

Object Design Document

Green Leaf

|  |  |
| --- | --- |
| Riferimento |  |
| Versione |  |
| Data |  |
| Destinatario | Prof.ssa F. Ferrucci,  Prof. F. Palomba |
| Presentato da | Alessandro Borrelli,  Vincenzo Cerciello,  Michela Faella,  Gerardo Napolitano,  Mirko Vitale |
| Approvato da |  |

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 23/12/2022 | 0.1 | Stesura del punto 1 e 2 | Team |
| 27/12/2022 | 0.2 | Stesura dei rimanenti capitoli | Team |

**Sommario**

[Revision History 2](#_bookmark0)

1. [Introduzione 5](#_bookmark2)
   1. [Object design goals 5](#_bookmark3)
   2. [Linee guida per la documentazione dell’interfaccia 5](#_bookmark4)
   3. [Definizioni, acronimi, e abbreviazioni 5](#_bookmark5)
   4. [Riferimenti 6](#_bookmark6)
2. [Packages 6](#_bookmark7)
3. [Class Interfaces 12](#_bookmark8)
4. [Design Patterns 23](#_bookmark9)
5. [Glossario 27](#_bookmark10)

# **Introduzione**

### Object design trade-offs

Durante la fase di Object Design sono sorti diversi compromessi di progettazione, di seguito sono riportati i trade-off:

## **Prestazioni vs Costi**

Tenuto conto del budget stanziato, e dalla necessità di avere funzioni chiave perfettamente operative nei tempi prestabiliti, si preferisce dedicare il monte ore a disposizione all’implementazione e revisione di quest’ultime.

## **Prestazioni vs Affidabilità**

Dovendo il sistema gestire dati sensibili, si preferisce garantire un maggior controllo di input e consistenza a scapito della latenza.

### Componenti off-the-shelf

### Green leaf farà affidamento su varie componenti off-the-shelf sia per il front-end che per il back-end.

### Per quanto riguarda il front-end verranno usate le librerie……

### Lato back-end saranno utilizzate le seguenti componenti off-the-shelf: MySQL, Tomcat, Maven.

### MySQL è un celebre RDBMS di Oracle. Il suo utilizzo sarà fondamentale per la realizzazione, la gestione e l'implementazione della base di dati di GreenLeaf. Tutti i dati che per necessità di business devono essere salvati persistentemente saranno affidati ad un database basato su MySQL. Sarà inoltre utilizzata la componente MySQL Connector/J , il driver JDBC ufficiale per MySQL necessario per la comunicazione con la base di dati da Java.

### Tomcat è un server web open-source sviluppato dalla Apache Software Foundation. Il suo impiego è necessario, poiché tutto il back-end di GreenLeaf (nucleo di elaborazione principale, comunicazione con la base di dati) sarà eseguito in ambiente Tomcat. Esso permetterà di utilizzare le Java Servlet per l'implementazione della logica di business e le Java Servlet Pages per la generazione dinamica delle pagine web utilizzate dal client. Maven è uno strumento di gestione dei progetti software basati su Java, anch'esso sviluppato dalla Apache Software Foundation. Agevolerà la gestione delle varie librerie utilizzate, delle loro versioni e delle dipendenze tra esse, permettendo agli sviluppatori di concentrarsi esclusivamente sulla scrittura del codice.

### 1.3 Linee guida per la documentazione dell’interfaccia

### È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell’intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

|  |  |  |
| --- | --- | --- |
| Tipi | Regole per la denominazione | Esempi |
| Package | Il prefisso di un nome di pacchetto univoco sempre scritto in minuscolo e tutte le lettere ASCII minuscole tranne le abbreviazioni come UI che sarebbe User Interface in modo da non rendere il nome del pacchetto lungo | package greenleaf;package pluto; |
| Classi | I nomi delle classi dovrebbero essere sostantivi, scritti in CamelCase. Abbiamo mantenuto i nomi delle nostre classi semplici e descrittivi. Tuttavia, abbiamo utilizzato parole intere ed evitato per quanto possibile acronimi e abbreviazioni (a meno che l'abbreviazione non sia molto più utilizzata rispetto alla forma lunga, come DB per database e UI per interfaccia utente) | class UtenteDB;class Tracciamento; |
| Interfacce | I nomi delle interfacce devono essere scritti in CamelCase come i nomi delle classi. | interface Identificativo; |
| Metodi | I nostri metodi sono verbi, scritti nella forma camelCase (da notare la prima lettera minuscola). | insert();createReport(); |
| Variabili | I nomi delle variabili sono brevi ma significativi. La scelta del nome di una variabile è mnemonica, cioè finalizzata a indicare all'osservatore casuale l'intento del suo utilizzo. I nomi delle variabili composte da una sola lettera vengono evitati tranne che per le variabili "usa e getta" temporanee. Se i nomi delle varabili sono composti da due nomi allora verranno scritti nella forma camelCase (da notare la prima lettera minuscola). | int i;string nomeRegione;float costo; |
| Costanti | I nomi delle variabili dichiarate costanti di classe e delle costanti ANSI devono essere tutti maiuscoli con parole separate da trattini bassi ("\_"). (Le costanti ANSI dovrebbero essere evitate, per facilitare il debug.) | static finali int MAX = 5; |

### Riferimenti

Bernd Bruegge, Allen H. Dutoit - Object-Oriented Software Engineering

GL\_SDD\_V\_2

# Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

##### .idea

* **.mvn**, contiene tutti i file di configurazione per Maven
* **src**, contiene tutti i file sorgente

