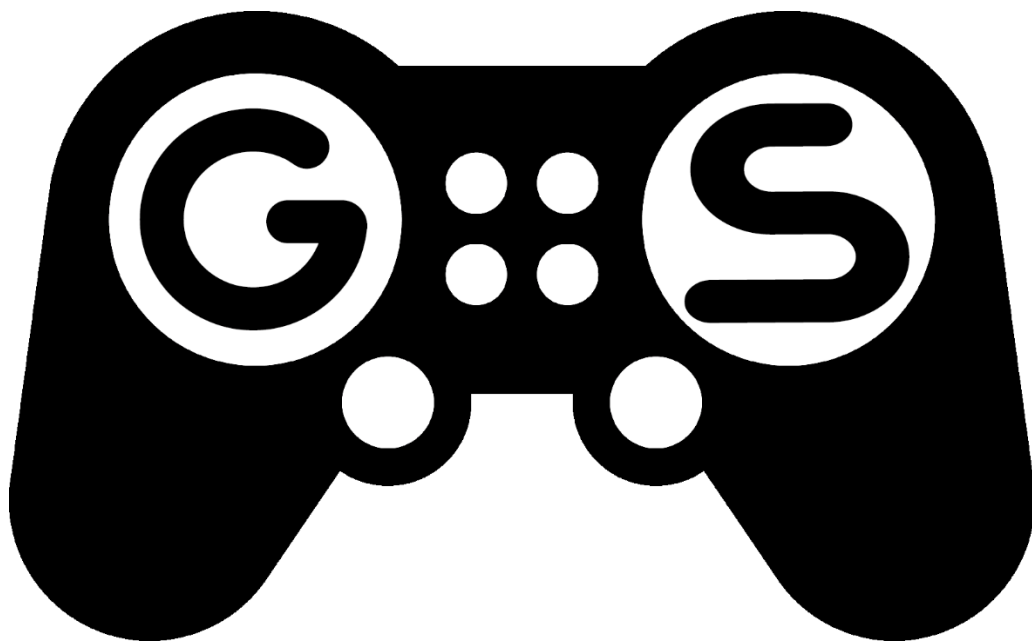


ObjectDesign Document

Progetto GameSquare



Partecipanti:

Francesco Galasso 0512105314

Cosimo Botticelli 0512105460

Aniello Pio La Pietra 0512105716

Sommario

| | |
|--|----|
| 1. Introduzione | 3 |
| 1.1 Trade-off | 3 |
| 1.2 Componenti off-the-shelf | 3 |
| 1.3 Linee Guida per la documentazione dell'interfaccia | 4 |
| 1.4 Definizioni, acronimi e abbreviazioni | 6 |
| 2. Packages | 8 |
| 3. Interfacce delle classi | 11 |
| 3.1 Entity | 11 |
| 3.2 Manager | 16 |
| 3.3 Control | 23 |
| 4. Class Diagram | 30 |
| 5. Glossario dei termini | 30 |

1. Introduzione

1.1 Trade-off

Nella fase di Object Design è necessario stabilire quali caratteristiche rispettare e quali rendere opzionali, a fronte dei diversi compromessi progettuali sorti. Pertanto, sono stati individuati i seguenti “trade-off” per la realizzazione del sistema.

Tempo di risposta vs Affidabilità: Verrà preferita l’affidabilità del sistema rispetto al tempo di risposta, così che i dati inseriti in input possano essere controllati con più attenzione.

Criteri di manutenzione vs Criteri di performance: Il sistema verrà implementato preferendo i criteri di manutenzione a discapito di quelli di performance, al fine di agevolare lo sviluppatore durante l’aggiornamento del software.

Comprensibilità vs Costi: Durante l’implementazione il codice verrà scritto rispettando lo standard del linguaggio di programmazione Java, favorendo la comprensibilità e facilitando il processo di manutenzione e modifica. Ciò comporterà un incremento dei costi per quanto riguarda la documentazione, ma garantirà manutenibilità e chiarezza dei contenuti anche a chi non è strettamente coinvolto in una specifica parte del progetto.

1.2 Componenti off-the-shelf

Per il sistema che si intende realizzare verranno utilizzati componenti **off-the-shelf**, ovvero componenti software già disponibili utilizzate per facilitare la creazione del progetto. In particolare, per velocizzare il sistema e ottenere effetti visuali verranno usati Javascript e la sua libreria JQuery, che permetteranno all’interfaccia di rispondere alle azioni dell’utente in maniera dinamica, e le tecniche basate su AJAX miglioreranno l’esperienza utente offrendo la possibilità di invocare le operazioni delle servlet e ottenere risultati in modo più interattivo, senza dover necessariamente ricaricare la pagina web. Poiché risulta più efficiente e veloce utilizzare una componente preesistente e fornita da un ente riconosciuto, per la gestione delle connessioni verrà utilizzato un sistema di connection pool, fornito da Apache Tomcat (riferimenti: <https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html>).

1.3 Linee Guida per la documentazione dell'interfaccia

Di seguito, sono riportate le linee guida alle quali ciascun programmatore dovrà attenersi per l'implementazione del sistema.

1.3.1 Classi e interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà prevalentemente allo standard Google Java (<http://google.github.io/styleguide/javaguide.html#s4.6-whitespace>). In più, ogni metodo e file deve essere preceduto da un commento che riporti il nome dell'autore e che documenti l'obiettivo che si vuole raggiungere, e possono essere presenti ulteriori commenti e giustificazioni in merito a particolari decisioni o calcoli. La convenzione prevalentemente utilizzata per quanto riguarda i nomi delle variabili è la nota lowerCamelCase, che consiste nello scrivere parole composte o frasi unendo tutte le parole tra loro.

Nella codifica di classi e interfacce Java, è opportuno rispettare le seguenti regole di formattazione:

1. Non inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.
2. La parentesi graffa aperta "{" si trova alla fine della stessa linea dell'istruzione di dichiarazione.
3. La parentesi graffa chiusa "}" inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione, mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

1. Le istruzioni racchiuse all'interno di un blocco (esempio: for), devono essere indentate di un'unità all'interno dell'istruzione composta.
2. La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta.
3. La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta
4. Le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi.

I nomi di classe devono essere sostantivi, con lettere minuscole, ma, rispettando la notazione CamelCase, la prima lettera del nome, così come la prima lettera di ogni parola interna ad esso, dev'essere maiuscola, e le parole interne non devono essere separate da underscore. I nomi delle classi devono essere semplici, descrittivi e che rispettino il dominio applicativo, così come i nomi dei metodi, i quali iniziano con una lettera minuscola e seguono la notazione lowerCamelCase.

