|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

***SocialBook  
Documento Object Design***

****

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Barbato Alessia | 0512105858 |
| Proietto Angelica | 0512105762 |
| Russo Luca | 0512105840 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 25/01/2021 | 1.0 | Stesura introduzione (Paragrafi 1.1, 1.2, 1.3, 1.4, 1.5) | Proietto Angelica |
| 26/01/2021 | 2.0 | Stesura packages | Proietto Angelica |

Indice

1.Introduzione 4

1.1 Object Design trade-offs 4

1.2 Linee guida per la documentazione delle interfacce 4

1.3 Definizioni, acronimi, abbreviazioni 6

1.4 Design Pattern 6

1.5 Riferimenti 7

2. Packages 7

2.1 Package Controller 7

2.2 Package Model 10

2.3 Package Utility 11

2.4 Package Webapp 11

3. Interfacce delle classi

4. Glossario

1. Introduzione

Dopo aver stilato i documenti Requirements Analysis e System Design è necessario porre attenzione sugli aspetti implementativi. Questo documento ha l’obiettivo di produrre un modello che integri in modo coerente tutte le informazioni collezionate nelle fasi precedenti. In particolar modo, verranno definite le interfacce delle classi, le operazioni supportate, i tipi dei dati, i parametri delle procedure, le signatures dei sottosistemi definiti nel documento di System Design, i trade-offs e le linee guida.

* 1. Object Design trade-offs

***FUNZIONALITA’ vs AFFIDABILITA’***

L’integrazione di molteplici funzionalità potrebbe portare con maggiore facilità a malfunzionamenti di componenti software e quindi comprometterne drasticamente l’affidabilità.

***COMPRENSIBILITA’ vs TEMPO DI SVILUPPO***

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

**TEMPO DI RISPOSTA vs AFFIDABILITA’**

Per garantire che il sistema non si blocchi in caso di picchi di carico elevati, si preferisce rinunciare a un ottimo tempo di risposta. Si è deciso di tollerare questa problematica al fine di offrire agli utenti un sistema per l’appunto affidabile.

* 1. Linee guida per la documentazione delle interfacce

In questa sezione vengono definite le linee guida che ogni sviluppatore rispetterà per la leggibilità del codice. Per la formattazione dei file XML, HTML, CSS e JS si userà il formatter di IntelliJ di default, mentre per i file Java si seguiranno le convenzioni della Sun di Java . Tra cui evidenziamo alcuni aspetti fondamentali:

1.2.1 Java Naming Convention

I nomi delle classi Java devono:

● Essere singolari;

● Essere descrittivi;

● Utilizzare solo caratteri consentiti (a-z, A-Z, 0-9);

● Iniziare con la lettera maiuscola, così come le parole successive all’interno del nome.

Esempio: *NomeClasse.java*

I nomi delle variabili devono:

● Iniziare con la lettera minuscola

- Se sono variabili locali composte da più nomi, le parole successive devono iniziare con la lettera maiuscola;

Esempio: *nomeVariabileLocale*

- Se sono variabili d’istanza (ad esempio, nelle classi bean) composte da più nomi, le parole sono divise dal simbolo \_ (underscore);

Esempio: nome\_variabile\_d\_istanza

● Descrivere il significato della variabile in questione;

● Essere scritti in inglese.

I nomi dei metodi devono:

● Iniziare con la lettera minuscola e le parole successive all’interno del nome con la lettera maiuscola;

Esempio: *nomeMetodo()*

● Essere chiari e descrivere l’azione che il metodo eseguirà;

● Rispettare il pattern getNomeVariabile e setNomeVariabile nel caso in cui si tratti rispettivamente di metodi getter o setter.

Dichiarazioni:

● variabile locale

1) dichiarata e inizializzata nel blocco di codice in cui è necessaria;

2) dichiarata all’esterno di un blocco di codice, nel caso in cui ci sia il bisogno di inizializzarla in un secondo momento e di riutilizzarla anche in blocchi differenti.

