|  |
| --- |
| **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 |
| 28/01/2021 | 3.0 | Stesura Class Interface Controller | Barbato Alessia |
| 29/01/2021 | 4.0 | Stesura Class Interface Model | Proietto Angelica |
| 30/01/2021 | 5.0 | Realizzazione Class Diagram e stesura glossario | Russo Luca |

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 13

3.1 Controller - Gestione Acquisto 13

3.2 Controller - Gestione Customer 16

3.3 Controller - Gestione Interazione 17

3.4 Controller - Gestione Libri 19

3.5 Controller - Gestione Ticket 20

3.6 Controller - Gestione Utente 24

3.7 Controller - Gestione Visualizzazione 35

3.8 Model 48

4. Class Diagram 37

5. Glossario 54

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 e metodi di supporto, utilizzati in particolare dalle varie servlet per svolgere alcune 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

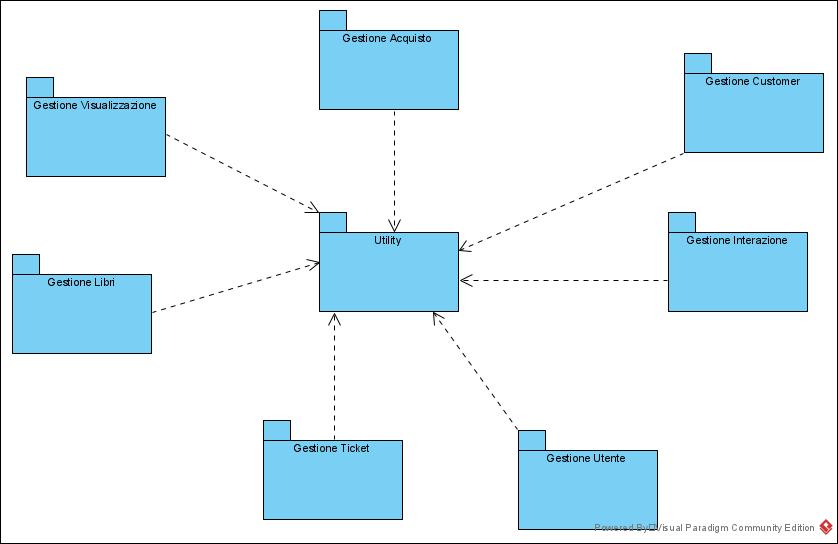
Il package model contiene le classi che verranno utilizzate per interfacciarsi con il DB e le classi che rappresentano i concetti chiave del sito, cioè le entità ognuna con le proprie informazioni.

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 |
| doGet(HttpServletRequest request, HttpServletResponseresponse):  **context** AllOrdersViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllOrdersViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute("orders", orders)  && request.setAttribute("products", orderDetail) |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null  && request.getParameter(“id”) == “elimina” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doGet(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::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null  && request.getParameter("id”) == “procedi" |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**modifica il carrello esistente e aggiorna gli ordini dell’utente |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null  &&request.getParameter("id”) == “pagamento” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.getSession().setAttribute("info", info) |

|  |
| --- |
| **Nome Classe** |
| PaymentServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null  &&request.getParameter("pay”) == "Procedi al pagamento" |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** PaymentServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **post:**memorizza le informazioni sul metodo di pagamento dell’utente e aggiorna il suo carrello |

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

* 1. Class Interface Controller – Gestione Customer

|  |
| --- |
| **Nome Classe** |
| CustomerManagerReviewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerManagerReviewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("customerManager") != null  && request.getParameter(“r\_id”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerManagerReviewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**elimina la recensione selezionata dal database |

|  |
| --- |
| **Nome Classe** |
| CustomerManagerServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerManagerServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("customerManager") != null  && request.getParameter(“id”) == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerManagerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute(“customers”, customers) |

* 1. Class Interface Controller – Gestione Interazione

|  |
| --- |
| **Nome Classe** |
| FollowBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowBooklistServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null  && [request.getParameter(“follow”) != null || request.getParameter(“unfollow”) != null] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowBooklistServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**viene effettuato il follow/unfollow di una booklist da parte dell’utente loggato |

|  |
| --- |
| **Nome Classe** |
| FollowEditServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowEditServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null  && [request.getParameter("follow") != null || request.getParameter("unFollow") != null || request.getParameter("editProfile") != null] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowEditServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**viene effettuato il follow/unfollow di un utente da parte dell’utente loggato oppure l’utente loggato viene rediretto alla pagina della modifica del profilo |

|  |
| --- |
| **Nome Classe** |
| FollowersServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowersServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getParameter("Following") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowersServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute("Following", “Following”)  && request.setAttribute("follows", follows) |

|  |
| --- |
| **Nome Classe** |
| FollowersServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowersServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getParameter("Followers") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** FollowersServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute("Followers", “Followers”)  && request.setAttribute("follows", follows) |

* 1. Class Interface Controller – Gestione Libri

|  |
| --- |
| **Nome Classe** |
| CatalogueManagerCreazioneModificaServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerCreazioneModificaServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("catalogueManager") != null  && [request.getParameter(“isbn\_modifica”) != “”  || request.getParameter(“isbn\_modifica”) == “”] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerCreazioneModificaServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**viene aggiornato il prezzo di un libro già presente nel database oppure viene memorizzata un’istanza di un nuovo libro |

|  |
| --- |
| **Nome Classe** |
| CatalogueManagerInserimentoRimozioneServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerInserimentoRimozioneServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("catalogueManager") != null  && request.getParameter(“operazione”) == “modifica” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerInserimentoRimozioneServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:** request.setAttribute("operazione", “modifica”)  && request.setAttribute("book", book) && request.setAttribute("authors", authors) |

|  |
| --- |
| **Nome Classe** |
| CatalogueManagerInserimentoRimozioneServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerInserimentoRimozioneServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("catalogueManager") != null  && request.getParameter(“operazione”) != “modifica” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CatalogueManagerInserimentoRimozioneServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:** request.setAttribute("operazione", “creazione”) |

