UNIVERSITÀ DEGLI STUDI
DI SALERNO

Ingegneria del Software

A.A. 2017/2018

Book a Book

Object Design Document

Versione 1.2

Top Manager:

Professore
Andrea De Lucia

Project Manager:

Nome	Matricola
Pizza Antonio	05121 02367

Partecipanti:

Nome	Matricola
Benincasa Mirko	05121 03524
De Stefano Manuel	05121 03896
Monaco Salvatore	05121 03456
Pangaro Luca	05121 03846
Soldà Stefano	05121 03576
Trerè Marialuisa	05121 03770

Revision History

Data	Versione	Descrizione	Autore
07/12/2017	1.0	Prima stesura	Tutto il team
29/12/2017	1.1	Revisione Prima stesura	Tutto il team
27/01/2018	1.2	Seconda Revisione	Tutto il team

Indice

1.Introduzione

1.1 Object Design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design.
Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:
Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

Prestazioni vs Costi:
Essendo il progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, per alcune funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.

Interfaccia vs Usabilità:
L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa. Fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:
La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

Naming Convention

- **È** buona norma utilizzare nomi:

    1. Descrittivi
    2. Pronunciabili
    3. Di uso comune
    4. Lunghezza medio-corta
    5. Non abbreviati
    6. Evitando la notazione ungherese
    7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili

- I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.

Esempio: elaboratoSessioneId

- E’ inoltre possibile, in alcuni casi, utilizzare il carattere underscore “_”, ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Esempio: CREATE_ARGOMENTO

Metodi:

- I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica un’azione, seguito dal nome di un oggetto. I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitarne la leggibilità. Esistono però casi particolari come ad esempio nell’implementazione dei model, dove viene utilizzata l’interfaccia CRUD.

Esempio: getId(), setId()

- I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno e, se applicabile, sulle eccezioni.

Classi e pagine:

- I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo.
- I nomi delle pagine devono cominciare con una lettera minuscola. Tutte le parole al loro interno sono scritte in minuscolo e separate dal carattere '-'.
- I nomi delle servlet sono analoghi a quelli delle classi, con l'aggiunta della parola "Servlet" come ultima parola.

Esempio: ManagerAutenticazione.java, registra-biblioteca.jsp, LoginServlet.java

Ogni file sorgente java contiene una singola classe e dev'essere strutturato in un determinato modo:

- Una breve introduzione alla classe

  L'introduzione indica: l'autore, la versione e la data.

```
/**
*
* @author nome dell'autore
* @version numero di versione della classe
* @since data dell'implementazione
*/
```

- L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.

- La dichiarazione di classe caratterizzata da:

  1. Dichiarazione della classe pubblica
  2. Dichiarazioni di costanti
  3. Dichiarazioni di variabili di classe
  4. Dichiarazioni di variabili d'istanza
  5. Costruttore
  6. Commento e dichiarazione metodi.

```java
package classiProgetto;

import java.io.Serializable;

/**
 *
 * @author Antonio Pizza
 *
 */
public class Libro implements Serializable{



	private String titolo;
	private String autore;
	private int ISBN;

	/**
	 * Questo è il costruttore di un libro dove vengono passati in input il titolo, l'autore ed il codice ISBN
	 *
	 * @param titolo è il titolo
	 * @param autore è l'autore
	 * @param ISBN è il codice ISBN
	 */
	public Libro(String titolo, String autore, int ISBN) {
		this.setTitolo(titolo);
		this.setAutore(autore);
		this.setISBN(ISBN);
	}

	/**
	 *
	 * @return titolo ritorna il titolo
	 */
	public String getTitolo() {
		return titolo;
	}

	/**
	 *
	 * @param titolo è il titolo
	 */
	public void setTitolo(String titolo) {
		this.titolo = titolo;
	}
```

```
	/**
	 *
	 * @return autore ritorna l'autore
	 */
	public String getAutore() {
		return autore;
	}

	/**
	 *
	 * @param autore è l'autore
	 */
	public void setAutore(String autore) {
		this.autore = autore;
	}

	/**
	 *
	 * @return ISBN ritorna il codice ISBN
	 */
	public int getISBN() {
		return ISBN;
	}

	/**
	 *
	 * @param iSBN è il codice ISBN
	 */
	public void setISBN(int iSBN) {
		ISBN = iSBN;
	}

	@Override
	public String toString() {
		return getClass().getSimpleName() + "{titolo=" + titolo + ", autore=" + autore +
		", ISBN=" + ISBN + "}";
	}

}
```

1.3 Definizioni, acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- CRUD: Create Read Update Delete

Abbreviazioni:

- DB: DataBase

1.4 Riferimenti

- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java,Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto BookaBook
- Documento RAD del progetto BookaBook

2. Packages

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

- Presentation layer
- Application layer
- Storage layer

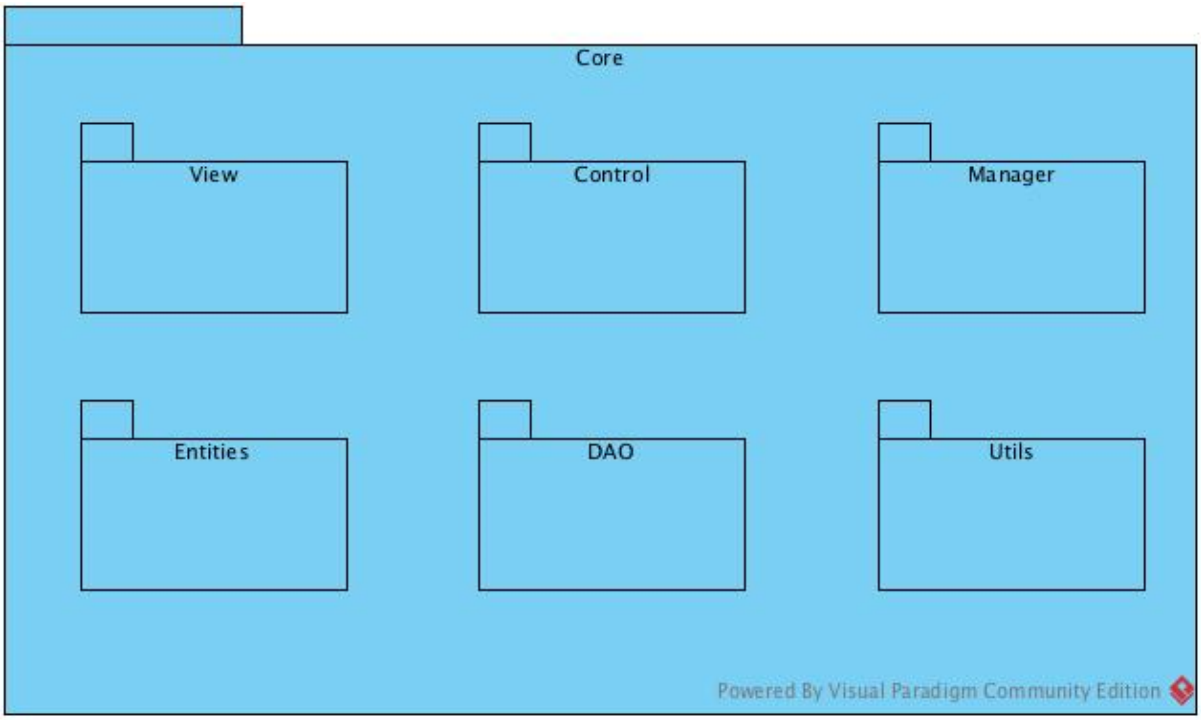
Il package BookaBook contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Presentation layer	Include tutte le interfacce grafiche e in generale i boundary objects, come le form con cui interagisce l'utente. L'interfaccia verso l'utente è rappresentata da un Web server e da eventuali contenuti statici (es. pagine HTML).