##### main

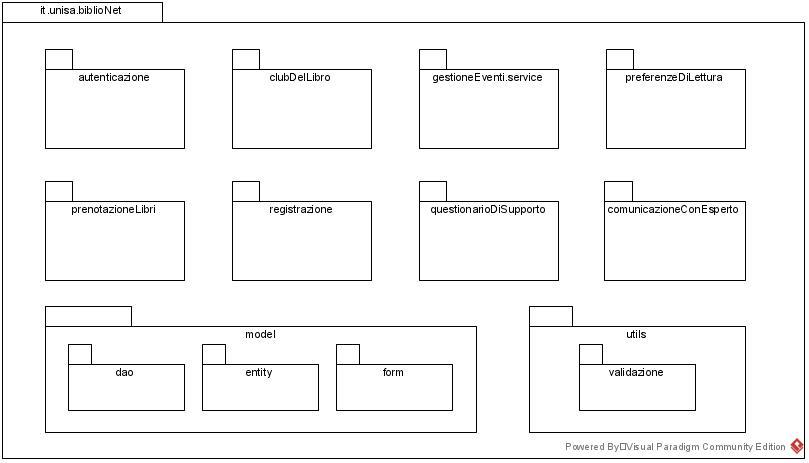
* + - **java**, contiene le classi Java relative alle componenti Control e Model
    - **resources**, contiene i file relativi alle componenti View
      * **static**, contiene i fogli di stile CSS e gli script JS
      * **templates**, contiene i file HTML renderizzati da Thymeleaf
  + **test**, contiene tutto il necessario per il testing
    - **java**, contiene le classi Java per l’implementazione del testing
* **target**, contiene tutti i file prodotti dal sistema di build di Maven

#### Package biblioNet

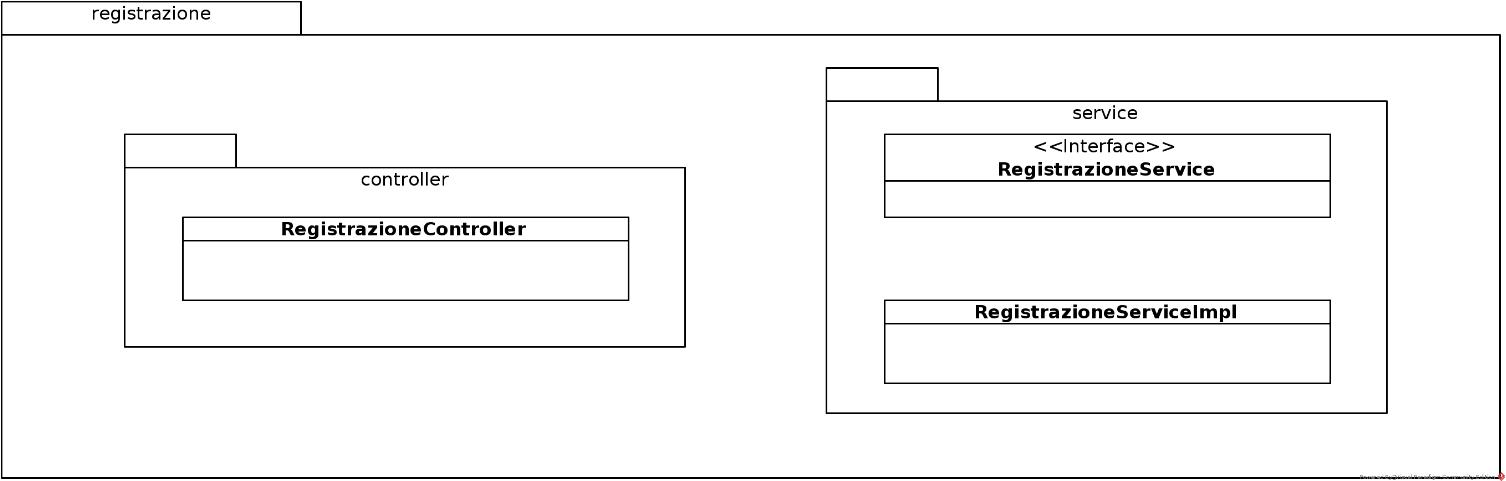
Nella presente sezione si mostra la struttura del package pricipale di biblioNet. La struttura generale è stata ottenuta a partire da tre principali scelte:

1. Creare un package separato per ogni sottosistema, contenente le classi service e controller del sottosistema, ed eventuali classi di utilità usate unicamente da esso.
2. Creare un package separato per le classi del *model*, contenente le classi entity e i DAO per l’accesso al DB. Tale scelta è stata presa vista l’elevata complessità del database di BiblioNet che prevede numerose relazioni tra le entità. Si è quindi preferito tenere tutto in un package separato e collegato a tutti gli altri package dei sottosistemi.
3. Creare un package chiamato *utils* in cui inserire eventuli classi di utlità per il sistema e usabili da più sottosistem.

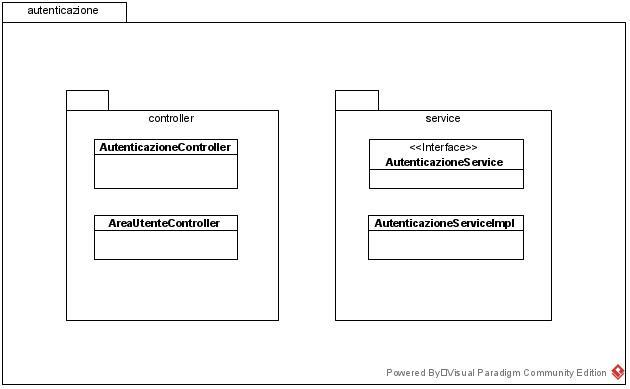
Per ciò che concerne la dipendenza tra i packages, la suddivisione precedentemente illustrata a portato alla creazione di una relazione tra il package model e tutti gli altri package del sistema.



