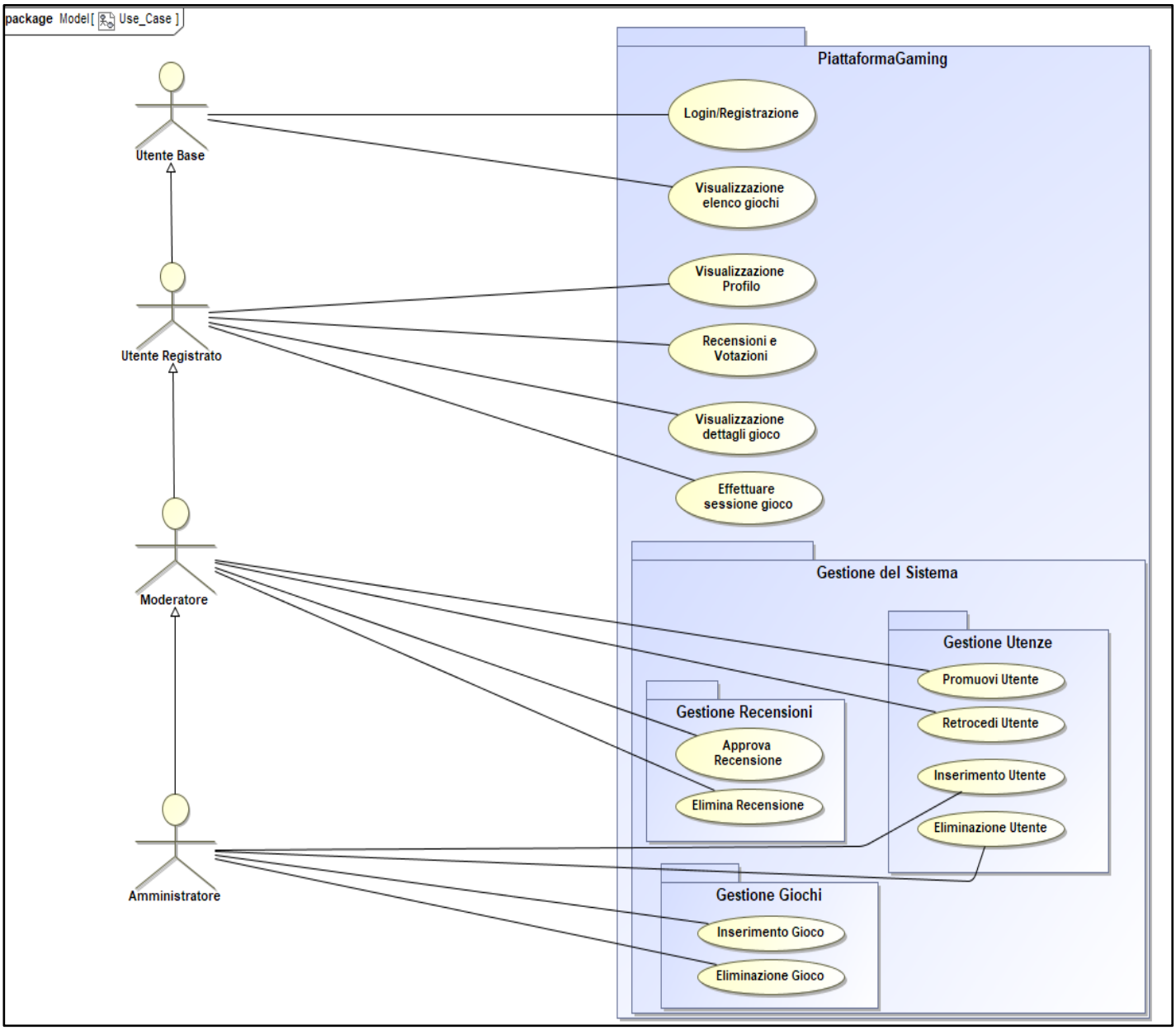
- **Affidabilità**
  Il sistema non deve incorrere in malfunzionamenti e dovrà garantire in qualsiasi momento tutte le funzionalità.

- **Robustezza**
  Il sistema dovrà essere in grado di rispondere correttamente a situazioni impreviste, come ad esempio errori ed eccezioni di varia natura (input scorretti – fallimenti di componenti software o hardware).

- **Usabilità**
  Il sistema dovrà risultare facile da utilizzare; avrà quindi una struttura semplice e chiara.

Use Case Diagram

La figura seguente rappresenta il diagramma Use Case della nostra piattaforma



Attori del sistema

Il primo attore identificato nel sistema è l'Utente, che abbiamo diviso in:

- **Utente base**: è un utente non ancora registrato nella piattaforma; potrà quindi registrarsi o consultare l'elenco dei titolo dei giochi;
- **Utente registrato**: è un utente iscritto al sistema; potrà visualizzare in dettaglio l'insieme dei giochi, effettuare una sessione di gaming e fornire giudizi tramite votazioni e recensioni. Avrà inoltre a disposizione una propria pagina personale che racchiuderà dati anagrafici, punti esperienza e trofei guadagnati.

Tra i due utenti c'è una *generalizzazione*, ossia il "sottoattore", l'Utente registrato, deve essere in grado di partecipare a tutti gli use case a cui partecipa il "superattore", l'Utente base, oltre ad avere le funzionalità aggiuntive descritte.

Il secondo attore identificato è il **Moderatore**. Questo particolare attore ha il compito di gestire l'utenza, promuovendo o retrocedendo gli utenti ad un particolare ruolo. Inoltre si occupa della gestione delle recensioni, approvandole o eliminandole.

Anche tra il Moderatore e l'Utente registrato vi è una generalizzazione.

Per ultimo c'è la figura dell' **Amministratore**. Questo specifico utente si occupa della gestione generale del sistema. Ha sia il compito di gestire le utenze, con le funzionalità di inserimento ed eliminazione di un utente; sia della gestione dei giochi, con le funzioni di inserimento di un nuovo gioco ed eliminazione di uno già presente nella piattaforma.

Anche tra l'Amministratore e il Moderatore è presente una generalizzazione; l'Amministratore copre quindi il ruolo principale dell'intera piattaforma, potendo accedere a tutte le funzionalità messe a disposizione.

Descrizione degli Use Cases

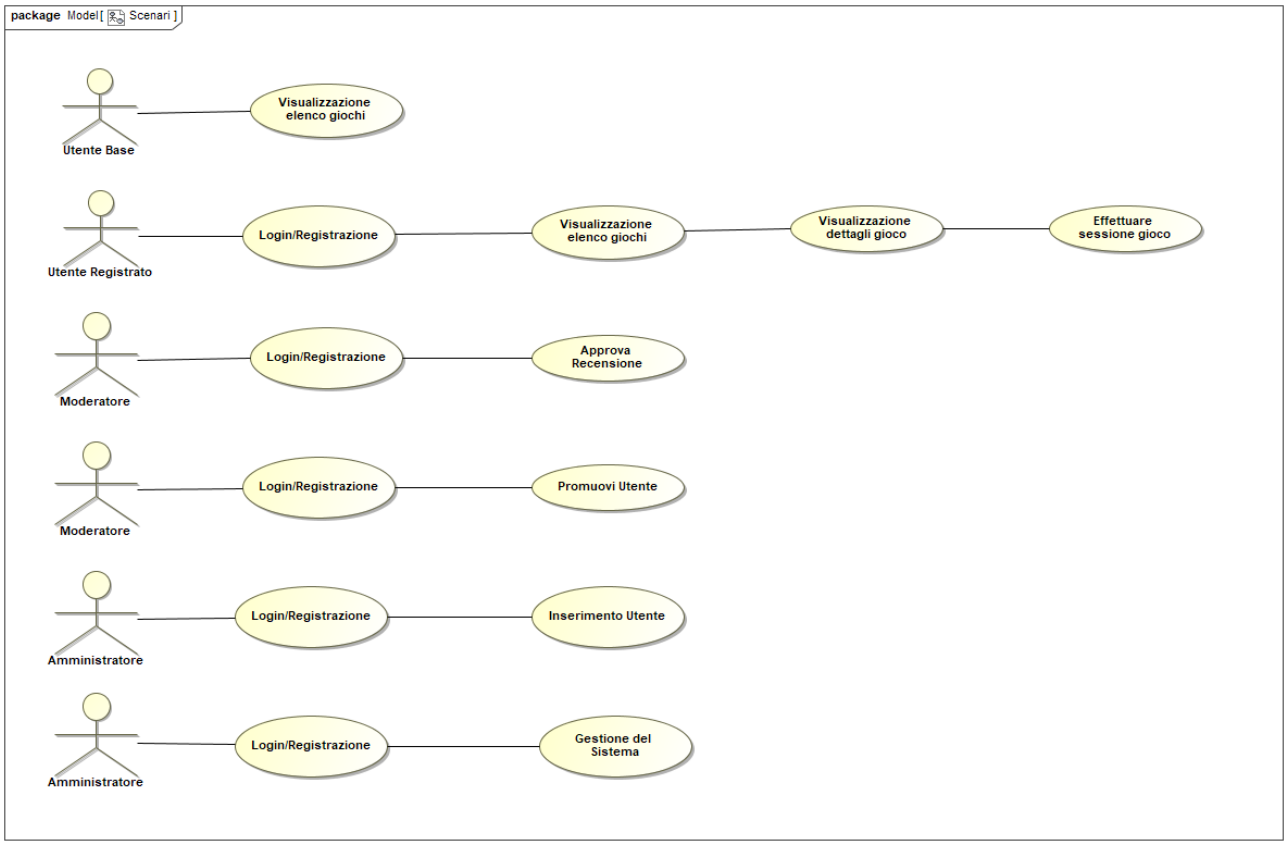
Nome Use-Case	**Login/Registrazione**
Attori Partecipanti	Utente Base - Utente Registrato - Moderatore - Amministratore
Descrizione	*Permette ad un Utente/Moderatore/Amministratore registrato di accedere al sistema e svolgere le proprie funzioni, o ad un Utente Base di registrarsi al sistema.*
Evento scatenante	Click sul relativo pulsante.
Usi	Login/Registrazione effettuati con successo o errore.