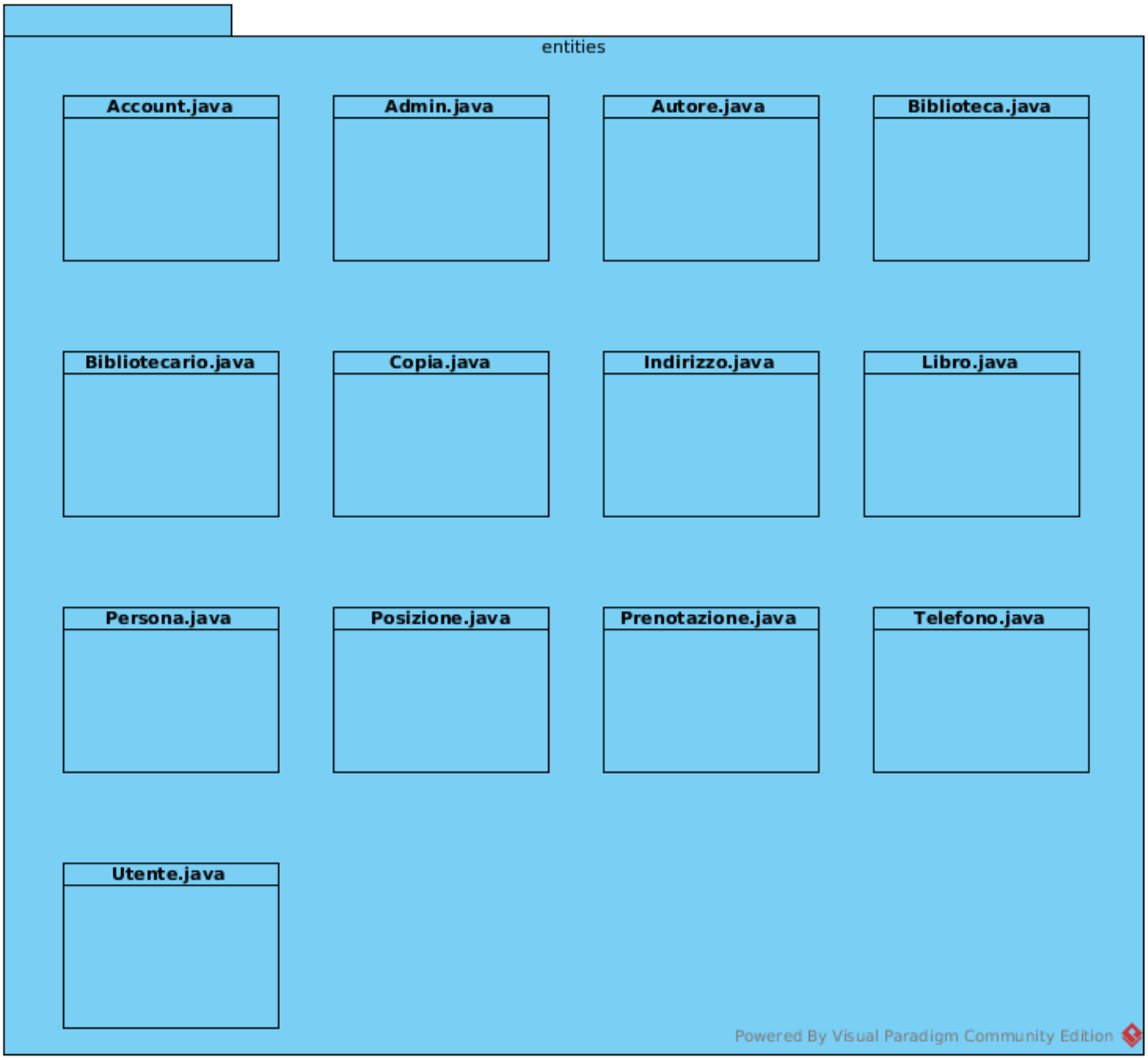
Application layer	Include tutti gli oggetti relativi al controllo e all'elaborazione dei dati. Questo avviene interrogando il database tramite lo storage layer per generare contenuti dinamici e accedere a dati persistenti. Si occupa di varie gestioni quali: • **Gestione Autenticazione** • **Gestione Account** • **Gestione Registrazione** • **Gestione Libri** • **Gestione Prenotazione** • **Gestione Utenti**

Storage layer	Ha il compito di effettuare memorizzazione, il recupero e l'interrogazione degli oggetti persistenti. I dati, i quali possono essere acceduti dall'application layer, sono depositati in maniera persistente su un database tramite DBMS.

2.1 Packages core



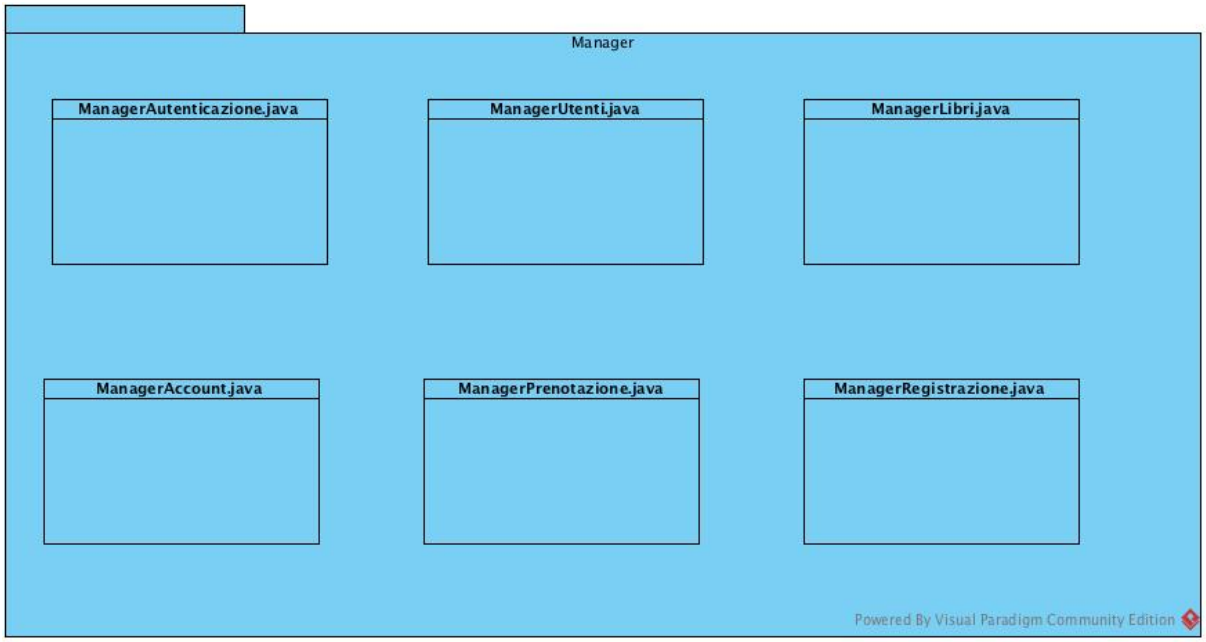
2.1.1 Packages Entities



Classe:	**Descrizione:**
Admin.java	Descrive un amministratore del sistema.
Persona.java	Descrive una persona che fa uso del sistema.
Prenotazione.java	Descrive una prenotazione all'interno del sistema.
Autore.java	Descrive un autore di un libro.
Account.java	Descrive un Account all'interno del sistema.