* 1. Class Interface Controller – Gestione Ticket

|  |
| --- |
| **Nome Classe** |
| AllTicketsServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllTicketsServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("personalCustomer") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllTicketsServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“tickets”, tickets) |

|  |
| --- |
| **Nome Classe** |
| AllTicketsServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllTicketsServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute("customerManager") != null  || request.getSession().getAttribute(“systemManager”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllTicketsServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“ticketsR”, ticketsByAdminRole) &&request.setAttribute(“tickets”, tickets) |

|  |
| --- |
| **Nome Classe** |
| NewMessageServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewMessageServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute(“customerManager”) != null  || request.getSession().getAttribute(“systemManager”) != null |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewMessageServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“ticketsR”, ticketsR)  &&session.removeAttribute("ticket") && request.setAttribute("tickets", tickets) |

|  |
| --- |
| **Nome Classe** |
| NewMessageServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewMessageServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewMessageServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **post:**session.removeAttribute("ticket") && request.setAttribute("tickets", tickets) |

|  |
| --- |
| **Nome Classe** |
| NewTicketServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewTicketServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null  &&request.getSession().getAttribute("customerManager") == null && request.getSession().getAttribute("systemManager") == null |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** NewTicketServlet::  doPost(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“tickets”, tickets) |

|  |
| --- |
| **Nome Classe** |
| TicketViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request.getParameter("name") == null &&  [request.getSession().getAttribute("personalCustomer") != null  || request.getSession().getAttribute("customerManager") != null  || request.getSession().getAttribute("systemManager") != null] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“messages”, messages) |

|  |
| --- |
| **Nome Classe** |
| TicketViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request.getParameter("name") == “accept” &&  [request.getSession().getAttribute("customerManager") != null  || request.getSession().getAttribute("systemManager") != null] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“ticket”, ticket) && request.setAttribute(“ticketR”, ticketR) |

|  |
| --- |
| **Nome Classe** |
| TicketViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request.getParameter("name") == “delete” &&  [request.getSession().getAttribute("customerManager") != null  || request.getSession().getAttribute("systemManager") != null  || request.getSession().getAttribute(“personalCustomer”) != null] |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** TicketViewServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**viene eliminato dal database il ticket con id = request.getParameter(“id”) |

* 1. Class Interface Controller – Gestione Utente

|  |
| --- |
| **Nome Classe** |
| AddBookBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AddBookBooklistServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request.getSession().getAttribute("personalCustomer") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AddBookBooklistServlet::  doGet(request: HttpServletRequest, response: HttpServletResponse)  **post:**request.setAttribute(“book”, book) |

|  |
| --- |
| **Nome Classe** |
| AllBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllBooklistServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getParameter("id") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** AllBooklistServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("idCustomer", idCustomer)  && request.setAttribute("booklists", personalBooklists)  && request.setAttribute("followed", followedBooklists) |

|  |
| --- |
| **Nome Classe** |
| BooklistViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** BooklistViewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null  && request.getParameter(“idCustomer”) == customer.getId() |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** BooklistViewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("booklist", booklist)  && request.setAttribute("books", books)  && request.setAttribute(“idCustomer”, idCustomer) |

|  |
| --- |
| **Nome Classe** |
| BooklistViewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** BooklistViewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null  && checkFollower(request.getParameter(“id”), customer.getId()) == true |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** BooklistViewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("booklist", booklist)  && request.setAttribute("books", books)  && request.setAttribute(“follow”, true) |

|  |
| --- |
| **Nome Classe** |
| CustomerEditServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** CustomerEditServlet::  doPost(request:HttpServletRequest,response:HttpServletResponse)  **pre:r**equest.getSession().getAttribute("personalCustomer") != null |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** CustomerEditServlet::  doPost(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute(“idCustomer”, idCustomer) |

|  |
| --- |
| **Nome Classe** |
| CustomerServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getParameter("customerView") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("customer", customer)  && request.setAttribute("preferiti", preferiti) |

|  |
| --- |
| **Nome Classe** |
| CustomerServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getParameter("customerView") != null  && request.getSession().getAttribute(“personalCustomer”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("customer", customer)  && request.setAttribute("preferiti", preferiti)  && request.setAttribute(“idCustomer”, idCustomer) |

|  |
| --- |
| **Nome Classe** |
| CustomerServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getParameter("personalView") != null  && request.getSession().getAttribute(“personalCustomer”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** CustomerServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute("idCustomer", idCustomer)  && request.setAttribute("preferiti", preferiti) |

|  |
| --- |
| **Nome Classe** |
| EditCreaBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getParameter("personalCustomer") == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**message("HEYY, devi fare l'accesso prima!!") |

|  |
| --- |
| **Nome Classe** |
| EditDeleteCreaBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditDeleteCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getParameter("personalCustomer") != null  && request.getParameter(“delete”) == null  && request.getParameter(“addPreferiti”) == null  && request.getParameter(“edit”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditDeleteCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute(“operazione”, “edit”)  && request.setAttribute(“books”, books)  && request.setAttribute(“booklist”, booklist) |

|  |
| --- |
| **Nome Classe** |
| EditDeleteCreaBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditDeleteCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getParameter("personalCustomer") != null  && request.getParameter(“delete”) == null  && request.getParameter(“addPreferiti”) == null  && request. |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** EditDeleteCreaBooklistServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**request.setAttribute(“operazione”, “Create”) |

|  |
| --- |
| **Nome Classe** |
| LoginServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** LoginServlet::  doPost(request:HttpServletRequest,response:HttpServletResponse)  **pre**:request.getSession().getAttribute(“personalCustomer”) == null  && request.getSession().getAttribute(“customerManager”) == null  && request.getAttribute(“catalogueManager”) == null  && request.getAttribute(“systemManager”) == null |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** LoginServlet::  doPost(request:HttpServletRequest,response:HttpServletResponse)  **post:**[request.getSession().setAttribute("personalCustomer", customer)  && request.getSession().setAttribute("cart", cart)]  || request.getSession().setAttribute(“customerManager”)  || request.getSession().setAttribute(“catalogueManager”)  || request.getSession().setAttribute(“systemManager”)  || message("Le credenziali inserite non sono valide!!") |

|  |
| --- |
| **Nome Classe** |
| LogoutServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession.getAttribute("personalCustomer") == null && request.getSession.getAttribute("customerManager") == null && request.getSession.getAttribute("catalogueManager") == null && request.getSession.getAttribute("systemManager") == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**message("Bisogna aver effettuato l’accesso!!") |

|  |
| --- |
| **Nome Classe** |
| LogoutServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession.getAttribute("personalCustomer") != null && request.getSession.getAttribute("customerManager") == null && request.getSession.getAttribute("catalogueManager") == null && request.getSession.getAttribute("systemManager") == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**session.removeAttribute(“personalCustomer”) |

|  |
| --- |
| **Nome Classe** |
| LogoutServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession.getAttribute("personalCustomer") == null && request.getSession.getAttribute("customerManager") != null && request.getSession.getAttribute("catalogueManager") == null && request.getSession.getAttribute("systemManager") == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**session.removeAttribute(“customerManager”) |