1.3.2 Pagine HTML

Le pagine HTML, sia in forma statica che dinamica, devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;
2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;
4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

1.3.3 Pagine lato Server (JSP)

Anche le pagine JSP, quando eseguite, devono produrre un documento conforme allo standard HTML 5. Il codice Java delle pagine deve aderire alle convenzioni per la codifica in Java, con i seguenti accorgimenti:

1. Il tag di apertura (<%) si trova all'inizio di una riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga;
3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione.

1.3.4 Script JavaScript

Gli Script in JavaScript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni differenti dal rendering della pagina dovrebbero essere collocati in file appositi.
2. Il codice JavaScript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.
3. I documenti JavaScript devono iniziare con un commento analogo a quello presente nei file Java.
4. Le funzioni JavaScript devono essere documentate in modo analogo ai metodi Java.
5. Gli oggetti JavaScript devono essere preceduti da un commento in stile JavaDoc.

1.3.5 Fogli di stile (CSS)

I fogli di stile devono seguire le seguenti convenzioni:

Tutti gli stili non in-line devono essere collocati in fogli di stile separati.

Ogni foglio di stile deve iniziare con un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({});
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga.

1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole;
2. Se il nome è costituito da più parole, è previsto l'uso di underscore (_).

I nomi dei campi devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole;
2. Se il nome è costituito da più parole, è previsto l'uso di underscore (_);

1.4 Definizioni, acronimi e abbreviazioni

ODD: Object Design Document.

DBMS: DataBase Management System.

Off-The-Shelf: componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.

Bootstrap: una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

HTML: linguaggio di markup utilizzato per lo sviluppo di pagine Web.

CSS: acronimo di Cascading Style Sheets, è un linguaggio usato per definire la formattazione delle pagine Web.

JavaScript: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: JQuery è una libreria JavaScript per applicazioni web.

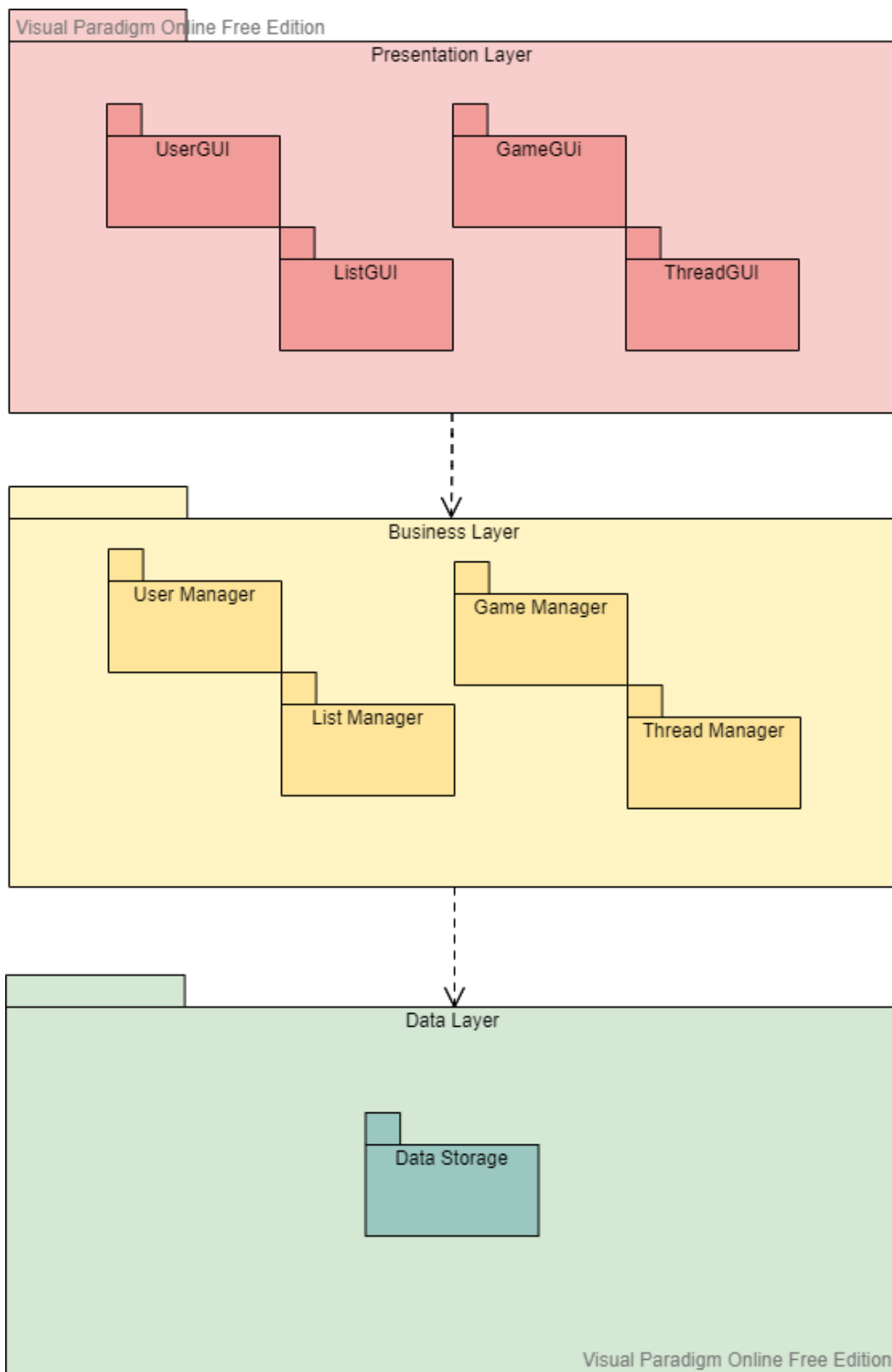
AJAX: acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.

lowerCamelCase: tecnica di naming delle variabili, adottata dallo standard Google Java, che consiste nello scrivere più parole insieme delimitando la fine e l'inizio di una nuova parola con una lettera maiuscola.

Servlet: oggetti scritti in linguaggio Java che operano all'interno di un server web.