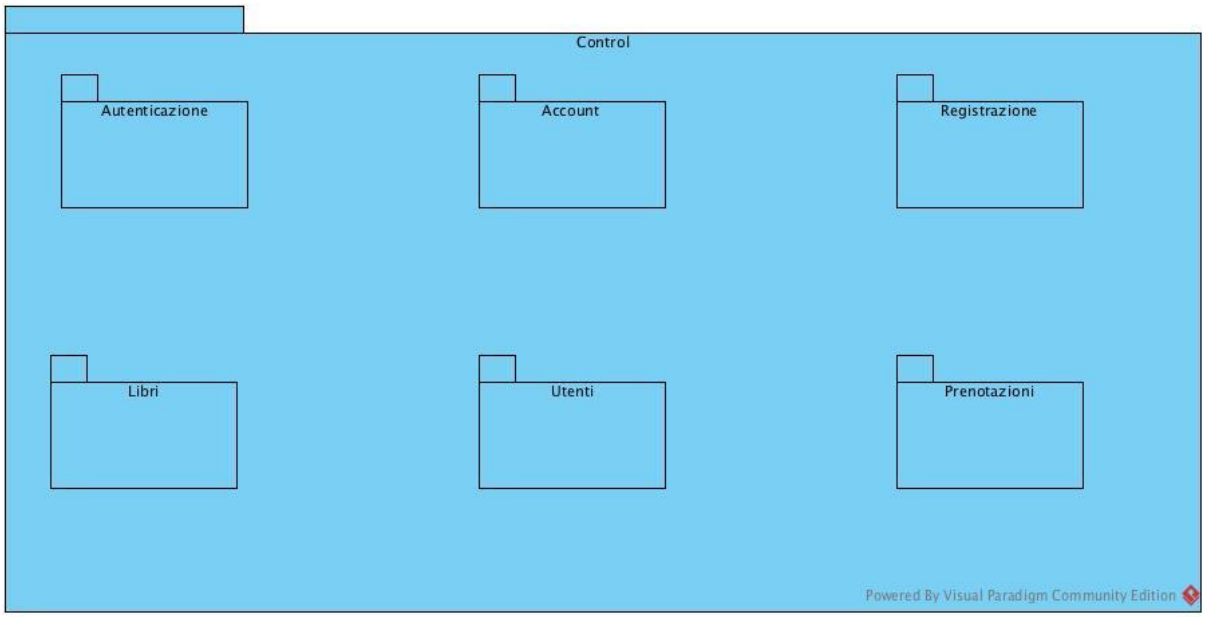
Biblioteca.java	Descrive una biblioteca all’interno del sistema.
Indirizzo.java	Descrive un indirizzo.
Posizione.java	Descrive la posizione di un libro.
Libro.java	Descrive un libro all’interno del sistema.
Bibliotecario.java	Descrive un bibliotecario all’interno del sistema.
Telefono.java	Descrive un numero di telefono all’interno del sistema.
Copia.java	Descrive una copia di un libro all’interno del sistema
Utente.java	Superclasse per Persona,Admin e Bibliotecario. Ne definisce gli attributi comuni.

2.1.2 Packages Manager

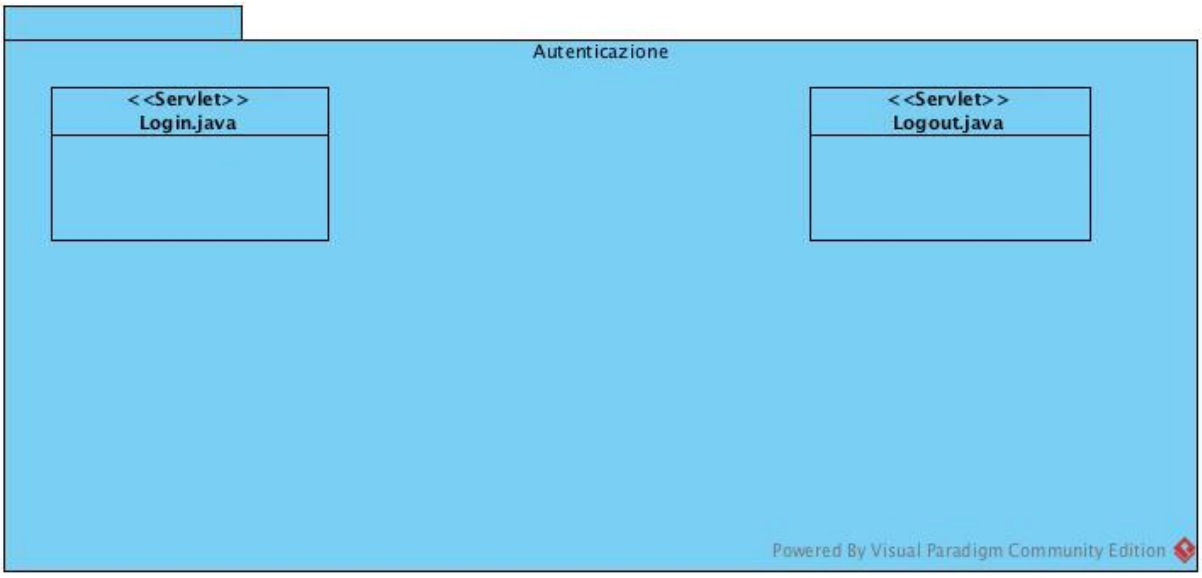


Classe:	**Descrizione:**
ManagerAutenticazione.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione.
ManagerAccount.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli account.
ManagerLibri.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce i libri.
ManagerPrenotazione.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le prenotazioni.
ManagerRegistrazione.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la registrazione
ManagerUtenti.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli utenti.