|  |
| --- |
| **Nome Classe** |
| LogoutServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession.getAttribute("personalCustomer") == null && request.getSession.getAttribute("customerManager") == null && request.getSession.getAttribute("catalogueManager") != null && request.getSession.getAttribute("systemManager") == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**session.removeAttribute(“catalogueManager”) |

|  |
| --- |
| **Nome Classe** |
| LogoutServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession.getAttribute("personalCustomer") == null && request.getSession.getAttribute("customerManager") == null && request.getSession.getAttribute("catalogueManager") == null && request.getSession.getAttribute("systemManager") != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** LogoutServlet:: doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**session.removeAttribute(“systemManager”) |

|  |
| --- |
| **Nome Classe** |
| RegistrationServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** RegistrationServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse)  **pre:**  request.getParamter(“name”).matches("([a-zA-Z]{1,15})") == false || request.getParameter(“surname”).matches("([a-zA-Z]{1,15})") == false || request.getParameter(“username”).matches("([a-zA-Z0-9]{1,15})") == false || request.getParameter("email").matches("^\\w+([\\.-]?\\w+)\*@\\w+([\\.-]?\\w+)\*(\\.\\w{2,3})+$") == false ||  request.getParameter("password").matches("^(?=.\*[A**-**Za**-**z])(?=.\*\\d)[A**-**Za**-**z\\d]{8,}$") == false ||  request.getParameter(“description”).matches("[a-zA-Z0-9- !?:.,()]{0,150}") == false |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** RegistrationServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse)  **post:**stampa un messaggio di errore sul formato |

|  |
| --- |
| **Nome Classe** |
| ReviewServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** ReviewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getSession().getAttribute(“personalCustomer”) != null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** ReviewServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** request.setAttribute("book", book)  && request.setAttribute("recensioni", recensioni)  && request.setAttribute("customers", customers) |

|  |
| --- |
| **Nome Classe** |
| ScegliBooklistServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** ScegliBooklistServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:** request.getSession().getAttribute(“personalCustomer”) == null |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** ScegliBooklistServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:**  request.setAttribute("isbn", isbn)  && request.setAttribute("booklists", booklists) |

|  |
| --- |
| **Nome Classe** |
| VerifyMail |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** VerifyMail::  doGet(request:HttpServletRequest,response:HttpServletResponse)  - |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** ScegliBooklistServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** response.setContentType("text/xml") |

* 1. Class Interface Controller – Gestione Visualizzazione

|  |
| --- |
| **Nome Classe** |
| HomeServlet |
| Attributi |
| - |
| Metodi |
| # doPost(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** HomeServlet::  doPost(request:HttpServletRequest,response:HttpServletResponse)  - |
| Post-condizione |
| doPost(HttpServletRequest request, HttpServletResponse response):  **context** HomeServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse)  **post:** request.getSession().setAttribute("utentiHome", utentiHome)  && request.getSession().setAttribute("libriHome", libriHome) |

|  |
| --- |
| **Nome Classe** |
| MostraLibriServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** MostraLibriServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  - |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** MostraLibriServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** request.setAttribute("books", books) |

|  |
| --- |
| **Nome Classe** |
| PaginaLibroServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaginaLibroServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.setAttribute("recensioni", recensioni) && request.setAttribute("book", book) && request.setAttribute("customers", customers) |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** PaginaLibroServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** request.setAttribute("books", books) |

|  |
| --- |
| **Nome Classe** |
| RicercaServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** RicercaServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getAttribute("query") == “” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** RicercaServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** message("Non hai inserito nessuna parola!!") |

|  |
| --- |
| **Nome Classe** |
| RicercaServlet |
| Attributi |
| - |
| Metodi |
| # doGet(HttpServletRequest request, HttpServletResponse response) : void |
| Pre-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** RicercaServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **pre:**request.getAttribute("query") != “” |
| Post-condizione |
| doGet(HttpServletRequest request, HttpServletResponse response):  **context** RicercaServlet::  doGet(request:HttpServletRequest,response:HttpServletResponse)  **post:** request.setAttribute("parolaCercata", parolaCercata)  && request.setAttribute("books", books) |

* 1. Class Interface Model

|  |
| --- |
| **Nome Classe** |
| AdminDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveByUsrEPwd(String username, String password) : Admin |
| Pre-condizione |
| doRetrieveByUsrEPwd(String username, String password):  **context** AdminDAO:: doRetrieveByUsrEPwd(username: String, password: String)  **pre:**username != null && password != null |
| Post-condizione |
| doRetrieveByUsrEPwd(String username, String password):  **context** AdminDAO:: doRetrieveByUsrEPwd(username: String, password: String)  **post:**restituisce null se non esiste nessun admin nel database con le credenziali passate, altrimenti restituisce l’admin in questione |

|  |
| --- |
| **Nome Classe** |
| AuthorDAO |
| Attributi |
| - |
| Metodi |
| + doSave(ArrayList<Author> authors, String isbn) : void  + doRetrieveAuthorsByIsbn(String isbn) : ArrayList<Author>  + doRetrieveIdAuthorLike (String like, int offset, int limit) : ArrayList<Integer> |
| Pre-condizione |
| doSave(ArrayList<Author> authors, String isbn):  **context** AuthorDAO:: doSave(authors: ArrayList<Author>, isbn: String)  **pre:**authors != null && authors.size() > 0 && isbn != null  doRetrieveAuthorsByIsbn(String isbn):  **context** AuthorDAO:: doRetrieveAuthorsByIsbn(isbn: String)  **pre:**isbn != null  doRetrieveIdAuthorLike (String like, int offset, int limit):  **context** AuthorDAO:: doRetrieveIdAuthorLike (like: String, offset: int, limit: int)  **pre:**like != null && offset > 0 && limit > 0 |
| Post-condizione |
| doSave(ArrayList<Author> authors, String isbn):  **context** AuthorDAO:: doSave(authors: ArrayList<Author>, isbn: String)  **post:**sul database vengono memorizzati tutti gli autori che si trovano in authors, riferiti al libro con l’isbn passato  doRetrieveAuthorsByIsbn(String isbn):  **context** AuthorDAO:: doRetrieveAuthorsByIsbn(isbn: String)  **post:** restituisce la lista di autori del libro con l’isbn passato  doRetrieveIdAuthorLike (String like, int offset, int limit):  **context** AuthorDAO:: doRetrieveIdAuthorLike(like: String, offset: int, limit: int)  **post:** restituisce una lista di (offset – limit) interi, in cui ogni intero rappresenta l’id di un autore il cui nome o cognome contiene la stringa like |