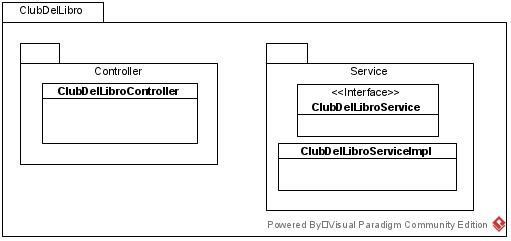
#### Package Registrazione



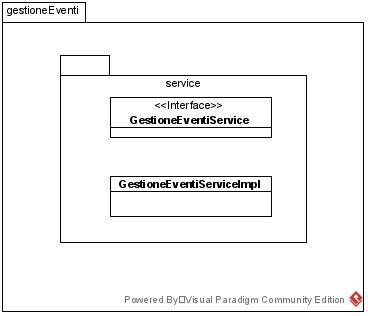
#### Package Autenticazione



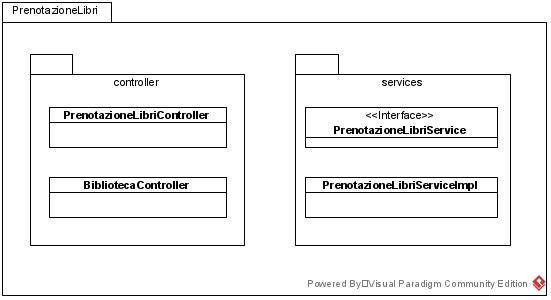
#### Package Club del Libro



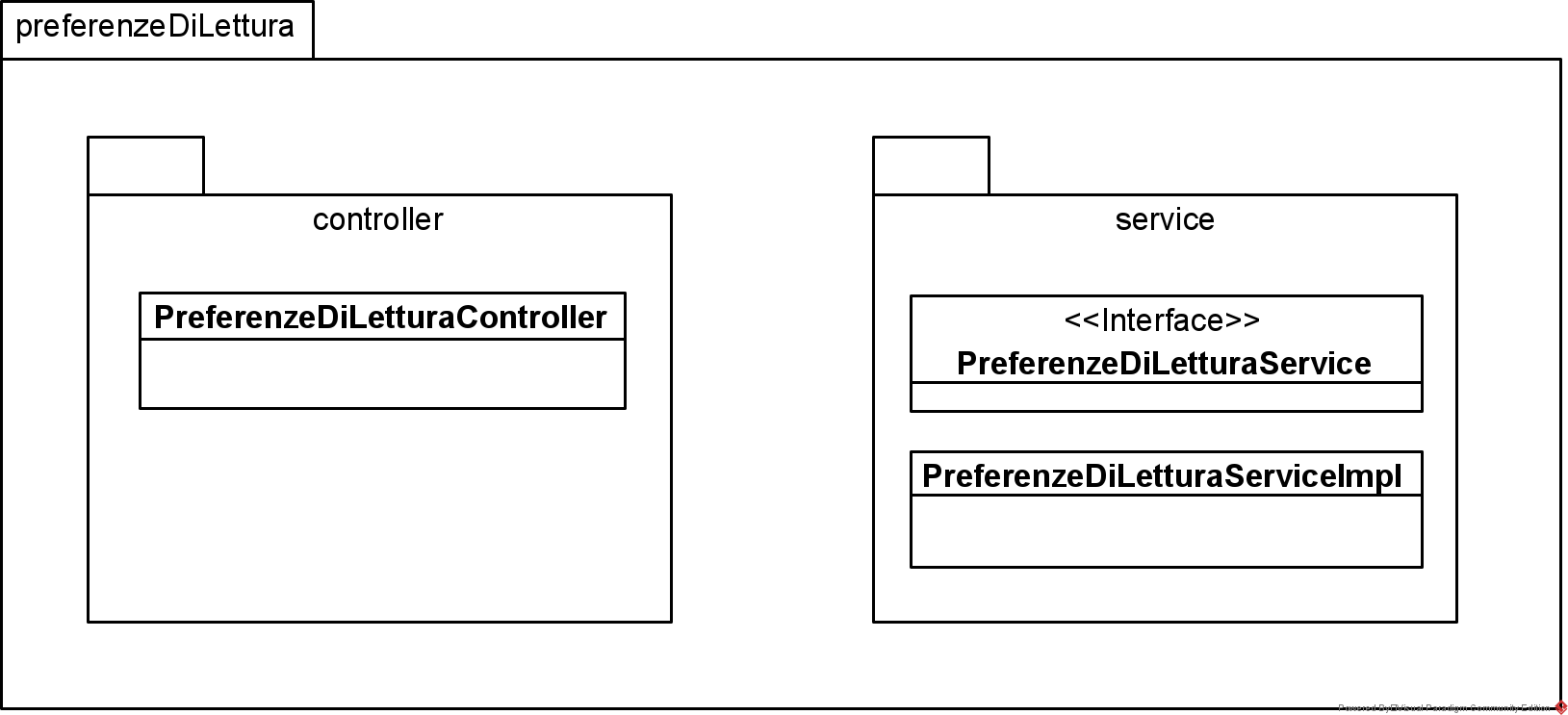
#### Package Gestione eventi



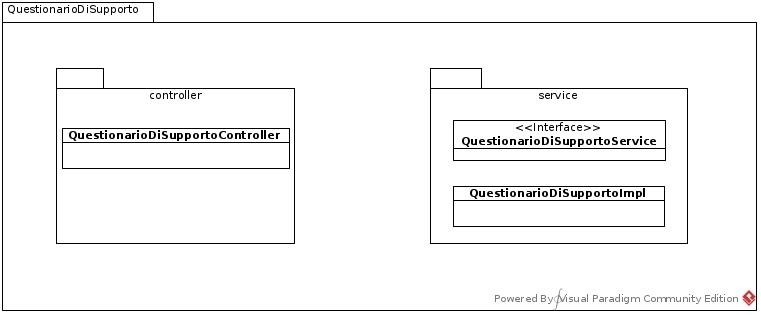
#### Package Prenotazione libri



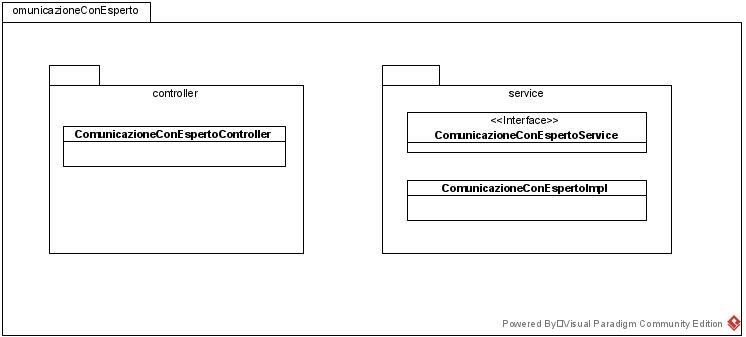
#### Package Preferenze di lettura



#### Package Questionario di supporto



#### Package Comunicazione con esperto



# Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package. [Non sarà il package model, le classi controller]

#### Javadoc di BiblioNet

Per motivi di leggibilità si è scelto di creare un sito, hostato tramite GitHub pages, contenente la JavaDoc di BiblioNet. In tale maniera, chiunque può consultare la documentazione aggiornata dell’intero sistema.

Di seguito, il link al sito in questione: <https://stefanolambiase.github.io/biblionet/>

#### Package Registrazione

|  |  |
| --- | --- |
| **Nome classe** | RegistrazioneService |
| **Descrizione** | Questa classe permette di gestire le operazioni relative alla registrazione. |
| **Metodi** | +registraEsperto(Esperto esperto): UtenteRegistrato  +registraBiblioteca(Biblioteca biblioteca): UtenteRegistrato  +registraLettore(Lettore lettore): UtenteRegistrato |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +registraLettore(Lettore lettore) |
| **Descrizione** | Questo metodo consente di registrare un nuovo lettore. |
| **Pre-condizione** | / |
| **Post-condizione** | **context:** RegistrazioneService:: registraLettore(Lettore lettore)  **post:** LettoreDAO.save(lettore) == true |
| **Nome Metodo** | +registraEsperto(Esperto esperto) |
| **Descrizione** | Questo metodo consente di registrare un nuovo esperto. |
| **Pre-condizione** | / |

|  |  |
| --- | --- |
| **Post-condizione** | **context:** RegistrazioneService:: registraEsperto(Esperto esperto)  **post:** EspertoDAO.save(esperto) == true |
| **Nome Metodo** | +registraBiblioteca(Biblioteca biblioteca) |
| **Descrizione** | Questo metodo consente di registrare una nuova biblioteca. |
| **Pre-condizione** | / |
| **Post-condizione** | **context:** RegistrazioneService:: registraLettore(Lettore lettore)  **post:** BibliotecaDAO.save(biblioteca) == true |

#### Package Autenticazione

### it.unisa.biblioNet.autenticazione.service.AutenticazioneService

|  |  |
| --- | --- |
| **Nome classe** | AutenticazioneService |
| **Descrizione** | Questa classe permette di gestire le operazioni relative all’autenticazione. |
