



Laurea Triennale in Informatica-Università di Salerno
Corso di *Ingegneria del Software*

OBJECT DESIGN DOCUMENT

“Peek A Book”



Versione	1.0
Data	03/07/2024
Presentato da	Iacomino Domenico, De Luca Ciro



Revision History

Data	Versione	Descrizione	Autori
29/06/2024	0.1	Prima Stesura	Iacomino Domenico, De Luca Ciro
02/07/2024	0.2	Seconda stesura	Iacomino Domenico, De Luca Ciro
03/07/2024	1.0	Terza stesura	Iacomino Domenico, De Luca Ciro

Indice

1.	Introduction	3
1.1.	Object design goals.....	3
1.2.	Object design trade-offs	3
1.3.	Tecnologie utilizzate.....	4
2.	Packages.....	5
	Package Autenticazione	6
	Package Carrello	7
	Package Catalogo.....	8
	Package Registrazione	9
	Package Storage	10
	Package Gestione_Admin	11
3.	Class Interfaces	12
	ArticoloDAO	12
	AutoreDAO	17
	CarrelloDAO	19
	LibroDAO	20
	OrdineDAO	23
	UtenteDAO	26
4.	Class Diagram	31
5.	Design Pattern	35



1. Introduction

Il gruppo Peek A Book intende sviluppare una applicazione web che mira ad essere il punto di riferimento per l'acquisto di libri in Italia ponendo l'accento sulla tempestività dell'approvvigionamento dei prodotti e sulla vicinanza al fruitore.

1.1. Object design goals

Robustezza: Il sistema deve essere in grado di gestire input imprevisti attraverso validazione, controllo degli errori e gestione delle eccezioni lanciate.

Modificabilità ed Estensibilità: La suddivisione in package deve garantire la modificabilità del sistema e la possibilità di aggiungere ulteriori funzionalità nelle fasi avanzate del rilascio.

1.2. Object design trade-offs

Come già specificato nella sezione Design goals nell'SDD, il sistema pone l'accento sulla modularità delle funzionalità da implementare (secondo la scala di priorità dei requisiti proposta all'interno del RAD), sulla responsività del sito web e sul controllo delle form inviate dall'utente (con particolare attenzione a quelle compilate dagli utenti non amministratori).

Al fine di adempiere a tali propositi e di intercettare i requisiti non funzionali, sono state effettuate le seguenti scelte:

- Salvataggio dell'utente nella sessione HTTP una volta loggato, per accelerare l'accesso ai dati personali. Ne risulta che la sessione HTTP deve prevedere un meccanismo di sincronizzazione.
- Gestione del carrello utente nella sessione HTTP una volta effettuato il login, in modo tale da aggiornare il carrello presente nel database soltanto una volta invalidata la sessione in seguito ad un logout. Ne risulta che la sessione HTTP deve prevedere un meccanismo di sincronizzazione.
- La validazione delle form viene effettuata frontend attraverso degli script in modo da escludere le form invalide dal flusso di richieste verso il server.
- Le immagini dei prodotti non potranno superare una grandezza massima di 1300x1600. Le immagini non potranno avere una risoluzione migliore al fine di diminuire l'occupazione di memoria secondaria del server e il tempo di caricamento delle pagine web.



1.3. Tecnologie utilizzate

Il sistema sarà sviluppato utilizzando il linguaggio di programmazione Java con l'implementazione OpenJDK-17. Le classi che intercetteranno le richieste utente estenderanno la gerarchia `HttpServlet` e si occuperanno di gestire il flusso di controllo del sistema inviando pagine web coerenti come risposta.

Per la struttura delle pagine del sito si utilizzerà HTML5, si curerà l'aspetto estetico mediante l'utilizzo di stylesheet CSS3 e si renderanno interattive tramite script in JS.

Le pagine dinamiche, invece, saranno implementate mediante attraverso JSP. Per la responsiveness del sito web sono state utilizzate le media queries di CSS.

Il DBMS relazionale scelto per lo storing dei dati persistenti è MySQL. Il database sarà definito tramite script SQL e le connessioni ad esso saranno effettuate con l'utilizzo di `jdbc/mysql-connector.jar`.

Il deployment del sito avverrà in locale su un server HTTP attraverso Apache Tomcat v9.0.71.



2. Packages

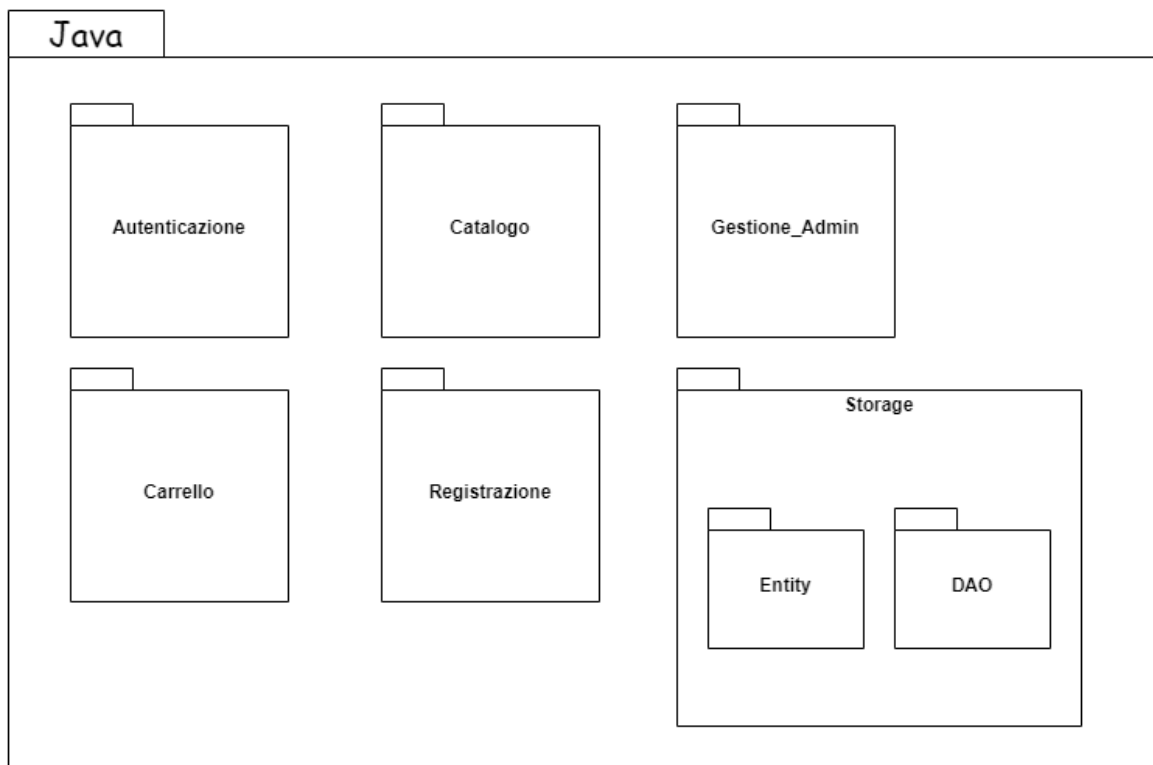
In questa sezione verrà mostrato l'albero dei file del progetto in accordo ai packages individuati nel system design.