|  |
| --- |
| **Nome Classe** |
| BookDAO |
| Attributi |
| - |
| Metodi |
| + doUpdateCatalogue(String isbn) : void  + doUpdatePrice(String isbn, int price) : void  + doSave(Book book) : void  + doRetrieveAll() : ArrayList<Book>  + doRetrieveByIsbn(String isbn) : Book  + doRetrieveByTitleOrGenre(String like, int offset, int limit) : ArrayList<Book>  + doRetrieveByIdAuthor(int id) : ArrayList<Book>  - createBook(ResultSet resultSet) : Book  - doRetrieveCatalogueByIsbn(String isbn) : boolean |
| Pre-condizione |
| doUpdateCatalogue(String isbn):  **context** BookDAO:: doUpdateCatalogue(isbn: String)  **pre:**isbn != null  doUpdatePrice(String isbn, int price):  **context** BookDAO:: doUpdatePrice(isbn: String, price: int)  **pre:**isbn != null && price > 0  doSave(Book book):  **context** BookDAO:: doSave(book: Book)  **pre:**book != null  doRetrieveAll():  **context** BookDAO:: doRetrieveAll()  -  doRetrieveByIsbn(String isbn):  **context** BookDAO:: doRetrieveByIsbn(isbn: String)  **pre:**isbn != null  doRetrieveByTitleOrGenre(String like, int offset, int limit):  **context** BookDAO:: doRetrieveByTitleOrGenre(like: String, offset: int, limit: int)  **pre:**like != null && offset > 0 && limit > 0  doRetrieveByIdAuthor(int id):  **context** BookDAO:: doRetrieveByIdAuthor(id: int)  **pre:**id > 0  createBook(ResultSet resultSet):  **context** BookDAO:: createBook(resultSet: ResultSet)  **pre:**resultSet != null  doRetrieveCatalogueByIsbn(String isbn):  **context** BookDAO:: doRetrieveCatalogueByIsbn(isbn: String)  **pre:**isbn != null |
| Post-condizione |
| doUpdateCatalogue(String isbn):  **context** BookDAO:: doUpdateCatalogue(isbn: String)  **post:**viene aggiornato il campo catalogue del libro con l’isbn passato  (impostato a true se è false, a false se è true) se il libro è memorizzato nel database, altrimenti viene lanciata un’eccezione  doUpdatePrice(String isbn, int price):  **context** BookDAO:: doUpdatePrice(isbn: String, price: int)  **post:** viene aggiornato il libro con l’isbn passato, modificando il campo price\_cent con price, se il libro è memorizzato nel database, altrimenti viene lanciata un’eccezione  doSave(Book book):  **context** BookDAO:: doSave(book: Book)  **post:**sul database viene memorizzato il libro passato se non esiste già, altrimenti viene lanciata un’eccezione  doRetrieveAll()  **context** BookDAO:: doRetrieveAll()  **post:**restituisce la lista di tutti i libri memorizzati sul database  doRetrieveByIsbn(String isbn):  **context** BookDAO:: doRetrieveByIsbn(isbn: String)  **post:**restituisce null se non esiste sul database nessun libro con l’isbn passato,  altrimenti restituisce il libro in questione  doRetrieveByTitleOrGenre(String like, int offset, int limit):  **context** BookDAO:: doRetrieveByTitleOrGenre(like: String, offset: int, limit: int)  **post:** restituisce una lista di (offset – limit) libri, in cui ogni libro contiene la stringa like nel titolo o nel genere o in entrambi  doRetrieveByIdAuthor(int id):  **context** BookDAO:: doRetrieveByIdAuthor(id: int)  **post:**restituisce la lista di libri dell’autore con chiave id se l’autore è memorizzato nel database, altrimenti lancia un’eccezione  doRetrieveCatalogueByIsbn(String isbn):  **context** BookDAO:: doRetrieveCatalogueByIsbn(isbn: String)  **post:** restituisce il campo catalogue (true/false) del libro con l’isbn passato se il libro è memorizzato nel database, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| BookListDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveBooklist(int idBooklist) : BookList  + doRetrieveFollowed(int idCustomer) : ArrayList<BookList>  + doRetrieveFromCustomer(int idCustomer) : ArrayList<BookList>  + doFollow(int idCustomer, int idBooklist) : void  + doUnfollow(int idCustomer, int idBooklist) : void  + checkFollower(int idCustomer, int idBooklist) : boolean  + doDelete(int idBooklist) : void  + doRetrieveBooks(int idBooklist) : ArrayList<Book>  + doSave(BookList booklist, int idCustomer) : void  + doSaveBook(int idBooklist, String isbn) : void  + doUpdate(BookList booklist) : void  + doRetrieveFavourite(int idCustomer) : BookList  + addFavourite(int idCustomer, String isbn) : void |
| Pre-condizione |
| doRetrieveBooklist(int idBooklist):  **context** BookListDAO:: doRetrieveBooklist(idBooklist: int)  **pre:**idBooklist > 0  doRetrieveFollowed(int idCustomer):  **context** BookListDAO:: doRetrieveFollowed(idCustomer: int)  **pre:**idCustomer > 0  doRetrieveFromCustomer(int idCustomer):  **context** BookListDAO:: doRetrieveFromCustomer(idCustomer: int)  **pre:**idCustomer > 0  doFollow(int idCustomer, int idBooklist):  **context** BookListDAO:: doFollow(idCustomer: int, idBooklist: int)  **pre:**idCustomer > 0 &&idBooklist > 0  checkFollower(int idCustomer, int idBooklist):  **context** BookListDAO:: checkFollower(idCustomer: int, idBooklist: int)  **pre:**idCustomer > 0 &&idBooklist > 0  doUnfollow(int idCustomer, int idBooklist):  **context** BookListDAO:: doUnfollow(idCustomer: int, idBooklist: int)  **pre:**idCustomer > 0 &&idBooklist > 0 && checkFollower(idCustomer, idBooklist) == true  doDelete(int idBooklist):  **context** BookListDAO:: doDelete(idBooklist: int)  **pre:**idBooklist > 0  doRetrieveBooks(int idBooklist):  **context** BookListDAO:: doRetrieveBooks(idBooklist: int)  **pre:**idBooklist > 0  doSave(BookList booklist, int idCustomer):  **context** BookListDAO:: doSave(booklist: BookList, idCustomer: int)  **pre:**booklist != null &&idCustomer > 0  doSaveBook(int idBooklist, String isbn):  **context** BookListDAO:: doSaveBook(idBooklist: int, isbn: String)  **pre:**idBooklist > 0 && isbn != null  doUpdate(BookList booklist):  **context** BookListDAO:: doUpdate(booklist: BookList)  **pre:**booklist != null  doRetrieveFavourite(int idCustomer)  **context** BookListDAO:: doRetrieveFavourite(idCustomer: int)  **pre:**idCustomer > 0  addFavourite(int idCustomer, String isbn)  **context** BookListDAO:: addFavourite(idCustomer: int, isbn: String)  **pre:**idBooklist > 0 |
| Post-condizione |
| doRetrieveBooklist(int idBooklist):  **context** BookListDAO:: doRetrieveBooklist(idBooklist: int)  **post:** restituisce null se non esiste nessuna booklist nel database con l’id passato, altrimenti restituisce la booklist in questione  doRetrieveFollowed(int idCustomer):  **context** BookListDAO:: doRetrieveFollowed(idCustomer: int)  **post:**restituisce la lista di booklist seguite dal customer con chiave idCustomer se il customer è memorizzato nel database, altrimenti lancia un’eccezione  doRetrieveFromCustomer(int idCustomer):  **context** BookListDAO:: doRetrieveFromCustomer(idCustomer: int)  **post:**restituisce la lista di booklist create dal customer con chiave idCustomerse il customer è memorizzato nel database, altrimenti lancia un’eccezione  doFollow(int idCustomer, int idBooklist):  **context** BookListDAO:: doFollow(idCustomer: int, idBooklist: int)  **post:**viene memorizzato nel database il follow da parte del customer con chiave idCustomer alla booklist con chiave idBooklist  checkFollower(int idCustomer, int idBooklist):  **context** BookListDAO:: checkFollower(idCustomer: int, idBooklist: int)  **post:**restituisce true se il customer con chiave idCustomer segue la booklist con chiave idBooklist, altrimenti restituisce false  doUnfollow(int idCustomer, int idBooklist):  **context** BookListDAO:: doUnfollow(idCustomer: int, idBooklist: int)  **post:**viene memorizzato nel database l’unfollow da parte del customer con