| **Metodi** | +login(String email, String password) : UtenteRegistrato  +aggiornaBiblioteca(Biblioteca biblioteca) : Biblioteca  +aggiornaEsperto(Esperto esperto) : Esperto  +aggiornaLettore(Lettore lettore) : Lettore |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +login(String email, String password) |
| **Descrizione** | Questo metodo consente di loggare un utente registrato. |
| **Pre-condizione** | / |
| **Post-condizione** | **context:** AutenticazioneService::login(email, password) |

|  |  |
| --- | --- |
|  | **post:** isLettore(loggedUser)||isEsperto(loggedUser)||isBiblioteca( loggedUser)==true |
| **Nome Metodo** | +aggiornaBiblioteca(Biblioteca biblioteca) |
| **Descrizione** | Questo metodo consente di aggiornare i dati di un account biblioteca. |
| **Pre-condizione** | / |
| **Post-condizione** | / |
| **Nome Metodo** | +aggiornaEsperto(Esperto esperto) |
| **Descrizione** | Questo metodo consente di aggiornare i dati di un account Esperto. |
| **Pre-condizione** | / |
| **Post-condizione** | / |
| **Nome Metodo** | +aggiornaLettore(Lettore lettore) |
| **Descrizione** | Questo metodo consente di aggiornare i dati di un account Lettore. |
| **Pre-condizione** | / |
| **Post-condizione** | / |

#### Package Club del Libro

### it.unisa.biblionet.clubDelLibro.service.ClubDelLibroService

|  |  |
| --- | --- |
| **Nome classe** | ClubDelLibroService |
| **Descrizione** | Questa classe offre delle funzionalità utili alla gestione di un Club del Libro. |
| **Metodi** | +creaClub(ClubDelLibro club): ClubDelLibro |

|  |  |
| --- | --- |
|  | +modficaDatiClub(ClubDelLibro club): ClubDelLibro  +visualizzaClubsDelLibro(Predicate<ClubDelLibro> filtro): List<ClubDelLibro>  +partecipaClub(ClubDelLibro club, Lettore lettore): Boolean  +findAllByLettori(Lettore lettore):List<ClubsDelLibro> |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +creaClub(ClubDelLibro club): ClubDelLibro |
| **Descrizione** | Questo metodo consente ad un esperto di creare un Club del Libro. |
| **Pre-condizione** | **context**: ClubDelLibroService::creaClub(club)  **pre:** not visualizzaClubsDelLibro (null).contains(club) |
| **Post-condizione** | **context**: ClubDelLibroService::creaClub(club)  **post:** visualizzaClubsDelLibro(null).contains(club) and visualizzaClubsDelLibro (null).size =  @pre. visualizzaClubsDelLibro (null).size+1 |
| **Nome Metodo** | +modficaDatiClub(ClubDelLibro clubClubDelLibro) |
| **Descrizione** | Questo metodo consente di modificare i dati di un club del libro. |
| **Pre-condizione** | / |
| **Post-condizione** | / |
| **Nome Metodo** | +visualizzaClubsDelLibro(Predicate<ClubDelLibro> filtro): List<ClubDelLibro> |