2.1.3 Packages Control

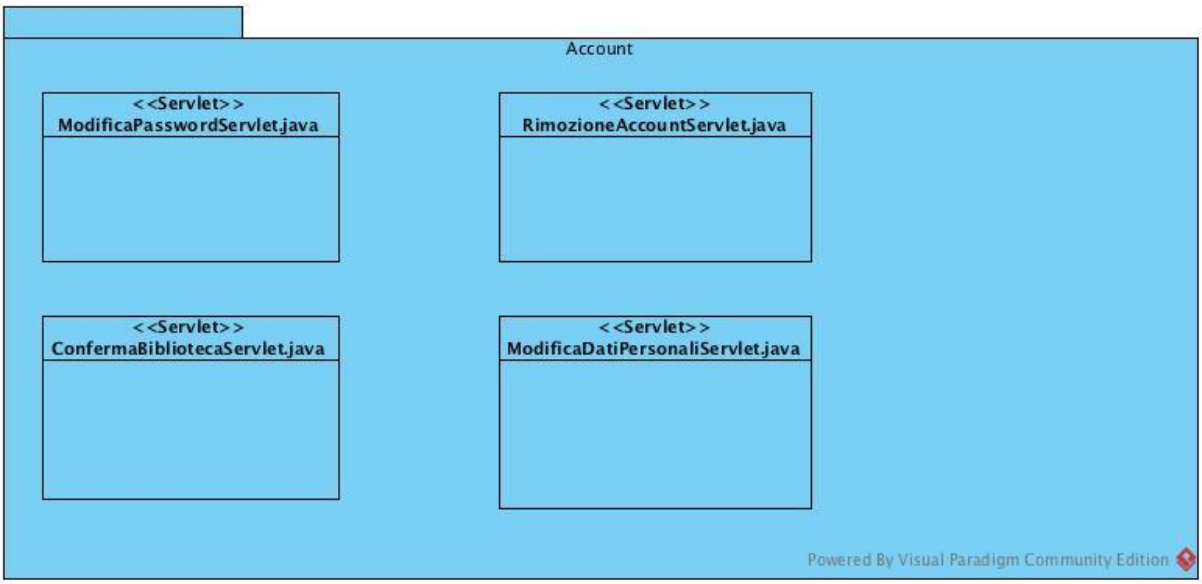


2.1.3.0 Autenticazione



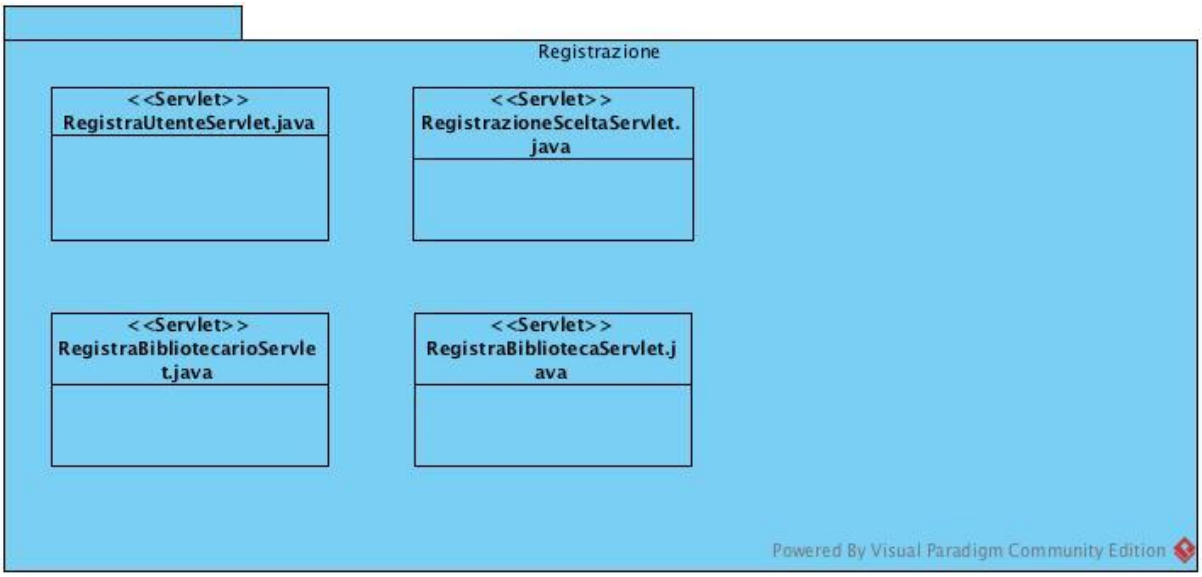
Classe:	**Descrizione:**
LoginServlet.java	Gestisce il login degli utenti.
LogoutServlet.java	Gestisce il logout degli utenti.

2.1.3.1 Account



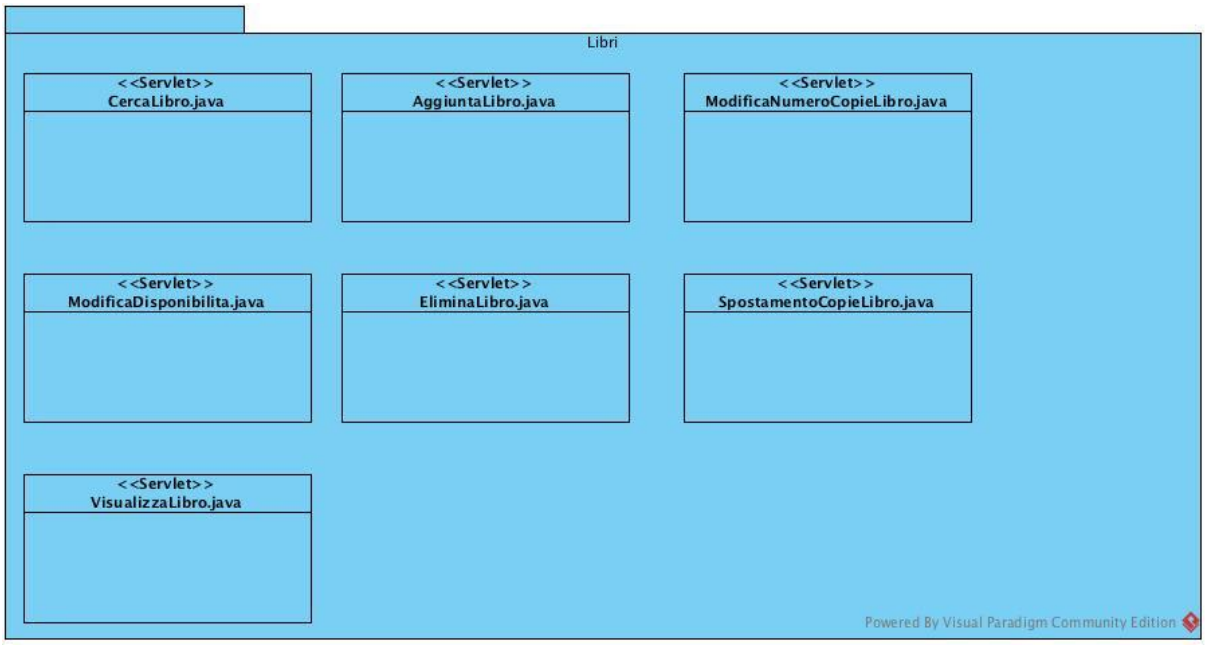
Classe:	**Descrizione:**
ModificaPasswordServlet.java	Permette di modificare la password.
ModificaDatiPersonaliServlet.java	Permette di modificare i dati personali.
RimozioneAccountServlet.java	Permette di richiedere la rimozione dell'account
ConfermaBibliotecaServlet.java	Permette di confermare la registrazione di nuova biblioteca.

2.1.3.2 Registrazione



Classe:	**Descrizione:**
RegistraUtenteServlet.java	Permette la registrazione di un nuovo utente
RegistrazioneSceltaServlet.java	Permette di scegliere il tipo di account da registrare
RegistraBibliotecaServlet.java	Permette la registrazione di una biblioteca.
RegistraBibliotecarioServlet.java	Permette di registrare un nuovo dipendente di una biblioteca.