- **src**, la cartella in cui sono contenuti tutti i file sorgente
 - ♦ **java**, contiene tutte le classi java e i suoi componenti
 - ♦ **test**, contiene tutte le classi di test
 - ♦ **webapp**, contiene tutti i file relativi alla parte front end
 - ♦ **target**, contiene il .war da deployare prodotto dalla build di Maven

Come esposto nell'SDD, il package principale è stato partizionato secondo il pattern architetturale MVC.

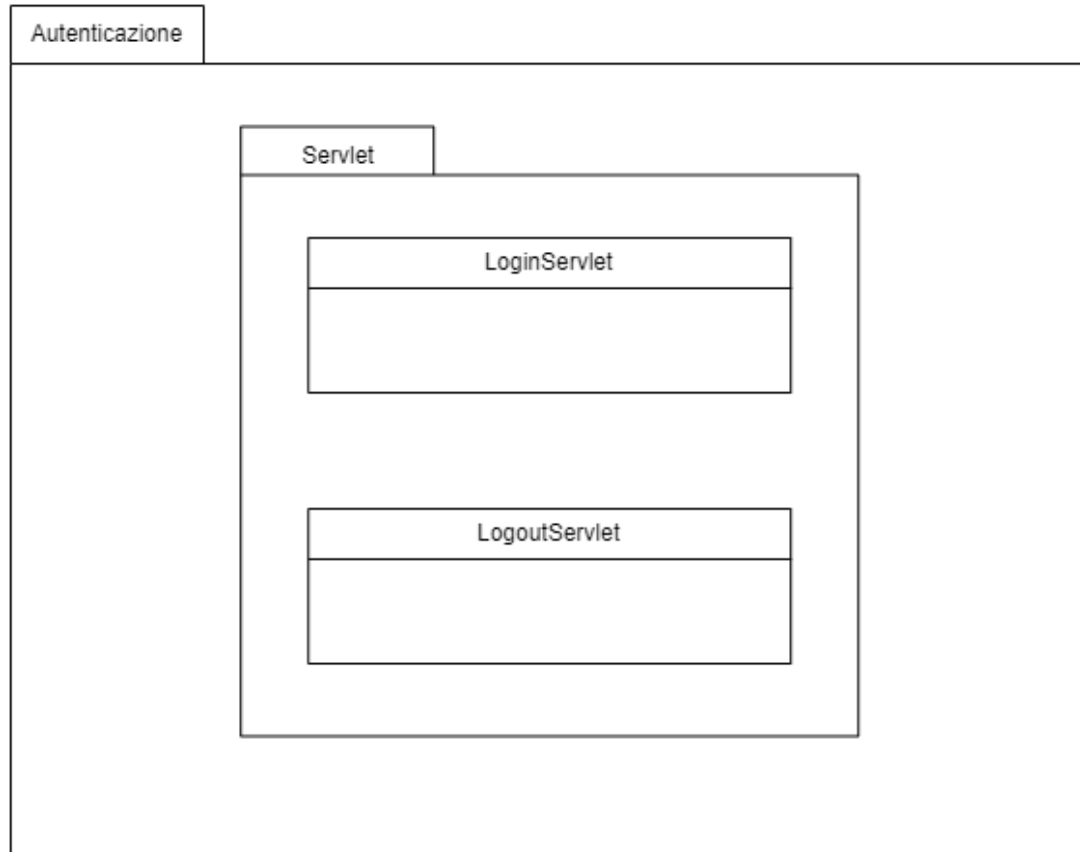
In particolare nel package Model troveremo le Entity e i DAO utilizzati per l'accesso al database.

La scelta del raggruppamento delle classi Entity è stata dettata dalle relazioni e dipendenze individuate tra gli oggetti persistenti.