Nome Use-Case	**Recensioni e Votazioni**
Attori Partecipanti	Utente Registrato
Descrizione	*Permette ad un Utente Registrato nel sistema di dare una valutazione al gioco da 1 a 5 e di esprimere un giudizio. (La recensione prima di essere visualizzata deve essere approvata da un Moderatore).*
Evento scatenante	Click sui bottoni dei voti per votare; Click su “Inserisci Recensione” per recensire.
Usi	Giudizio del gioco inserito correttamente ed in attesa di moderazione.

Nome Use-Case	**Promuovi Utente**
Attori Partecipanti	Moderatore
Descrizione	*Permette ad un Moderatore di aumentare il grado di un Utente Registrato, passandolo così ad un grado successivo.*
Evento scatenante	Click sul relativo pulsante.
Usi	Promozione di un utente effettuato con successo.

Nome Use-Case	**Approva Recensione**
Attori Partecipanti	Moderatore
Descrizione	*Permette ad un Moderatore di rendere pubbliche le recensioni effettuate da utenti registrati nel sistema.*
Evento scatenante	Click sul relativo pulsante.
Usi	Approvazione effettuata con successo.

Scenari

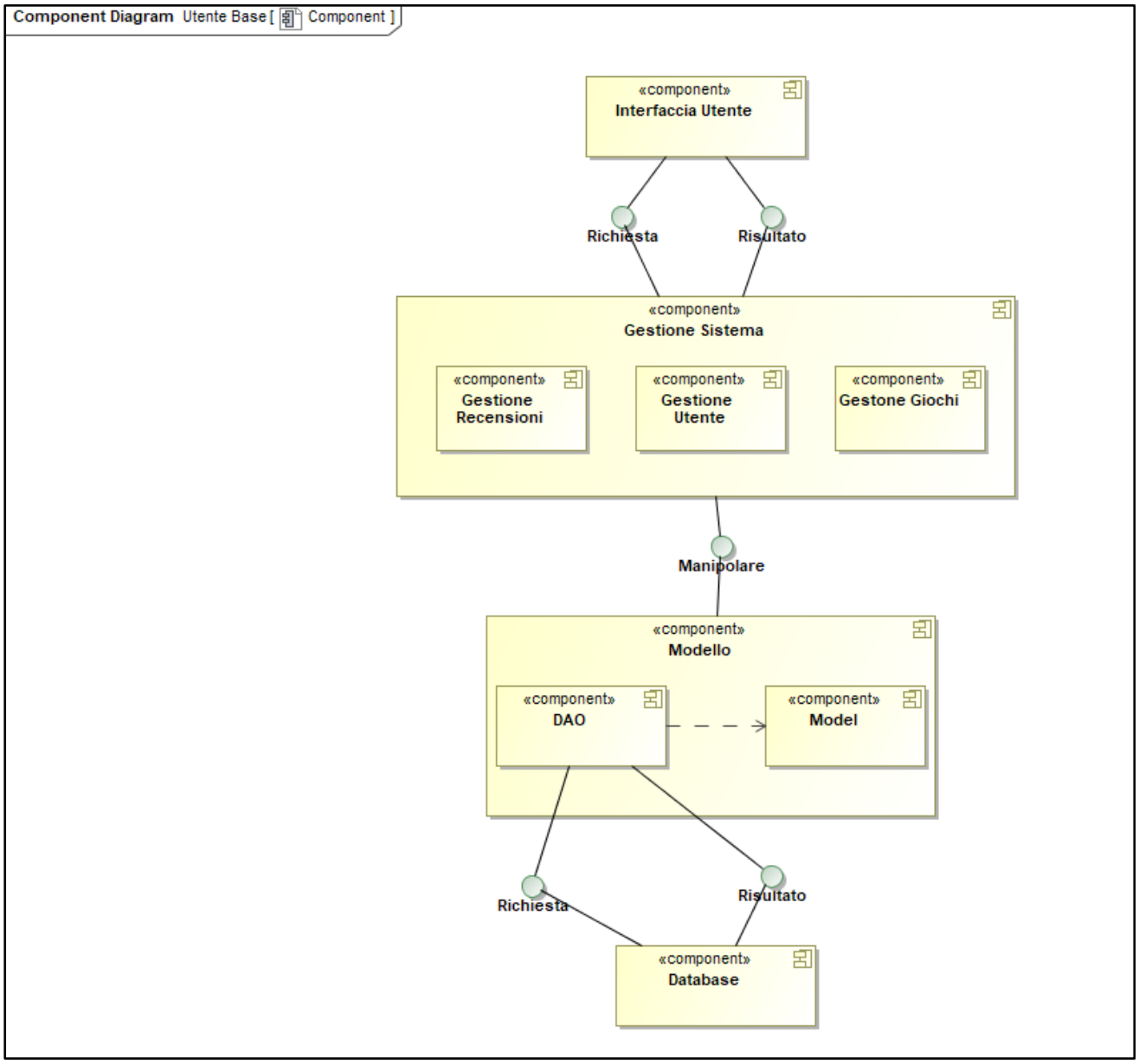


L'immagine seguente rappresenta degli esempi di scenari, ossia interazioni tra entità esterne al sistema (attori) e il sistema stesso, o sue componenti.

Gli scenari separano il flusso principale dalle varianti alternative, rappresentando una specifica serie di azioni effettuate dall'attore.

System Design

Architettura del sistema



La seguente immagine mostra l'architettura software del nostro sistema espressa attraverso un Component Diagram.

Essa è composta da 4 parti principali:

- **Interfaccia Utente**
- **Gestione del sistema**
- **Modello**
- **Database**

L'**interfaccia Utente** rappresenta la nostra GUI (Graphical Unit Interface). Questa interfaccia grafica consente agli utenti di interagire con il sistema manipolando graficamente gli oggetti in maniera molto user-friendly.

La Gestione del sistema non è altro che un contenitore di meccanismi e funzioni per elaborare i dati.
Abbiamo deciso di dividere la Gestione del sistema in 3 sotto-moduli:

- **Gestione recensioni**
- **Gestione utente**
- **Gestione giochi**

Abbiamo gestito il sistema in questa maniera per separare contestualmente le operazioni di elaborazione dati. Ciascuno di essi si occuperà quindi di gestire una specifica sezione.

La componente **Modello** racchiude gli oggetti della nostra piattaforma.
Si frappone tra la componente di Gestione del sistema e il Database; la prima infatti utilizza gli opportuni metodi per gestire gli oggetti presenti nel Model ed effettuare specifiche operazioni, come ad esempio inserimenti o estrapolazioni nel e dal Database.
Abbiamo deciso di suddividere la componente Model in 2 sotto-componenti:

- Il **DAO**
- **Model**

Ne parleremo più avanti nella descrizione dei Pattern utilizzati.

Il **Database** è la componente utilizzata per immagazzinare le informazioni principali della nostra piattaforma.
E' collegato alla componente Modello proprio perché il Database conterrà delle istanze di essi.

Descrizione delle scelte e strategie adottate

Abbiamo deciso di dividere le scelte e strategie adottate in 2 sezioni: le *scelte architetturali* e quelle *implementative*.

Scelte architetturali

- Applicazione Web
- Utilizzo del Pattern MVC
- Utilizzo del Pattern Front Controller
- Utilizzo del Pattern DAO

● *Applicazione Web*

Abbiamo deciso di realizzare la nostra piattaforma come un'Applicazione Web, quindi come un'architettura client-server.
I dati che andremo a manipolare saranno inseriti all'interno di un database in un server; per la creazione e la gestione di esso ci affideremo a *phpMyAdmin,* un'applicazione web che consente di amministrare un database tramite un qualsiasi browser.