● variabile d’istanza

1) dichiarata in un rigo e inizializzata successivamente nel costruttore della classe oppure tramite il metodo setter associato a quella variabile;

2) una sola dichiarazione di una variabile per ogni riga.

1.2.2 HTML Convention

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

● Un’indentazione consiste in una tabulazione;

● Ogni tag deve avere un’indentazione maggior del tag che lo contiene;

● Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

● I tag di commento, se presenti, devono seguire le stesse regole che si applicano ai tag normali.

1.2.3 CSS Convention

I fogli di stile (CSS) devono seguire le seguenti convenzioni:

● Tutti gli stili non inline devono essere collocati in fogli di stile separati;

● Ogni regola CSS deve essere formattata come segue:

1 – I selettori della regola si trovano a livello 0 di indentazione, separati da una virgola;

2 – L’ultimo selettore della riga è seguito da una 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 determinata dalla parentesi graffa chiusa (}), collocata sulla riga successiva all’ultima proprietà elencata.

1.2.4 Database SQL Convention

I nomi delle tabelle devono seguire le seguenti regole:

● Devono essere costituiti di sole lettere;

● Devono iniziare con una lettera minuscola e se si tratta di nomi composti, le parole successive devono iniziare con una lettera maiuscola;

● Devono essere sostantivi singolari tratti dal dominio del problema ed esplicativi del contenuto.

I nomi dei campi devono seguire le seguenti regole:

● Devono essere costituiti di sole lettere minuscole;

● Se si tratta di nomi composti, le parole devono essere separate dal simbolo \_ (underscore).

* 1. Definizioni, acronimi, abbreviazioni

**RAD:** Requirements Analysis Document.

**SDD:** System Design Document.

**JSP:** Java Server Page.

**HTML:** Hyper Text Markup Language.

**CSS:** Cascading Style Sheets.

**SQL:** Structured Query Language.

* 1. Design Pattern

1.4.1 Singleton Pattern

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, così da fornire un punto di accesso globale a tale istanza. Abbiamo progettato una classe Singleton (**ConPool**) per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe.

Per realizzare il singleton pattern occorre avere:

● Una variabile privata statica della classe, nel nostro caso dataSource, che rappresenta l’unica istanza creata;

● Un metodo statico e pubblico getConnection(), che restituisce l’istanza in questione.

Il suo scopo è:

● Avere un accesso controllato all’unica istanza della classe;

● Centralizzare informazioni e comportamenti in un’unica entità condivisa dagli utilizzatori.

Il principale vantaggio è la mutua esclusione.

* 1. Riferimenti

● Object-Oriented Software Engineering Using UML, Patterns, and Java, Bernd Bruegge & Allen H. Dutoit.

● SocialBook RAD.

● SocialBook SDD.

1. Packages

Il sistema SocialBook sarà distribuito in package, rispettando la suddivisione in sottosistemi effettuata in fase di stesura del documento System Design.

Più nello specifico, abbiamo 3 package principali:

● il package “controller”, contenente ulteriori package, uno per ogni sottosistema individuato;

● il package “model”, contenente le risorse per l’accesso e la gestione dei dati persistenti;

● il package “webapp” (view), contenente le risorse utili per l’interazione con l’utente (ad esempio,

le pagine degli stili css, le pagine contenenti le funzioni js, le immagini presenti nel sito ecc.).

Inoltre, è stato introdotto un quarto package “utility”, contenente classi, metodi di supporto e utilizzati in particolare dalle varie servlet per svolgere determinate operazioni.

* 1. Package controller

Il package controller contiene tutte quelle classi che si occupano dell’elaborazione e

processamento dei dati, servendosi delle classi DAO per effettuare le modifiche e soddisfare i comandi dell'utente (ricevuti attraverso il view), occupandosi del reindirizzamento a nuove pagine JSP.