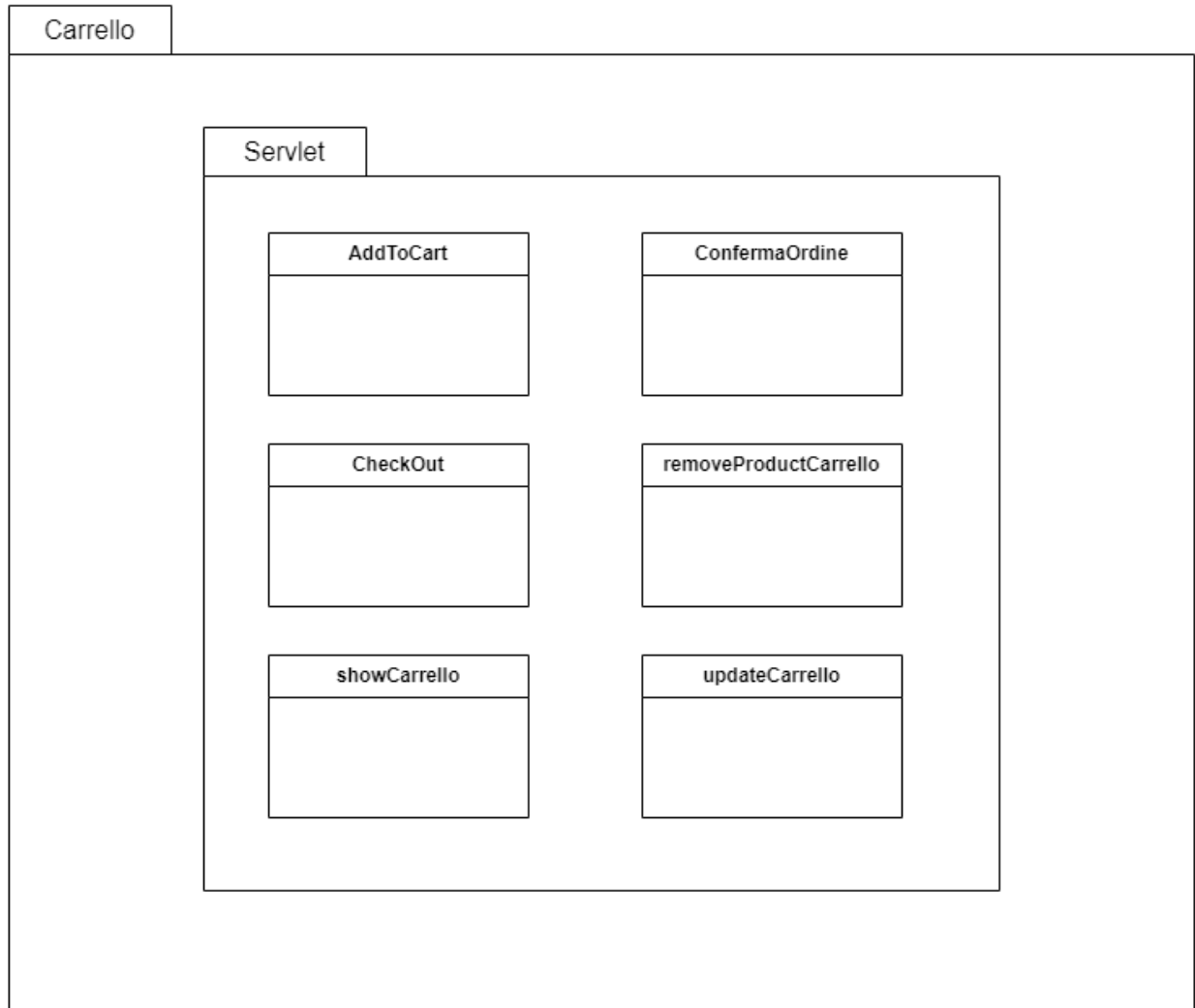


Package Autenticazione



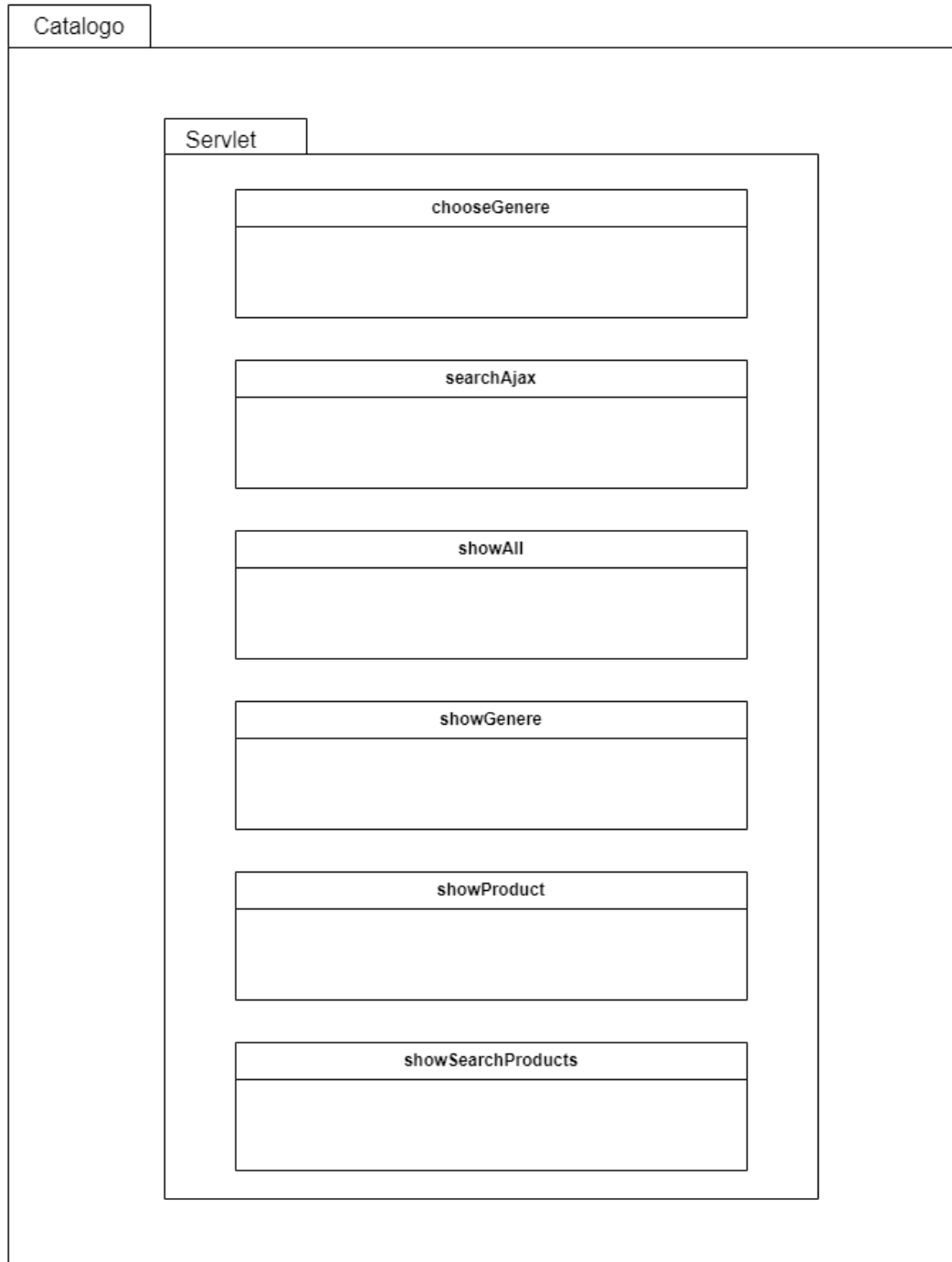


Package Carrello



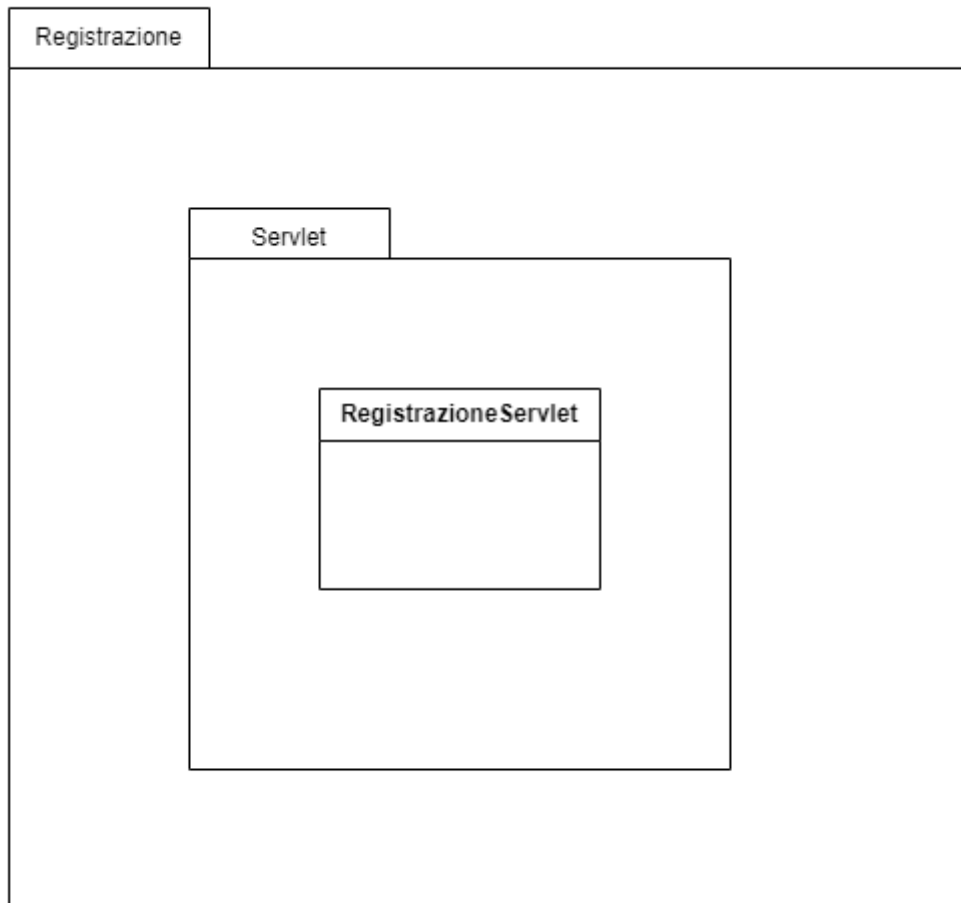


Package Catalogo



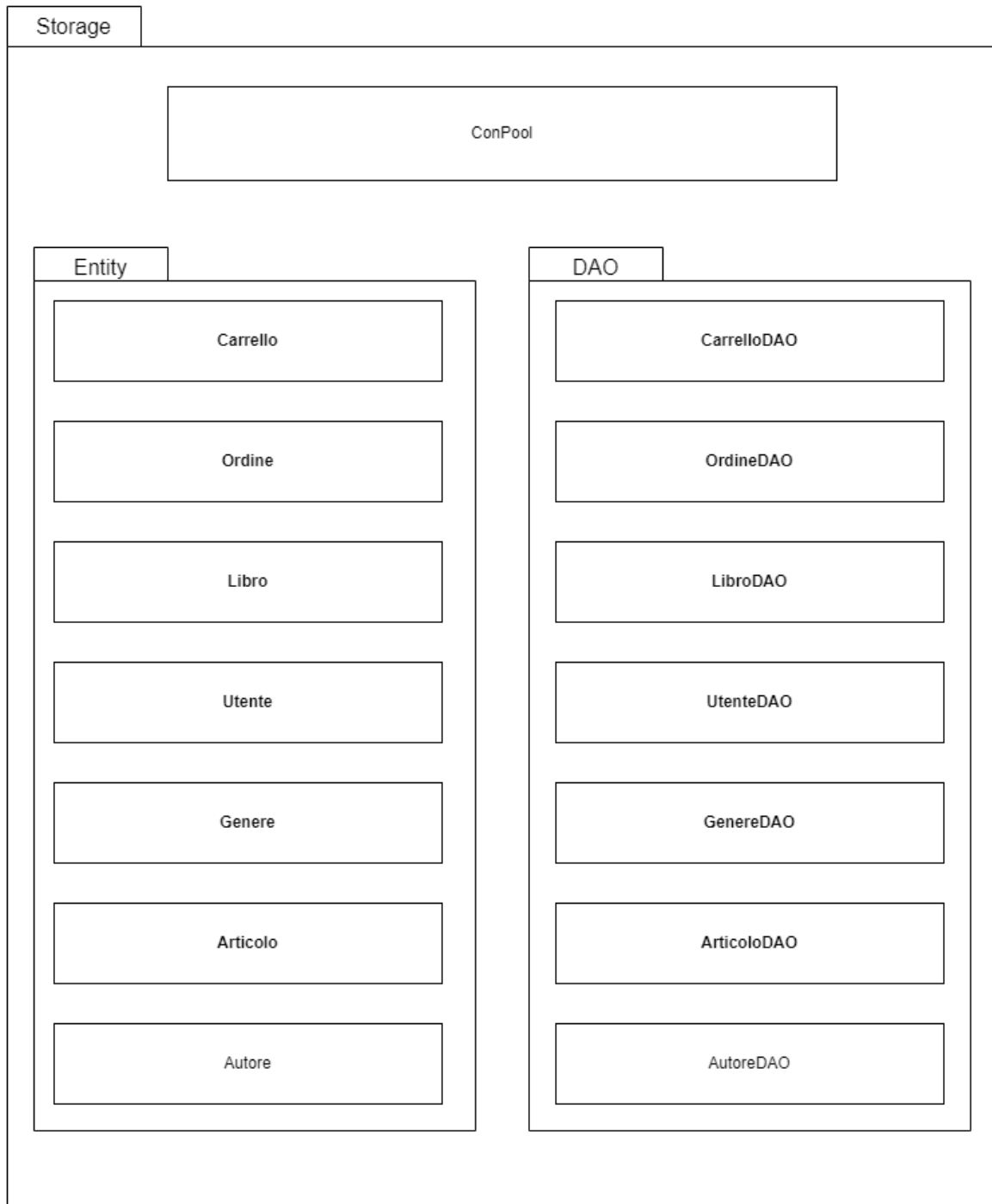


Package Registrazione



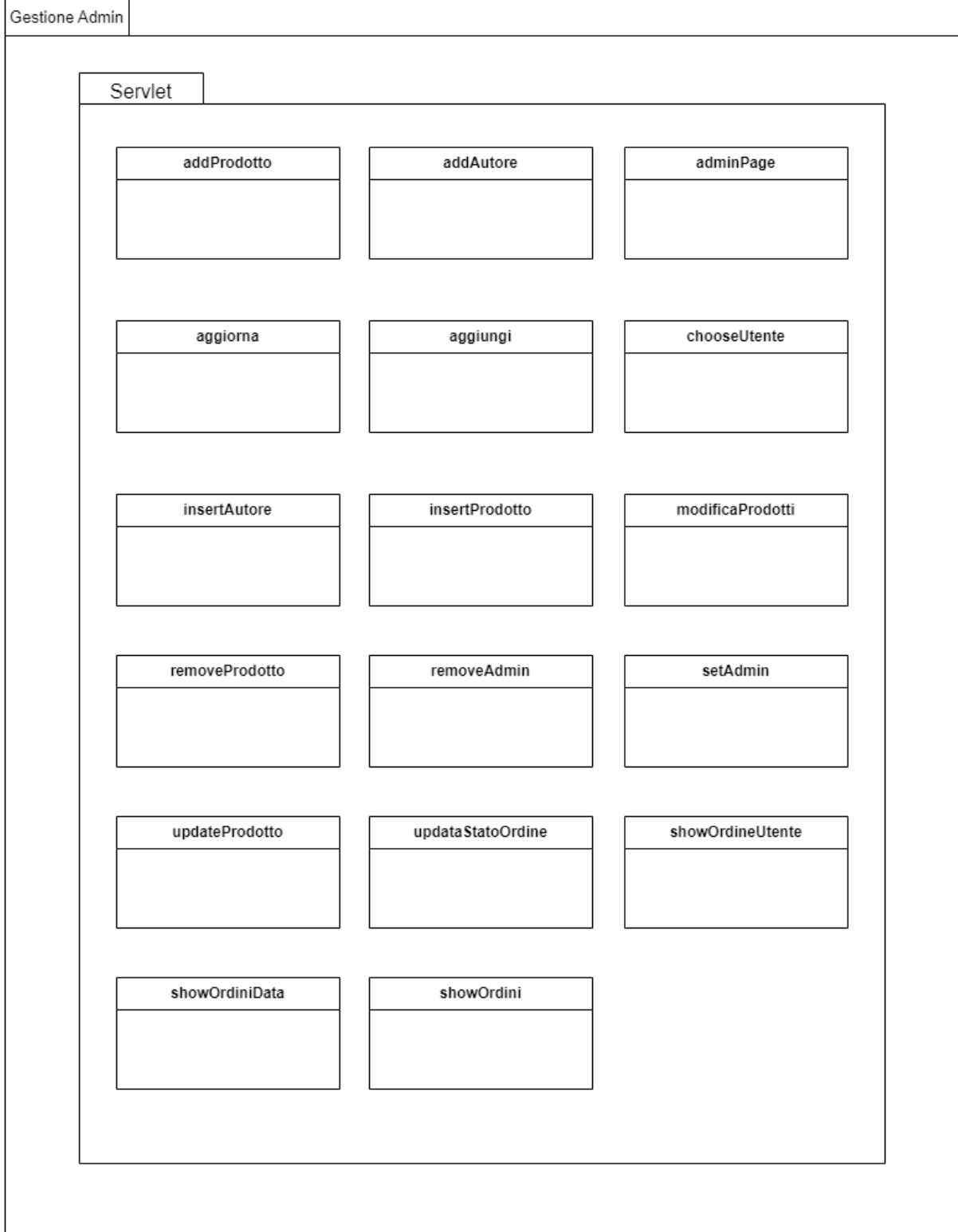


Package Storage





Package Gestione_Admin





3. Class Interfaces

Di seguito saranno presentate le interfacce di ciascun DAO.

ArticoloDAO

Nome metodo	doUpdateArticolo(int codice, int quantita, double prezzo):int
Descrizione	Questo metodo aggiorna i dati dell'articolo nel database.
Pre-condizione	context: ArticoloDAO::doUpdateArticolo(int codice, int quantita, double prezzo) pre: int codice = not null int quantita = not null double prezzo = not null
Post-condizione	context: ArticoloDAO::doUpdateArticolo(int codice, int quantita, double prezzo) post: articoloDAO.doUpdateArticolo(...) == 1

Nome metodo	doUpdateQuantita(Articolo articolo, int quantita):int
Descrizione	Questo metodo sottrae la quantità indicata dalla quantità disponibile di un determinato articolo nel database.
Pre-condizione	context: ArticoloDAO::doUpdateQuantita(Articolo articolo, int quantita) pre: Articolo articolo = not null int quantita = not null
Post-condizione	context: ArticoloDAO::doUpdateQuantita(Articolo articolo, int quantita) post: articoloDAO.doUpdateQuantita(...) == 1



Nome metodo	doUpdatePrezzo(Articolo articolo, double prezzo):int
Descrizione	Questo metodo modifica il prezzo di un determinato articolo presente nel database.
