Documentazione Object Design
“GLITCH”

Progetto presentato da:

Annunziata Elefante

Ferdinando Napolitano

Santolo Mutone

Docente:

Prof. Andrea De Lucia

Indice

1. Introduzione

Con l'Object Design Document specifichiamo in modo dettagliato le decisioni prese in fase di analisi e di system 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. Verranno, inoltre, vdefiniti i packages, le class interfaces e i class diagram.

1.1. 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.

1.2. 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, esempio di sito di e-commerce generico, e 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.

1.3. 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.

1.4. Definizioni, acronimi e abbreviazioni

- **ODD:** Object Design Document
- **Componente off-the-shelf:** componenti hardware e software disponibili sul mercato

- **Tab:** tabulatore
- **JPA:** Java Persistence API

2. 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 dell' Application layer, individuati nel SDD.
- Il package **model** contiene al suo interno i *dao* che rappresentano i servizi offerti dal sottosistema DataManager dello Storage layer, individuato nel SDD.
- Il package **view** contiene al suo interno le *view* che rappresentano i servizi offerti dai sottosistemi InterfacciaUtente, InterfacciaGestoreAssistenza, InterfacciaGestoreCatalogo, InterfacciaGestoreAccount del Presentation layer, 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: *dao*, che contiene le operazioni per la gestione dei dati persistenti, e *bean*, che contiene gli oggetti veri e propri.

2.1. Package controller

In questo package sono presenti tutte le **servlet** che implementeranno la logica di business e che con l'utilizzo dei dao, contenuti all'interno del package model.dao, gestiranno i dati contenuti nei bean (nel package model.bean).

- **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:**
    - BaseServlet.java;
    - GestioneProdottoServlet.java;
    - GestioneOffertaServlet.java.

Classe	Descrizione
BaseServlet.java	Gestisce l'inizializzazione della home del sito
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, modifica quantità e rimozione di un prodotto nel carrello
RiepilogoOrdineServlet.java	Gestisce le operazioni per la creazione dell'Ordine
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
VisualizzazioneEmailServlet.java	
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

2.2. Package model

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

2.2.1. Package dao

Questo package contiene le interfacce che definiscono tutte le operazioni effettuabili sulle entità bean e i dao 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;
- UtenteDB.java;
- RichiestaDB.java;
- CartaDiCreditoDB.java;
- ProdottoDB.java;
- VideogiocoDB.java;
- ConsoleDB.java;
- CarrelloDB.java;
- OffertaDB.java;
- OrdineDB.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
UtenteDB.java	Classe che implementa UtenteDAO.java
RichiestaDB.java	Classe che implementa RichiestaDAO.java
CartaDiCreditoDB.java	Classe che implementa CartaDiCreditoDAO.java
ProdottoDB.java	Classe che implementa ProdottoDAO.java
VideogiocoDB.java	Classe che implementa VideogiocoDAO.java
ConsoleDB.java	Classe che implementa ConsoleDAO.java
CarrelloDB.java	Classe che implementa CarrelloDAO.java
OffertaDB.java	Classe che implementa OffertaDAO.java
OrdineDb.java	Classe che implementa OrdineDAO.java

2.2.2. Package bean

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

- Utente.java;
- Richiesta.java;
- CartaDiCredito.java;

- Prodotto.java;
- Videogioco.java;
- Console.java;
- Carrello.java;
- Offerta.java;
- Ordine.java;
- ValidazioneUtente.java;
- ValidazioneRichiesta.java;
- ValidazioneCartaDiCredito.java;
- ValidazioneProdotto.java;
- ValidazioneConsole.java;
- ValidazioneVideogioco.java;
- ValidazioneOfferta.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
ValidazioneUtente.java	Classe statica che permette di verificare la correttezza del formato degli oggetti Utente
ValidazioneRichiesta.java	Classe statica che permette di verificare la correttezza del formato degli oggetti Richiesta

ValidazioneCartaDiCredito.java	Classe statica che permette di verificare la correttezza del formato degli oggetti Carta di credito
ValidazioneProdotto.java	Classe statica che permette di verificare la correttezza del formato degli oggetti Prodotto
ValidazioneConsole.java	Classe statica che estende ValidazioneProdotto e permette di verificare la correttezza del formato degli oggetti Console
ValidazioneVideogioco.java	Classe statica che estende ValidazioneProdotto e permette di verificare la correttezza del formato degli oggetti Videogioco
ValidazioneOfferta.java	Classe statica che permette di verificare la correttezza del formato degli oggetti Offerta