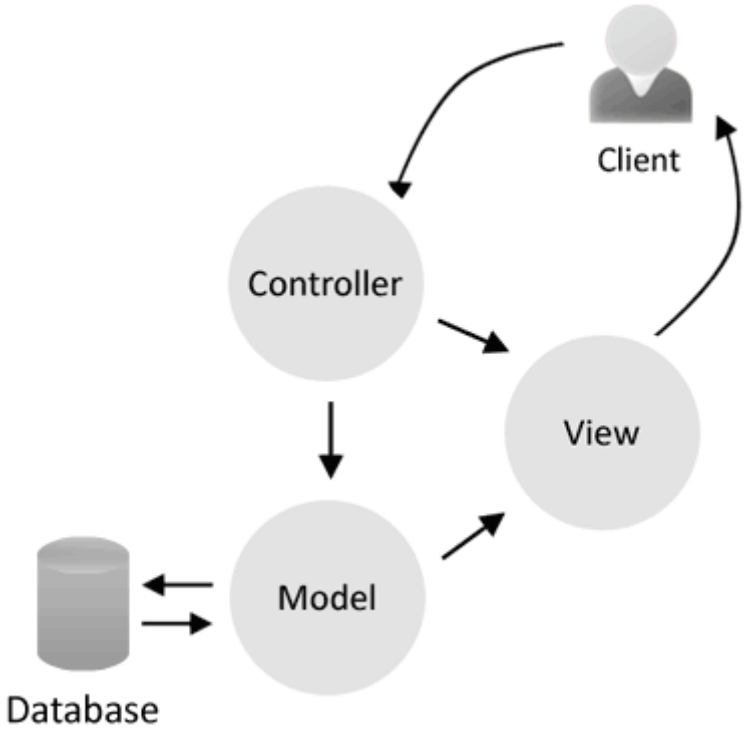
- *Pattern MVC (Model View Controller)*

Parlando di pattern architetturali la nostra prima scelta è ricaduta sul modello MVC, un pattern molto diffuso e importante nella programmazione orientata agli oggetti.

L' MVC permette di organizzare il codice di un'applicazione in 3 parti principali seguendo un criterio logico:

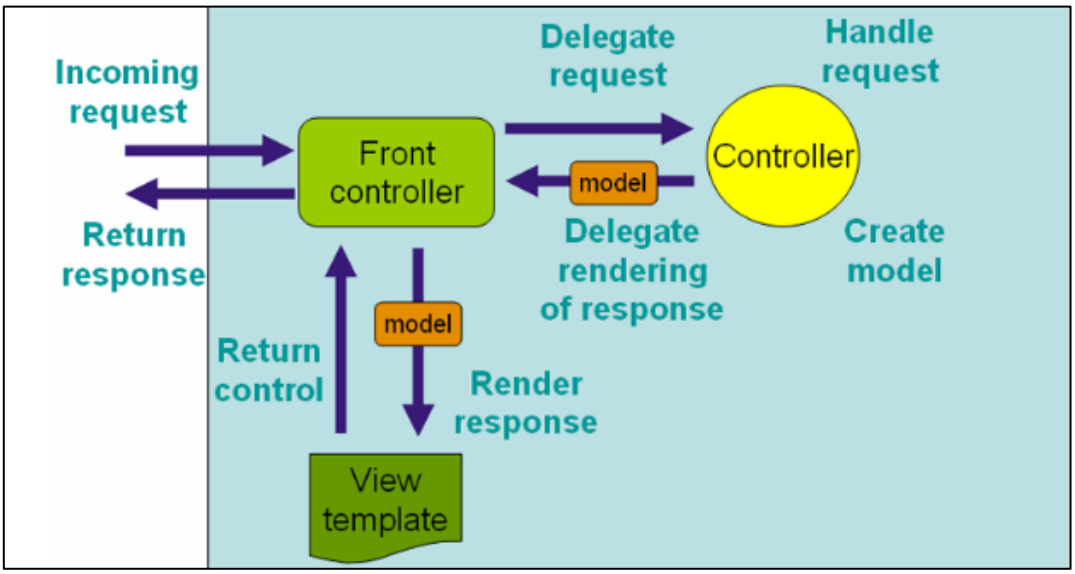
- **Model** : si compone degli oggetti principali dell'applicazione e contiene i metodi di accesso ai dati.
- **View** : si occupa della visualizzazione dei dati all'utente; gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante.
- **Controller** : riceve degli input dall'utente tramite la View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato della View.

Abbiamo adottato questo design pattern perché organizzando il codice secondo questo schema, potremo concentrarci su un problema specifico ed avere la sicurezza che un possibile intervento rimanga circoscritto al blocco di codice di cui ci stiamo occupando, lasciando intatti gli altri.



- *Pattern Front Controller*

Il pattern architetturale MVC implementa un modello denominato Front Controller.
Questo design pattern si applica alla progettazione di applicazioni web e fornisce un punto di ingresso centralizzato per la gestione delle richieste.
Al front controller arriveranno quindi tutte le richieste, successivamente verranno delegate allo specifico controller che si occuperà della loro gestione.
Il response verrà inviato nuovamente al front controller e successivamente instradato verso la componente view.



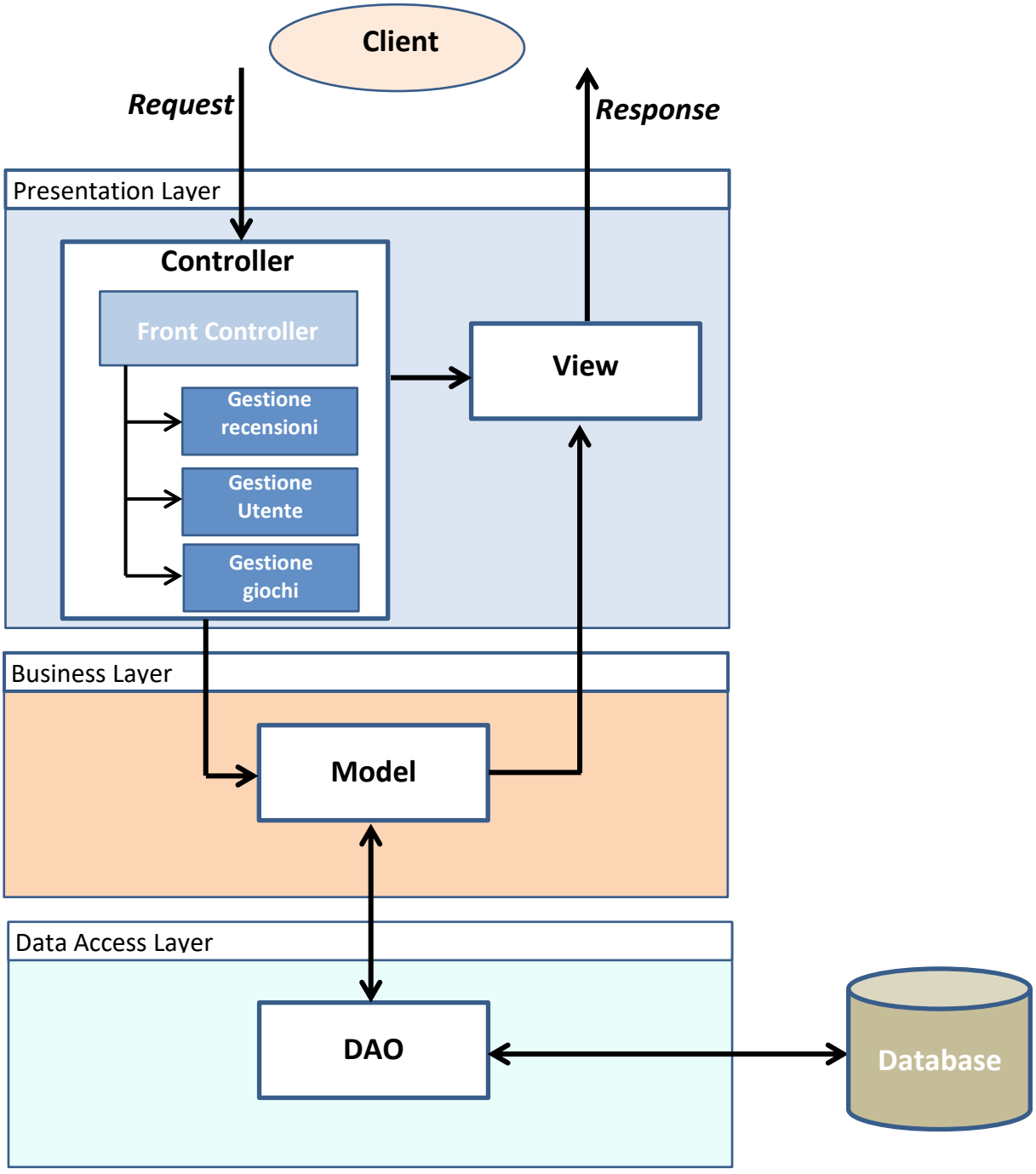
- *Pattern DAO (Data Access Object)*

Un'altra scelta architetturale è ricaduta sull'utilizzo del design pattern DAO, modello architetturale per la gestione della persistenza.
Si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS.
Questo pattern è utilizzato principalmente in applicazioni web per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe).