chiave idCustomer alla booklist con chiave idBooklist  doDelete(int idBooklist):  **context** BookListDAO:: doDelete(idBooklist: int)  **post:**elimina la booklist con chiave idBooklist se è già memorizzata nel database, altrimenti lancia un’eccezione  doRetrieveBooks(int idBooklist):  **context** BookListDAO:: doRetrieveBooks(idBooklist: int)  **post:**restituisce la lista di libri presenti nella booklist con chiave idBooklist se la booklist è memorizzata nel database, altrimenti lancia un’eccezione  doSave(BookList booklist, int idCustomer):  **context** BookListDAO:: doSave(booklist: BookList, idCustomer: int)  **post:**memorizza nel database la booklist associandola al customer con chiave idCustomer se il customer è presente nel database, altrimenti lancia un’eccezione  doSaveBook(int idBooklist, String isbn):  **context** BookListDAO:: doSaveBook(idBooklist: int, isbn: String)  **post:**aggiorna la booklist con chiave idBooklist aggiungendo il libro con l’isbn passato, se entrambi sono presenti nel database altrimenti lancia un’eccezione  doUpdate(BookList booklist):  **context** BookListDAO:: doUpdate(booklist: BookList)  **post:**aggiorna il nome e l’immagine della booklist se è presente nel database, altrimenti lancia un’eccezione  doRetrieveFavourite(int idCustomer)  **context** BookListDAO:: doRetrieveFavourite(idCustomer: int)  **post:**restituisce la lista dei preferiti (booklist) del customer con chiave idCustomer se il customer è presente nel database altrimenti lancia un’eccezione  addFavourite(int idCustomer, String isbn)  **context** BookListDAO:: addFavourite(idCustomer: int, isbn: String)  **post:**aggiorna la lista dei preferiti (booklist) del customer con chiave idCustomer aggiungendo il libro con l’isbn passato, se entrambi sono presenti nel database, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| CartDAO |
| Attributi |
| - |
| Metodi |
| + doSave(Cart cart, int idCustomer) : void  + doUpdateCustomerCart(Cart cart) : void  + doSaveBookCart(int idCart, String isbn) : void  + doDeleteFromCart(int idCart, String isbn) : void  + doRetrieveByCustomer(int idCustomer) : Optional<Cart> |
| Pre-condizione |
| doSave(Cart cart, int idCustomer):  **context** CartDAO:: doSave(cart: Cart, idCustomer: int)  **pre:**cart != null && idCustomer > 0  doUpdateCustomerCart(Cart cart):  **context** CartDAO:: doUpdateCustomerCart(cart: Cart)  **pre:**cart != null  doSaveBookCart(int idCart, String isbn)  **context** CartDAO:: doSaveBookCart(idCart: int, isbn: String)  **pre:**idCart > 0 && isbn != null  doDeleteFromCart(int idCart, String isbn)  **context** CartDAO:: doDeleteFromCart(idCart: int, isbn: String)  **pre:**idCart > 0 && isbn != null  doRetrieveByCustomer(int idCustomer)  **context** CartDAO:: doRetrieveByCustomer(idCustomer: int)  **pre:**idCustomer > 0 |
| Post-condizione |
| doSave(Cart cart, int idCustomer):  **context** CartDAO:: doSave(cart: Cart, idCustomer: int)  **post:**memorizza il carrello cart, associato al customer con chiave idCustomer, se quest’utlimo è presente nel database, altrimenti lancia un’eccezione  doUpdateCustomerCart(Cart cart):  **context** CartDAO:: doUpdateCustomerCart(cart: Cart)  **post:**aggiorna il prezzo totale del carrello cart se è presente nel database, altrimenti lancia un’eccezione  doSaveBookCart(int idCart, String isbn)  **context** CartDAO:: doSaveBookCart(idCart: int, isbn: String)  **post:**aggiorna il carrello con chiave idCart aggiungendo il libro con l’isbn passato se entrambi sono presenti nel database, altrimenti lancia un’eccezione  doDeleteFromCart(int idCart, String isbn)  **context** CartDAO:: doDeleteFromCart(idCart: int, isbn: String)  **post:**aggiorna il carrello con chiave idCart eliminando il libro con l’isbn passato se entrambi sono presenti nel database e se il libro è già presente nel carrello,  altrimenti lancia un’eccezione  doRetrieveByCustomer(int idCustomer)  **context** CartDAO:: doRetrieveByCustomer(idCustomer: int)  **post:**restituisce il carrello associato al customer con chiave idCustomer se il customer è presente nel database, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| CustomerDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveById(int id) : Customer  + doRetrieveAll() : ArrayList<Customer>  + doRetrieveByEmail(String email) : Customer  + doSave(Customer customer) : void  + validate(String username, String password) : boolean  + doRetrieveByUsername(String username) : Customer  + doUpdate(Customer customer) : void  + doDeleteById(int id) : void  + doRetrieveByReviews(String isbn) : ArrayList<Customer> |
| Pre-condizione |
| doRetrieveById(int id):  **context** CustomerDAO:: doRetrieveById(id: int)  **pre:**id > 0  doRetrieveAll():  **context** CustomerDAO:: doRetrieveAll()  -  doRetrieveByEmail(String email):  **context** CustomerDAO:: doRetrieveByEmail(email: String)  **pre:**email != null  doSave(Customer customer):  **context** CustomerDAO:: doSave(customer: Customer)  **pre:**customer != null  validate(String username, String password):  **context** CustomerDAO:: validate(username: String, password: String)  **pre:**username != null && password != null  doRetrieveByUsername(String username):  **context** CustomerDAO:: doRetrieveByUsername(username: String)  **pre:**username != null  doUpdate(Customer customer):  **context** CustomerDAO:: doUpdate(customer: Customer)  **pre:**customer != null  doDeleteById(int id):  **context** CustomerDAO:: doDeleteById(id: int)  **pre:**id != null  doRetrieveByReviews(String isbn):  **context** CustomerDAO:: doRetrieveByReviews(isbn: String)  **pre:**isbn != null |
| Post-condizione |
| doRetrieveById(int id):  **context** CustomerDAO:: doRetrieveById(id: int)  **post:** restituisce null se non esiste nessun customer nel database con chiave id, altrimenti restituisce il customer in questione  doRetrieveAll():  **context** CustomerDAO:: doRetrieveAll()  **post:** restituisce tutti i customer memorizzati nel database  doRetrieveByEmail(String email):  **context** CustomerDAO:: doRetrieveByEmail(email: String)  **post:**restituisce null se non esiste nessun customer nel database con l’email passata, altrimenti restituisce il customer in questione  doSave(Customer customer):  **context** CustomerDAO:: doSave(customer: Customer)  **post:**memorizza il customer nel database  validate(String username, String password):  **context** CustomerDAO:: validate(username: String, password: String)  **pre:**restituisce false se non esiste nessun customer nel database con l’username passato oppure se la password è errata, altrimenti restituisce true  doRetrieveByUsername(String username):  **context** CustomerDAO:: doRetrieveByUsername(username: String)  **post:**restituisce null se non esiste nessun customer nel database con l’username passato, altrimenti restituisce il customer in questione  doUpdate(Customer customer):  **context** CustomerDAO:: doUpdate(customer: Customer)  **post:** aggiorna la password, la descrizione e l’immagine del customer se è presente nel database, altrimenti lancia un’eccezione  doDeleteById(int id):  **context** CustomerDAO:: doDeleteById(id: int)  **post:**elimina dal database il customer con chiave id se è già memorizzato, altrimenti lancia un’eccezione  doRetrieveByReviews(String isbn):  **context** CustomerDAO:: doRetrieveByReviews(isbn: String)  **post:**restituisce la lista di customer che hanno recensito il libro con l’isbn passato se quest’ultimo è presente nel database, altrimenti restituisce null |