2.3. 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;
  - RichiestaAssistenza.jsp;
  - PaginaPagamento.jsp;
  - PaginaOrdini.jsp;
  - Login.jsp;
  - Registrazione.jsp;
- **InterfacciaGestoreCatalogo:**
  - GestioneProdotti.jsp;
  - GestioneOfferte.jsp;
- **InterfacciaGestoreAccount:**
  - GestioneAccount.jsp;
- **InterfacciaGestoreAssistenza**:
  - GestioneAssistenza.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, i dati modificabili dall'utente e un pulsante per eliminare il proprio profilo
Carrello.jsp	Mostra la pagina che contiene i prodotti inseriti nel carrello dall'Utente
RichiestaAssistenza.jsp	Mostra la pagina dove l'Utente può compilare un form per l'invio di un'e-mail di assistenza
PaginaPagamento.jsp	Mostra la pagina riassuntiva coi prodotti scelti e la modalità di pagamento
PaginaOrdini.jsp	Mostra uno storico di tutti gli acquisti effettuati dall'utente
Login.jsp	Mostra il modal di inserimento delle credenziali per poter accedere al sito
Registrazione.jsp	Mostra il modal di inserimento dei dati personali per potersi registrare al sito
GestioneProdotti.jsp	Mostra la pagina di inserimento e rimozione di un prodotto dal catalogo
GestioneOfferte.jsp	Mostra la pagina di inserimento e rimozione di un'offerta dal catalogo
GestioneAccount.jsp	Mostra la pagina di inserimento e rimozione di un ruolo Utente
GestioneAssistenza.jsp	Mostra la pagina dove il gestore assistenza può rispondere all'e-mail

3. Interfacce delle classi

- **Vincoli servlet**

Nome Classe	RegistrazioneServlet
Descrizione	Gestisce l'operazione di registrazione di un nuovo utente
Pre-condizione	**context:** RegistrazioneServlet :: doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && checkNome(nome : String)==true && checkCognome(cognome: String) ==true && checkEmail(email : email) ==true && checkProvincia(provincia: String)==true && checkCAP(cap: Integer)==true && checkCitta(citta: String)==true && checkStrada(strada: String)==true && checkNumeroCivico(numeroCivico: Integer)==true && checkUsername(username: String)==true && checkPassword(password: String)==true **context:** RegistrazioneServlet :: doPost(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && checkNome(nome : String)==true && checkCognome(cognome: String) ==true && checkEmail(email : email) ==true && checkProvincia(provincia: String)==true && checkCAP(cap: Integer)==true && checkCitta(citta: String)==true && checkStrada(strada: String)==true && checkNumeroCivico(numeroCivico: Integer)==true && checkUsername(username: String)==true && checkPassword(password: String)==true
Post-condizione	
Invarianti	

Nome Classe	LoginServlet
Descrizione	Gestisce l'operazione di accesso di un utente

Pre-condizione	**context** LoginServlet::doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null **context** LoginServlet::doPost(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	
Invarianti	

Nome Classe	**LogoutServlet**
Descrizione	Gestisce l'operazione di logout di un utente
Pre-condizione	**context** LogoutServlet::doGet(request::HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null **context** LogoutServlet::doPost(request::HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	
Invarianti	

Nome Classe	GestioneCarrelloServlet
Descrizione	Gestisce le operazioni di aggiunta e rimozione di un prodotto nel carrello
Pre-condizione	**context** GestioneCarrelloServlet::doGet(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && **context** GestioneCarrelloServlet::doPost(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null &&
Post-condizione	
Invarianti	

Nome Classe	AcquistoServlet
Descrizione	Gestisce l'operazione di acquisto prodotti nel carrello
Pre-condizione	**context** AcquistoServlet::doGet(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && **context** AcquistoServlet::doPost(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null &&
Post-condizione	
Invarianti	

Nome Classe	InserimentoCartaServlet
Descrizione	Gestisce le operazioni di inserimento della carta di credito
Pre-condizione	**context** InserimentoCartaServlet::doGet(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkIntestatario(intestatario : String)==true && checkNumeroIdentificativo(numeroIdentificativo : Integer)==true && checkScadenza(scadenza != Integer)==true checkCVV(cvv : Integer)==true

	context InserimentoCartaServlet::doPost(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkIntestatario(intestatario : String)==true && checkNumeroIdentificativo(numeroIdentificativo : Integer)==true && checkScadenza(scadenza != Integer)==true checkCVV(cvv : Integer)==true
Post-condizione	
Invarianti	