Utilizzando già un paradigma MVC, il vantaggio relativo all'utilizzo del modello Data Access Object nella nostra applicazione è un ulteriore mantenimento di una rigida separazione tra le componenti Modello e Gestione del sistema, che corrispondono rispettivamente al Model e Controller della struttura MVC.

Ciclo Request → Response

Di seguito mostriamo tramite un diagramma le fasi che intercorrono da una richiesta del client e una risposta da parte dell'applicazione seguendo la struttura architetturale del nostro sistema.



Tra una richiesta e una risposta quindi possiamo notare le seguenti fasi:

1. Il Controller, in particolare il Front Controller riceve la richiesta dal Client.
2. Il Front Controller chiama il metodo appropriato sui Modelli e si mette in attesa del risultato.
3. Se il Model interessato ha la necessità di comunicare con il database lo fa tramite il DAO.
4. Il Controller chiamerà in seguito la View con i risultati del Model.
5. La View tramite un motore di template farà visualizzare l'opportuna risposta al Client.

Scelte implementative

- Promozione e retrocessione degli utenti
- Inserimento di un nuovo gioco
- Gruppi di utenza
- Recensione e votazione di un gioco

● *Promozione e retrocessione degli utenti*

La piattaforma prevede la funzionalità legata alla promozione o retrocessione di un utente in un determinato ruolo.
Attenendoci alla specifica del progetto questa funzionalità era legata agli utenti *Moderatori.* Essendo però un utente *Amministratore* una generalizzazione di un *Moderatore*, la funzionalità è estesa anche ad esso.
Ci è sorto però un dubbio: *un Moderatore può decidere quindi di promuovere un utente ad Amministratore o addirittura di retrocedere quest'ultimo?*

A questo punto abbiamo fatto una scelta:
Essendo l'Amministratore la figura più importante dell'intero sistema, si trova un gradino sopra agli altri gruppi di utenza; perciò abbiamo imposto che un Amministratore può decidere di promuovere un qualsiasi gruppo di utenza al ruolo che desidera, senza alcun vincolo.

Per il Moderatore invece abbiamo aggiunto un vincolo gerarchico; egli infatti può compiere solamente 2 azioni:

1. Promuovere un Utente Registrato a Moderatore.
2. Retrocedere un Moderatore ad Utente Registrato.

● *Inserimento di un nuovo gioco*

Tra le funzionalità legate ad un utente *Amministratore* c'è anche quella di poter aggiungere un nuovo gioco alla piattaforma o eliminarne uno esistente; soffermiamoci sull'aggiunta.

Questa operazione comprende 2 figure: l'*Amministratore* e lo *Sviluppatore della piattaforma* di gaming.
Queste sono le fasi da seguire per poter aggiungere un nuovo gioco nella piattaforma:

1. Il gioco dev'essere finito e sviluppato correttamente.
2. L'amministratore valuta se il gioco rispetta gli standard di correttezza per la piattaforma.
3. Qualora l'Amministratore decidesse di inserirlo, sarà compito dello sviluppatore della piattaforma di gaming inserire il codice del gioco all'interno dell'applicazione web.
4. Solamente a questo punto l'Amministratore potrà aggiungere il nuovo gioco dal BackOffice e renderlo disponibile agli utenti.

- *Gruppi di Utenza*

Visto che il nostro sistema prevede 3 tipologie di utenti (Utente registrato – Moderatore – Amministratore) abbiamo deciso di gestire questa gerarchia tramite un identificativo (da 0 a 2) associato a ciascun utente.

Avremo quindi questa corrispondenza:

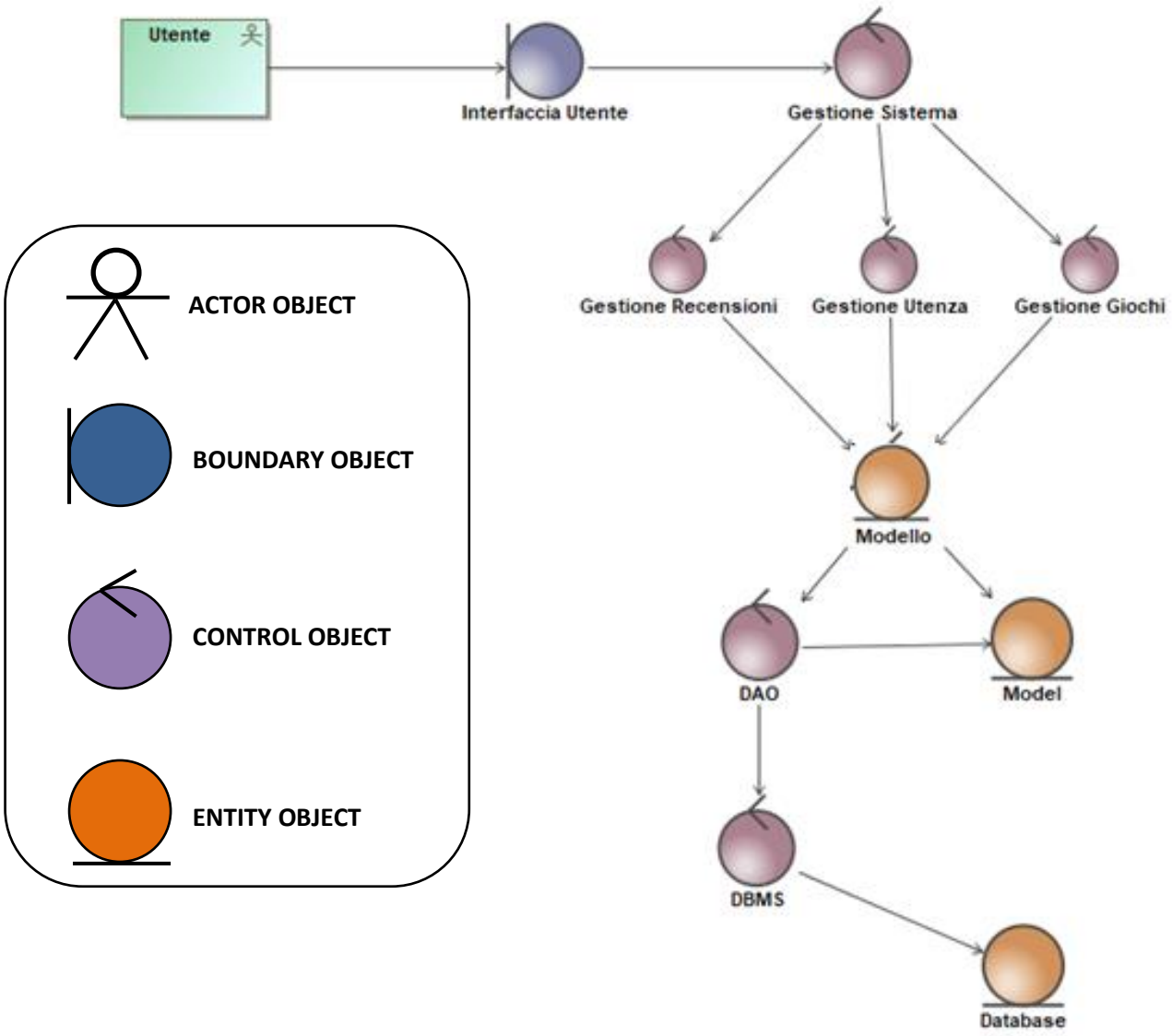
Id	Categoria
0	Utente registrato
1	Moderatore
2	Amministratore

- *Recensione e votazione di un gioco*

La piattaforma prevede sia la possibilità di votare un gioco, sia quella di recensirlo tramite una descrizione. Prendendo spunto da altre piattaforme ed essendo le due funzionalità contestualmente simili abbiamo deciso di considerare una votazione e una recensione come unico oggetto.

Quando un utente si troverà nell'apposita pagina dedicata all'inserimento di un parere sul gioco avrà a disposizione un form per inserire una recensione, legato alla possibilità di esprimere un voto da 0 a 5. Vediamo un esempio.