|  |
| --- |
| **Nome Classe** |
| FollowDAO |
| Attributi |
| - |
| Metodi |
| + doFollow(int idCustomer, int idFollower) : void  + doRetrieveAllFollowers(int idCustomer) : ArrayList<Follow>  + doRetrieveAllFollowed(int idCustomer) : ArrayList<Follow>  + checkFollower(int idCustomer, int idFollower) : boolean  + doDelete(int idCustomer, int idFollower) : void |
| Pre-condizione |
| doFollow(int idCustomer, int idFollower):  **context** FollowDAO:: doFollow(idCustomer: int, idFollower: int)  **pre:**idCustomer > 0 && idFollower > 0 && idCustomer != idFollower  doRetrieveAllFollowers(int idCustomer):  **context** FollowDAO:: doRetrieveAllFollowers(idCustomer: int)  **pre:**idCustomer > 0  doRetrieveAllFollowed(int idCustomer):  **context** FollowDAO:: doRetrieveAllFollowed(idCustomer: int)  **pre:**idCustomer > 0  checkFollower(int idCustomer, int idFollower):  **context** FollowDAO:: checkFollower(idCustomer: int, idFollower: int)  **pre:**idCustomer > 0 && idFollower > 0 && idCustomer != idFollower  doDelete(int idCustomer, int idFollower):  **context** FollowDAO:: doDelete(doDelete(int idCustomer, int idFollower)  **pre:**idCustomer > 0 && idFollower > 0 && idCustomer != idFollower |
| Post-condizione |
| doFollow(int idCustomer, int idFollower):  **context** FollowDAO:: doFollow(idCustomer: int, idFollower: int)  **post:**memorizza nel database il follow da parte del customer con chiave idCustomer nei confronti del customer con chiave idFollower se entrambi i customer sono memorizzati nel database e non si seguono già, altrimenti lancia un’eccezione  doRetrieveAllFollowers(int idCustomer):  **context** FollowDAO:: doRetrieveAllFollowers(idCustomer: int)  **post:**restituisce la lista dei followers del customer con chiave idCustomer se il customer è presente nel database, altrimenti lancia un’eccezione  doRetrieveAllFollowed(int idCustomer):  **context** FollowDAO:: doRetrieveAllFollowed(idCustomer: int)  **post:**restituisce la lista dei following del customer con chiave idCustomer se il customer è presente nel database, altrimenti lancia un’eccezione  checkFollower(int idCustomer, int idFollower):  **context** FollowDAO:: checkFollower(idCustomer: int, idFollower: int)  **post:**restituisce true se il customer con chiave idCustomer segue il customer con chiave idFollower, altrimenti restituisce false  doDelete(int idCustomer, int idFollower):  **context** FollowDAO:: doDelete(doDelete(int idCustomer, int idFollower)  **post:**elimina dal database il follow da parte del customer con chiave idCustomer nei confronti del customer con chiave idFollower se entrambi i customer sono memorizzati nel database e si seguono già, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| InfoPaymentDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveByCustomer(int idCustomer) : Optional<InfoPayment>  + doDeleteById(int idCustomer) : void  + doSave(InfoPayment infoPayment) : void |
| Pre-condizione |
| doRetrieveByCustomer(int idCustomer):  **context** InfoPaymentDAO:: doRetrieveByCustomer(idCustomer: int)  **pre:**idCustomer > 0  doDeleteById(int idCustomer):  **context** InfoPaymentDAO:: doDeleteById(idCustomer: int)  **pre:**idCustomer > 0  doSave(InfoPayment infoPayment):  **context** InfoPaymentDAO:: doSave(infoPayment: InfoPayment)  **pre:**infoPayment != null |
| Post-condizione |
| doRetrieveByCustomer(int idCustomer):  **context** InfoPaymentDAO:: doRetrieveByCustomer(idCustomer: int)  **post:**restituisce un oggetto di tipo InfoPayment se per il customer con chiave idCustomer esiste, altrimenti restituisce un oggetto vuoto  doDeleteById(int idCustomer):  **context** InfoPaymentDAO:: doDeleteById(idCustomer: int)  **post:**elimina l’InfoPayment associato al customer con chiave idCustomer se esiste, altrimenti lancia un’eccezione  doSave(InfoPayment infoPayment):  **context** InfoPaymentDAO:: doSave(infoPayment: InfoPayment)  **post:**memorizza l’InfoPayment associato al customer con chiave idCustomer se non esiste già, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| MessageDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveByTicket(int idTicket) : ArrayList<Message>  + doSave(Message message) : void |
| Pre-condizione |
| doRetrieveByTicket(int idTicket):  **context** MessageDAO:: doRetrieveByTicket(idTicket: int)  **pre:**idTicket > 0  doSave(Message message):  **context** MessageDAO:: doSave(message: Message)  **pre:**message != null |
| Post-condizione |
| doRetrieveByTicket(int idTicket):  **context** MessageDAO:: doRetrieveByTicket(idTicket: int)  **post:**restituisce la lista di messaggi associata al ticket con chiave idTicket se quest’ultimo è presente sul database, altrimenti restituisce null  doSave(Message message):  **context** MessageDAO:: doSave(message: Message)  **post:**memorizza message nel database se non è già presente, altrimenti lancia un’eccezione |