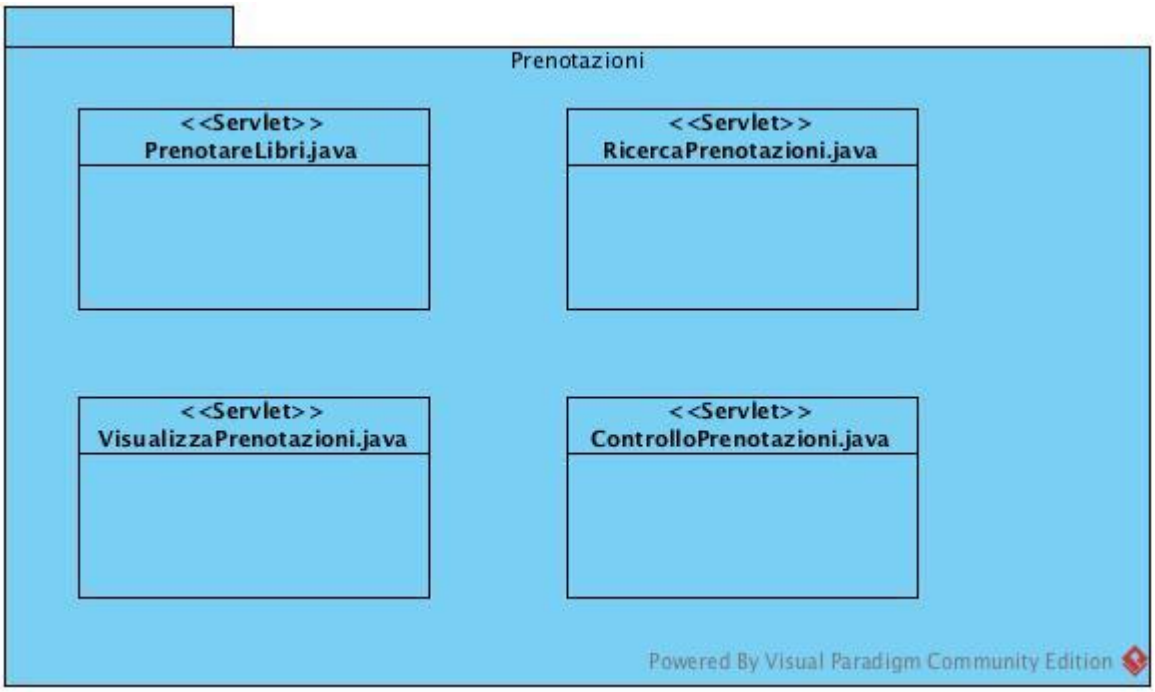
2.1.3.3 Libri



Classe:	**Descrizione:**
AggiuntaCopieServlet.java	Permette di aggiungere copie e creare scaffali in una biblioteca al momento dell'aggiunta di un nuovo libro.
AggiuntaLibroServlet.java	Permette l'aggiunta di un libro.
AggiuntaPosizioneServlet.java	Permette l'aggiunta di uno scaffale in una biblioteca.
AggiungiCopiaServlet.java	Permette di aggiungere una singola copia ad uno scaffale di una biblioteca.
CercaLibroServlet.java	Permette la ricerca di un libro.
EliminaCopiaServlet.java	Permette l'eliminazione di una copia.
EliminaLibroServlet.java	Permette l'eliminazione di un libro.
EliminaPosizioneServlet.java	Permette l'eliminazione di uno scaffale.
GestisciCopieServlet.java	Permette la visualizzazione di tutti gli scaffali con le relative copie di un determinato libro.
SpostaCopieServlet.java	Permette di spostare una copia da uno scaffale ad un altro nella singola biblioteca.
VerificaIsbnServlet.java	Permette di verificare tramite l'isbn se il libro è presente nel database globale.

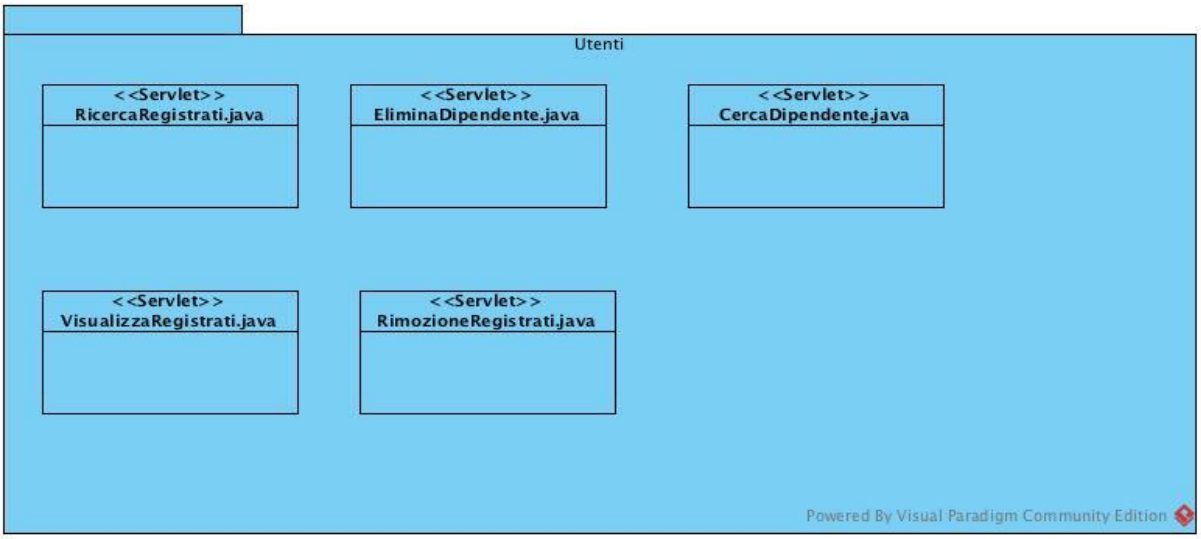
VisualizzaLibroServlet.java	Permette di visualizzare un libro e tutte le sue informazioni
VisualizzaPosizioniServlet.java	Permette la visualizzazione di tutti gli scaffali di una biblioteca.

2.1.3.4 Prenotazioni



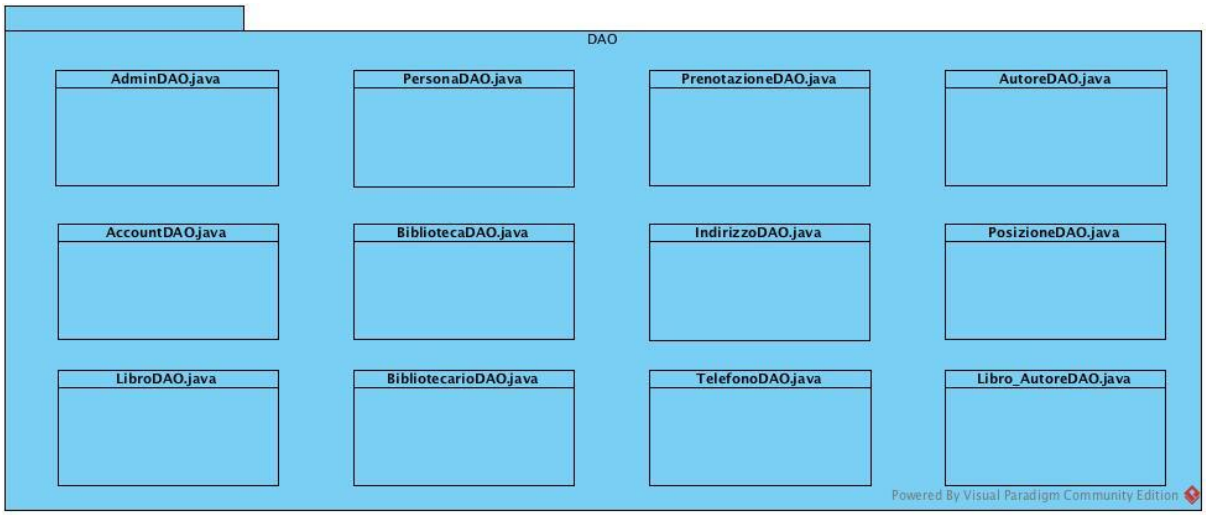
Classe:	**Descrizione:**
PrenotazioneLibroServlet.java	Permette di prenotare un libro.
RicercaPrenotazioniPerCriterioServlet.java	Permette la ricerca di un libro tramite un criterio.
VisualizzaPrenotazioniServlet.java	Permette di visualizzare una lista di prenotazioni.
AnnullaPrenotazioneServlet.java	Permette di annullare una prenotazione.
ConfermaRestituzioneLibroServlet.java	Permette di restituire un libro alla biblioteca.
ConfermaRitiroLibroServlet.java	Permette il ritiro di un libro dalla biblioteca.
DettagliPrenotazioneServlet.java	Permette la visualizzazione dei dettagli di una prenotazione.