Pre-condizione	context: ArticoloDAO::doUpdatePrezzo(Articolo articolo, double prezzo) pre: Articolo articolo = not null double prezzo = not null
Post-condizione	context: ArticoloDAO::doUpdatePrezzo(Articolo articolo, double prezzo) post: articoloDAO.doUpdatePrezzo(...) == 1

Nome metodo	doSaveArticolo(Articolo articolo):int
Descrizione	Questo metodo aggiunge un nuovo articolo al database.
Pre-condizione	context: ArticoloDAO::doSaveArticolo(Articolo articolo) pre: Articolo articolo = not null
Post-condizione	context: ArticoloDAO::doSaveArticolo(Articolo articolo) post: articoloDAO.doSaveArticolo(...) == 1



Nome metodo	doRetrieveArticoli(int limit, int offset):List<Articolo>
Descrizione	Questo metodo restituisce tutti gli articoli presenti nel database rispettando i limiti passati come parametri.
Pre-condizione	context: ArticoloDAO::doRetrieveArticoli(int limit, int offset) pre: int limit > 0 int offset > 0
Post-condizione	context: ArticoloDAO::doRetrieveArticoli(int limit, int offset) post: List<Articolo> articoli == not null

Nome metodo	doRetrieveArticoliByNomeLike(String nome, int offset, int limit):List<Articolo>
Descrizione	Questo metodo restituisce tutti gli articoli presenti nel database, con il nome che contiene la stringa passata e rispettando i limiti passati come parametri.
Pre-condizione	context: ArticoloDAO::doRetrieveArticoliByNomeLike(,String nome, int limit, int offset) pre: int limit > 0 int offset > 0 nome = not null
Post-condizione	context: ArticoloDAO::doRetrieveArticoliByNomeLike(,String nome, int limit, int offset) post: List<Articolo> articoli == not null