Nome Classe	**GestioneProfiloServlet**
Descrizione	Gestisce le operazioni di modifica e rimozione profilo utente
Pre-condizione	**context** GestioneProfiloServlet::doGet(request::HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkEmail(email : email)==true && checkProvincia(provincia: String)==true && checkCAP(cap: Integer)==true && checkCitta(citta: String)==true && checkStrada(strada: String)==true && checkNumeroCivico(numeroCivico: Integer)==true && checkPassword(password: String)==true **context** GestioneProfiloServlet::doPost(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkEmail(email : email)==true && checkProvincia(provincia: String)==true && checkCAP(cap: Integer)==true && checkCitta(citta: String)==true && checkStrada(strada: String)==true && checkNumeroCivico(numeroCivico: Integer)==true && checkPassword(password: String)==true
Post-condizione	
Invarianti	

Nome Classe	GestioneRuoloServlet
Descrizione	Gestisce le operazioni di modifica e rimozione profilo utente
Pre-condizione	**context** GestioneRuoloServlet::doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Account") **context** GestioneRuoloServlet::doPost(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Account")
Post-condizione	
Invarianti	

Nome Classe	RichiestaAssistenzaServlet
Descrizione	Gestisce l'operazione di richiesta assistenza utente
Pre-condizione	**context** RichiestaAssistenzaServlet::doGet(request::HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkMittente(mittente)==true && checkNomeProblema(nomeProblema)==true && checkDescrizione(descrizione)==true && checkDestinatario(destinatario)==true **context** RichiestaAssistenzaServlet::doPost(request:: HttpServletRequest, response: HttpServletResponse); **pre**: request!=null && response!=null && checkMittente(mittente)==true && checkNomeProblema(nomeProblema)==true && checkDescrizione(descrizione)==true && checkDestinatario(destinatario)==true
Post-condizione	
Invarianti	

Nome Classe	RispostaAssistenzaServlet
Descrizione	Gestisce l'operazione di risposta assistenza utente
Pre-condizione	**context** RispostaAssistenzaServlet::doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Assistenza") && checkMittente(mittente)==true && checkNomeProblema(nomeProblema)==true && checkSoluzione(soluzione)==true && checkDestinatario(destinatario)==true **context** RispostaAssistenzaServlet::doPost(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Assistenza") && checkMittente(mittente)==true && checkNomeProblema(nomeProblema)==true && checkSoluzione(soluzione)==true && checkDestinatario(destinatario)==true
Post-condizione	
Invarianti	

Nome Classe	GestioneProdottoServlet
Descrizione	Gestisce le operazioni di inserimento e rimozione prodotto dal catalogo
Pre-condizione	**context** GestioneProdottoServlet::doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Catalogo") && checkCodice(codice)==true (if addConsole) && checkModello(modello)==true (if addConsole) && checkCasaProduttrice(casaProduttrice)==true (if addConsole) && checkNome(nome)==true (if addVideogioco) && checkGenere(genere)==true (if addVideogioco) && checkPiattaforma(piattaforma)== true (if addVideogioco) &&

	checkImmagine(immagine)==true && checkPrezzo(prezzo)==true && checkDescrizione(descrizione)==true **context** GestioneProdottoServlet::doPost(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Catalogo") && checkCodice(codice)==true (if addConsole) && checkModello(modello)==true (if addConsole) && checkCasaProduttrice(casaProduttrice)==true (if addConsole) && checkNome(nome)==true (if addVideogioco) && checkGenere(genere)==true (if addVideogioco) && checkPiattaforma(piattaforma)== true (if addVideogioco) && checkImmagine(immagine)==true && checkPrezzo(prezzo)==true && checkDescrizione(descrizione)==true
Post-condizione	
Invarianti	