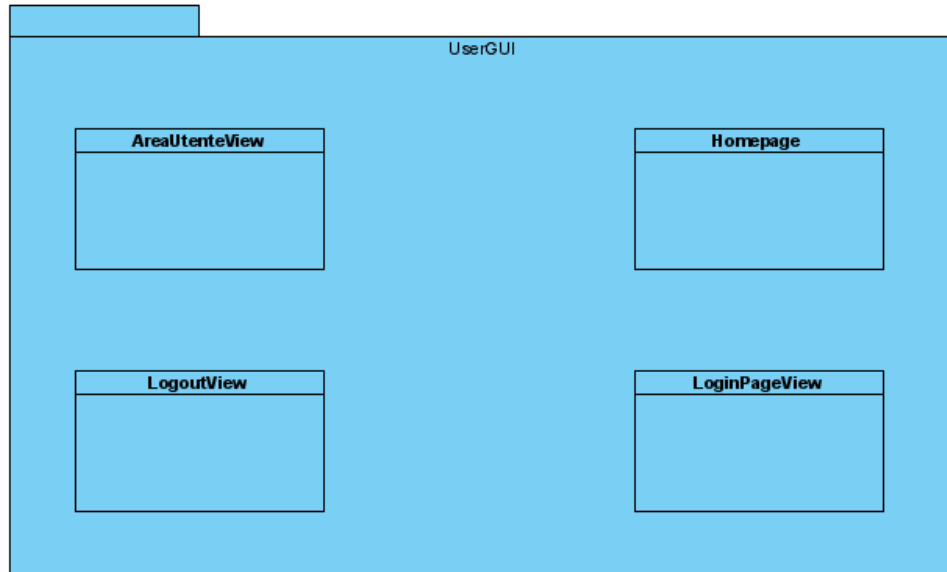
Tomcat: un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

2. Packages

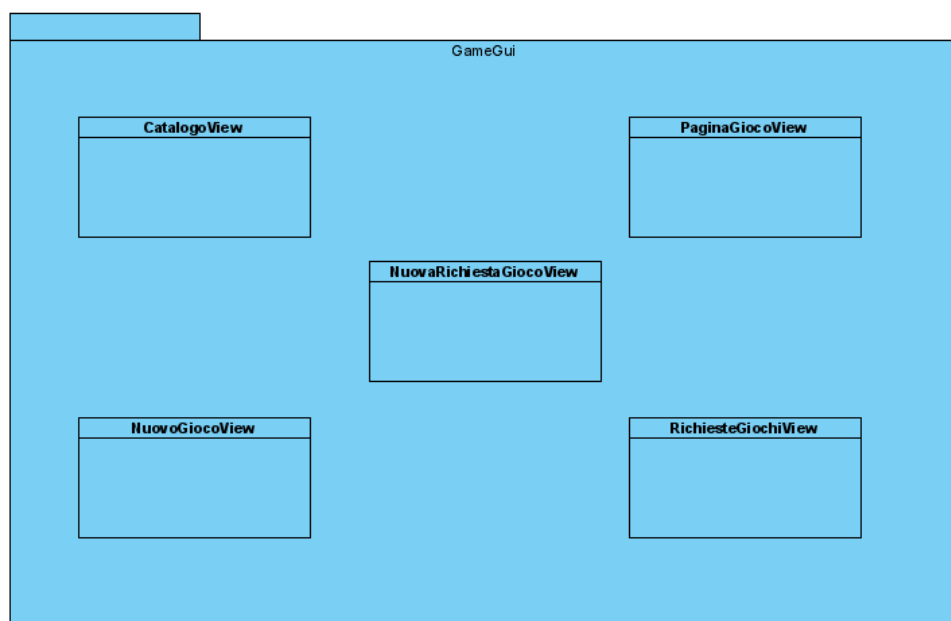


Il diagramma descrive la suddivisione in tre layer del sistema.