2.1.3.5 Utenti



Classe:	**Descrizione:**
RicercaRegistratiServlet.java	Permette di ricercare un registrato al sito.
EliminaDipendenteServlet.java	Permette l'eliminazione di un dipendente della biblioteca.
CercaDipendenteServlet.java	Permette di cercare un dipendente di una biblioteca.
RimozioneRegistratiServlet.java	Permette la rimozione di un utente registrato.

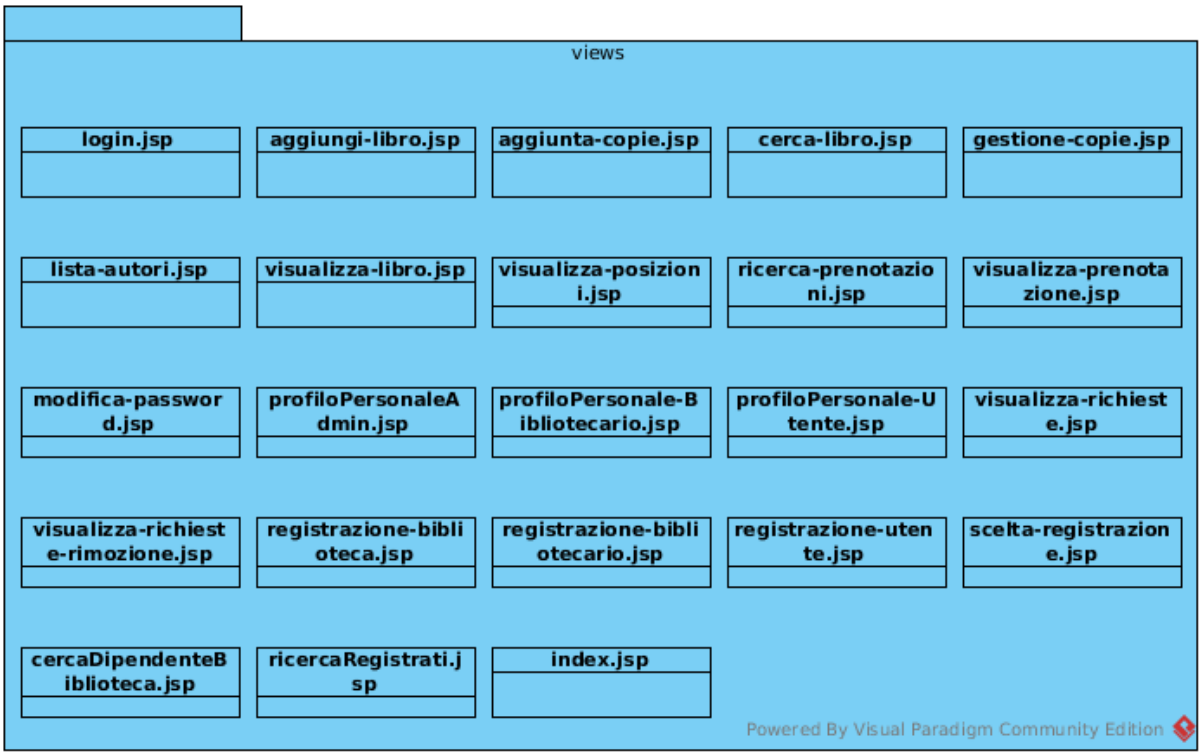
2.1.4 PackagesDAO



Classe:	**Descrizione:**
AdminDAO.java	Permette di interagire con la tabella Admin all'interno del database.
PersonaDAO.java	Permette di interagire con la tabella Persona all'interno del database.
PrenotazioneDAO.java	Permette di interagire con la tabella Prenotazione all'interno del database.
AutoreDAO.java	Permette di interagire con la tabella Autore all'interno del database.
AccountDAO.java	Permette di interagire con la tabella Account all'interno del database.
BibliotecaDAO.java	Permette di interagire con la tabella Biblioteca all'interno del database.
IndirizzoDAO.java	Permette di interagire con la tabella Indirizzo all'interno del database.
PosizioneDAO.java	Permette di interagire con la tabella Posizione all'interno del database.
LibroDAO.java	Permette di interagire con la tabella Libro all'interno del database.
BibliotecarioDAO.java	Permette di interagire con la tabella Bibliotecario all'interno del database.

TelefonoDAO.java	Permette di interagire con la tabella Telefono all'interno del database.
Libro_AutoreDAO.java	Permette di interagire con la tabella Libro_Autore all'interno del database.
CopiaDAO.java	Permette di interagire con la tabella Copia all'interno del database.