Nome Classe	**GestioneOffertaServlet**
Descrizione	Gestisce le operazioni di inserimento e rimozione offerta dal catalogo
Pre-condizione	**context** GestioneOffertaServlet::doGet(request: HttpServletRequest, response: HttpServletResponse) **pre**: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Catalogo") && checkNome(nome)!= null && checkSconto(percentualeDiSconto)!=null && checkCategoria(categoria) != null && getOffertaCatalogo(categoria) == null **context** GestioneOffertaServlet::doPost(request: HttpServletRequest, response: HttpServletResponse)

	pre: request!=null && response!=null && request.getSession().getAttribute("ruoloUtente"). equals("Gestore_Catalogo") && checkNome(nome)!= null && checkSconto(percentualeDiSconto)!=null && checkCategoria(categoria) != null && getOffertaCatalogo(categoria) == null
Post-condizione	
Invarianti	

- **Vincoli dao**

Nome Classe	**UtenteDAO**
Descrizione	Definisce le operazioni CRUD per l'Utente
Pre-Condizione	**context:** UtenteDAO::createUtente(u: Utente) : void **pre:** u != null **context:** UtenteDAO::deleteUtente(username: String): Utente **pre:** username != null **context:** UtenteDAO::updateUtente(u: Utente): Utente **pre:** u != null **context:** UtenteDAO::retriveByUsername (username: String): Utente **pre:** username != null **context:** UtenteDAO::retriveByEmail (e-mail: String): Utente **pre:** e-mail != null
Post-Condizione	
Invariante	

Nome Classe	RichiestaDAO
Descrizione	Definisce operazioni di CRD per la Richiesta
Pre-Condizione	**context:** RichiestaDAO::createRichiesta(email: Richiesta) : void **pre:** email != null **context:** RichiestaDAO::deleteRichiesta(id: int): Richiesta **pre:** id != null **context:** RichiestaDAO::retriveByMittente (emailMittente: String): List<Richiesta> **pre:** emailMittente != null **context:** RichiestaDAO::retriveById (id: int): Richiesta **pre:** id != null
Post-Condizione	
Invariante	

Nome Classe	CartaDiCreditoDAO
Descrizione	Definisce operazioni di CRD per la Carta di credito
Pre-Condizione	**context:** CartaDiCreditoDAO::createCarta(carta: CartaDiCredito): void **pre:** carta != null **context:** CartaDiCreditoDAO::deleteCarta (numeroCarta: int): CartaDiCredito **pre:** numeroCarta != null **context:** CartaDiCreditoDAO::retriveByUtente

	(username: String): List<CartaDiCredito> **pre:** username != null
Post-Condizione	
Invariante	

Nome Classe	**ProdottoDAO**
Descrizione	Definisce operazioni di CRD per tutti i Prodotti
Pre-Condizione	**context:** ProdottoDAO::createProdotto(prodotto: Prodotto): void **pre:** prodotto != null **context:** ProdottoDAO::deleteProdotto (ID: int): Void **pre:** ID != null **context:** ProdottoDAO::findProdottoById (ID: int): Prodotto **pre:** ID != null
Post-Condizione	
Invariante	

Nome Classe	**VideogiocoDAO**
Descrizione	Estende ProdottoDAO e aggiunge nuove operazioni effettuabili su Videogioco
Pre-Condizione	**context:** VideogiocoDAO::createProdotto (prodotto: Prodotto): void **pre:** prodotto != null **context:** VideogiocoDAO::retriveByNome (nome: String): List<Videogioco> **pre:** nome != null

	context: VideogiocoDAO::retriveByGenere (genere: String): List<Videogioco> **pre:** genere != null **context:** VideogiocoDAO::retriveByPiattaforma (piattaforma: String): List<Videogioco> **pre:** piattaforma != null **context:** VideogiocoDAO::doRetriveVideogiocoAllRange (min : int , max : int): List<Videogioco> **pre:** min != null && max != null
Post-Condizione	
Invariante	

Nome Classe	**ConsoleDAO**
Descrizione	Estende ProdottoDAO e aggiunge nuove operazioni effettuabili su Console
Pre-Condizione	**context:** ConsoleDAO::createProdotto (prodotto: Prodotto): void **pre:** prodotto != null **context:** ConsoleDAO::retriveByModello (modello: String): Prodotto **pre:** modello != null **context:** ConsoleDAO::retriveByCasaProduttrice (casaProduttrice: String): List<Prodotto> **pre:** genere != null **context:** ConsoleDAO::doRetriveVideogiocoAllRange (min : int , max : int): List<Videogioco> **pre:** min != null && max != null
Post-Condizione	

Invariante	

Nome Classe	CarrelloDAO
Descrizione	Definisce operazioni di CRUD per ogni Carrello
Pre-Condizione	**context:** CarrelloDAO::createCarrello (carrello: Carrello): void **pre:** carrello != null **context:** CarrelloDAO::deleteCarrello (username: String): void **pre:** username != null **context:** CarrelloDAO::updateCarrello (cart: Carrello): void **pre:** username != null **context:** CarrelloDAO::retriveByUtente (username: String): Carrello **pre:** username != null
Post-Condizione	
Invariante	

Nome Classe	OffertaDAO
Descrizione	Definisce operazioni di CRD per l'Offerta
Pre-Condizione	**context:** OffertaDAO::createOfferta (offerta: Offerta): void **pre:** offerta != null **context:** OffertaDAO::deleteOfferta(codice: int): Offerta **pre:** codice != null **context:** OffertaDAO::retriveByCategoria (categoria : String): List<Offerta> **pre:** categoria != null

Post-Condizione	
Invariante	

Nome Classe	**OrdineDAO**
Descrizione	Definisce operazioni di CRD per l'Ordine
Pre-Condizione	**context:** OrdineDAO::createOrdine (ordine: Ordine): void **pre:** ordine != null **context:** OrdineDAO::deleteOrdine (ID: int): Ordine **pre:** ID != null **context:** OrdineDAO::retriveByUtente (username: String): List<Ordine> **pre:** username != null
Post-Condizione	
Invariante	

- **Vincoli bean**

Nome Classe	**Utente**
Descrizione	Classe che rappresenta le informazioni relative all'Utente
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Utente **inv**: checkUsername(username) == true && checkPassword(password) == true

Nome Classe	Richiesta
Descrizione	Classe che rappresenta le informazioni relative alle Richieste
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Richiesta **inv**: checkUsername(username) == true && checkPassword(password) == true && (self.stato == false || self.stato == true)

Nome Classe	CaratDiCredito
Descrizione	Classe che rappresenta le informazioni relative alla Carta di credito
Pre-condizione	-
Post-condizione	-
Invarianti	**context** CartaDiCredito **inv**: checkUsername(username) == true && checkPassword(password) == true

Nome Classe	Prodotto
Descrizione	Classe che rappresenta le informazioni relative al Prodotto
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Prodotto **inv**: checkUsername(username) == true && checkPassword(password) == true && (utente.getRuolo = “Gestore catalogo”) == true

Nome Classe	Videogioco
Descrizione	Classe che rappresenta le informazioni relative al Videogioco
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Videogioco **inv**: checkUsername(username) == true && checkPassword(password) == true && (utente.getRuolo = “Gestore catalogo”) == true

Nome Classe	Console
Descrizione	Classe che rappresenta le informazioni relative alla Console
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Console **inv**: checkUsername(username) == true && checkPassword(password) == true && (utente.getRuolo = “Gestore catalogo”) == true

Nome Classe	Carrello
Descrizione	Classe che rappresenta le informazioni relative al Carrello
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Carrello **inv**: checkUsername(username) == true && checkPassword(password) == true

Nome Classe	Offerta
Descrizione	Classe che rappresenta le informazioni relative all'Offerta
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Offerta **inv**: checkUsername(username) == true && checkPassword(password) == true && (utente.getRuolo = "Gestore catalogo") == true

Nome Classe	Ordine
Descrizione	Classe che rappresenta le informazioni relative all'Ordine
Pre-condizione	-
Post-condizione	-
Invarianti	**context** Ordine **inv**: checkUsername(username) == true && checkPassword(password) == true

Nome Classe	ValidazioneUtente
Descrizione	Classe statica che permette di verificare la correttezza del formato degli oggetti Utente
Pre-condizione	**context** ValidazioneUtente :: checkNome(nome : String) : Boolean **pre:** nome != null **context** ValidazioneUtente :: checkCognome(cognome : String) : Boolean **pre**: cognome != null **context** ValidazioneUtente :: checkEmail(e-mail : e-mail) : Boolean

	pre: e-mail != null **context** ValidazioneUtente :: checkProvincia(provincia : String) : Boolean **pre:** provincia != null **context** ValidazioneUtente :: checkCAP(cap : Integer) : Boolean **pre:** cap != null **context** ValidazioneUtente :: checkCittà(città : String) : Boolean **pre:** città != null **context** ValidazioneUtente :: checkNumero(numero : Integer) : Boolean **pre:** numero != null **context** ValidazioneUtente :: checkUsername(username : String) : Boolean **pre:** username != null **context** ValidazioneUtente :: checkPassword(password : String) : Boolean **pre:** password != null
Post-condizione	-
Invarianti	-

Nome Classe	**ValidazioneRichiesta**
Descrizione	Classe statica che permette di verificare la correttezza del formato degli oggetti Richiesta
Pre-condizione	**context** ValidazioneRichiesta :: checkDescrizione(descrizione : String) : Boolean **pre**: descrizione != null
Post-condizione	-
Invarianti	-

Nome Classe	ValidazioneCartaDiCredito
Descrizione	Classe statica che permette di verificare la correttezza del formato degli oggetti Carta di credito
Pre-condizione	**context** ValidazioneCartaDiCredito :: checkNome(nome : String) : Boolean **pre**: nome != null **context** ValidazioneCartaDiCredito :: checkCognome(cognome : String) : Boolean **pre**: cognome != null **context** ValidazioneCartaDiCredito :: checkNumeroIdentificativo(numeroIdentificativo : Integer) : Boolean **pre**: numeroIdentificativo != null **context** ValidazioneCartaDiCredito :: checkScadenza(scadenza : Date) : Boolean **pre**: scadenza != null **context** ValidazioneCartaDiCredito :: checkCVV(cvv : Integer) : Boolean **pre**: cvv != null
Post-condizione	-
Invarianti	-

Nome Classe	ValidazioneProdotto
Descrizione	Classe statica che permette di verificare la correttezza del formato degli oggetti Prodotto
Pre-condizione	**context** ValidazioneProdotto :: checkImmagine(immagine : String) : Boolean **pre**: immagine != null **context** ValidazioneProdotto :: checkPrezzo(prezzo : Real) : Boolean **pre**: prezzo != null

	context ValidazioneProdotto :: checkDescrizione(descrizione : String) : Boolean **pre**: descrizione != null
Post-condizione	-
Invarianti	-

Nome Classe	**ValidazioneConsole**
Descrizione	Classe statica che estende ValidazioneProdotto e permette di verificare la correttezza del formato degli oggetti Console
Pre-condizione	**context** ValidazioneConsole :: checkImmagine(immagine : String) : Boolean **pre**: immagine != null **context** ValidazioneConsole:: checkPrezzo(prezzo : Real) : Boolean **pre**: prezzo != null **context** ValidazioneConsole :: checkDescrizione(descrizione : String) : Boolean **pre**: descrizione != null **context** ValidazioneConsole :: checkDescrizione(descrizione : String) : Boolean **pre**: descrizione != null **context** ValidazioneConsole :: checkModello(modello : String) : Boolean **pre**: modello != null **context** ValidazioneConsole :: checkCasaProduttrice(casaProduttrice : String) : Boolean **pre**: casaProduttrice!= null
Post-condizione	-
Invarianti	-

Nome Classe	**ValidazioneVideogioco**
Descrizione	Classe statica che estende ValidazioneProdotto e permette di verificare la correttezza del formato degli oggetti Videogioco
Pre-condizione	**context** ValidazioneVideogioco :: checkImmagine(immagine : String) : Boolean **pre**: immagine != null **context** ValidazioneVideogioco :: checkPrezzo(prezzo : Real) : Boolean **pre**: prezzo != null **context** ValidazioneVideogioco :: checkDescrizione(descrizione : String) : Boolean **pre**: descrizione != null **context** ValidazioneVideogioco :: checkNome(nome : String) : Boolean **pre**: nome != null **context** ValidazioneVideogioco :: checkGenere(genere : String) : Boolean **pre**: genere != null **context** ValidazioneVideogioco :: checkPiattaforma(piattaforma : String) : Boolean **pre**: piattaforma != null
Post-condizione	-
Invarianti	-
Nome Classe	**ValidazioneOfferta**
Descrizione	Classe statica che permette di verificare la correttezza del formato degli oggetti Offerta
Pre-condizione	**context** ValidazioneOfferta :: checkNome(nome : String) : Boolean **pre**: nome != null **context** ValidazioneOfferta : : checkPercentualeSconto(percentualeSconto : Integer) : Boolean

	pre: percentualeSconto!= null **context** ValidazioneOfferta :: checkCategoria(categoria : String) : Boolean **pre**: categoria != null
Post-condizione	-
Invarianti	-

4. Diagramma delle classi