2.1.1 Gestione Utente

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| AddBookBooklistServlet.java | Servlet che gestisce l’inserimento di un nuovo libro a una booklist. |
| AllBooklistServlet.java | Servlet che si occupa del recupero delle booklist (create/seguite) associate ad un certo utente e memorizzate sul database, così da permetterne la visualizzazione. |
| BooklistViewServlet.java | Servlet che si occupa del recupero dei libri presenti in una determinata booklist, memorizzata sul database, così da permetterne la visualizzazione. |
| CustomerEditServlet.java | Servlet che si occupa di effettuare le modifiche alle informazioni personali dell’utente. |
| CustomerServlet.java | Servlet che permette all’utente di visualizzare la propria pagina personale oppure la pagina personale di un altro utente. |
| EditCreaBooklistServlet.java | Servlet che si occupa della creazione/modifica di una booklist. |
| LoginServlet.java | Servlet che gestisce il login di un utente registrato/admin. |
| Logout.java | Servlet che gestisce il logout di un utente registrato/admin che abbia precedentemente effettuato il login. |
| NewCustomerServlet.java | Servlet che si occupa dell’instradamento alla pagina della registrazione. |
| RegistrationServlet.java | Servlet che gestisce la registrazione di un nuovo utente alla piattaforma. |
| ReviewServlet.java | Servlet che gestisce l’inserimento/rimozione di una recensione da parte di un utente a un determinato libro. |
| ScegliBooklistServlet.java | Servlet che reindirizza l’utente verso una pagina in cui si trovano tutte le booklist tra cui può scegliere per effettuare le modifiche. |
| VerifyMain.java | Servlet che controlla il formato e l’unicità dell’email, per verificarne la correttezza. |

2.1.2 Gestione Visualizzazione

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| HomeServlet.java | Servlet che reindirizza l’utente verso l’homepage al momento dell’avvio dell’applicazione. |
| MostraLibriServlet.java | Servlet che reindirizza l’utente verso una pagina in cui si trovano tutti i libri. |
| PaginaLibroServlet.java | Servlet che si occupa del recupero delle informazioni riguardanti un determinato libro, da mostrare all’utente. |
| RicercaServlet.java | Servlet che gestisce la ricerca da parte dell’utente, restituendo risultati che contengono la parola cercata nel titolo, genere o autore. |

2.1.3 Gestione Interazione

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| FollowEditServlet.java | Servlet che gestisce i follow/unfollow di un utente rispetto ad altri utenti. |
| FollowersServlet.java | Servlet che gestisce e permette la visualizzazione delle liste di seguiti/seguaci di ogni utente. |

2.1.4 Gestione Acquisto

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| PaymentServlet.java | Servlet che gestisce e processa le informazioni di pagamento inserite dall’utente e le informazioni sull’ordine, così da effettuarlo. |
| ShowCartServlet.java | Servlet che si occupa dell’inserimento/rimozione di un libro al carrello e della visualizzazione di quest’ultimo. |

2.1.5 Gestione Ticket

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| AllTicketsServlet.java | Servlet che smista i ticket memorizzati, permettendone la visualizzazione in base all’utente che ha effettuato l’accesso e sta effettuando la richiesta in quel momento. |
| NewMessageServlet.java | Servlet che gestisce lo scambio di messaggi tra utente registrato/admin che riceve i ticket di una certa categoria. |
| NewTicketServlet.java | Servlet che si occupa della creazione e successiva visualizzazione (solo nel caso in cui l’utente in questione sia registrato e abbia effettuato l’accesso) del nuovo ticket e di quelli pregressi. |
| TicketViewServlet.java | Servlet che mostra un determinato ticket dando la possibilità agli admin di accettare il ticket o cancellarlo. |

2.1.6 Gestione Customer

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CustomerManagerServlet.java | Servlet che permette al Customer Manager di visualizzare tutti gli utenti registrati e di eliminare gli utenti segnalati dai ticket ricevuti. |
| CustomerManagerReviewServlet.java | Servlet che permette al Customer Manager di eliminare qualsiasi recensione scritta da qualsiasi utente registrato. |