Robustness Diagram



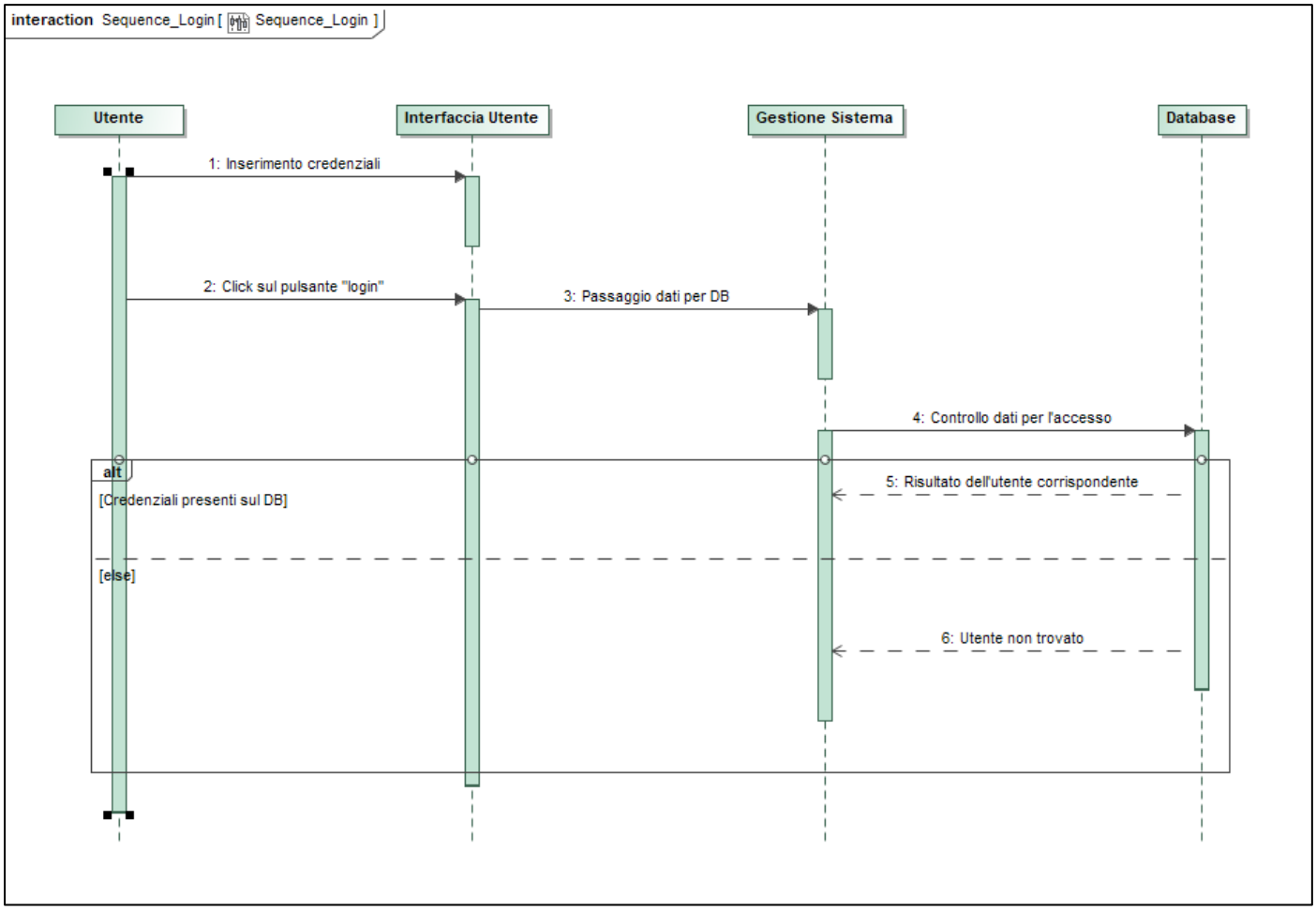
Lo schema seguente rappresenta il Robustness Diagram della nostra applicazione.
Non fa parte dei diagrammi standard UML, tuttavia ne utilizza i concetti; come si può vedere dalla legenda utilizza soprattutto 4 oggetti:

- **Actor** : Corrisponde allo stesso concetto di attore in uno Use Case Diagram UML
- **Boundary** : Corrisponde alla User Interface; oggetti di questo tipo rappresentano in genere il Presentation Layer con cui l'attore interagisce, come in questo caso.
- **Entity** : Gli oggetti di tipo Entity rappresentano di solito oggetti che fanno parte del Model.
- **Control** : Gli oggetti di tipo Control rappresentano "la colla" che tiene uniti gli oggetti Boundary e Entity; possiamo ricondurlo alla componente Controller del paradigma MVC.

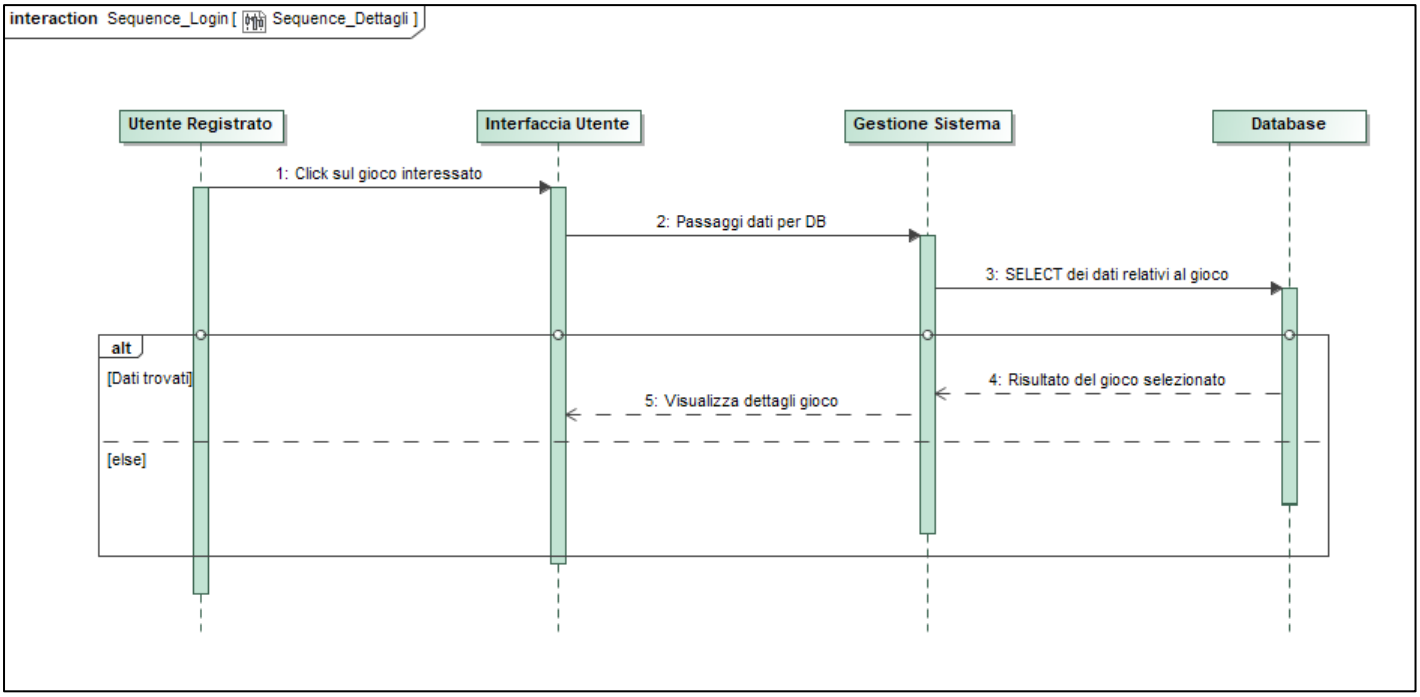
Sequence Diagram

Il Sequence Diagram è un diagramma previsto dall'UML utilizzato per descrivere uno scenario, ossia una determinata sequenza di azioni compiute dagli attori del nostro sistema in cui tutte le scelte sono già state effettuate.
Mostriamo di seguito una serie di possibili scenari.

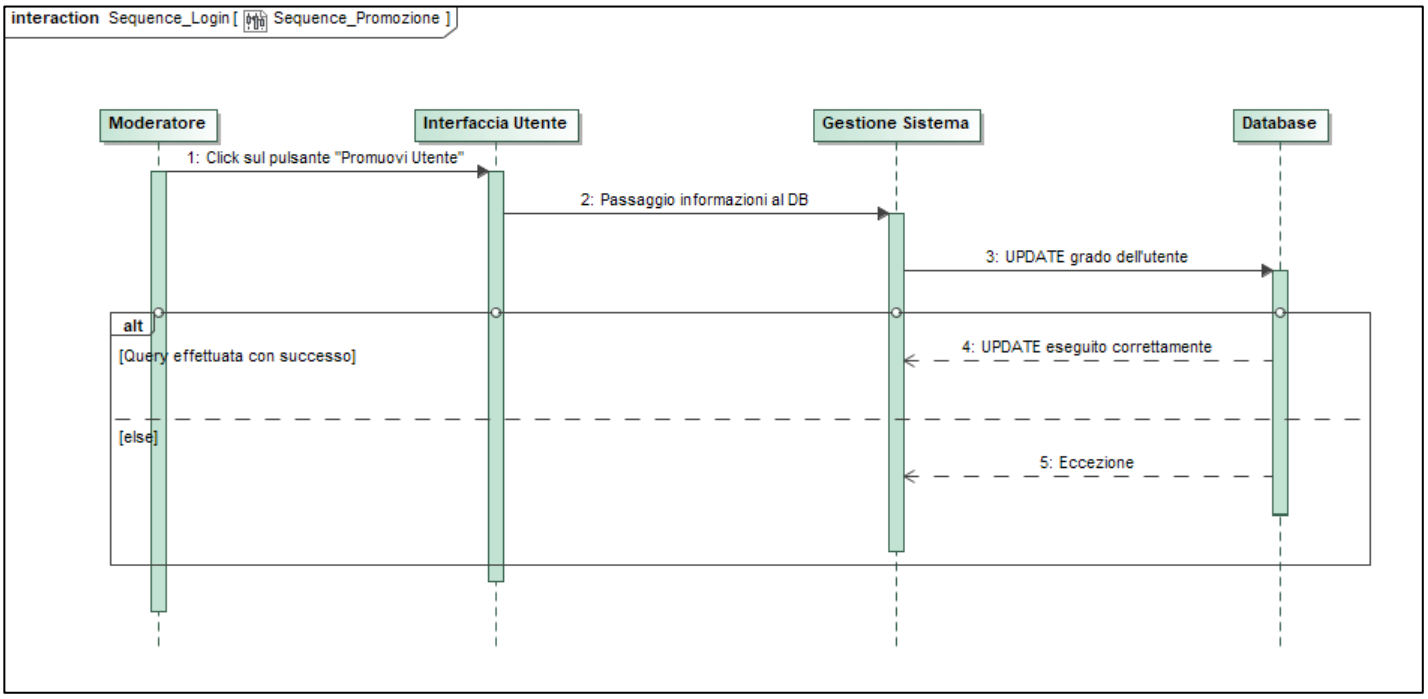
Login



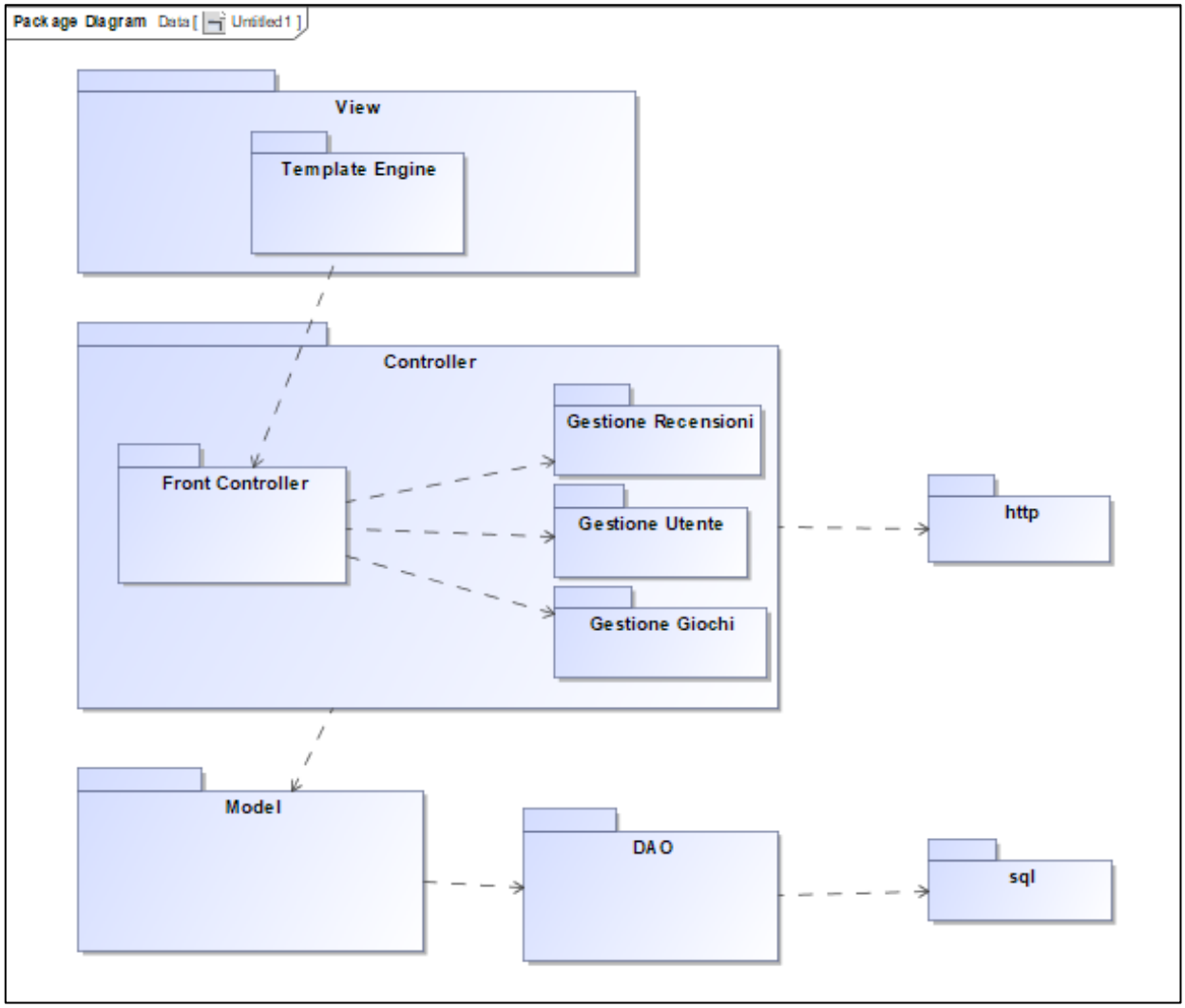
Dettagli gioco



Promozione utente



Package Diagram

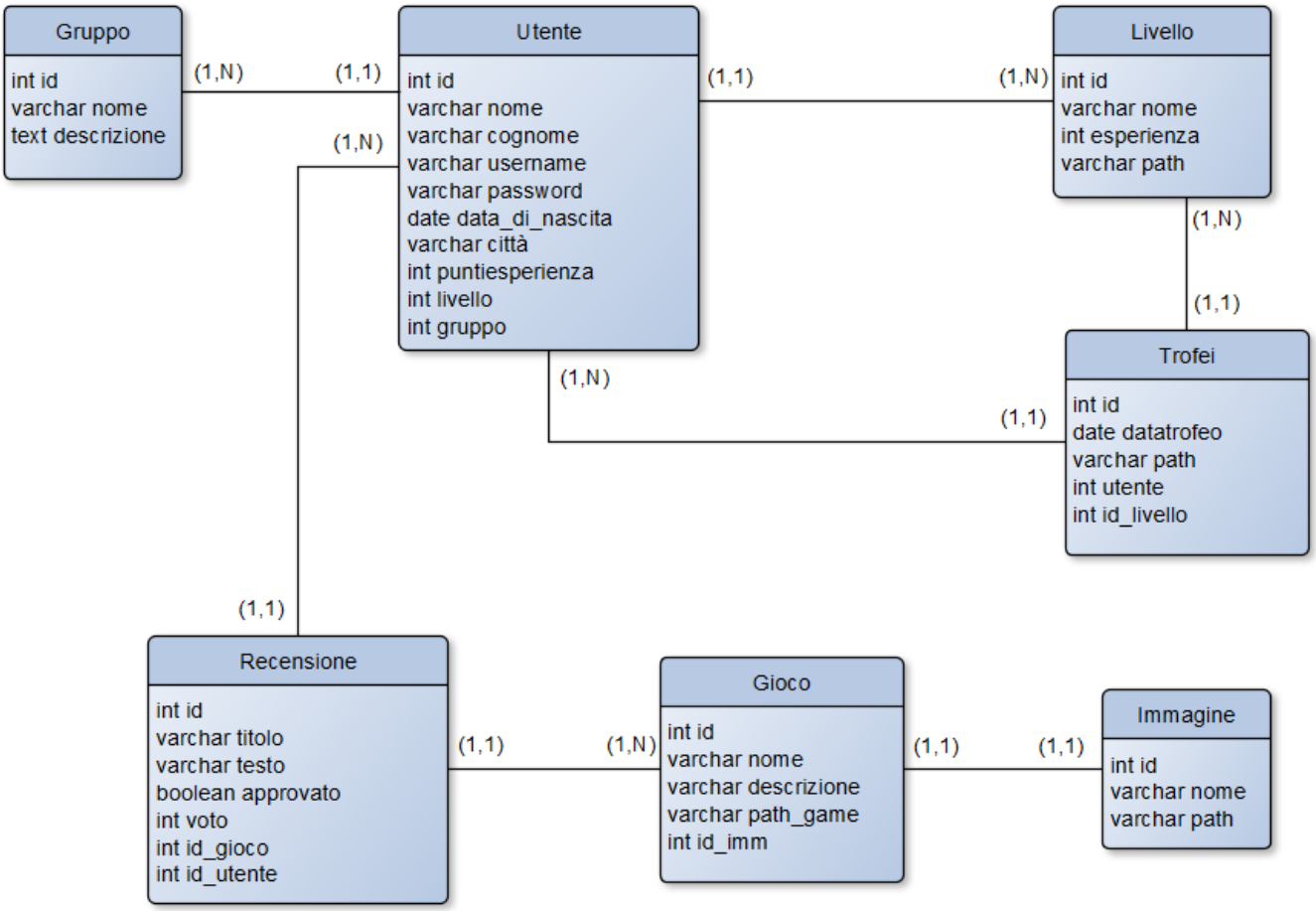


Quello in figura corrisponde al Package Diagram della nostra applicazione.

Un Package Diagram è utilizzato per raggruppare elementi e fornire una sorta di namespace in base agli elementi raggruppati; si viene a creare quindi un'organizzazione gerarchica del sistema.

E' un diagramma che permette di visualizzare in modo più chiaro le componenti della nostra piattaforma; in questo caso abbiamo rappresentato e diviso i layers in base ai pattern architetturali adottati.

Modello ER



Descrizione del modello ER

Utente

Rappresenta un utente regolarmente iscritto al sistema; è caratterizzato da un identificativo (**id**), da dati anagrafici quali **nome**, **cognome**, **data_di_nascita** e **città**, dalle credenziali di accesso, rappresentate dagli attributi **username** e **password** e da **puntiesperienza**, campo di tipo intero che rappresenta appunto i punti esperienza raggiunti da quell'utente.
Utente è in relazione 1 a 1 con Gruppo perché a 1 utente è associato 1 solo gruppo di utenza.
Utente è in relazione 1 a 1 con Livello perché 1 utente appartiene a 1 determinato livello.

Livello

L'entità Livello contiene i vari record relativi ai livelli che un certo utente può raggiungere.
Abbiamo considerato nella nostra piattaforma un insieme di 10 Livelli; si parte dal livello 1 di partenza fino ad arrivare al livello massimo, il 10° appunto.
L'entità livello è caratterizzata dai seguenti attributi: un **id** per identificare il livello di appartenenza, un **nome**, un campo **esperienza** che sta ad indicare l'esperienza massima per quel livello (ad esempio ipotizzando che dal livello 1 occorrano 100 punti esperienza per passare al livello 2, il campo esperienza del livello 1 è settato a 100).
Infine è presente un attributo **path** che corrisponde all'immagine del trofeo di quel determinato livello.
Livello è in relazione 1 a 1 con utente perché 1 utente appartiene a 1 determinato livello.
Livello è in relazione 1 a N con Trofei perché lo stesso livello può apparire più volte nell'entità Trofei.

Trofei

L'entità Trofei rappresenta uno storico di tutti i trofei conquistati dagli utenti. E' composta dai seguenti attributi: un **id**, un campo **date** che corrisponde alla data in cui l'utente ha conquistato quel trofeo, l' **utente** protagonista, il **livello** conquistato e il **path** di riferimento all'immagine del trofeo.
Trofei è in relazione 1 a 1 con Livello perché un record di Trofei è legato ad 1 solo livello.
Trofei è in relazione 1 a 1 con Utente perché un record di Trofei si riferisce ad 1 solo utente.

Gruppo

La tabella Gruppo contiene tutti i gruppi di utenza del sistema, abbiamo deciso di separarla dall'entità Utente per rendere la struttura più efficiente.
Essa contiene un attributo **id** per identificare il gruppo di appartenenza, il **nome** del gruppo di utenza e una breve **descrizione**.
E' in relazione 1 a N con Utente perché più utenti possono appartenere allo stesso gruppo di utenza.

Recensione

L'entità recensione conterrà tutte le istanze che rappresentano un giudizio su un gioco.
E' formata dai seguenti attributi: un **id,** il **titolo** e il **testo** della recensione, un campo booleano **approvato** settato inizialmente a false e un attributo **voto** per fornire un giudizio del gioco con una scala da 0 a 5 stelle.
E' caratterizzata infine dalle chiavi esterne **id_gioco** e **id_utente** per conoscere l'utente che ha inserito la recensione e il gioco a cui ha fatto riferimento.
Recensione è in relazione 1 a 1 con Utente perché 1 recensione è attribuita ad 1 solo utente.
Recensione è in relazione 1 a 1 con Gioco perché 1 recensione si riferisce a 1 singolo gioco.

Gioco

L'entità Gioco contiene l'insieme dei record di riferimento ai giochi presenti nella piattaforma.
E' composta da un identificativo **id,** dal **nome** del gioco, da una **descrizione** e da un campo **path** che si riferisce al percorso del gioco all'interno dell'applicazione; è formata inoltre da un **id_imm** che si riferisce all'id dell'immagine del gioco contenuta nell'apposita tabella Immagine.
Gioco è in relazione 1 a N con Recensione perché un gioco può contenere più recensioni.
Gioco è in relazione 1 a 1 con Immagine perché 1 gioco è caratterizzato da 1 unica immagine.

Immagine

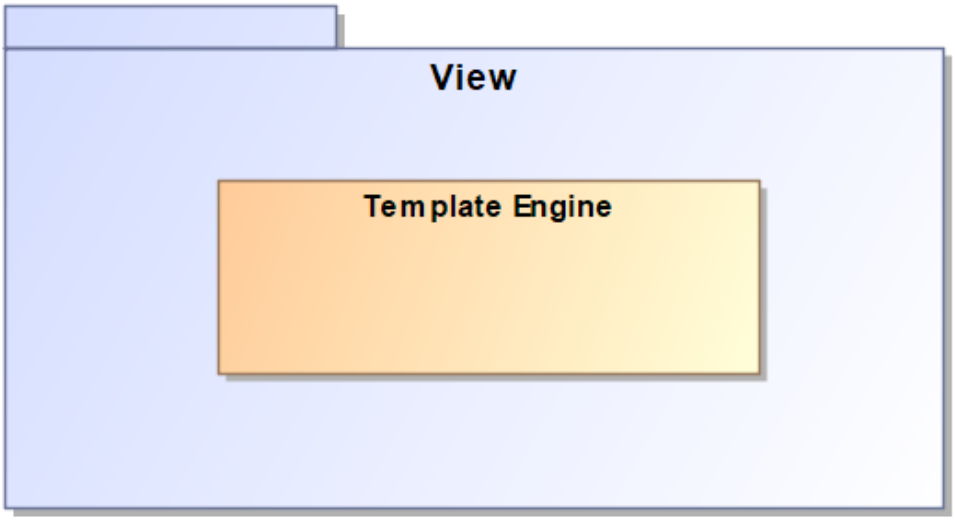
La tabella immagine conterrà tutte le immagini di copertina dei giochi.
Sarà formata da un campo **id,** un **nome** e l'attributo **path** che appunto si riferisce all'immagine relativa al gioco.
Immagine è in relazione 1 a 1 con Gioco proprio perché a ciascun gioco è attribuita 1 unica immagine.

Software Design

Class Diagram

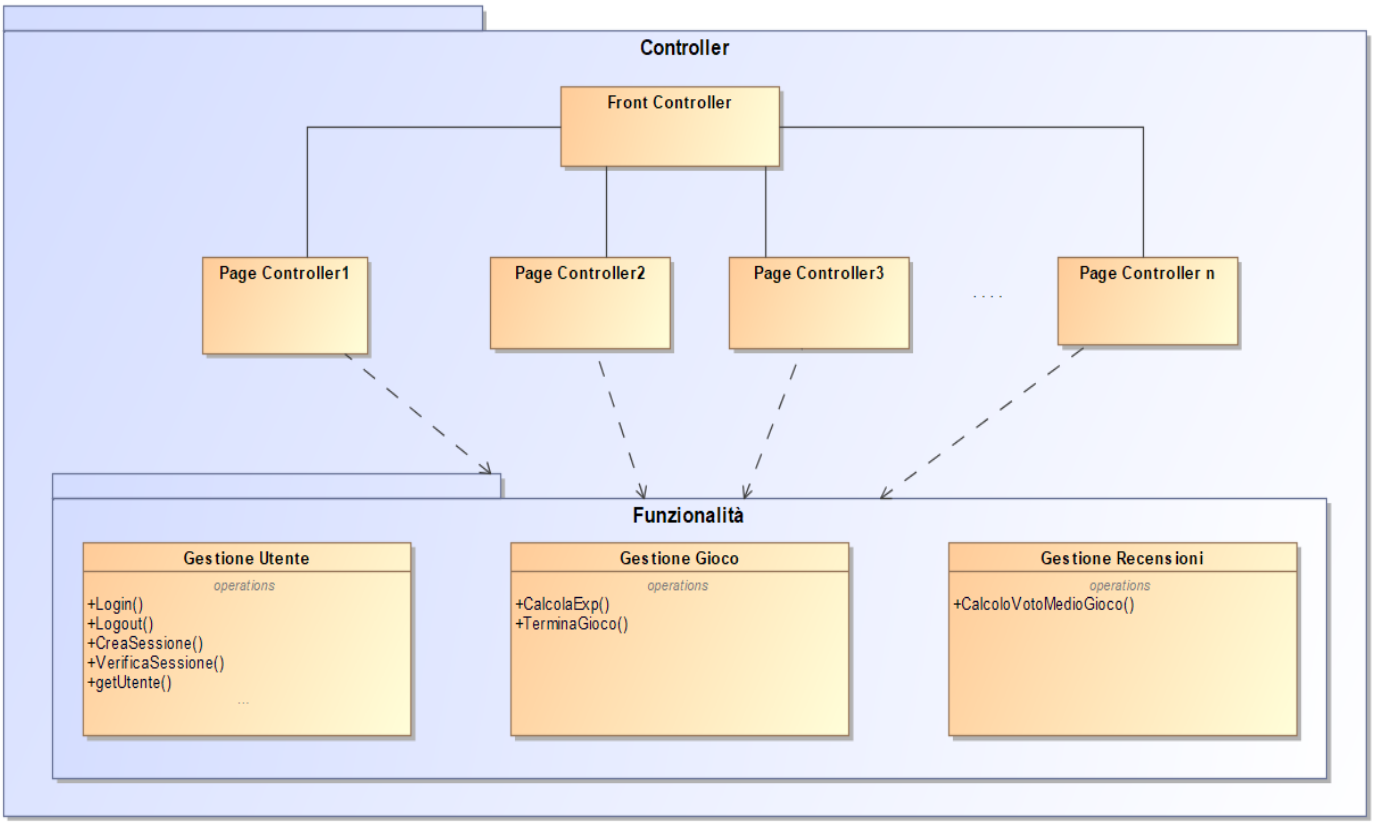
Le figure seguenti rappresentano le componenti principali del Class Diagram della nostra applicazione.
Abbiamo deciso di dividerle in sezioni distinte seguendo l'architettura e i design pattern utilizzati.
Infine verrà visualizzato il Class Diagram completo.