|  |  |
| --- | --- |
| **Descrizione** | Questo metodo restituisce una lista di tutti I club del libro che soddisfano I filtra inseriti. Se viene passato un filtro null, restituisce tutti I club del libro. |
| **Pre-condizione** | / |
| **Post-condizione** | / |
| **Nome Metodo** | +partecipaClub(ClubDelLibro club, Lettore lettore): Boolean |
| **Descrizione** | Questo metodo permette di far partecipare un lettore ad un club del libro. |
| **Pre-condizione** | **context**: ClubDelLibro:: partecipaClub(club,lettore)  **pre:** not findAllByLettori(Lettore).contains(club) |
| **Post-condizione** | **context**: ClubDelLibro:: partecipaClub(club,lettore)  **post:** findAllByLettori(Lettore).contains(club) |

#### Package Gestione Eventi

### it.unisa.biblioNet.gestioneEventi.service.GestioneEventiService

|  |  |
| --- | --- |
| **Nome classe** | GestioneEventiService |
| **Descrizione** | Questa classe fornisce I metodi per effettuare le operazioni riguardanti la creazione e la gestione degli Eventi dei Club del Libro. |
| **Metodi** | +creaEvento(Evento evento): Evento  +modificaEvento(Evento evento): Evento  +eliminaEvento(Evento evento): Evento |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +creaEvento(Evento evento): Evento |
| **Descrizione** | Questo metodo permette di creare un Evento relativo a un Club del Libro. |
| **Pre-condizione** | **context:** GestioneEventoService::creaEvento(Evento evento)  **pre:** not visualizzaListaEventiClub(evento.club).includes(evento) |
| **Post-condizione** | **context:** GestioneEventoService::creaEvento(Evento evento)  **post:** visualizzaListaEventiClub(evento.club).includes(evento) and  visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size+1 |
| **Nome Metodo** | +modificaEvento(Evento evento): Evento |
| **Descrizione** | Questo metodo permette di modificare un Evento già esistente relativo a un Club del Libro. |
| **Pre-condizione** | **context:** GestioneEventoService::modificaEvento(Evento evento)  **pre:** visualizzaListaEventiClub(evento.club). includes(evento) |
| **Post-condizione** | **context:** GestioneEventoService::modificaEvento(Evento evento)  **post:** visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size |
| **Nome Metodo** | +cancellaEvento(Evento evento): Evento |
| **Descrizione** | Questo metodo permette la cancellazione di un Evento già esistente relativo a un Club del Libro. |

|  |  |
| --- | --- |
| **Pre-condizione** | **context:** GestioneEventoService::cancellaEvento(Evento evento)  **pre:** visualizzaListaEventiClub(evento.club). includes(evento) |
| **Post-condizione** | **context:** GestioneEventoService::cancellaEvento(Evento evento)  **post:** not visualizzaListaEventiClub(evento.club).includes(evento) and  visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size-1 |

#### Package Prenotazione libri

### it.unisa.biblioNet.prenotazioneLibri.service.PrenotazioneService

|  |  |
| --- | --- |
| **Nome classe** | PrenotazioneService |
| **Descrizione** | Questa classe permette di gestire le operazioni relative alla prenotazione di libri. |
| **Metodi** | +aggiungiLibro(Libro libro): Libro  +getTicketsByBiblioteca(Utente biblioteca): List<TicketPrestito>  +richiediPrestito(Lettore lettore, String idBiblioteca, int idLibro)  +accettaRichiesta(TicketPrestito ticket): TicketPrestito  +rifiutaRichiesta(TicketPrestito ticket): TicketPrestito  +visualizzaListaLibriCompleta(): List<Libro> |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +inserimentoPerIsbn(String isbn,String idBiblioteca, int numCopie, List<String> generi): Libro |
| **Descrizione** | Questo metodo consente di aggiungere un libro ad una lista di libri tramite ISBN. |
| **Pre-condizione** | **context:**PrestitoLibriService::inserimentoPerIsbn(String isbn, String idBiblioteca, int numCopie, List<String> generi) **pre**:not visualizzaListaLibriCompleta(null).contains(libro) |
| **Post-condizione** | **context:**PrestitoLibriService::inserimentoPerIsbn(String isbn, String idBiblioteca, int numCopie, List<String> generi)  **post**: visualizzaListaLibriCompleta (null).contains(libro) and [visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibri](mailto:visualizzaListaLibriCompleta(null).size%3D@pre.visualizzaListaLibri) Completa(null).size+1 |
| **Nome Metodo** | +inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie |
| **Descrizione** | Questo metodo consente di aggiungere un libro di quelli già presenti nel database. |
| **Pre-condizione** | **context:**PrestitoLibriService::inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie)  **pre**: visualizzaListaLibriCompleta (null).contains(libro) |
| **Post-condizione** | **context:**PrestitoLibriService::inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie)  **post**: visualizzaListaLibriCompleta (null).contains(libro) and [visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibri](mailto:visualizzaListaLibriCompleta(null).size%3D@pre.visualizzaListaLibri) Completa(null).size+1 |
| **Nome Metodo** | +inserimentoManuale(Libro libro,String idBiblioteca, int numCopie, List<String> generi): Libro |
| **Descrizione** | Questo metodo consente di aggiungere un libro ad una lista di libri manualmente. |
| **Pre-condizione** | **context:**PrestitoLibriService::inserimentoManuale(Libro libro, String idBiblioteca, int numCopie, List<String> generi)  **pre**:not visualizzaListaLibriCompleta(null).contains(libro) |
| **Post-condizione** | **context:**PrestitoLibriService::inserimentoManuale(Libro libro, String idBiblioteca, int numCopie, List<String> generi) |