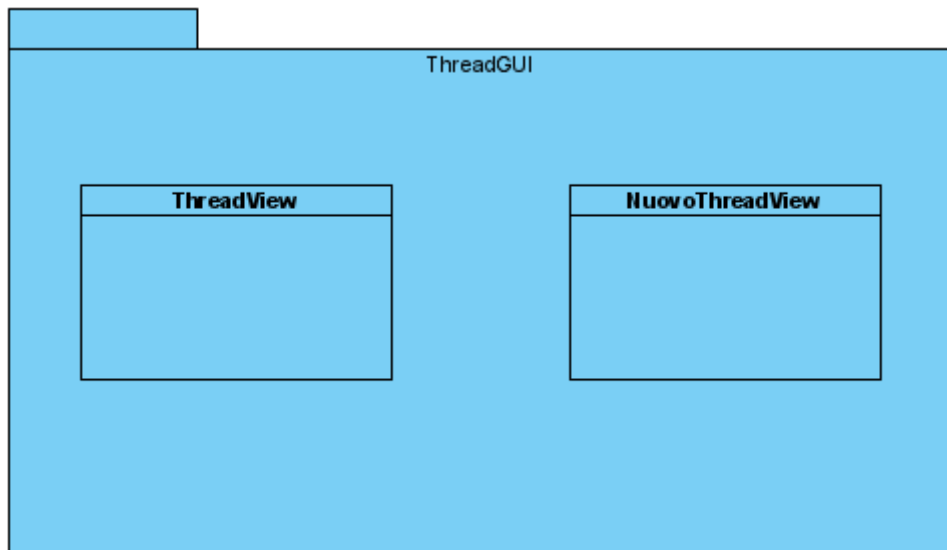
2.1 Package Interface UserManager



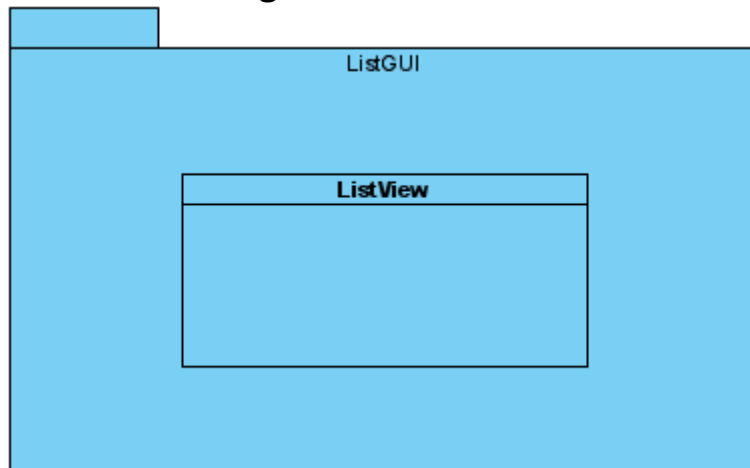
2.2 Package Interface GameManager



2.3 Package Interface ThreadManager

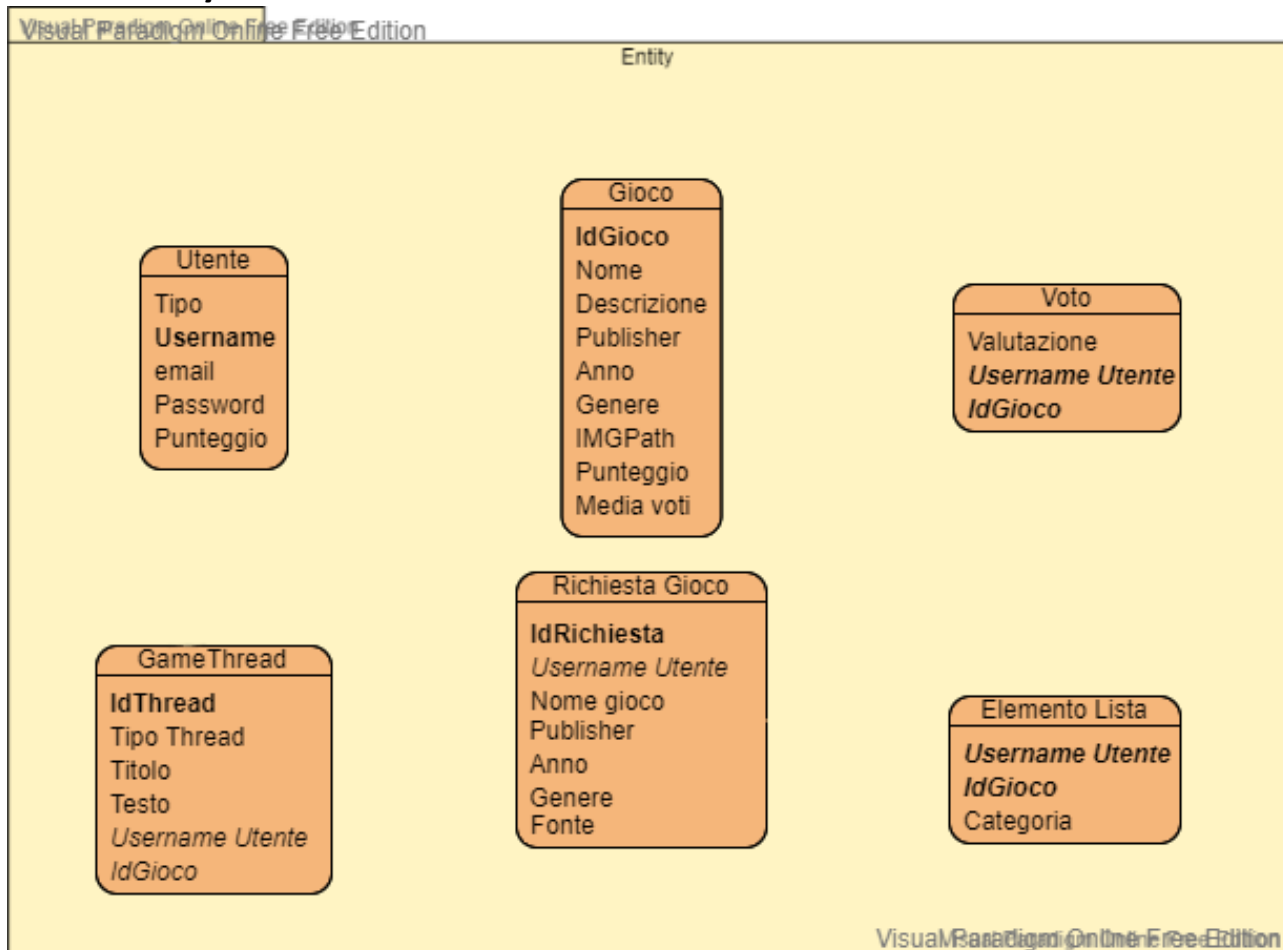


2.4 Package Interface ListManager



3. Interfacce delle classi

3.1 Entity



| | |
|----------------------|--|
| Nome classe | Utente |
| Descrizione | Questa classe rappresenta l'oggetto Utente |
| Signature dei metodi | + getTipo(): String + setTipo(tipo: String): void + getEmail(): String + setEmail(email: String): void + getUsername(): String + setUsername(username: String): void + getPassword(): String + setPassword(password: String): void + getPunteggio(): int + setPunteggio(punteggio: int): void + toString(): String |
| Pre-condizioni | Context Utente::setTipo(tipo) Pre: tipo=="user" OR tipo=="mod" OR tipo=="dev" OR tipo=="manager" |

| | |
|-----------------|--|
| | Context Utente::setEmail(email) Pre: email non deve avere altre corrispondenze nel database Context Utente::setUsername (username) Pre: username non deve avere altre corrispondenze nel database |
| Post-condizioni | Context Utente::setEmail(email) Post: email è presente nel database Context Utente::setUsername (username) Post: username è presente nel database |
| Invariante | Context Utente Inv: self.tipo=="user" OR self.tipo=="mod" OR self.tipo=="dev" OR self.tipo=="manager" |

| | |
|----------------------|--|
| Nome classe | Gioco |
| Descrizione | Questa classe rappresenta l'oggetto Gioco |
| Signature dei metodi | + getIdGioco(): int + setIdGioco(idGioco: int): void + getNome(): String + setNome(nome: String): void + getDescrizione(): String + setDescrizione(descrizione: String): void + getPublisher(): String + setPublisher(publisher: String): void + getAnno(): String + setAnno(anno: String): void + getGenere(): String + setGenere(genere: String): void + getImgpath(): String + setImgpath(imgpath: String): void + getPunteggio(): int + setPunteggio(punteggio: int): void + getMediaVoti(): float + setMediaVoti(mediaVoti: float): void + toString(): String |
| Pre-condizioni | Context Gioco::setIdGioco(idGioco) Pre: idGioco non deve avere altre corrispondenze nel database |
| Post-condizioni | Context Gioco::setIdGioco(idGioco) Post: idGioco è presente nel database |
| Invariante | Context Gioco Inv: self.mediaVoti >= 0 |

| | |
|----------------------|--|
| Nome classe | GameThread |
| Descrizione | Questa classe rappresenta l'oggetto Thread (Discussione/Post relativo ad un determinato gioco) |
| Signature dei metodi | + getIdThread(): int + setIdThread(idThread: int): void + getTipoThread(): String + setTipoThread(tipoThread: String): void + getTitolo(): String + setTitolo(titolo: String): void + getTesto(): String + setTesto(testo: String): void + getUsernameUtente(): String + setUsernameUtente(usernameUtente: String): void + getIdGioco(): int + setIdGioco(idGioco: int): void |
| Pre-condizioni | Context: GameThread::setIdThread(idThread) Pre: idThread non deve avere altre corrispondenze nel database Context: GameThread::setUsernameUtente(usernameUtente) Pre: usernameUtente deve avere una corrispondenza nel database come username di un oggetto Utente Context: GameThread::setIdGioco(idGioco) Pre: idGioco deve avere una corrispondenza nel database come id di un oggetto Gioco |
| Post-condizioni | Context: GameThread::setIdThread(idThread) Post: idThread è presente nel database Context: GameThread::setUsernameUtente(usernameUtente) Post: usernameUtente corrisponde a username di un oggetto Utente esistente Context: GameThread::setIdGioco(idGioco) Post: idGioco corrisponde a idGioco di un oggetto Gioco esistente |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | ElementoLista |
| Descrizione | Questa classe rappresenta l'oggetto Elemento Lista |
| Signature dei metodi | + getUsernameUtente(): String + setUsernameUtente(usernameUtente: String): void + getIdGioco(): int + setIdGioco(idGioco: int): void |

| | |
|-----------------|---|
| | + getCategory(): String + setCategoria(categoria: String): void |
| Pre-condizioni | Context: ElementoLista::setUsernameUtente(usernameUtente) Pre: usernameUtente deve avere una corrispondenza nel database come username di un oggetto Utente Context: ElementoLista::setIdGioco(idGioco) Pre: idGioco deve avere una corrispondenza nel database come id di un oggetto Gioco Context: ElementoLista::setCategoria(categoria) Pre: categoria="Acquistato" or categoria="In corso" or categoria="Completato" or categoria="Platinato" or categoria="Sviluppato" |
| Post-condizioni | Context: ElementoLista::setUsernameUtente(usernameUtente) Post: usernameUtente corrisponde a username di un oggetto Utente esistente Context: ElementoLista::setIdGioco(idGioco) Post: idGioco corrisponde a idGioco di un oggetto Gioco esistente |
| Invariante | Context: ElementoLista Inv: self.categoria="Acquistato" or self.categoria="In corso" or self.categoria="Completato" or self.categoria="Platinato" or self.categoria="Sviluppato" |

| | |
|----------------------|---|
| Nome classe | RichiestaGioco |
| Descrizione | Questa classe rappresenta l'oggetto Richiesta Gioco |
| Signature dei metodi | + getIdRichiesta(): int + setIdRichiesta(idRichiesta: int): void + getUsernameUtente(): String + setUsernameUtente(usernameUtente: String): void + getNomeGioco(): String + setNomeGioco(nomeGioco: String): void + getFonte(): String + setFonte(fonte: String): void + getRisposta(): boolean + setRisposta(risposta: boolean): void |
| Pre-condizioni | Context: RichiestaGioco::setUsernameUtente(usernameUtente) Pre: usernameUtente deve avere una corrispondenza nel database come username di un oggetto Utente Context: RichiestaGioco::setIdRichiesta(idRichiesta) Pre: idRichiesta non deve avere altre corrispondenze nel database |

| | |
|-----------------|--|
| Post-condizioni | Context: RichiestaGioco::setUsernameUtente(usernameUtente) Post: usernameUtente corrisponde a username di un oggetto Utente esistente Context: RichiestaGioco::setIdRichiesta(idRichiesta) Pre: idRichiesta è presente nel database |
| Invariante | |

| | |
|----------------------|--|
| Nome classe | Voto |
| Descrizione | Questa classe rappresenta l'oggetto Voto |
| Signature dei metodi | + getValutazione(): int + setValutazione(valutazione: int): void + getUsernameUtente(): String + setUsernameUtente(usernameUtente: String): void + getIdGioco(): int + setIdGioco(idGioco: int): void |
| Pre-condizioni | Context: Voto::setUsernameUtente(usernameUtente) Pre: usernameUtente deve avere una corrispondenza nel database come username di un oggetto Utente Context: Voto::setIdGioco(idGioco) Pre: idGioco deve avere una corrispondenza nel database come id di un oggetto Gioco |
| Post-condizioni | Context: Voto::setUsernameUtente(usernameUtente) Post: usernameUtente corrisponde a username di un oggetto Utente esistente Context: Voto::setIdGioco(idGioco) Post: idGioco corrisponde a idGioco di un oggetto Gioco esistente |
| Invariante | |

3.2 Manager

| | |
|----------------------|---|
| Nome classe | UtenteDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'utente. |
| Signature dei metodi | + checkLogin(Email: String, password: String): Utente + checkEmail(Email: String): Boolean + checkPassword(Email:String, pass: String): Boolean + getScore(username:String): Integer + findTopUsers(): Collection(Utente) |
| Pre-condizioni | Context UtenteDAO::checkLogin(Email, password) Pre: Email!=null AND password!=null Context UtenteDAO::checkEmail(Email) Pre: Email!=null Context UtenteDAO::checkPassword(Email, pass) Pre: Email!=null AND pass!=null Context UtenteDAO::getScore(username) Pre: username!=null |
| Post-condizioni | Context UtenteDAO::checkLogin(Email, password) Post: utenteLoggato = database.utente-> select(u utente.email=email and utente.password=password) Context UtenteDAO::checkEmail(Email) Post: true if database.utente->includes (select(u utente.email=Email)), false altrimenti Context UtenteDAO::checkPassword(Email, pass) Post: true if database.utente->includes (select(u utente.email=email and utente.password=pass)), false altrimenti Context UtenteDAO::getScore(username) Post: score = database.utente -> select(u.punteggio utente.username=username) AND score>=0 Context UtenteDAO::findTopUsers() Post: utenti->asOrderedSet() ->size()<=10 AND utenti-> sortedBy(u u.punteggio)->reverse() |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | GiocoDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'oggetto Gioco. |
| Signature dei metodi | + findAllGames(): Collection(Gioco) + getTopRated(): Collection(Gioco) + findGameById(id:Integer): Gioco + findByPublisher(publisher:String): Collection(Gioco) + findByGenre(genre:String): Collection(Gioco) + checkExistingGame(nome:String, publisher:String, anno:String): Boolean + getPublishers(): Collection(String) + getGenres(): Collection(String) + addGame(game: Gioco): Boolean + updateAverage(gameID:Integer, media:Float): Boolean |
| Pre-condizioni | <p>Context GiocoDAO::findGameById(id) Pre: id>0</p> <p>Context GiocoDAO::checkExistingGame(nome, publisher, anno) Pre: nome!=null, publisher!=null, anno!=null</p> <p>Context GiocoDAO::addGame(game) Pre: game!=null</p> <p>Context GiocoDAO::updateAverage(gameID, media) Pre: database.gioco->includes(select(g gioco.id=gameID) AND id>0 media>=0</p> |
| Post-condizioni | <p>Context GiocoDAO::findAllGames() Post: giochi->database.gioco->select(g)->asSet() and giochi->size()>=0</p> <p>Context GiocoDAO::getTopRated() Post: giochi->asOrderedSet()->size()>=0 AND giochi->size()<=11 AND giochi->sortedBy(g g.media_voti)->reverse()</p> <p>Context GiocoDAO::findGameById(id) Post: gioco->select(g g.id=id)</p> <p>Context GiocoDAO::findByPublisher(publisher) Post: giochi->select(g g.publisher=publisher) AND giochi->size>=0</p> <p>Context GiocoDAO::findByGenre(genre) Post: giochi->select(g g.genere=genre) AND giochi->size>=0</p> <p>Context GiocoDAO::checkExistingGame(nome, publisher, anno)</p> |

| | |
|------------|--|
| | <p>Post: true if database.gioco->includes(select(g gioco.nome=nome and gioco.publisher=publisher and gioco.anno=anno), false altrimenti</p> <p>Context GiocoDAO::getPublishers() Post: results->size()>=0</p> <p>Context GiocoDAO::getGenres() Post: results->size()>=0</p> <p>Context GiocoDAO::addGame(game) Post: database.gioco->includes(select(g gioco.nome=game.getNome() and gioco.publisher=fame.getPublisher() and gioco.anno=game.getAnno()))</p> <p>Context GiocoDAO::updateAverage(gameID, media) Post: true if database.gioco->includes(select(g gioco.id=gameID and gioco.media_voti=media), false altrimenti</p> |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | ThreadDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'oggetto GameThread. |
| Signature dei metodi | + viewThreadById(idThread:Integer): GameThread + viewThreadByGame(idGioco:Integer): Collection(GameThread) + addThread(thread:GameThread): boolean |
| Pre-condizioni | <p>Context ThreadDAO::addThread(thread) Pre: thread!=null AND database.gioco->includes(select(g gioco.id=thread.getIdGioco())) AND database.utente->includes (select(u utente.username=thread.getUsernameUtente()))</p> |
| Post-condizioni | <p>Context ThreadDAO::viewThreadById(idThread) Post: thread->select(t thread.id=id)</p> <p>Context ThreadDAO::viewThreadByGame(idGioco) Post: threads->select(t thread.id_gioco=idGioco) AND threads->size()>=0</p> <p>Context ThreadDAO::addThread(thread) Post: database.thread->includes(thread)</p> |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | VotoDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'oggetto Voto. |
| Signature dei metodi | + getVote(idGioco:Integer, utente:String): Voto + addVote(voto: Voto): Boolean + deleteVote(gameid: Integer, user: String): Boolean + updateVote(gameid: Integer, user: String, voto: Integer): Boolean + calculateAverage(gameid: Integer): Float + findAllVotes(): Collection(Voto) |
| Pre-condizioni | <p>Context VotoDAO::getVote(idGioco, utente) Pre: idGioco>0 AND utente!=null</p> <p>Context VotoDAO::addVote(voto) Pre: voto!=null AND database.gioco->includes(select(g gioco.id=voto.getIdGioco())) AND database.utente->includes (select(u utente.username=voto.getUsernameUtente())) AND database.voto-> not includes(select(v voto.id_gioco=voto.getIdGioco() AND voto.utente=voto.getUsernameUtente)</p> <p>Context VotoDAO::deleteVote(gameid, user) Pre: gameid>0 AND user!=null and database.voto -> includes(select(v voto.id_gioco=gameid AND voto.utente=user)</p> <p>Context VotoDAO::updateVote(gameid, user, voto) Pre: voto>=0 AND gameid>0 AND user!=null AND database.voto -> includes(select(v voto.id_gioco=gameid AND voto.utente=user)</p> |
| Post-condizioni | <p>Context VotoDAO::getVote(idGioco, utente) Post: voto-> select(v voto.id_gioco=idGioco AND voto.utente=utente)</p> <p>Context VotoDAO::addVote(voto) Post: database.voto-> includes(select(v voto.id_gioco=voto.getIdGioco() AND voto.utente=voto.getUsernameUtente)</p> <p>Context VotoDAO::deleteVote(gameid, user) Post: database.voto-> not includes(select(v voto.id_gioco=gameid AND voto.utente=user)</p> <p>Context VotoDAO::updateVote(gameid, user, voto) Post: database.voto-> includes(select(v voto.id_gioco=gameid AND voto.utente=user and voto.valutazione=voto)</p> |

| | |
|------------|--|
| | Context VotoDAO::calculateAverage(gameid) Post: avg>=0 Context VotoDAO::findAllVotes() Post: voti->database.voto->select(v) AND voti->size()>=0 |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | RichiestaGiocoDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'oggetto RichiestaGioco |
| Signature dei metodi | + addGameRequest(req: RichiestaGioco): Boolean + viewRequestById(id: Integer): RichiestaGioco + getAllRequests(): Collection(RichiestaGioco) + deleteGameRequest(id: Integer): Boolean |
| Pre-condizioni | Context RichiestaGiocoDAO::addGameRequest(req) Pre: req!=null AND database.utente->includes(u utente.username=req.getUsernameUtente()) Context RichiestaGiocoDAO::deleteGameRequest(id) Pre: database.richiesta_gioco->includes(select(r richiesta_gioco.id=id) |
| Post-condizioni | Context RichiestaGiocoDAO::addGameRequest(req) Post: database.richiesta_gioco-> includes(req) Context RichiestaGiocoDAO::viewRequestById(id) Post: richiesta->select(r richiesta_gioco.id=id) if database.richiesta_gioco->includes(r richiesta_gioco.id=id), null altrimenti Context RichiestaGiocoDAO::deleteGameRequest(id) Post: database.richiesta_gioco-> not includes(select(r richiesta_gioco.id=id) Context RichiestaGiocoDAO::getAllRequests() Post: richieste->database.richiesta_gioco->select(r) and richieste->size()>=0 |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | ElementoListaDAO |
| Descrizione | Questa classe è un manager che si occupa di interagire col database. Gestisce le query riguardanti l'oggetto ElementoLista (elemento che fa riferimento a uno specifico gioco per uno specifico utente, con una determinata categoria). |
| Signature dei metodi | + addListElement(elem: ElementoLista): Boolean + getUserList(username: String): Collection(ElementoLista) + getListElement(username: String, gameid: Integer): ElementoLista + deleteListElement(gameid: Integer, user: String): Boolean + updateCategory(user: String, gameid: Integer, categoria: String): Boolean + getGameName(gameid: Integer): String + getGameScore(gameid: Integer): Integer + updateUserScore(user: String, punteggio: Integer): Boolean + getCategoryStats(gameid: Integer, categoria: String): Integer |
| Pre-condizioni | <p>Context ElementoListaDAO::addListElement(elem) Pre: elem!=null AND database.gioco->includes(select(g gioco.id=elem.getIdGioco())) AND database.utente->includes (select(u utente.username=elem.getUsernameUtente())) AND database.elemento_lista->not includes(select(e elemento_lista.utente=elem.getUsernameUtente() AND elemento_lista.id_gioco=elem.getIdGioco()))</p> <p>Context ElementoListaDAO::deleteListElement(gameid, user) Pre: user!=null AND gameid>0</p> <p>Context ElementoListaDAO::updateCategory(user, gameid, categoria) Pre: user!=null AND gameid>0 AND categoria!=null AND database.elemento_lista->includes(select(e elemento_lista.utente=user and elemento_lista.id_gioco=gameid)</p> <p>Context ElementoListaDAO::getGameName(gameid) Pre: gameid>0</p> <p>Context ElementoListaDAO::getGameScore(gameid) Pre: gameid>0</p> <p>Context ElementoListaDAO::updateUserScore(user, punteggio) Pre: punteggio>=0 AND user!=null AND database.utente->includes(select(u utente.username=username)</p> |
| Post-condizioni | <p>Context ElementoListaDAO::addListElement(elem) Post: database.elemento_lista->includes(select(e elemento_lista.utente=elem.getUsernameUtente() and elemento_lista.id_gioco=elem.getIdGioco()))</p> |

| | |
|------------|--|
| | <p>Context ElementoListaDAO::getUserList(username) Post: results->size()>=0</p> <p>Context ElementoListaDAO::getListElement(username, gameid) Post: elemento_lista->select(e elemento_lista.utente=username and elemento_lista.id_gioco=gameid) if database.elemento_lista ->includes(e elemento_lista.utente=username and elemento_lista.id_gioco=gameid), null altrimenti</p> <p>Context ElementoListaDAO::deleteListElement(gameid, user) Post: database.elemento_lista-> not includes(select(e elemento_lista.utente=user and elemento_lista.id_gioco=gameid))</p> <p>Context ElementoListaDAO::updateCategory(user, gameid, categoria) Post: database.elemento_lista-> includes(select(e elemento_lista.utente=user and elemento_lista.id_gioco=gameid and elemento_lista.categoria=categoria))</p> <p>Context ElementoListaDAO::getGameName(gameid) Post: name=database.gioco->select(g.nome gioco.id=gameid) if database.gioco->includes(g gioco.id=gameid), null altrimenti</p> <p>Context ElementoListaDAO::getGameScore(gameid) Post: score=database.gioco->select(g.punteggio gioco.id=gameid) if database.gioco->includes(g gioco.id=gameid), 0 altrimenti</p> <p>Context ElementoListaDAO::updateUserScore(user, punteggio) Post: database.utente-> includes(select(u u.username=user and u.punteggio=punteggio))</p> <p>Context ElementoListaDAO::getCategoryStats(gameid, categoria) Post: count>=0</p> |
| Invariante | |

3.3 Control

| | |
|----------------------|---|
| Nome classe | AddGame |
| Descrizione | Questa classe è un control che si occupa di passare a GiocoDAO i dati di un gioco da aggiungere. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context AddGame:doPost(request, response) Pre: request.getParameter("game_title")!=null AND request.getParameter("publisher")!=null AND request.getParameter("game_genre")!=null AND request.getParameter(game_year)!=null AND request.getParameter("game_desc")!=null AND request.getParameter("score")!=null |
| Post-condizioni | Context AddGame:doPost(request, response) Post: request.getAttribute("aggiuntaGioco")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | FindAllServlet |
| Descrizione | Questa classe è un control che si occupa di prelevare l'intero catalogo da GiocoDAO. |
| Signature dei metodi | # doGet (request: HttpServletRequest, response: HttpServletResponse): void + doPost (request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | |
| Post-condizioni | Context FindAllServlet::doGet (request, response) Post: request.getAttribute("giochi")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | FindByGenreServlet |
| Descrizione | Questa classe è un control che si occupa di interfacciarsi con GiocoDAO per ottenere i giochi appartenenti a un genere selezionato dall'utente. |
| Signature dei metodi | # doGet (request: HttpServletRequest, response: HttpServletResponse): void + doPost (request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context FindByGenreServlet::doGet(request, response) Pre: request.getParameter("gen")!=null |
| Post-condizioni | Context FindByGenreServlet::doGet(request, response) Post: request.getAttribute("giochi")!=null AND dispatcher!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | FindByPublisherServlet |
| Descrizione | Questa classe è un control che si occupa di interfacciarsi con GiocoDAO per ottenere i giochi pubblicati da un editore selezionato dall'utente. |
| Signature dei metodi | # doGet (request: HttpServletRequest, response: HttpServletResponse): void + doPost (request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context FindByPublisherServlet::doGet(request, response) Pre: request.getParameter("pub")!=null |
| Post-condizioni | Context FindByPublisherServlet::doGet(request, response) Post: request.getAttribute("giochi")!=null AND dispatcher!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | GamePageServlet |
| Descrizione | Questa classe è un control che si occupa della visualizzazione di un gioco in base alle interazioni di un utente. |
| Signature dei metodi | # doGet (request: HttpServletRequest, response: HttpServletResponse): void + doPost (request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context GamePageServlet::doPost(request, response) Pre: request.getParameter("id")!=null |
| Post-condizioni | Context GamePageServlet::doPost(request, response) Post: dispatcher!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | Ricerca |
| Descrizione | Questa classe è un control che si occupa di gestire la ricerca per parola chiave di un gioco. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | |
| Post-condizioni | Context Ricerca::doPost(request, response) Post: request.getAttribute("result")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | AddVoteServlet |
| Descrizione | Questa classe è un control che si occupa di passare a VotoDAO i dati di un voto da registrare. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context AddVoteServlet::doPost(request, response) Pre: request.getParameter("username")!=null AND request.getParameter("game_id")!=null AND request.getParameter("vote_value")!=null |
| Post-condizioni | Context AddVoteServlet::doPost(request, response) Post: request.getAttribute("result")!=null |
| Invariante | |

| | |
|----------------------|--|
| Nome classe | DeleteVoteServlet |
| Descrizione | Questa classe è un control che si occupa di passare a VotoDAO i dati di un voto da rimuovere. |
| Signature dei metodi | # doGet(request: HttpServletRequest, response: HttpServletResponse): void + doPost(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context DeleteVoteServlet::doGet(request, response) Pre: request.getParameter("username")!=null AND request.getParameter("game_id")!=null |
| Post-condizioni | Context DeleteVoteServlet::doGet(request, response) Post: request.getAttribute("result")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | CreateGameReq |
| Descrizione | Questa classe è un control che si occupa di passare a RichiestaGiocoDAO i dati di una richiesta gioco da pubblicare. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context CreateGameRequest::doPost(request, response) Pre: request.getParameter("gamereq_title")!=null AND request.getParameter("gamereq_source")!=null AND request.getParameter("publisher")!=null AND request.getParameter("game_genre")!=null AND request.getParameter("game_year")!=null AND request.getParameter("username")!=null |
| Post-condizioni | Context CreateGameRequest::doPost(request, response) |

| | |
|------------|--|
| | Post: request.getAttribute(" richiestagioco")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | AcceptRequestServlet |
| Descrizione | Questa classe è un control che si occupa di gestire l'accettazione di una richiesta per un nuovo gioco, passa l'id della richiesta a RichiestaGiocoDAO, che ne ricava i dati, passandoli a GiocoDAO per aggiungere il titolo, e successivamente elimina la richiesta. |
| Signature dei metodi | # doGet(request: HttpServletRequest, response: HttpServletResponse): void + doPost(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context AcceptRequestServlet::doGet(request, response) Pre: request.getParameter("reqid")!=null |
| Post-condizioni | Context AcceptRequestServlet::doGet(request, response) Post: request.getAttribute("result")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | RefuseRequestServlet |
| Descrizione | Questa classe è un control che si occupa di gestire il rifiuto di una richiesta per un nuovo gioco, passa l'id della richiesta a RichiestaGiocoDAO, che si occuperà di eliminarla dal database. |
| Signature dei metodi | # doGet (request: HttpServletRequest, response: HttpServletResponse): void + doPost(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context RefuseRequestServlet::doGet(request, response) Pre: request.getParameter("reqid")!=null |
| Post-condizioni | Context RefuseRequestServlet::doGet(request, response) Post: request.getAttribute("result")!=null |
| Invariante | |

| | |
|----------------------|--|
| Nome classe | CreateThread |
| Descrizione | Questa classe è un control che si occupa di passare a ThreadDAO i dati di un thread da pubblicare. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context CreateThread::doPost(request, response) Pre: request.getParameter("thread_title")!=null AND request.getParameter("thread_type")!=null AND |

| | |
|-----------------|---|
| | request.getParameter("thread_text")!=null AND request.getParameter("gameId")!=null AND request.getParameter("username")!=null |
| Post-condizioni | Context CreateThread::doPost(request, response) Post: request.getAttribute("inserimentoThread")!=null |
| Invariante | |

| | |
|----------------------|--|
| Nome classe | ViewThread |
| Descrizione | Questa classe è un control che si occupa della visualizzazione di un thread in base alle interazioni di un utente |
| Signature dei metodi | #doGet(request: HttpServletRequest, response: HttpServletResponse): void + doPost(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context ThreadControl::doGet(request, response) Pre: request.getSession().getAttribute("threadid")!=null |
| Post-condizioni | Context ThreadControl::doGet(request, response) Post: dispatcher!=null |
| Invariante | |

| | |
|----------------------|--|
| Nome classe | UserLogin |
| Descrizione | Questa classe è un control che si occupa di gestire l'operazione di login. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context UserLogin::doPost(request, response) Pre: request.getParameter("user_email")!=null AND request.getParameter(user_password)!=null |
| Post-condizioni | Context UserLogin::doPost(request, response) Post: request.getAttribute("loginResult")!=null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | UserLogout |
| Descrizione | Questa classe è un control che si occupa di gestire l'operazione di logout |
| Signature dei metodi | # doGet(request: HttpServletRequest, response: HttpServletResponse): void + doPost(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context UserLogout::doPost(request, response) Pre: Request.getSession(false).getAttribute("utenteLoggato")!=null AND Request.getSession(false).getAttribute("utente")!=null |

| | |
|-----------------|---|
| Post-condizioni | Context UserLogout::doPost(request, response) Post: request.getSession().getAttribute("utenteLoggato")==null AND Request.getSession(false).getAttribute("utente")==null |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | AddToListServlet |
| Descrizione | Questa classe è un control che si occupa di passare a ElementoListaDAO i dati di un elemento lista da aggiungere. |
| Signature dei metodi | + doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context AddToListServlet::doGet(request, response) Pre: request.getParameter("username")!=null AND request.getParameter("game_id")!=null AND request.getParameter("category_value")!=null AND request.getParameter("score")!=null AND request.getParameter("usrScore")!=null |
| Post-condizioni | Context AddToListServlet::doGet(request, response) Post: ElementoListaDAO.addListElement(username)==true AND ElementoListaDAO.updateUserScore(user, punteggio)==true |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | ChangeCategoryServlet |
| Descrizione | Questa classe è un control che si occupa di passare a ElementoListaDAO i dati di un Elemento Lista per cui cambiare la categoria. |
| Signature dei metodi | # doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context ChangeCategoryServlet::doGet(request, response) Pre: request.getParameter("username")!=null AND request.getParameter("game_id")!=null AND request.getParameter("category_value")!=null AND request.getParameter("user_category")!=null AND request.getParameter("score")!=null AND request.getParameter("usrScore")!=null |
| Post-condizioni | Context ChangeCategoryServlet::doGet(request, response) Post: ElementoListaDAO.updateCategory(user, gameid, categoria)==true AND ElementoListaDAO.updateUserScore(user, punteggio)==true |
| Invariante | |

| | |
|----------------------|---|
| Nome classe | DeleteFromListServlet |
| Descrizione | Questa classe è un control che si occupa di passare a ElementoListaDAO i dati di un Elemento Lista da rimuovere. |
| Signature dei metodi | + doPost(request: HttpServletRequest, response: HttpServletResponse): void + doGet(request: HttpServletRequest, response: HttpServletResponse): void |
| Pre-condizioni | Context DeleteFromServlet::doGet(request, response) Pre: request.getParameter("username")!=null AND request.getParameter("game_id")!=null AND request.getParameter("category_value")!=null AND request.getParameter("user_category")!=null AND request.getParameter("score")!=null AND request.getParameter("usrScore")!=null |
| Post-condizioni | Context DeleteFromServlet:doGet(request, response) Pre: ElementoListaDAO.deleteListElement(gameid, user)==true ElementoListaDAO.updateUserScore(user, punteggio)==true |
| Invariante | |

| | |
|-----------------------------|---|
| Lista | Un insieme di giochi appartenenti ad un utente iscritto al sito e catalogati in base al loro livello di completamento. |
| Voto | Una valutazione in decimi da parte di un utente per un determinato gioco. |
| Utente Standard | Utente comune iscritto al sito, può gestire la sua lista, valutare i giochi e partecipare a discussioni. |
| Game Developer/Sviluppatore | Uno sviluppatore di videogiochi, che può aggiungere liberamente giochi e thread all'applicazione. |
| Moderatore | Un utente di GameSquare che si occupa della moderazione dell'aspetto social del sito, aggiungendo/rimuovendo thread e accettando richieste per la loro pubblicazione. |
| Gestore Catalogo | Un utente di GameSquare che si occupa esclusivamente della gestione dei giochi nel sito, aggiungendoli, modificandoli e accettando richieste per la loro aggiunta. |