2.1.7 Gestione Libri

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CatalogueManagerInserimentoRimozioneServlet.java | Servlet che permette al Catalogue Manager di effettuare l’inserimento/rimozione di un libro al/dal catalogo. |
| CatalogueManagerCreazioneModificaServlet.java | Servlet che permette al Catalogue Manager di modificare il prezzo di un qualsiasi libro e di creare una nuova istanza di un libro non memorizzato sul database. |

* 1. Package Model

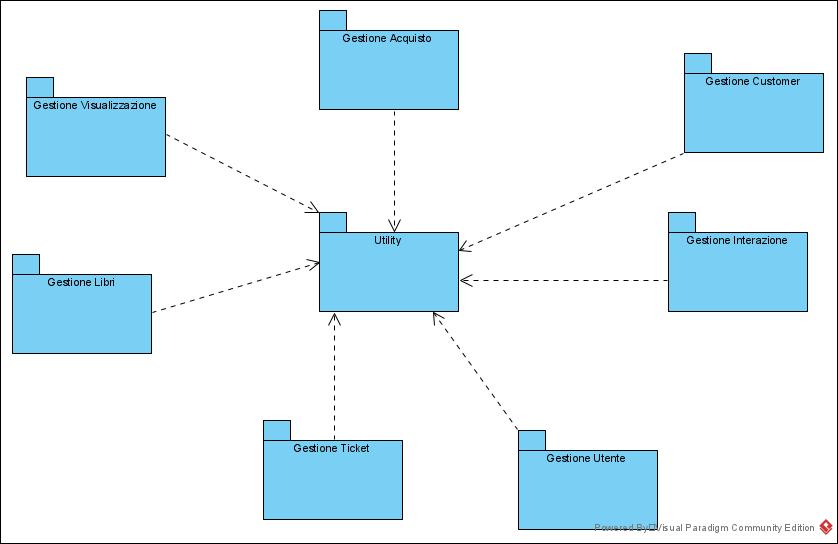
Il package model contiene le classi che verranno utilizzate per interfacciarsi con il DB e le classi che rappresentano i concetti chiave del sito, cioè le entità ognuna con le proprie informazioni.

2.2.1 Gestione Database

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| ConPool.java | Model la cui istanza rappresenta la connessione con il database. |
| Admin.java | Model che rappresenta l’admin e le relative informazioni. |
| AdminDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Admin. |
| Author.java | Model che rappresenta l’autore e le relative informazioni. |
| AuthorDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Author. |
| Book.java | Model che rappresenta il libro e le relative informazioni. |
| BookDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Book. |
| BookList.java | Model che rappresenta la booklist e le relative informazioni. |
| BookListDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità BookList. |
| Cart.java | Model che rappresenta il carrello e le relative informazioni. |
| CartDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Cart. |
| Customer.java | Model che rappresenta il customer (utente registrato) e le relative informazioni. |
| CustomerDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Customer. |
| Follow.java | Model che rappresenta la relazione di follow che c’è tra gli utenti registrati. |
| FollowDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Customer. |
| InfoPayment.java | Model che rappresenta il metodo di pagamento e le relative informazioni. |
| InfoPaymentDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità InfoPayment. |
| Message.java | Model che rappresenta il messaggio e le relative informazioni. |
| MessageDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Message. |
| Order.java | Model che rappresenta l’ordine e le relative informazioni. |
| OrderDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Order. |
| OrderDetailDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Order. |
| Review.java | Model che rappresenta la recensione e le relative informazioni. |
| ReviewDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Review. |
| Ticket.java | Model che rappresenta il ticket e le relative informazioni. |
| TicketDAO.java | Model che permette di eseguire le operazioni sul DB riferite all’entità Ticket. |

* 1. Package Utility

Il package Utility contiene risorse sfruttabili da più componenti, favorendone così il riutilizzo e facilitando i vari controlli.