|  |  |
| --- | --- |
|  | **post**: visualizzaListaLibriCompleta (null).contains(libro) and [visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibri](mailto:visualizzaListaLibriCompleta(null).size%3D@pre.visualizzaListaLibri) Completa(null).size+1 |
| **Nome Metodo** | +richiediPrestito(Lettore lettore, String idBiblioteca, int idLibro) |
| **Descrizione** | Questo metodo consente di richiedere il prestito di un libro. |
| **Pre-condizione** | / |
| **Post-condizione** | / |
| **Nome Metodo** | +accettaRichiesta(TicketPrestito ticket, int giorni): TicketPrestito |
| **Descrizione** | Questo metodo consente di accettare una richiesta di prenotazione. |
| **Pre-condizione** | **context:**PrestitoLibriService::accettaRichiesta(ticket, giorni)  **pre**:visualizzaTicketsByBiblioteca().contains(ticket) |
| **Post-condizione** | **context:**PrestitoLibriService::accettaRichiesta(ticket)  **post**: not visualizzaTicketsByBiblioteca().contains(ticket) and [visualizzaRichiestePrestiti().size=@pre.visualizzaRichiestePrestit](mailto:visualizzaRichiestePrestiti().size%3D@pre.visualizzaRichiestePrestit) i(null).size-1 |
| **Nome Metodo** | +rifiutaRichiesta(Ticket ticket): Boolean |
| **Descrizione** | Questo metodo consente di rifiutare una richiesta di prenotazione. |
| **Pre-condizione** | **context:**PrestitoLibriService::accettaRichiesta(ticket) **pre**:visualizzaTicketsByBiblioteca().contains(ticket) |
| **Post-condizione** | **context:**PrestitoLibriService::accettaRichiesta(ticket)  **post**: not visualizzaTicketsByBiblioteca().contains(ticket) and [visualizzaTicketsByBiblioteca().size=@pre.visualizzaTicketsByBi](mailto:visualizzaTicketsByBiblioteca().size%3D@pre.visualizzaTicketsByBi) blioteca(null).size-1 |
| **Nome Metodo** | +visualizzaListaLibriCompleta(): List<Libro> |
| **Descrizione** | Questo metodo consente di visualizzare la lista dei libri prenotabile completa. |

|  |  |
| --- | --- |
| **Pre-condizione** | / |
| **Post-condizione** | / |

#### Package Preferenze di lettura

### it.unisa.biblioNet.preferenzeDiLettura.service.PreferenzeDiLetturaService

|  |  |
| --- | --- |
| **Nome classe** | PreferenzeDiLetturaService |
| **Descrizione** | Questa classe offre i metodi necessari per la gestione delle preferenze di lettura selezionabili da Lettore ed Esperto. |
| **Metodi** | +addGeneriEsperto(List<Genere> generi, Esperto esperto): void  +addGeneriLettore(List<Genere> generi, Lettore lettore). void |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +addGeneriEsperto(List<Genere> generi, Esperto esperto): void |
| **Descrizione** | Questo metodo si occupa di permettere ad un esperto di inserire un genere alla lista di quelli che conosce o preferisce. |
| **Pre-condizione** | **/** |
| **Post-condizione** | **context:** PreferenzeDiLetturaService::addGeneriEsperto(List<Genere> generi, Esperto esperto)  **post:** Arrays.compare(generi, esperto.getGeneri) == 0 |
| **Nome Metodo** | +addGeneriLettore(List>Genere> generi, Lettore lettore): void |

|  |  |
| --- | --- |
| **Descrizione** | Questo metodo si occupa di permettere ad un lettore di inserire un genere alla lista di quelli che conosce o preferisce. |
| **Pre-condizione** | **/** |
| **Post-condizione** | **context:** PreferenzeDiLetturaService::addGeneriLettore(List<Genere> generi, Lettore lettore)  **post:** Arrays.compare(generi, lettore.getGeneri) == 0 |

#### Package Questionario di supporto

#### it.unisa.biblioNet.QuestionarioDiSupporto.service.QuestionarioDiSupportoServi ce

|  |  |
| --- | --- |
| **Nome classe** | QuestionarioDiSupportoService |
| **Descrizione** | Questa classe offre i metodi per la compilazione da parte di un lettore e della gestione da parte dell’Amministratore del Questionario di Supporto.  Queste funzionalità saranno implementate in una futura release. |
| **Metodi** | +ConfermaQuestionario():boolean |
| **Invariante di classe** | / |

#### Package Comunicazione con esperto

### it.unisa.biblioNet.comunicazioneconEsperto.service.ComunicazioneServic e

|  |  |
| --- | --- |
| **Nome classe** | ComunicazioneService |
| **Descrizione** | Questa classe permette di gestire le operazioni relative alla comunicazione con un esperto. |

|  |  |
| --- | --- |
| **Metodi** | +visualizzaEspertiPerGenere():List<Esperto> |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | +visualizzaEspertiPerGenere():List<Esperto> |
| **Descrizione** | Questo metodo permette di visualizzare la lista di esperti per genere con cui è possibile iniziare una comunicazione. |
| **Pre-condizione** | / |
| **Post-condizione** | / |

# Design Patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo

dell’applicativo BiblioNet. Per ogni pattern si darà:

* + Una brevissima introduzione teorica.
  + Il problema che doveva risolvere all’interno di BiblioNet.
  + Una brevissima spiegazione di come si è risolto il problema in BiblioNet.
  + Un grafico della struttura delle classi che implementano il pattern.