Nome metodo	doRetrieveArticoliByCodice(int codice):Articolo
Descrizione	Questo metodo restituisce l'articolo il cui codice è quello passato come parametro.
Pre-condizione	context: ArticoloDAO::doRetrieveArticoliByCodice(int codice) pre: codice = not null
Post-condizione	context: ArticoloDAO::doRetrieveArticoliByCodice(int codice) post: Articolo articolo == not null

Nome metodo	doRetrieveArticoliByValutazione(int limit, int offset):List<Articolo>
Descrizione	Questo metodo restituisce gli articoli in base alla valutazione.
Pre-condizione	context: ArticoloDAO::doRetrieveArticoliByValutazione(int limit, int offset) pre: int limit > 0 int offset > 0
Post-condizione	context: ArticoloDAO::doRetrieveArticoliByValutazione(int limit, int offset) post: List<Articolo> articoli == not null



Nome metodo	doUpdateArticoloFull(Articolo articolo):int
Descrizione	Questo metodo aggiorna tutte le informazioni di un determinato articolo presente nel database.
Pre-condizione	context: ArticoloDAO::doUpdateArticoloFull(Articolo articolo) pre: Articolo articolo = not null
Post-condizione	context: ArticoloDAO::doUpdateArticoloFull(Articolo articolo) post: articoloDAO.doUpdateArticoloFull(...) == 1

Nome metodo	doRemoveArticolo(int codice):int
Descrizione	Questo metodo rimuove un determinato articolo presente nel database in base al suo codice.
Pre-condizione	text: ArticoloDAO::doRemoveArticolo(int codice) pre: int codice = not null
Post-condizione	context: ArticoloDAO::doRemoveArticolo(int codice) post: articoloDAO.doRemoveArticolo(...) == 1



AutoreDAO

Nome metodo	loRetrieveAutori(int limit, int offset):List<Autore>
Descrizione	Questo metodo restituisce tutti gli autori presenti nel database rispettando i limiti passati come parametri.
Pre-condizione	context: AutoreDAO::doRetrieveAutori(int limit, int offset) pre: int limit > 0 int offset > 0
Post-condizione	context: AutoreDAO::doRetrieveAutori(int limit, int offset) post: List<Autore> autore == not null

Nome metodo	doUpdateAutore(Autore a):int
Descrizione	Questo metodo modifica le informazioni di un determinato autore presente nel database.
Pre-condizione	context: AutoreDAO::doUpdateAutore(Autore a) pre: Autore autore = not null
Post-condizione	context: AutoreDAO::doUpdateAutore(Autore a) post: autoreDAO.doUpdateAutore(...) == 1



Nome metodo	doUpdateAutore(Autore a):int
Descrizione	Questo metodo modifica le informazioni di un determinato autore presente nel database.
Pre-condizione	context: AutoreDAO::doUpdateAutore(Autore a) pre: Autore autore = not null
Post-condizione	context: AutoreDAO::doUpdateAutore(Autore a) post: autoreDAO.doUpdateAutore(...) == 1

Nome metodo	doSaveAutore(Autore a):int
Descrizione	Questo metodo salva un nuovo autore nel database.
Pre-condizione	context: AutoreDAO::doSaveAutore(Autore a) pre: Autore autore = not null
Post-condizione	context: AutoreDAO::doSaveAutore(Autore a) post: autoreDAO.doSaveAutore(...) == 1



CarrelloDAO

Nome metodo	doCreateCarrello(Utente u):int
Descrizione	Questo metodo crea un nuovo carrello e lo associa ad un Utente presente nel database.
Pre-condizione	context: CarrelloDAO::doCreateCarrello(Utente u) pre: Utente utente = not null
Post-condizione	context: CarrelloDAO::doCreateCarrello(Utente u) post: carrelloDAO.doCreateCarrello(...) == 1

Nome metodo	doClearCarrello(Utente u):int
Descrizione	Questo metodo azzerava il carrello di un determinato utente passato come parametro.
Pre-condizione	context: CarrelloDAO::doClearCarrello(Utente u) pre: Utente utente = not null
Post-condizione	context: CarrelloDAO::doClearCarrello(Utente u) post: carrelloDAO.doClearCarrello(...) == 1



Nome metodo	doUpdateCarrello(Carrello carrello):int
Descrizione	Questo metodo modifica le informazioni di un determinato carrello presente nel database.
Pre-condizione	context: CarrelloDAO::doUpdateCarrello(Carrello carrello) pre: Carrello carrello = not null
Post-condizione	context: CarrelloDAO::doUpdateCarrello(Carrello carrello) post: carrelloDAO.doUpdateCarrello(...) == 1

LibroDAO

Nome metodo	doRetrieveLibro(int limit, int offset):List<Libro>
Descrizione	Questo metodo restituisce tutti i libri presenti nel database rispettando i limiti passati come parametri.
Pre-condizione	context: LibroDAO::doRetrieveLibro(int limit, int offset) pre: int limit > 0 int offset > 0
Post-condizione	context: LibroDAO::doRetrieveArticoli(int limit, int offset) post: List<Libro> libri == not null



Nome metodo	doRetrieveLibroByArticolo(int codice):Libro
Descrizione	Questo metodo restituisce un libro in base al codice dell'articolo ad esso collegato.
Pre-condizione	context: LibroDAO::doRetrieveLibroByArticolo(int codice) pre: int codice = not null
Post-condizione	context: LibroDAO::doRetrieveLibroByArticolo(int codice) post: Libro libro == not null

Nome metodo	doRetrieveLibroByGenere(Genere genere, int limit, int offset):List<Libro>
Descrizione	Questo metodo restituisce tutti i libri di un determinato genere.
Pre-condizione	context: LibroDAO::doRetrieveLibroByGenere(Genere genere, int limit, int offset) pre: int limit > 0 int offset > 0 Genere genere = not null
Post-condizione	context: LibroDAO::doRetrieveLibroByGenere(Genere genere, int limit, int offset) post: List<Libro> libri == not null



Nome metodo	doSaveLibro(Libro l):int
Descrizione	Questo metodo salva un nuovo libro nel database.
Pre-condizione	context: LibroDAO::doSaveLibro(Libro l) pre: Libro l = not null
Post-condizione	context: LibroDAO::doSaveLibro(Libro l) post: libroDAO.doSaveLibro(...) = 1

Nome metodo	doUpdateLibro(Libro l):int
Descrizione	Questo metodo aggiorna le informazioni di un libro presente nel database passato come parametro.
Pre-condizione	context: LibroDAO::doUpdateLibro(Libro l) pre: Libro l = not null
Post-condizione	context: LibroDAO::doUpdateLibro(Libro l) post: libroDAO.doUpdateLibro(...) = 1



Nome metodo	doRemoveLibro(int codice):int
Descrizione	Questo metodo rimuove un determinato libro dal database in base al suo codice passato come parametro.
Pre-condizione	context: LibroDAO::doRemoveLibro(int codice) pre: int codice = not null
Post-condizione	context: LibroDAO::doRemoveLibro(int codice) post: libroDAO.doRemoveLibro(...) = 1

OrdineDAO

Nome metodo	doRetrieveOrdiniByUtente(Utente u):List<Ordine>
Descrizione	Questo metodo restituisce tutti gli ordini di un determinato utente.
Pre-condizione	context: OrdineDAO::doRetrieveOrdiniByUtente(Utente u) pre: Utente u = not null
Post-condizione	context: OrdineDAO::doRetrieveOrdiniByUtente(Utente u) post: List<Ordine> ordini == not null



Nome metodo	doRetrieveOrdineByNumero(int n):Ordine
Descrizione	Questo metodo restituisce l'ordine il cui numero è quello passato come parametro.
Pre-condizione	context: OrdineDAO::doRetrieveOrdineByNumero(int n) pre: int n = not null
Post-condizione	context: OrdineDAO::doRetrieveOrdineByNumero(int n) post: ordineDao.doRetrieveOrdineByNumero(...) == not null

Nome metodo	doRetrieveAllOrdini(int offset, int limit):List<Ordine>
Descrizione	Questo metodo restituisce tutti gli ordini presenti nel database.
Pre-condizione	context: OrdineDAO::doRetrieveAllOrdini(int offset, int limit) pre: int limit > 0 int offset > 0
Post-condizione	context: OrdineDAO::doRetrieveAllOrdini(int offset, int limit) post: List<Ordine> ordini == not null



Nome metodo	doSaveOrdine(Utente utente, Carrello carrello, Ordine ordine):int
Descrizione	Questo metodo inserisce un nuovo ordine nel database.
Pre-condizione	context: OrdineDAO::doSaveOrdine(Utente utente, Carrello carrello, Ordine ordine) pre: Utente utente = not null Carrello carrello = not null Ordine ordine = not null
Post-condizione	context: OrdineDAO::doSaveOrdine(Utente utente, Carrello carrello, Ordine ordine) post: ordineDAO.doSaveOrdine(...) == 1

Nome metodo	doUpdateStatoOrdine(Ordine o):int
Descrizione	Questo metodo modifica lo stato di un determinato ordine presente nel database.
Pre-condizione	context: OrdineDAO::doUpdateStatoOrdine(Ordine o) pre: Ordine ordine = not null
Post-condizione	context: OrdineDAO::doUpdateStatoOrdine(Ordine o) post: ordineDAO.doUpdateStatoOrdine(...) == 1



Nome metodo	doRetrieveOrdiniByData(Date start, Date end):List<Ordine>
Descrizione	Questo metodo restituisce tutti gli ordini effettuati tra due date passate come parametro.
Pre-condizione	context: OrdineDAO::doRetrieveOrdiniByData(Date start, Date end) pre: Date start = not null Date end = not null
Post-condizione	context: OrdineDAO::doRetrieveOrdiniByData(Date start, Date end) post: List<Ordine> ordini == not null

UtenteDAO

Nome metodo	doRetrieveIdByUsername(String user):int
Descrizione	Questo metodo restituisce l'id di un utente in base al suo username.
Pre-condizione	context: UtenteDAO::doRetrieveIdByUsername(String user) pre: String user = not null
Post-condizione	context: UtenteDAO::doRetrieveIdByUsername(String user) post: int id == not null



Nome metodo	doRetrieveByUsernamePassword(String username, String password):Utente
Descrizione	Questo metodo restituisce un Utente presente nel database in base ad una combinazione di username e password passata come parametro.
Pre-condizione	context: UtenteDAO::doRetrieveByUsernamePassword(String username, String password) pre: String username = not null String password = not null
Post-condizione	context: UtenteDAO::doRetrieveByUsernamePassword(String username, String password) post: Utente utente == not null

Nome metodo	doRetrieveAllUtenti(int limit, int offset):List<Utente>
Descrizione	Questo metodo restituisce tutti gli utenti presenti nel database.
Pre-condizione	context: UtenteDAO::doRetrieveAllUtenti(int limit, int offset) pre: int limit > 0 int offset > 0
Post-condizione	context: UtenteDAO::doRetrieveAllUtenti(int limit, int offset) post: List<Utente> utenti == not null



Nome metodo	doSaveUtente(Utente u):int
Descrizione	Questo metodo inserisce un nuovo Utente nel database.
Pre-condizione	context: UtenteDAO::doSaveUtente(Utente u) pre: Utente u = not null
Post-condizione	context: UtenteDAO::doSaveUtente(Utente u) post: utenteDAO.doSaveUtente(...) == 1

Nome metodo	doUpdateUtente(Utente u):int
Descrizione	Questo metodo aggiorna le informazioni di un utente presente nel database.
Pre-condizione	context: UtenteDAO::doUpdateUtente(Utente u) pre: Utente u = not null
Post-condizione	context: UtenteDAO::doUpdateUtente(Utente u) post: utenteDAO.doUpdateUtente(...) == 1



Nome metodo	doSetAdmin(Utente u):int
Descrizione	Questo metodo rende admin un Utente presente nel database.
Pre-condizione	context: UtenteDAO::doSetAdmin(Utente u) pre: Utente u = not null
Post-condizione	context: UtenteDAO::doSetAdmin(Utente u) post: utenteDAO.doSetAdmin(...) == 1

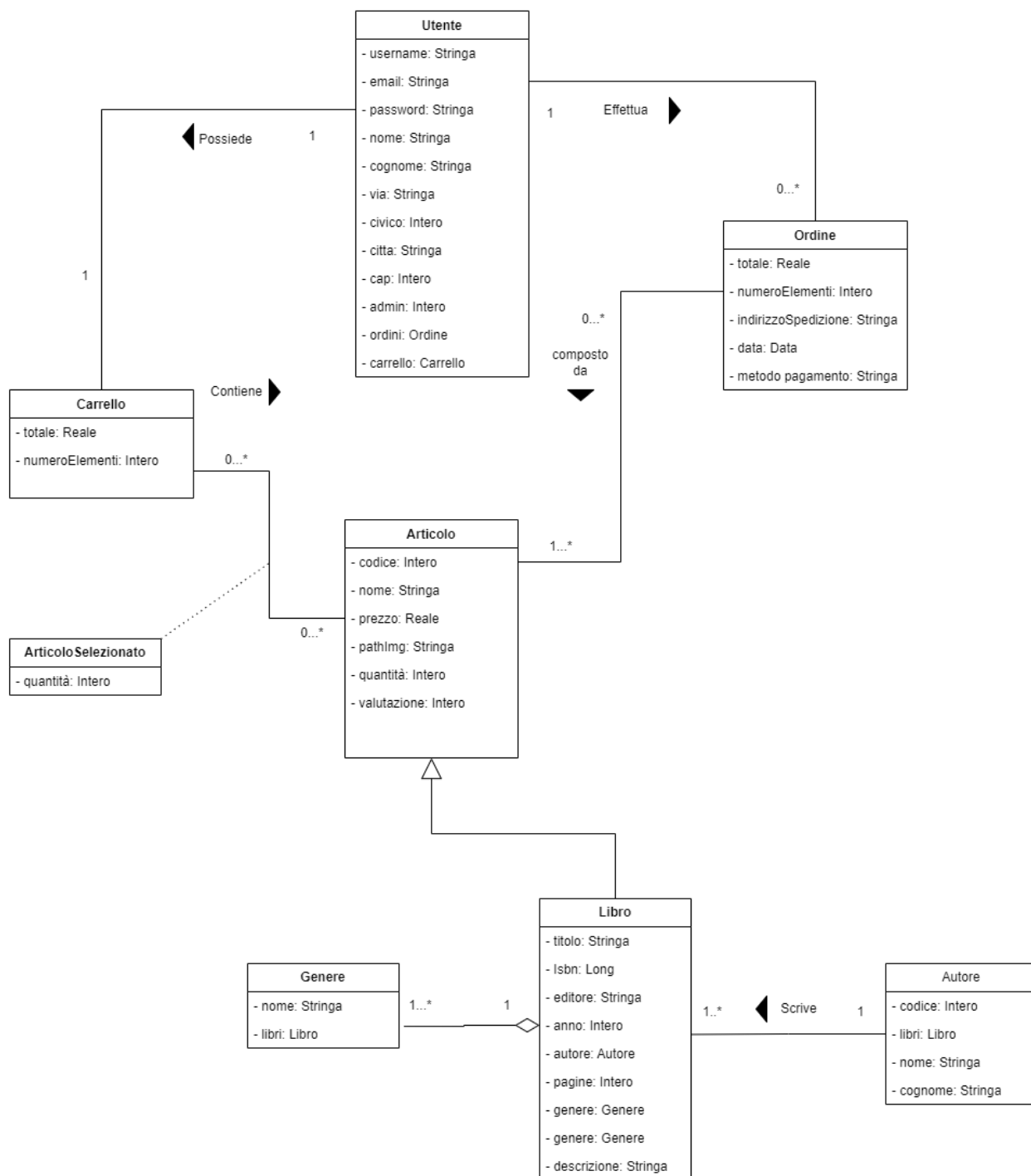
Nome metodo	doRemoveAdmin(Utente u):int
Descrizione	Questo metodo rende utente basico un Utente presente nel database.
Pre-condizione	context: UtenteDAO::doRemoveAdmin(Utente u) pre: Utente u = not null
Post-condizione	context: UtenteDAO::doRemoveAdmin(Utente u) post: utenteDAO.doRemoveAdmin(...) == 1



Nome metodo	doVerifyDuplicate(Utente utente):boolean
Descrizione	Questo metodo verifica se è già presente nel database un Utente con lo stesso username.
Pre-condizione	context: UtenteDAO::doVerifyDuplicate(Utente utente) pre: Utente utente = not null
Post-condizione	context: UtenteDAO::doVerifyDuplicate(Utente utente) post: utenteDAO.doVerifyDuplicate(...) == false

4. Class Diagram

Si riporta di seguito il class diagram aggiornato relativo ai dati persistenti che il sistema PeekABook gestisce.





La creazione del database, come espresso nella sezione 1.3 del documento corrente, è effettuata tramite il file di script peekDatabase.sql riportato di seguito.

```
1 • drop database if exists PeekABook;
2 • create database PeekABook;
3 • use PeekABook;
4
5 • CREATE TABLE Utente(
6     id int primary key auto_increment,
7     adm tinyint(1) not null default 0,
8     email varchar(40) not null,
9     username varchar(20) not null,
10    passwordHash varchar(256) not null,
11    nome varchar(40) not null,
12    cognome varchar(40) not null,
13    via varchar(40) not null,
14    civico int not null,
15    citta varchar(40) not null,
16    CAP int(5) not null);
17
18 • CREATE TABLE Autore(
19     codiceAutore int auto_increment primary key,
20     nome varchar(30) not null,
21     cognome varchar(30) not null);
22
23 • CREATE TABLE Carrello(
24     utente int,
25     totale double not null,
26     numeroArticoli int not null,
27     primary key(utente)
28 );
```



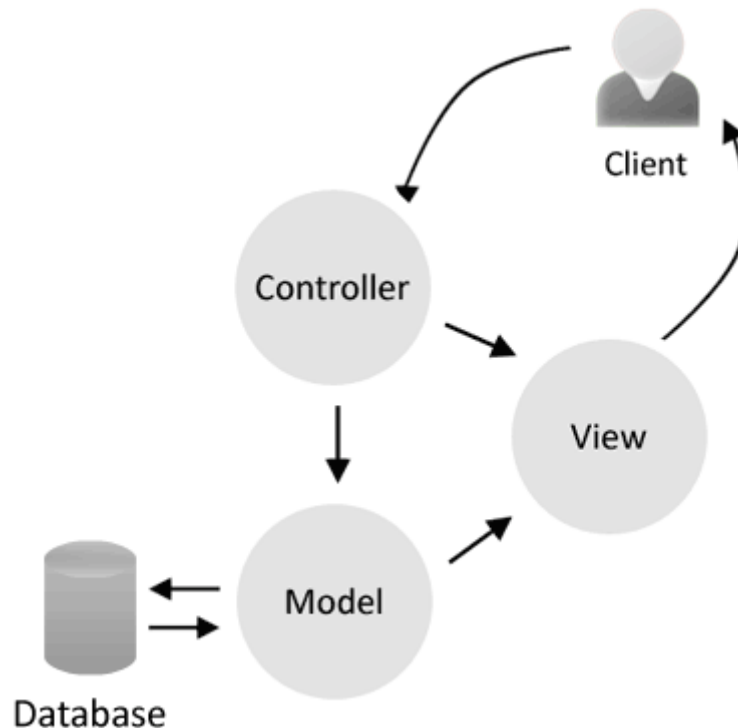

```
30 • CREATE TABLE Articolo(  
31     codice int primary key auto_increment,  
32     prezzo double not null,  
33     valutazione int not null,  
34     quantita int not null,  
35     copertina varchar(256),  
36     nome varchar(100) not null);  
37  
38 • CREATE TABLE ArticoloSelezionato(  
39     utente int,  
40     articolo int,  
41     quantita int not null,  
42     foreign key(articolo) references Articolo(codice) on update cascade on delete cascade,  
43     primary key(utente,articolo));  
44  
45 • CREATE TABLE Ordine(  
46     utente int,  
47     numero int auto_increment,  
48     via varchar(40) not null,  
49     civico int not null,  
50     citta varchar(40) not null,  
51     CAP int(5) not null,  
52     dataOrdine date not null,  
53     dataConsegna date,  
54     stato varchar(20) not null,  
55     metodoPagamento varchar(30) not null,  
56     primary key (numero));  
57
```



```
58 ● ○ CREATE TABLE Libro(  
59     autore int,  
60     articolo int auto_increment,  
61     titolo varchar(40) not null,  
62     ISBN long not null,  
63     anno int(4) not null,  
64     pagine int not null,  
65     editore varchar(30) not null,  
66     descrizione varchar(1000) not null,  
67     primary key(articolo),  
68     foreign key(articolo) references Articolo(codice) on update cascade on delete cascade,  
69     foreign key-autore references Autore(codiceAutore) on update cascade on delete cascade);  
70  
71 ● ○ CREATE TABLE Genere(  
72     nome varchar(30) primary key);  
73  
74 ● ○ CREATE TABLE GenLibro(  
75     libro int,  
76     genere varchar(30),  
77     foreign key(libro) references Libro(articolo) on update cascade on delete cascade,  
78     foreign key-genere references Genere(nome) on update cascade on delete cascade,  
79     primary key(libro, genere));
```

5. Design Pattern

Per lo sviluppo del progetto è stato utilizzato il pattern architetturale **MVC (Model-View-Controller)**. Esso permette di separare l'applicazione in tre componenti logici interconnessi ma allo stesso tempo indipendenti. Questa suddivisione aiuta a isolare in maniera chiara la logica di business, l'interfaccia utente, le interazioni utente e la gestione del dato rendendo il codice più facile da gestire e da estendere.



Inoltre è stato utilizzato il pattern **DAO** per separare la logica di business dalla logica di accesso ai dati. In questo modo, i componenti della logica di business non accedono mai direttamente al database. Solo gli oggetti previsti dal pattern Dao possono accedervi. Inoltre, se dovessimo modificare il tipo di memoria persistente utilizzata, non sarà necessario stravolgere il codice della nostra applicazione, ma basterà modificare i DAO utilizzati.



Di seguito è riportato un esempio di utilizzo del pattern DAO in PeekABook.