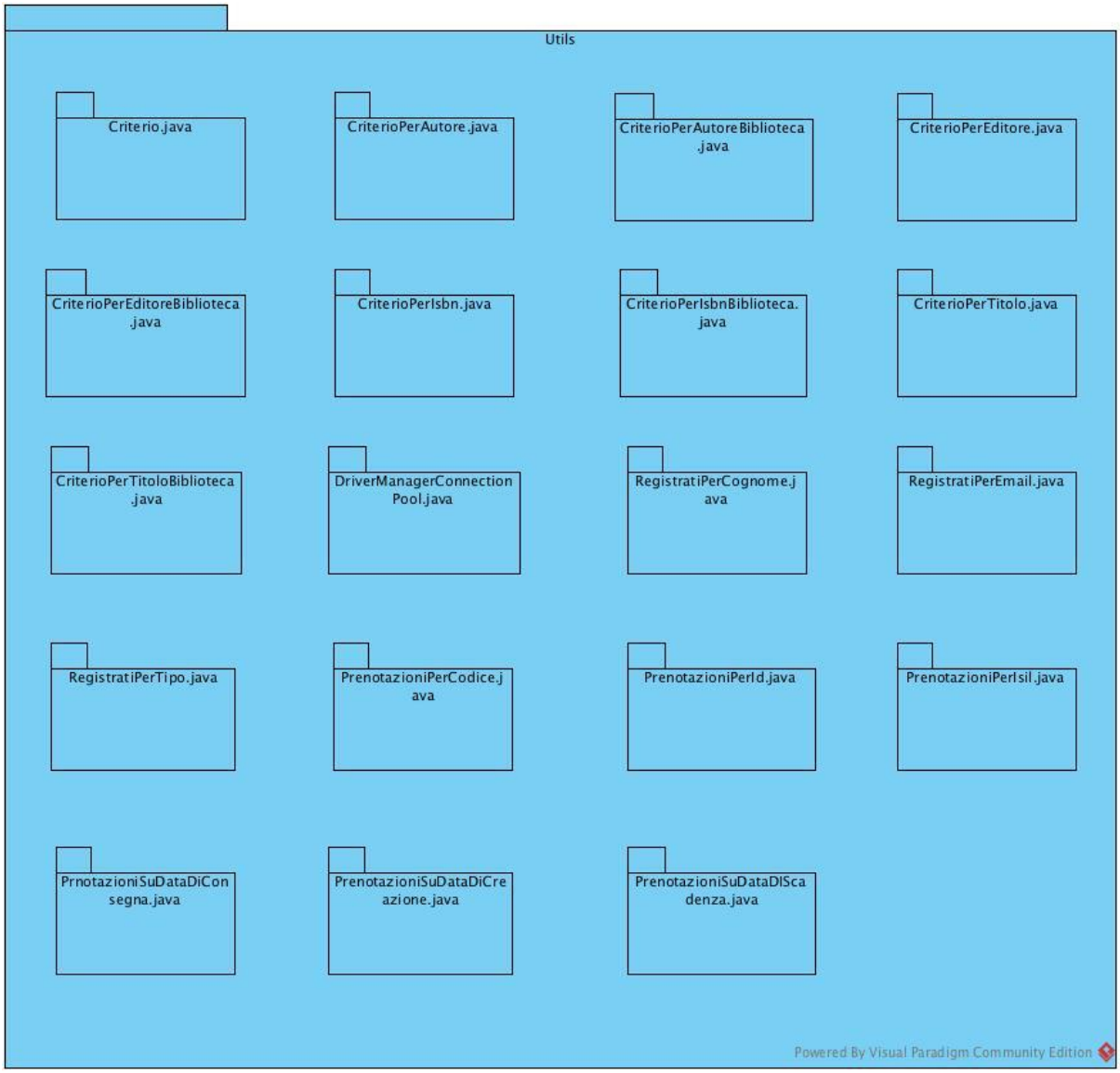
2.1.5 Packages View



Classe:	**Descrizione:**
login.jsp	Visualizza la pagina che permette di effettuare il login.
index.jsp	Visualizza la pagina iniziale del sistema.
profiloPersonale-Admin.jsp	Visualizza la pagina che permette di visualizzare e modificare il profilo personale dell'admin.
profiloPersonale-Bibiliotecario.jsp	Visualizza la pagina che permette di visualizzare e modificare il profilo personale del bibliotecario e qualora questo fosse un "Responsabile" anche quello della biblioteca .
profiloPersonale-Utente.jsp	Visualizza la pagina che permette di visualizzare e modificare il profilo personale dell'utente.
modifica-password.jsp	Visualizza la pagina che permette di modificare la propria password.
visualizza-richieste.jsp	Visualizza la pagina che permette di visualizzare le richieste di registrazione al sito.
visualizza-richiesteRimozione.jsp	Visualizza la pagina che permette di visualizzare le richieste di rimozione al sito.
registrazione-biblioteca.jsp	Visualizza la pagina che permette di registrare una biblioteca con il relativo responsabile al sito.
registrazione-utente.jsp	Visualizza la pagina che permette di registrare una persona al sito.
registrazione-bibliotecario.jsp	Visualizza la pagina che permette di registrare un nuovo dipendente di una biblioteca.
scelta-registrazione.jsp	Visualizza la scelta del tipo di account da registrare.
cerca-libro.jsp	Visualizza la pagina che permette di cercare un libro.
aggiunta-copie.jsp	Visualizza la pagina che permette di aggiungere copie di un libro ad una biblioteca.

aggiunta-libro.jsp	Visualizza la pagina che permette di aggiungere un libro ad una biblioteca.
gestione-copie.jsp	Visualizza la pagina che permette di gestire le copie di un libro.
lista-autori.jsp	Visualizza la lista di tutti gli autori dei libri presenti sul database.
visualizza-posizioni.jsp	Visualizza la pagina per gestire tutte le posizioni delle copie di una biblioteca.
visualizza-libro.jsp	Visualizza la pagina che permette di visualizzare i dettagli di un libro.
ricerca-prenotazioni.jsp	Visualizza la pagina che permette di ricercare una prenotazione.
visualizza-prenotazioni.jsp	Visualizza la pagina che permette di visualizzare le prenotazioni.
ricercaRegistrati.jsp	Visualizza la pagina che permette di ricercare i registrati.
cercaDipendenteBiblioteca.jsp	Visualizza la pagina che permette di cercare il dipendente di una biblioteca.

2.1.6 Package Utils



Classe:	**Descrizione:**
Criterio.java	Interfaccia che implementa un unico metodo usato per effettuare le ricerche.
CriterioPerAutore.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri in base all'autore.

CriterioPerAutoreBiblioteca.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri, all'interno di una biblioteca, in base all'autore.
CriterioPerEditore.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri in base all'editore.
CriterioPerEditoreBiblioteca.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri, all'interno di una biblioteca, in base all'editore.
CriterioPerIsbn.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri in base all'isbn.
CriterioPerIsbnBiblioteca.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri, all'interno di una biblioteca, in base all'isbn.
CriterioPerTitolo.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri in base al titolo.
CriterioPerTitoloBiblioteca.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare i libri, all'interno di una biblioteca, in base al titolo.
DriverManagerConnectionPool.java	Classe che implementa un pool di connessioni al database in maniera da ottimizzarne l'accesso.
RegistratiPerCognome.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare gli utenti registrati (persone e bibliotecari) in base al cognome.
RegistratiPerEmail.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare gli utenti registrati (persone e bibliotecari) in base all'email.
RegistratiPerTipo.java	Classe che implementa l'interfaccia Criterio. Viene usata per cercare gli utenti registrati (persone e bibliotecari) in base al tipo.

PrenotazioniPerCodice.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base al codice.
PrenotazioniPerId.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base all’id della persona.
PrenotazionePerIsil.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base all’isil della biblioteca.
PrenotazioniSuDataDiConsega.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base alla data di consegna.
PrenotazioniSuDataDiCreazioni.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base alla data di creazione.
PrenotazioniSuDataDiScadenza.java	Classe che implementa l’interfaccia Criterio. Viene usata per cercare prenotazioni in base alla data di scadenza.

3.Interfaccia delle classi

3.1 Package Manager

3.1.0 Manager Autenticazione

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione.

- public Utente login(String email, String password)
  Questo metodo permette di effettuare l'accesso.
- public boolean logout(HttpSession session)
  Questo metodo permette di effettuare il logout.

3.1.1 Manager Account

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account.

- public boolean modificaPassword(String email, String newPsw)
  Questo metodo permette di modificare la password corrente dell'utente e ritorna un booleano.

- public Bibliotecario modificaDatiPersonali(String vecchiaMail, String email, String path_foto)
  Questo metodo permette ad un Bibliotecario di modificare i suoi dati personali.

- public Persona modificaDatiPersonali(String vecchiaMail, String email, String nome, String cognome, String numDocumento, String provincia, String cap, String via, String numeroCivico, String citta, String recapitoTelefonico, String path_foto)
  Questo metodo permette ad una Persona di modificare i suoi dati personali.

- public Admin modificaDatiPersonali(String vecchiaMail, String email) {
  Questo metodo permette di modificare i dati personali dell'utente Amministratore, e ritorna un oggetto Amministratore.

- public Biblioteca modificaDatiBiblioteca(String isil, String nome, String via, String citta, String numeroCivico, String numeroTelefono, String provincia, String cap)
  Questo metodo permette ad un bibliotecario di modifica i dati relativi alla biblioteca.

- public boolean recuperaPassword(String email, String nuovaPassword)
  Questo metodo permette di recuperare la password.

- public boolean richiestaRimozioneAccount(String isil)
  Questo metodo permette di creare una richiesta di rimozione di un account da parte di una bibloteca.

- public boolean rimozioneAccountUtente(String email)
  Questo metodo permette di rimuovere un account utente.