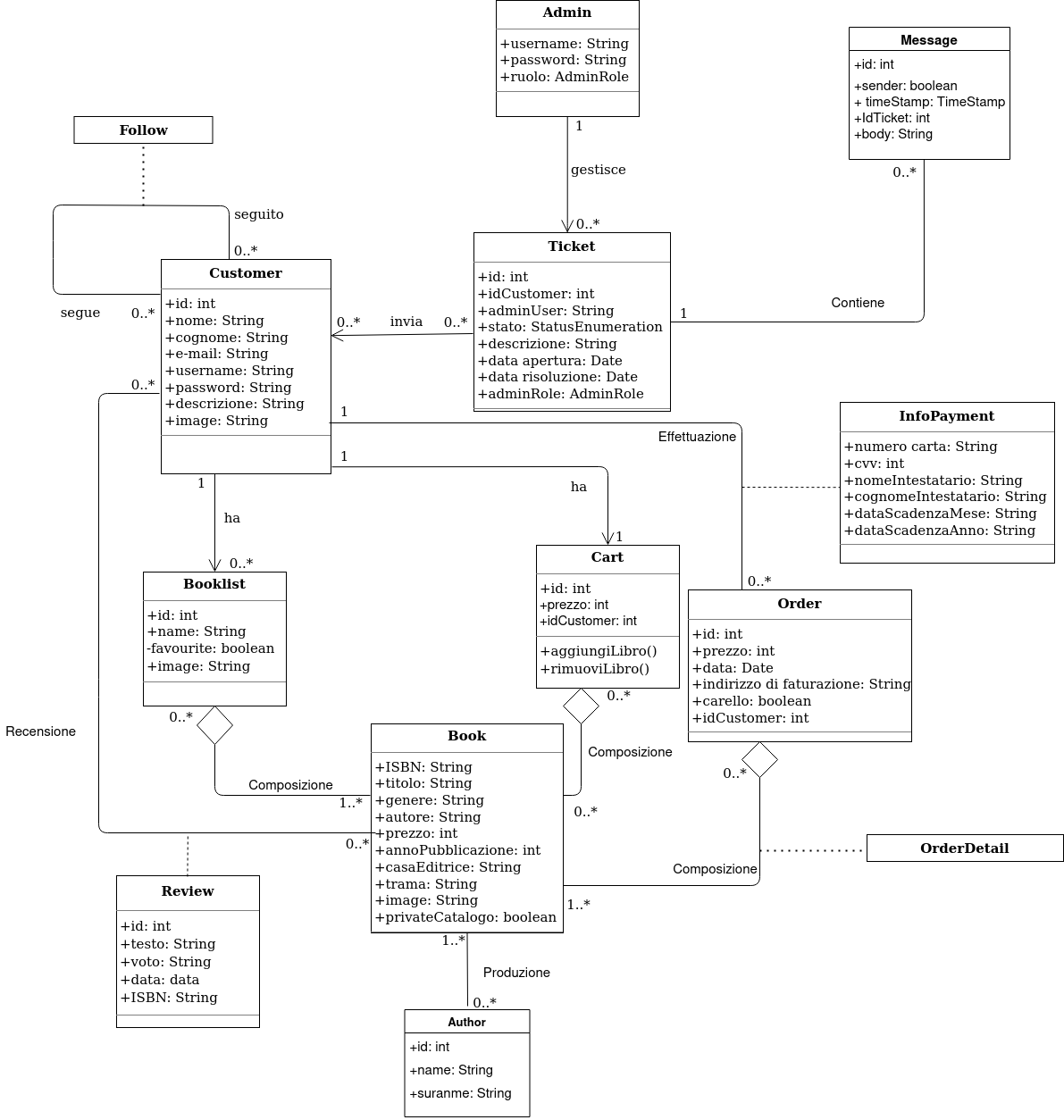
|  |
| --- |
| **Nome Classe** |
| OrderDAO |
| Attributi |
| - |
| Metodi |
| + doUpdate(Order order) : void  + doRetrieveByCustomer(int idCustomer) : ArrayList<Order>  + doRetrieveByCart(int idCustomer) : Order |
| Pre-condizione |
| doUpdate(Order order):  **context** OrderDAO:: doUpdate(order: Order)  **pre:**order != null  doRetrieveByCustomer(int idCustomer):  **context** OrderDAO:: doRetrieveByCustomer(idCustomer: int)  **pre:**idCustomer > 0  doRetrieveByCart(int idCustomer):  **context** OrderDAO:: doRetrieveByCart(idCustomer: int)  **pre:**idCustomer > 0 |
| Post-condizione |
| doUpdate(Order order):  **context** OrderDAO:: doUpdate(order: Order)  **post:** aggiorna l’ordine order se è presente nel database, altrimenti lancia un’eccezione  doRetrieveByCustomer(int idCustomer):  **context** OrderDAO:: doRetrieveByCustomer(idCustomer: int)  **post:**restituisce la lista degli ordini effettuati dal customer con chiave idCustomer se quest’ultimo è presente nel database, altrimenti lancia un’eccezione  doRetrieveByCart(int idCustomer):  **context** OrderDAO:: doRetrieveByCart(idCustomer: int)  **post:**restituisce il carrello del customer con chiave idCustomer se quest’ultimo è presente nel database, altrimenti lancia un’eccezione |

|  |
| --- |
| **Nome Classe** |
| OrderDetailDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveByCustomer(int idCustomer) : List<OrderDetail>  + doRetrieveByOrder(int idOrder) : ArrayList<Book> |
| Pre-condizione |
| doRetrieveByCustomer(int idCustomer)  **context** OrderDetailDAO:: doRetrieveByCustomer(idCustomer: int)  **pre:**idCustomer > 0  doRetrieveByOrder(int idOrder)  **context** OrderDetailDAO:: doRetrieveByOrder(idOrder: int)  **pre:**idOrder > 0 |
| Post-condizione |
| doRetrieveByCustomer(int idCustomer)  **context** OrderDetailDAO:: doRetrieveByCustomer(idCustomer: int)  **post:**restituisce la lista di OrderDetail associata al customer con chiave idCustomer, in cui OrderDetail contiene l’id dell’ordine e una struttura dati (hashmap) in cui l’isbn del libro è la chiave e il titolo è il valore  doRetrieveByOrder(int idOrder)  **context** OrderDetailDAO:: doRetrieveByOrder(idOrder: int)  **post:**restituisce la lista di libri dell’ordine con chiave idOrder |

|  |
| --- |
| **Nome Classe** |
| ReviewDAO |
| Attributi |
| - |
| Metodi |
| + doSave(Review review) : void  + doDeleteById(int idReview) : void  + doRetrieveByISBN(String isbn) : ArrayList<Review>  + doRetrieveByISBNCustomer(String isbn, int idCustomer) : Review  + doUpdateById(Review review) : void  - createReview(ResultSet resultSet) : Review |
| Pre-condizione |
| doSave(Review review):  **context** ReviewDAO:: doSave(review: Review)  **pre:**review != null  doDeleteById(int idReview):  **context** ReviewDAO:: doDeleteById(idReview: int)  **pre:**idReview > 0  doRetrieveByISBN(String isbn):  **context** ReviewDAO:: doRetrieveByISBN()  **pre:**isbn != null  doRetrieveByISBNCustomer(String isbn, int idCustomer):  **context** ReviewDAO:: doRetrieveByISBNCustomer(isbn: String, idCustomer: int)  **pre:**isbn != null && idCustomer > 0  doUpdateById(Review review):  **context** ReviewDAO:: doUpdateById(review: Review)  **pre:**review != null  createReview(ResultSet resultSet):  **context** ReviewDAO:: createReview(resultSet: ResultSet)  **pre:**resultSet != null |
| Post-condizione |
| doSave(Review review):  **context** ReviewDAO:: doSave(review: Review)  **post:** memorizza review nel database se non è già presente, altrimenti lancia un’eccezione  doDeleteById(int idReview):  **context** ReviewDAO:: doDeleteById(idReview: int)  **post:**elimina la recensione review con chiave idReview se esiste, altrimenti lancia un’eccezione  doRetrieveByISBN(String isbn):  **context** ReviewDAO:: doRetrieveByISBN()  **post:**restituisce la lista di review associata al libro con l’isbn passato se quest’ultimo è presente nel database, altrimenti lancia un’eccezione  doRetrieveByISBNCustomer(String isbn, int idCustomer):  **context** ReviewDAO:: doRetrieveByISBNCustomer(isbn: String, idCustomer: int)  **post:**restituisce la review effettuata dal customer con chiave idCustomer in riferimento al libro con l’isbn passato se esiste, altrimenti restituisce null  doUpdateById(Review review):  **context** ReviewDAO:: doUpdateById(review: Review)  **post:**aggiorna la recensione review se è presente nel database, altrimenti lancia un’eccezione  createReview(ResultSet resultSet):  **context** ReviewDAO:: createReview(resultSet: ResultSet)  **post:**restituisce un oggetto di tipo Review partendo dai dati presi dal ResultSet |

|  |
| --- |
| **Nome Classe** |
| TicketDAO |
| Attributi |
| - |
| Metodi |
| + doRetrieveByAdmin(String username) : ArrayList<Ticket>  + doRetrieveByRole(AdminRole role) : ArrayList<Ticket>  + doRetrieveByCustomer(int idCustomer) : ArrayList<Ticket>  + doRetrieveById(int idTicket) : Ticket  + doDeleteById(int idTicket) : void  + doSave(Ticket ticket) : void  + doUpdate(Ticket ticket) : void |
| Pre-condizione |
| doRetrieveByAdmin(String username):  **context** TicketDAO:: doRetrieveByAdmin(username: String)  **pre:**username != null  doRetrieveByRole(AdminRole role):  **context** TicketDAO:: doRetrieveByRole(role: AdminRole)  **pre:**role.equals(“SYSTEM\_MANAGER”) || role.equals(“CUSTOMER\_MANAGER”) ||  role.equals(“CATALOGUE\_MANAGER”)  doRetrieveByCustomer(int idCustomer):  **context** TicketDAO:: doRetrieveByCustomer(idCustomer: int)  **pre:**idCustomer > 0  doRetrieveById(int idTicket):  **context** TicketDAO:: doRetrieveById(int idTicket)  **pre:**idTicket > 0  doDeleteById(int idTicket):  **context** TicketDAO:: doDeleteById(int idTicket  **pre:**idTicket > 0  doSave(Ticket ticket):  **context** TicketDAO:: doSave(Ticket ticket)  **pre:**ticket != null  doUpdate(Ticket ticket):  **context** TicketDAO:: doUpdate(Ticket ticket)  **pre:**ticket != null |
| Post-condizione |
| doRetrieveByAdmin(String username):  **context** TicketDAO:: doRetrieveByAdmin(username: String)  **post:**restituisce la lista di ticket riferita all’admin con l’username passato  doRetrieveByRole(AdminRole role):  **context** TicketDAO:: doRetrieveByRole(role: AdminRole)  **post:**restituisce la lista di ticket in base al ruolo dell’admin  doRetrieveByCustomer(int idCustomer):  **context** TicketDAO:: doRetrieveByCustomer(idCustomer: int)  **post:**restituisce la lista di ticket creati dal customer con chiave idCustomer se quest’ultimo è presente nel database, altrimenti lancia un’eccezione  doRetrieveById(int idTicket):  **context** TicketDAO:: doRetrieveById(int idTicket)  **post:**restituisce il ticket con chiave idTicket se quest’ultimo è presente nel database, altrimenti restituisce null  doSave(Ticket ticket):  **context** TicketDAO:: doSave(Ticket ticket)  **post:**memorizza ticket nel database se non è già presente,  altrimenti lancia un’eccezione  doUpdate(Ticket ticket):  **context** TicketDAO:: doUpdate(Ticket ticket)  **post:**aggiorna ticket se è presente nel database, altrimenti lancia un’eccezione |

1. Class Diagram



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.