View



Questo Package corrisponde alla nostra componente View.
Rispettando il modello MVC adottato questa sezione risulta completamente separata dalla parte applicativa.
Per gestire l'interazione tra gli utenti e la piattaforma abbiamo deciso di utilizzare un template engine.

Controller

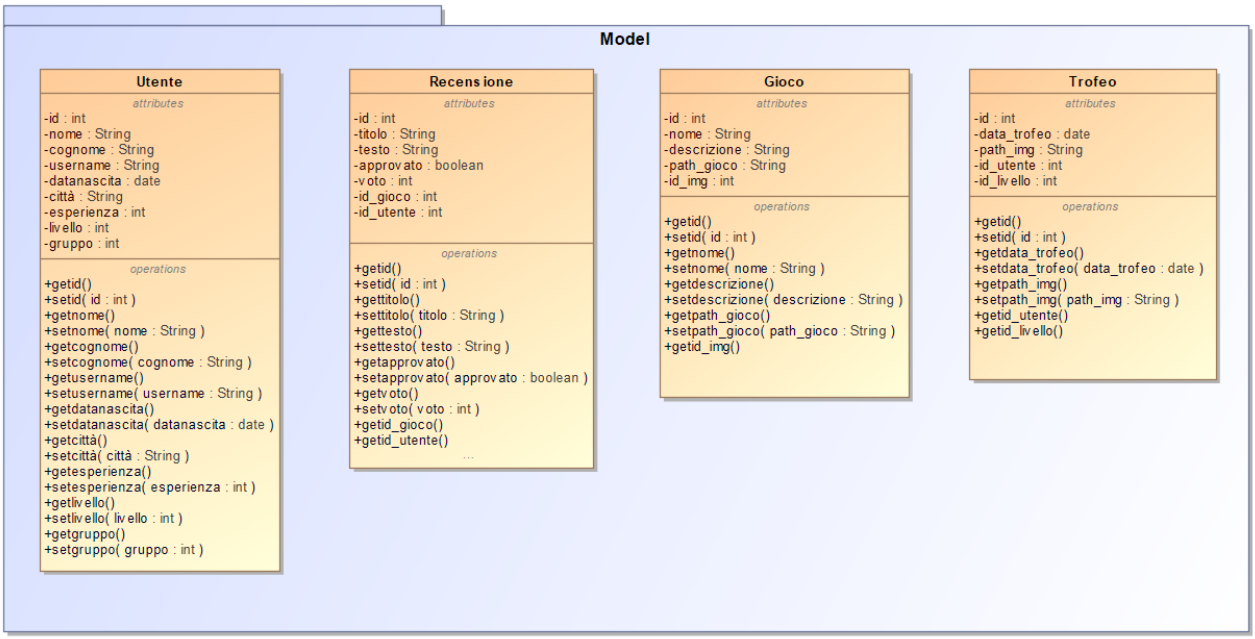


Il controller fa da mediatore tra la View ed il Model; esso riceve i comandi dell'utente attraverso la View ed esegue le opportune operazioni che possono interessare il Model.

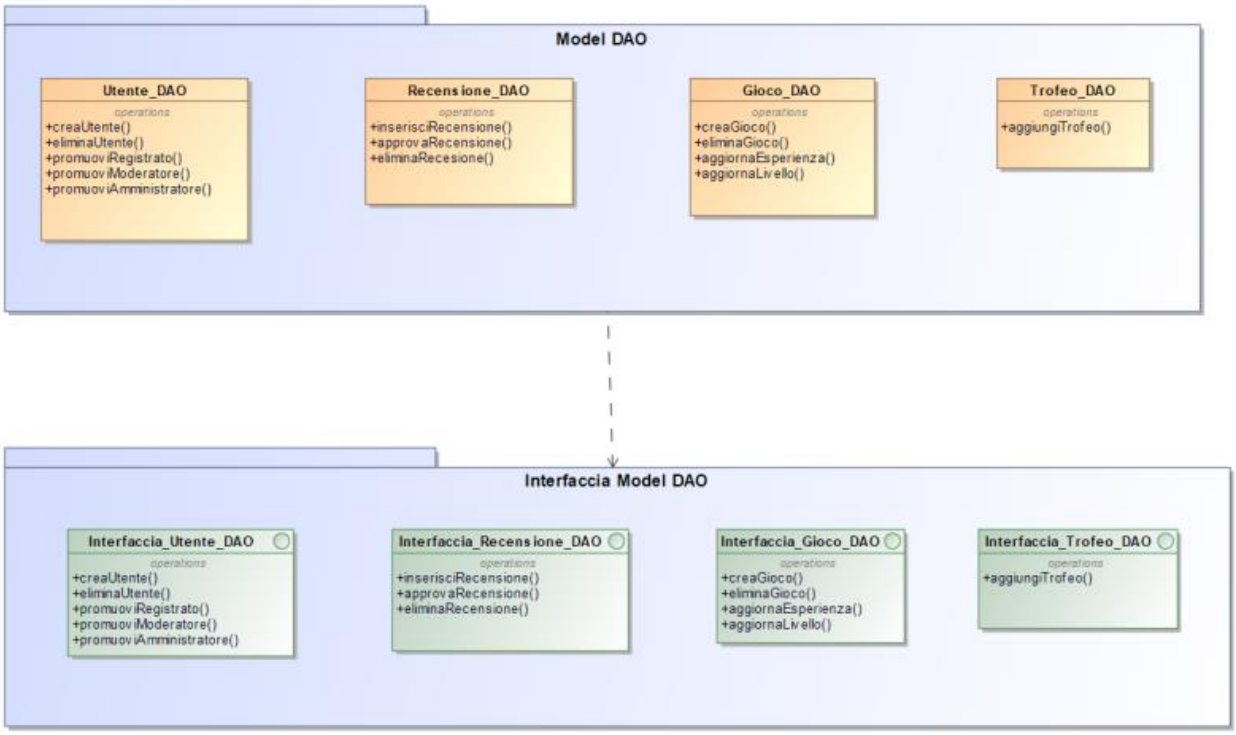
Nel nostro specifico caso, basandoci sul Front Controller Pattern, tutte le richieste da parte dell'utente arrivano al Front Controller.
Fornisce un punto di accesso centralizzato di tutte le request e le inoltra opportunamente ai controller specifici di ogni pagina; nel nostro caso i Page Controller sono rappresentati da Servlet

Il Controller è composto inoltre dalle funzionalità di gestione dell'utenza, del gioco e delle recensioni.
Sono delle Classi contenenti metodi specifici che possono essere richiamati ogni qual volta fosse necessario.

Model



Il Model è formato degli oggetti principali della nostra applicazione; insieme alla componente DAO, che mostreremo di seguito, si occupa di recuperare ed archiviare informazioni da o verso il database.



Avendo utilizzato il design pattern DAO, abbiamo isolato le classi contenenti tutti i metodi di interazione con il database e abbiamo pensato a un utilizzo delle interfacce per una struttura migliore del pattern.

Class Diagram Risultante