* 1. Package webapp

Il package webapp contiene tutte le pagine JSP, ovvero le pagine dinamiche con cui l’utente può interagire per utilizzare tutte le funzionalità del sistema.

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| AllBooklistsView.jsp | Pagina che permette all’utente registrato di |
| AllOrdersView.jsp | Pagina che permette all’utente registrato di visualizzare la lista dei propri ordini pregressi. |
| AllTicketsView.jsp | Pagina che permette all’utente registrato di visualizzare i ticket creati, mentre mostra all’admin la lista dei ticket disponibili e dei ticket in carico. |
| BooklistEditCreate.jsp | Pagina che permette all’utente registrato di creare una nuova booklist oppure di modificarne una già esistente. |
| BooklistView.jsp | Pagina che permette di visualizzare le informazioni di una booklist (il nome e l’immagine) e i libri di cui la booklist è composta. |
| CartView.jsp | Pagina che mostra all’utente i libri attualmente presenti nel proprio carrello. |
| Crea\_modifica\_libro.jsp | Pagina che permette al Catalogue Manager di memorizzare un nuovo libro oppure di modificare il prezzo di un libro esistente. |
| CustomerEdit.jsp | Pagina che permette all’utente registrato di modificare alcune delle proprie informazioni personali, quali l’immagine, la descrizione e la password. |
| Followers.jsp | Pagina che mostra la lista dei seguiti/seguaci di un utente registrato. |
| Header.jsp | Pagina che viene inclusa in tutte le altre pagine e che rappresenta il menu orizzontale navigazionale. |
| Index.jsp | Pagina che rappresenta l’homepage del sito ed è la prima pagina che l’utente vede all’avvio del sistema. |
| Libri.jsp | Pagina che mostra al Catalogue Manager la lista di tutti i libri, che siano essi presenti o meno nel catalogo. |
| NewTicket.jsp | Pagina che consente a un utente (registrato o meno) di inviare un ticket a un admin (Customer Manager oppure System Manager). |
| Pagina\_libro.jsp | Pagina che mostra il libro e tutte le sue informazioni, consentendo all’utente registrato di svolgere delle operazioni su di esso (ad esempio, aggiungerlo al carrello oppure ad una booklist, recensirlo …) |
| Payment\_info.jsp | Pagina che permette a un utente registrato di effettuare un ordine, inserendo il metodo di pagamento che si vuole utilizzare. |
| Registration.jsp | Pagina che permette ad un utente di poter effettuare la registrazione alla piattaforma, inserendo dati corretti (che rispettino il formato e i vincoli). |
| Ricerca.jsp | Pagina che mostra i risultati della ricerca effettuata oppure un messaggio di errore se non è stato trovato nessun libro. |
| ScegliBooklist.jsp | Pagina che mostra all’utente tutte le booklist di sua proprietà, a cui può aggiungere un libro. |
|  | Pagina che rappresenta l’homepage del sito ed è la prima pagina che l’utente vede all’avvio del sistema. |
| Utenti.jsp | Pagina che mostra al Customer Manager la lista di tutti gli utenti e le loro informazioni personali, ognuno dei quali può essere poi rimosso. |
| MessageTicket.jsp | Pagina che permette all’utente registrato/admin di scambiarsi messaggi nel ticket. |
| Exception.jsp/ Error.jsp | Pagine che mostrano messaggi di errore quando si verificano delle eccezioni (ad esempio, l’utente che prova a compiere un’azione per cui non è autorizzato). |