#### Singleton e Observer

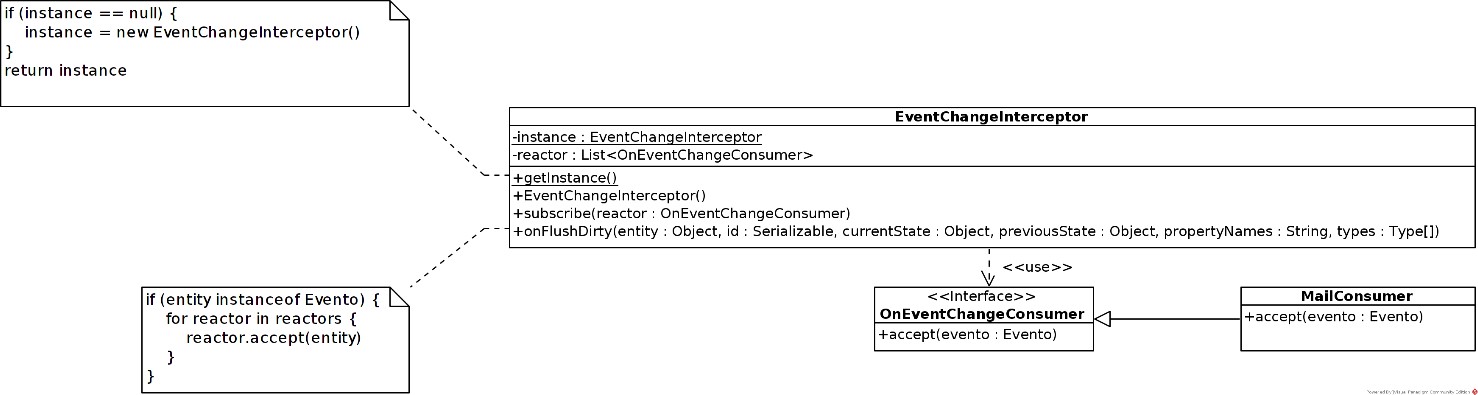
L’Observer è un design pattern comportamentale, ossia un design pattern che fornisce una soluzione a un problema di interazione tra più oggetti. Observer, nello specifico, permette a un oggetto “osservato” di generare (pubblicare) eventi ai quali una lista di oggetti “osservatori” (iscritti) possa poi reagire.

Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell’istanziazione degli oggetti, che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza.

BiblioNet prevede un sistema di notifica via email per la modifica degli eventi.

Per gestire ciò a livello implementativo senza accoppiare strettamente il sistema con un dato servizio di mailing o sistema di notifiche, viene creato un oggetto Singleton che conserva una propria lista di osservatori dotati di un metodo “accept” che accetti un singolo Evento come argomento. Ogni volta che viene modificato un evento sul database, l’oggetto, che è anche un intercettore di hibernate, “notifica” così la lista degli osservatori tramite il metodo onFlushDirty.

Ciò rende possibile la creazione in un momento successivo di multipli osservatori legati agli eventi (ad esempio per le notifiche push per un eventuale applicazione mobile).

