Object Design Document

GameSquare

1. Introduzione

## 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.

## 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, il framework che verrà usato per il comparto grafico è *Bootstrap*, un framework open source utile per creare la grafica di siti web e web application, contenente strumenti basati su HTML e CSS sia per la tipografia che per l’interfaccia, così come estensioni opzionali realizzate con Javascript.

Per velocizzare il sistema e ottenere effetti visuali verranno usati JQuery e Javascript, che permetteranno all’interfaccia di rispondere alle azioni dell’utente in maniera dinamica, e un miglioramento dell’esperienza con le tecniche basate su Ajax.

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>).

## 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. **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 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 (non sono consentiti caratteri speciali) e seguono la notazione lowerCamelCase.

* + 1. **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. **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. **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. **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. **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 (\_);

## Design Pattern

[…]

## 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

**2.1 Package Interface GestoreUtente**

**2.2 Package Interface GestoreGioco**

**2.3 Package Interface GestoreThread**

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: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 |  |

|  |  |
| --- | --- |
| 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 |  |

|  |  |
| --- | --- |
| Nome classe | Game Thread |
| Descrizione | Questa classe rappresenta l’oggetto Thread |
| 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:** Thread:setIdThread(idThread)  **Pre:** idThread non deve avere altre corrispondenze nel database  **Context:** Thread:setUsernameUtente(usernameUtente)  **Pre:** usernameUtente deve avere una corrispondenza nel database come username di un oggetto Utente  **Context:** Thread:setIdGioco(idGioco)  **Pre:** idGioco deve avere una corrispondenza nel database come idGioco di un oggetto gioco |
| Post-condizioni | **Context:** Thread:setIdThread(idThread)  **Post:** idThread è presente nel database  **Context:** Thread:setUsernameUtente(usernameUtente)  **Post:** usernameUtente corrisponde a username di un oggetto Utente esistente  **Context:** Thread: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  + getCategoria(): 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 idGioco di un oggetto gioco |
| 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 |  |

|  |  |
| --- | --- |
| 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 idGioco 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 Model**

|  |  |
| --- | --- |
| 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  + logout(session: HttpSession): void  + checkEmail(Email: String): boolean  + checkPassword(Pass: String): boolean |
| Pre-condizioni |  |
| Post-condizioni |  |
| 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 | + viewGame |
| Pre-condizioni |  |
| Post-condizioni |  |
| Invariante |  |