1. Class Interface
   1. Class Interface Controller – Gestione Acquisto

|  |
| --- |
| **Nome Classe** |
| AllOrdersViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| **context** AllOrdersViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null |
| Post-condizione |
| **context** AllOrdersViewServlet::  doGet(request:HttpServletRequest, response:HttpServletResponse)  **post:**request.setAttribute("orders") && req.setAttribute("products") |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| **context** PaymentServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre :** request.getParameter("id").equals("elimina") |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do  Get(request:HttpServletRequest,response:HttpServletResponse)  **post :**  request.getSession().removeAttribute("info") |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| +doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do Get(request:HttpServletRequest,response:HttpServletResponse)  **pre :** request.getParameter("id") = "procedi" |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do  Get(request:HttpServletRequest,response:HttpServletResponse) |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| +doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do Get(request:HttpServletRequest,response:HttpServletResponse)  **pre :** request.getParameter("id") = "pagamento" |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do  Get(request:HttpServletRequest,response:HttpServletResponse)  **post :**  request.getSession().setAttribute("info") |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| +doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do Post(request:HttpServletRequest,response:HttpServletResponse)  **pre :** request.getParameter("pay”) = "Procedi al pagamento" |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** PaymentServlet:: do  Post(request:HttpServletRequest,response:HttpServletResponse) |

|  |
| --- |
| **Nome Classe** |
| ShowCatServlet |
| Attributi |
| - |
| Metodi |
| +doGet(HttpServletRequest request, HttpServletResponse response) : void  +doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** ShowCartServlet:: do Get(request:HttpServletRequest,response:HttpServletResponse)  **pre :** [request.getParameter("customer”) != null && req.getParameter(“addCart”) != null ] ||  [request.getParameter("customer”) != null && req.getParameter(“addCart”) == null && req.getParameter(“delete”) != null ]  doPost(HttpServletRequest request, HttpServletResponse  response):  **context** ShowCartServlet:: do Post(request:HttpServletRequest,response:HttpServletResponse) |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse  response):  **context** ShowCartServlet:: do  Get(request:HttpServletRequest,response:HttpServletResponse)  **post:** req.getSession().setAttribute("cart", cart);  doPost(HttpServletRequest request, HttpServletResponse  response):  **context** ShowCartServlet:: do  Post(request:HttpServletRequest,response:HttpServletResponse) |

1. Glossario

**SDD (System Design Document):** Documento formalizzato alla definizione di obiettivi di progettazione del sistema, decomposizione del sistema in sottosistemi più piccoli e scelta di architettura software più adatta al sistema.

**RAD (Requirement Analysis Document):** documento contenente informazioni inerenti al sistema da realizzare raccolte durante la fase di Requirement Analysis e Requirement Elicitation.

**Servlet:** oggetti Java all'interno del server web che permettono di creare web applications in combinazione con JSP.

**JSP:** tecnologia di programmazione web utilizzata per fornire contenuti dinamici.

**HTML:** [linguaggio di markup](https://it.wikipedia.org/wiki/Linguaggio_di_markup) nato per la [formattazione](https://it.wikipedia.org/wiki/Formattazione_del_testo) e [impaginazione](https://it.wikipedia.org/wiki/Impaginazione) di documenti ipertestuali utilizzato principalmente per il disaccoppiamento della struttura logica di una [pagina web](https://it.wikipedia.org/wiki/Pagina_web) (definita appunto dal markup) e la sua rappresentazione.

**CSS:** usato separare i contenuti di documenti HTML dalla loro [formattazione](https://it.wikipedia.org/wiki/Formattazione_del_testo), permettendo una programmazione più chiara e facile da utilizzare e garantendo il riutilizzo di codice e una più facile manutenzione.

**SQL:** linguaggio standardizzato per database basati sul modello relazionale (RDBMS).

**Piattaforma:** Definisce l’insieme delle funzionalità fornite dal sistema attraverso l’applicazione web.

**Utente:** Un utilizzatore della piattaforma che non si è ancora registrato.

**Utente registrato:** Un utilizzatore iscritto alla piattaforma.

**Customer manager:** Amministratore che si occupa della gestione degli utenti registrati.

**System manager:** Amministratore che si occupa della gestione di bug, segnalati da utenti che utilizzano la piattaforma.

**Catalogue manager:** Amministratore che si occupa della gestione del catalogo libri.

**Admin:** Generalizzazione di customer manager, catalogue manager e system manager.