3.1.2 Manager Registrazione

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account

- public Persona registra (String nome,String cognome,String email,String numeroDocumento,String via,String citta,String numeroCivico,String password,String pathFoto, String provincia,String CAP,String numeroTelefono)
  Questo metodo permette ad una Persona di registrarsi sul sito, e creare quindi un account.

- public Biblioteca registra(String isil,String nomeBiblioteca,String via,String citta,String numeroCivico,String provincia, String CAP,String numeroTelefono)
  Questo metodo permette di effettuare la registrazione da parte di una Biblioteca, e quindi creare un account.

- public Bibliotecario registraDipendente(String isil,String nome,String cognome,String email,String password,String path_foto,String tipo)
  Questo metodo permette ad un Bibliotecario di registrare un dipendente della sua biblioteca,quindi creando un account.

- public List<Biblioteca> visualizzaRichieste()
  Questo metodo permette di far visualizzare all"amministratore una lista di richieste che deve confermare o rifiutare.

- public boolean modificaStatoBiblioteca(String idBiblioteca,String change,int id_admin)
  Questo metodo permette di modificare lo stato di una biblioteca.

- public int checkEmail(String email)
  Questo metodo permette di verificare se l'email è già registrata.

- public Telefono recuperaTelefonoPersona(Persona persona)
  Questo metodo permette di recuperare il telefono di una persona.

- public Telefono recuperaTelefonoBiblioteca(Biblioteca biblioteca)
  Questo metodo permette di recuperare il telefono di una persona.

3.1.3 Manager Libri

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account

- public Collection<Libro> cercaLibro(Criterio c, int offset)
  Questo metodo permette la ricerca di un libro all'interno di una o più biblioteche a partire dalla posizione indicata dalla variabile offset.

- public boolean modificaDisponibilita(String isbn,String idBiblioteca,boolean flag)
  Questo metodo permette di modificare la disponibilità di un libro all'interno di una biblioteca.

- public Libro aggiuntaLibro(String isbn, String titolo, String editore, Calendar dataPubblicazione, String descrizione, String pathFoto, List<Autore> autori)
  Questo metodo permette l'aggiunta di un libro all'interno del database di libri.

- public boolean aggiungiLibroBiblioteca(String isil, Libro book, Collection<Posizione> posizioni)
  Metodo che aggiunge un libro esistente nel database ad una biblioteca (creando scaffali e copie).

- public Copia aggiungiCopia(String isbn,String isil,String idPosizione, String idCopia)
  Questo metodo permette di aggiungere una copia di un libro che in una specifica posizione.

- public boolean eliminaCopia(String isbn,String isil,String idPosizione, String idCopia)
  Questo metodo permette di eliminare una copia di un libro da una posizione.

- public boolean eliminaLibro(String isbn,String isil)
  Questo metodo permette di eliminare un libro all'interno di una biblioteca. (Ovvero elimina tutte le sue copie)

- public boolean spostaCopia(String isbn,String isil,String idCopia,Posizione vecchiaPosizione,Posizione nuovaPosizione)
  Questo metodo permette di spostare una copia da una posizione ad un' altra all'interno di una biblioteca.

- public Libro visualizzaLibro(String isbn)
  Questo metodo permette di visualizzare uno specifico libro.

- public Posizione aggiungiPosizione(String etichetta, String isil)
  Questo metodo permette di aggiungere una posizione all'interno della biblioteca.

- public Collection<Posizione> visualizzaPosizioniLibro(String isbn, String isil)
  Questo metodo permette di visualizzare tutte le posizioni che contengono copie di un libro.
- public List<Posizione> visualizzaPosizioniBiblioteca(String isil)
  Questo metodo permette di visualizzare tutti gli scaffali di una biblioteca.

- public boolean eliminaPosizione(String etichetta, String isil)
  Questo metodo permette di eliminare uno scaffale vuoto in una biblioteca.
- public Collection<Biblioteca> getBibliotecheConLibro(String isbn)
  Questo metodo permette di visualizzare tutte le biblioteche che possiedono ed ed hanno disponibile un determinato libro.

3.1.4 Manager Prenotazioni

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account

- public Prenotazione prenotareLibro(Persona p,String isbn,String isil)
  Questo metodo permette di prenotare un libro da parte dell'utente loggato alla piattaforma.

- public Collection<Prenotazione> visualizzaPrenotazioni(Criterio cp)
  Questo metodo permette di visualizzare una lista di prenotazioni relative ad una biblioteca o ad un utente specifico.

- public boolean controlloPrenotazione(String idPrenotazione,String email, String status)
  Questo metodo permette di controllare le prenotazioni effettuate ad una biblioteca da parte di un utente.

- public Prenotazione visualizzaPrenotazione(int idPrenotazione)
  Questo metodo permette di visualizzare una prenotazione specifica.

3.1.5 Manager Utenti

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli utenti

- public Collection<Utente> visualizzaRegistrati(Criterio cr)
  Questo metodo permette di visualizzare la lista di registrati al sito.

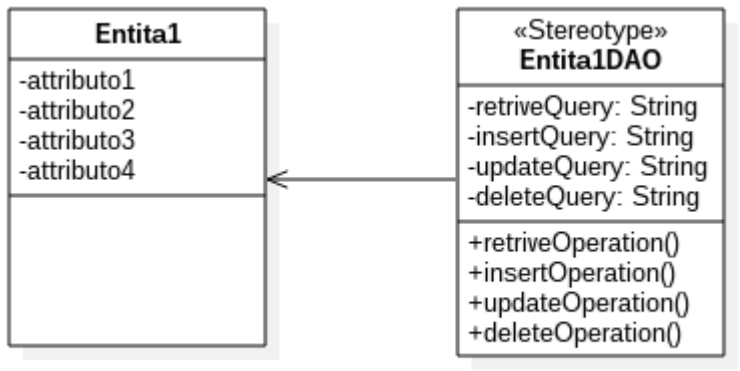
- public boolean eliminaRegistrati(String email)
  Questo metodo permette di eliminare un utente o una biblioteca dal sito.

- public Collection<Bibliotecario> visualizzaDipendenti(Criterio cr, String isil)
  Questo metodo permette di visualizzare una lista di dipendenti di una specifica biblioteca da parte del responsabile di biblioteca.

4.Design Pattern

4.1 - Dao Pattern

Per l’accesso al database è stato impiegato il pattern DAO (Data Access Object). Il pattern è usato per separare le API, di basso livello, di accesso ai dati, dalla logica di business di alto livello. Per ogni tabella presente nel DB esisterà una sua corrispondente classe DAO che possiederà metodi per effettuare le operazioni CRUD. Ogni classe DAO utilizzerà esclusivamente la classe Entity corrispondente alla tabella su cui effettua le operazioni.
Di seguito è presentato un modello di utilizzo del pattern:



4.2 - Façade Pattern

Per la realizzazione dei servizi dei sottosistemi è stato usato il pattern Façade. Il pattern si basa sull'utilizzo di un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Nel nostro caso, ogni sottosistema, identificato nell'SDD, possiede una classe chiamata manager. Questa implementa dei metodi che corrispondono ai servizi offerti dal sottosistema. tali metodi, nella loro implementazione, utilizzeranno le classi entity e le classi DAO per eseguire il servizio da essi offerto. In output presenteranno un oggetto che servirà alla servlet per la presentazione del risultato dell'operazione. In questo modo si svincola completamente la logica di business, implementata dai manager, e la logica di controllo, implementata dalla servlet. Viene qui presentato un modello di utilizzo del pattern: