**Immagine che contiene interni

Descrizione generata automaticamente**

**Documentazione Object Design**

**“GLITCH”**

**Progetto presentato da:**

Annunziata Elefante

Ferdinando Napolitano

Santolo Mutone

**Indice**

**1. Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3**

* 1. Trade-off . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  2. Componenti off-the-shelf. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  3. Linee guida per la documentazione dell’interfaccia. . . . . . . . . . . . . . . . . . . 3
  4. Design pattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  5. Definizioni, acronimi e abbreviazioni . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

**2. Packages**

**3. Interfacce delle classi**

1. **Introduzione**

Con l’Object Design Document specifichiamo in modo dettagliato le decisioni prese in fase di analisi e di design riguardo Glitch; in particolare verranno specificati i principali trade-offs, descritte le componenti off-the-shelfs utilizzate dal sistema, le linee guida per la documentazione delle interfacce e l’individuazione dei Design Patterns. Inoltre, verranno definiti i packages, le class interfaces e i class diagram.

* 1. **Trade-off**
* **Build VS Buy**

Questo trade-off ci mette davanti un quesito molto importante sul futuro del nostro software: costruire VS comprare. Per avere un sistema originale e che rispecchi al meglio i nostri bisogni e le nostre idee, si preferirebbe costruire interamente il sistema; tale scelta porta ad un aumento del lavoro da parte del gruppo che dovrebbe costruire il proprio software consentendo una migliore applicazione sulla realtà di interesse. Tale scelta viene presa in considerazione soprattutto quando non si dispone di un budget elevato. Quando però alcuni fattori vengono a mancare, come ad esempio tempo, tools o un gruppo tecnico, si preferisce comprare il sistema; tale scelta porta ad ottenere dei risultati ottimali che verrebbero consegnati in un tempo più ristretto. Pertanto, il nostro software verrà costruito da zero da parte dell’intero gruppo al fine di ottenere un sistema unico nel suo genere;

* **Security VS Performance**

Ogni sistema dovrebbe offrire in egual misura sia un’ottima sicurezza che alte performance, ma queste non sono direttamente correlate, anzi, guadagnare in una vuol dire perdere nell’altra. A tal proposito è importante raggiungere il giusto equilibrio e prendere in considerazione gli aspetti principali di entrambe per capire quale prediligere. Sebbene avere un sistema ad alte performance vuol dire avere tempi di risposta e compilazione molto bassi e un buon throughput, si è deciso di dare una maggiore importanza alla sicurezza così da garantire la protezione dei dati utente e una giusta gestione dei permessi sulle operazioni e sui servizi;

* **Understandability VS Costs**

Un sistema software deve garantire la comprensibilità. È importante, infatti, rendere il codice semplice da leggere così da semplificare le modifiche successive, non solo da chi l’ha realizzato ma anche da coloro che sono esterni al progetto o che magari non sono stati coinvolti in una determinata parte del codice. Per rendere il codice quanto più comprensibile verranno utilizzati dei commenti che permetteranno una maggiore leggibilità del sistema. Quindi daremo maggiore importanza alla comprensibilità anche se ciò comporta un aumento dei costi di sviluppo e di tempo.

* 1. **Componenti off-the-shelf**

Nella realizzazione di Glitch andremo ad utilizzare componenti off-the-shelf già disponibili per facilitare lo sviluppo del progetto.

Per la progettazione del lato front-end utilizzeremo Bootstrap 4, che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Tale framework include esempi di progettazione basati su HTML5, CSS3 e alcune estensioni di JavaScript.

I siti web a cui faremo riferimento per la definizione del nostro template sono: [www.amazon.it](http://www.amazon.it), esempio di sito di e-commerce generico, e [www.gamestop.it](http://www.gamestop.it) , come sito di vendita console e videogiochi.

Il framework che utilizzeremo per implementare il codice del nostro software sarà Spring e l’interazione con esso avverrà tramite linguaggio di programmazione Java. Per lo sviluppo del codice verrà utilizzata la libreria jQuery e la tecnica di sviluppo AJAX.

Per la creazione del database ci serviremo del software MySQL che sarà connesso all’ambiente di sviluppo tramite il driver JDBC e usufruiremo del framework Java Persistence API per la gestione dei dati persistenti.

* 1. **Linee guida per la documentazione dell’interfaccia**

**Linee guida per classi e interfacce java e Spring**

Per ogni sorgente di classe Java implementato, dovranno essere rispettati i seguenti parametri:

* Classi e interfacce:
  + la nomenclatura delle classi dovrà rispettare la notazione UpperCamelCase;
  + l’identificativo delle classi non dovrà essere ambiguo, ma congruo allo scopo della classe;
  + l’identificativo della classe dovrà essere formulato al singolare;
  + l’identificativo delle classi dovrà essere un nome (Es. Array) o al più un nome frasale (Es. ArrayList);
* Costanti:
  + i valori immutabili definiti in classi java dovranno essere definiti come “static final”;
  + i nomi di costanti dovranno essere definiti in maiuscolo;
  + è ammessa la separazione tramite \_ là dove necessario.
* Variabili d’istanza e variabili locali:
  + i nomi di parametri o variabili locali dovranno essere definiti secondo la notazione “*lowerCamelCase*”;
  + per ogni parametro d’istanza definito nella classe, dovrà essere definito il livello di visibilità.
* Metodi:
  + gli identificativi dei metodi dovranno seguire la notazione “*lowerCamelCase*”;
  + gli identificativi dei metodi dovranno iniziare con un verbo;
  + per ogni metodo di una classe sarà necessario specificare il livello di visibilità;
  + eventuali parametri nella firma del metodo, dovranno seguire le convenzioni adottate per le variabili d’istanza.
* Blocchi e indentazioni:
  + il codice dovrà essere accuratamente indentato, tramite un “*Tab*” per ogni livello d’indentazione;
  + le parentesi graffe per l’inizio di un nuovo blocco di codice dovranno essere riportate sulla stessa riga della definizione del blocco;
  + le parentesi graffe di fine blocco dovranno essere allineate con l’inizio della definizione del blocco.

* Blocchi eccezionali:
  + ogni blocco “*try/catch*” definito all’interno di un metodo dovrà essere indentato in maniera corretta secondo le specifiche sopra riportate;
  + le clausole “*catch*” dovranno essere riportate in maniera ordinata, dalla più specifica alla più generale, in caso di relazioni di estensione tra le tipologie di eccezioni coinvolte;
  + ogni messaggio di errore gestito da stampare a video dovrà riportare un messaggio specificato dal programmatore che ne identifichi con chiarezza il tipo di errore e la provenienza;
  + insiemi di operazioni comuni tra il blocco “*try”* e seguenti blocchi “*catch*” dovranno essere riportate nel blocco di chiusura “*finally*”.
* Annotazioni:
  + le annotazioni previste per una classe o per un metodo dovranno apparire una per riga, subito dopo il blocco di documentazione;
  + eventuali annotazioni prima di un parametro della classe dovranno essere definite una per linea, al disopra del parametro stesso, senza lasciare linee vuote.
* Commenti:
  + per ogni metodo della classe dovranno essere riportati blocchi di commenti che aiutino a capire il corretto flusso di operazioni del metodo;
  + sarà necessario chiarire, tramite commenti, operazioni innestate o eventuali blocchi poco chiari in prima lettura;
  + i commenti devono essere conformi ai seguenti esempi:

/\* //Secondo /\*

\* Primo // esempio \* Terzo esempio \*/

\* esempio

\*/

**Linee guida per pagine HTML 5**

Per ogni documento HTML 5 creato, dovranno essere rispettati i seguenti parametri:

* ogni documento creato dovrà riportare il tag “<!doctype html>” per identificare la tecnologia HTML 5 utilizzata;
* ogni tag aperto nel corpo del documento HTML, a meno di tag singoli, dovrà riportare il rispettivo tag di chiusura;
* la struttura base di una pagina html (head, body), a meno che non siano incluse in una JSP, dovrà essere rispettata;
* ogni documento html, in particolare nel corpo del documento, dovrà essere indentato (preferibilmente tramite una tabulazione per livello) ad ogni definizione di un nuovo tag;
* non potranno essere definiti più tag HTML sulla stessa riga;
* è preferibile definire tag in minuscolo.

**Linee guida per script JavaScript**

* ogni funzione Javascript dovrà essere riportata in un documento diverso dalla pagina html inclusa;
* i nomi di funzioni, variabili e costanti dovranno seguire le stesse specifiche definite per i documenti Java;
* ogni script dovrà essere incluso alla fine del body del relativo documento HTML.

**Linee guida per Fogli di stile CSS 3**

* ogni regola CSS non incline dovrà essere riportata su un documento differente rispetto a quello della pagina html di riferimento, in modo da garantire un più facile riuso senza duplicazione;
* ogni regola CSS dovrà iniziare all’inizio di una nuova linea, con la specifica dei selettori della regola;
* l’ultimo selettore di una regola CSS dovrà essere seguito dall’apertura del blocco con { ;
* l’indentazione dovrà seguire i seguenti criteri:
  + inizio di una nuova regola (#....{) e fine del blocco della regola (}) livello di indentazione 0;
  + le proprietà di ogni regola CSS dovranno essere indentate di un “*Tab*” rispetto all’inizio del blocco, e dovranno essere riportate 1 per riga.
  1. **Design pattern**
  2. **Definizioni, acronimi e abbreviazioni**

1. **Package**

La divisione in package proposta segue la divisione in sottosistemi individuata nella fase di system design. I package usati sono:

* Il package *Controller* che contiene al suo interno le servlet che rappresentano i servizi offerti dai sottosistemi ServiziAccesso, ServiziCarrello, ServiziRuolo, ServiziAccount, ServiziEmail e ServiziCatalogo del layer application, individuati nel SDD.
* Il package *Model* contiene al suo interno le jpa che rappresentano i servizi offerti dal sottosistema DataManager del layer storage, individuato nel SDD.
* Il package *View* rappresenta i servizi offerti dai sottosistemi InterfacciaUtente, InterfacciaGestoreAssistenza, InterfacciaGestoreCatalogo, InterfacciaGestoreAccount del layer presentation, individuati nel SDD.

Il motivo che ha portato alla prima suddivisione dei package (in particolare controller, model e view) è l’utilizzo dell’architettura MVC la quale ci permette di separare la logica di business (package controller) dalla presentazione (package view) e gestione dei dati (package model). Il package model è suddiviso in altri due package: persistence, che contiene le interfacce per la gestione dei dati persistenti e le jpa che implementano queste interfacce e bean, che contiene gli oggetti.

* 1. **Package controller**

In questo package sono presenti tutte le servlet che implementeranno la logica di business e che con l’utilizzo delle interfacce dao, contenute all’interno del package model, comunicheranno con le jpa (presenti nello stesso package) gestendo i dati contenuti nei bean (presenti nello stesso package).

* **ServiziAccesso**:
  + RegistrazioneServlet.java;
  + LoginServlet.java;
  + LogoutServlet.java;
* **ServiziCarrello:**
  + GestioneCarrelloServlet.java;
  + AcquistoServlet.java;
  + InserimentoCartaServlet.java;
* **ServiziAccount:**
  + GestioneProfiloServlet.java;
* **ServiziRuolo:**
  + GestioneRuoloServlet.java;
* **ServiziEmail:**
  + RichiestaAssistenzaServlet.java;
  + RispostaAssistenzaServlet.java;
* **ServiziCatalogo:**
  + GestioneProdottoServlet.java;
  + GestioneOffertaServlet.java.

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| RegistrazioneServlet.java | Gestisce l’operazione di registrazione di un nuovo utente |
| LoginServlet.java | Gestisce l’operazione di accesso di un utente |
| LogoutServlet.java | Gestisce l’operazione di logout di un utente |
| GestioneCarrelloServlet.java | Gestisce le operazioni di aggiunta e rimozione di un prodotto nel carrello |
| AcquistoServlet.java | Gestisce l’operazione di acquisto prodotti nel carrello |
| InserimentoCartaServlet.java | Gestisce le operazioni di inserimento della carta di credito |
| GestioneProfiloServlet.java | Gestisce le operazioni di modifica e rimozione profilo utente |
| GestioneRuoloServlet.java | Gestisce le operazioni di aggiunta e rimozione ruolo utente |
| RichiestaAssistenzaServlet.java | Gestisce l’operazione di richiesta assistenza utente |
| RispostaAssistenzaServlet.java | Gestisce l’operazione di risposta assistenza utente |
| GestioneProdottoServlet.java | Gestisce le operazioni di inserimento e rimozione prodotto dal catalogo |
| GestioneOffertaServlet.java | Gestisce le operazioni di inserimento e rimozione offerta dal catalogo |

* 1. **Package model**

Il package model è suddiviso a sua volta in altri due: persistence e bean.

* + 1. **Package dao**

Questo package contiene le interfacce che definiscono tutte le operazioni effettuabili sulle entità bean e le jpa che le implementano. All’interno vi sono:

* UtenteDAO.java;
* RichiestaDAO.java;
* CartaDiCreditoDAO.java;
* ProdottoDAO.java;
* VideogiocoDAO.java;
* ConsoleDAO.java;
* CarrelloDAO.java;
* OffertaDAO.java;
* OrdineDAO.java.
* UtenteJPA.java;
* RichiestaJPA.java;
* CartaDiCreditoJPA.java;
* ProdottoJPA.java;
* VideogiocoJPA.java;
* ConsoleJPA.java;
* CarrelloJPA.java;
* OffertaJPA.java;
* OrdineJPA.java.

|  |  |
| --- | --- |
| **Interfaccia** | **Descrizione** |
| UtenteDAO.java | Definisce le operazioni CRUD per l’Utente |
| RichiestaDAO.java | Definisce operazioni di CRD per la Richiesta |
| CartaDiCreditoDAO.java | Definisce operazioni di CRD per la Carta di credito |
| ProdottoDAO.java | Definisce operazioni di CRD per tutti i Prodotti |
| VideogiocoDAO.java | Estende ProdottoDAO e aggiunge nuove operazioni effettuabili su Videogioco |
| ConsoleDAO.java | Estende ProdottoDAO e aggiunge nuove operazioni effettuabili su Console |
| CarrelloDAO.java | Definisce operazioni di CRUD per ogni Carrello |
| OffertaDAO.java | Definisce operazioni di CRD per l’Offerta |
| OrdineDAO.java | Definisce operazioni di CRD per l’Ordine |

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| UtenteJPA.java | Classe che implementa UtenteDAO.java |
| RichiestaJPA.java | Classe che implementa RichiestaDAO.java |
| CartaDiCreditoJPA.java | Classe che implementa CartaDiCreditoDAO.java |
| ProdottoJPA.java | Classe che implementa ProdottoDAO.java |
| VideogiocoJPA.java | Classe che implementa VideogiocoDAO.java |
| ConsoleJPA.java | Classe che implementa ConsoleDAO.java |
| CarrelloJPA.java | Classe che implementa CarrelloDAO.java |
| OffertaJPA.java | Classe che implementa OffertaDAO.java |
| OrdineJPA.java | Classe che implementa OrdineDAO.java |

* + 1. **Package bean**

In questo package sono contenuti tutti gli entity che definiscono gli oggetti di dominio e che vengono usati come bean dalle servlet per gestire i dati e memorizzarli tramite le classi jpa.

* Utente.java;
* Richiesta.java;
* CartaDiCredito.java;
* Prodotto.java;
* Videogioco.java;
* Console.java;
* Carrello.java;
* Offerta.java;
* Ordine.java;
* Validazione.java

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| Utente.java | Classe che rappresenta le informazioni relative all’Utente |
| Richiesta.java | Classe che rappresenta le informazioni relative all’Richieste |
| CartaDiCredito.java | Classe che rappresenta le informazioni relative all’Carta di credito |
| Prodotto.java | Classe che rappresenta le informazioni relative all’Prodotto |
| Videogioco.java | Classe che rappresenta le informazioni relative all’Videogioco |
| Console.java | Classe che rappresenta le informazioni relative all’Console |
| Carrello.java | Classe che rappresenta le informazioni relative all’Carrello |
| Offerta.java | Classe che rappresenta le informazioni relative all’Offerta |
| Ordine.java | Classe che rappresenta le informazioni relative all’Ordine |
| Validazione.java | Classe statica che permette di verificare la correttezza del formato degli oggetti |

* 1. **Package view**

In questo package sono contenute tutte le pagine utilizzate per la visualizzazione dei dati processati dalle servlet e per la logica di presentazione dell’applicazione web.

Quindi avremo:

* **InterfacciaUtente:**
  + Header.jsp;
  + Footer.html;
  + Homepage.jsp;
  + PaginaPersonale.jsp;
  + Carrello.jsp;
  + Assistenza.jsp;
  + Pagamento.jsp;
  + InserimentoCarta.jsp;
  + Prodotto.jsp;
* **InterfacciaGestoreCatalogo:**
  + GestioneProdotto.jsp;
  + GestioneOfferta.jsp;
* **InterfacciaGestoreAccount:**
  + GestioneRuolo.jsp;
* **InterfacciaGestoreAssistenza**:
  + RispostaAssistenza.jsp.

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| Header.jsp | Sezione Header comune a tutte le pagine |
| Footer.html | Sezione Footer comune a tutte le pagine |
| Homepage.jsp | Mostra la pagina iniziale del sito contenente il catalogo (quest’ultimo varia in base al ruolo dell’Utente) |
| PaginaPersonale.jsp | Mostra la pagina con l’anagrafica |
| Carrello.jsp | Mostra la pagina che contiene i prodotti inseriti nel carrello dall’Utente |
| Assistenza.jsp | Mostra la pagina dove l’Utente può compilare un form per l’invio di un’ e-mail di assistenza |
| Pagamento.jsp | Mostra la pagina riassuntiva del pagamento |
| InserimentoCarta.jsp | Mostra il form di inserimento dati di una Carta di credito |
| Prodotto.jsp | Mostra la pagina contenente i dati relativi al prodotto |
| GestioneProdotto.jsp | Mostra la pagina di inserimento e rimozione di un prodotto dal catalogo |
| GestioneOfferta.jsp | Mostra la pagina di inserimento e rimozione di un’offerta dal catalogo |
| GestioneRuolo.jsp | Mostra la pagina di inserimento e rimozione di un ruolo Utente |
| RispostaAssistenza.jsp | Mostra la pagina dove il gestore assistenza può rispondere all’e-mail |