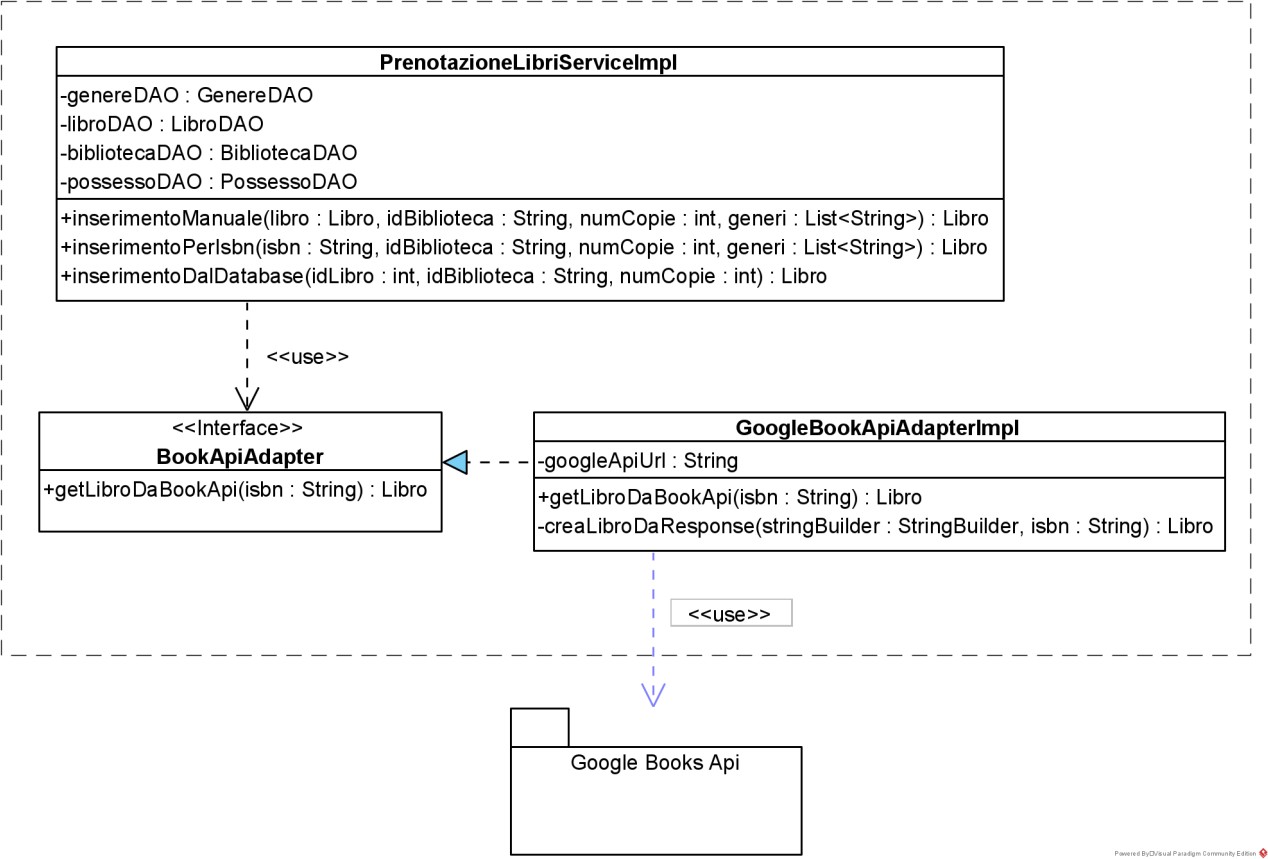


#### Adapter

L’Adapter è un design pattern strutturale, ovvero quei design pattern che facilitano la progettazione attraverso la semplificazione delle relazioni tra entità. Adapter, in particolare, permette ad oggetti con differenti interfacce di collaborare. Si implementa attraverso una classe “adapter”, che si occupa di convertire i dati in oggetti comprensibili dal sistema.

Biblionet si pone l’obiettivo di semplificare e digitalizzare l’interazione tra biblioteche e lettori. In questo ambito, offre anche la funzionalità di prenotazione di libri attraverso la piattaforma, il che presuppone che le biblioteche debbano rendere disponibile una lista di libri prenotabili dal sito stesso. Per l’inserimento dei libri all’interno di questo catalogo virtuale, oltre all’inserimento manuale, viene offerta ai gestori delle biblioteche la possibilità di inserire un libro tramite l’ISBN, senza doversi preoccupare di compilarne tutti i campi. Questa ricerca viene effettuata tramite un servizio esterno al sistema, in particolare il database di Google Books, attraverso una API offerta al pubblico sul web. Ovviamente l’API di ricerca dei libri, in quanto accessibile da chiunque, risponde alle query in un formato standard ed ampiamente utilizzato, ovvero il JSON.

Il design pattern Adapter si occupa della conversione dei dati di un libro da formato JSON ad oggetto della classe Libro, in modo da rendere le informazioni comprensibili dal sistema. La ricerca dei libri avviene in due fasi: la query all’API passando l’ISBN, e la traduzione del risultato grazie all’adapter.

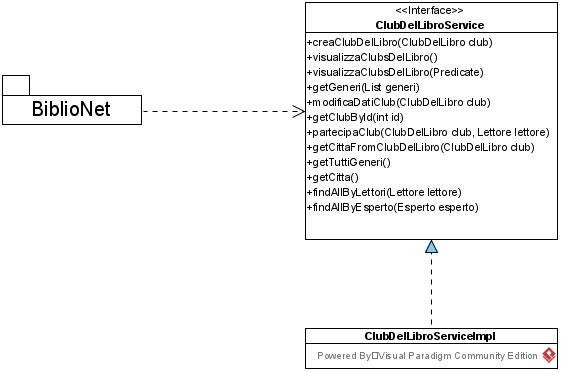


#### Facade

Il Facade è un design pattern che permette, implementando una interfaccia semplificata di accedere a sottosistemi più complessi. In questo modo si può nascondere al sistema la complessità delle librerie, dei framework o dei set di classi che si stanno usando. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l’implementazione dei metodi dell’interfaccia per implementare le modifiche.

BiblioNet, essendo un sistema molto complesso, sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile l’interfacciarsi con essa. Nello specifico BiblioNet utilizza il Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni.

Di seguito un esempio di Facade nel sistema BiblioNet:



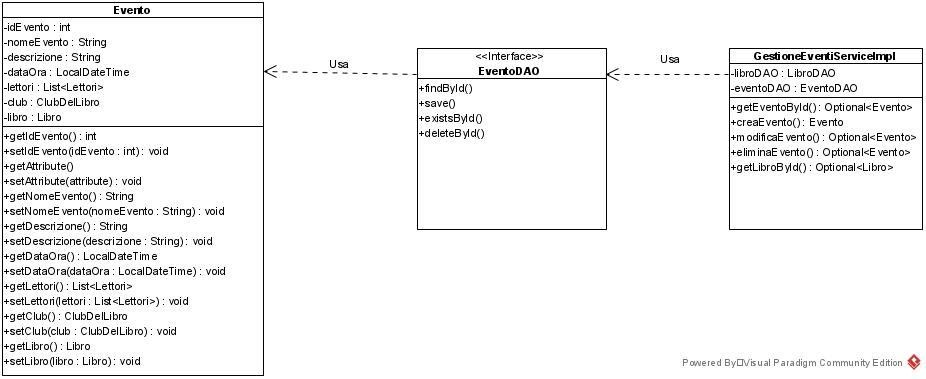
#### DAO

Un DAO (Data Access Object) è un pattern che offre un’interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell’applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali.

Essendo biblionet una web application che punta di gestire sia prenotazione di libri che numerosi club del libro presenta un database molto vasto, quindi ha bisogno di poter interagire con database in modo rapido e sicuro con numerosi query per quella che è la moltitudine di dati da gestire. Per questo motivo abbiamo usato varie interfacce DAO all’interno del nostro sistema le quali, grazie all’utilizzo del framework di Spring (SpringJPARepository), vengono implementate in maniera del tutto automatica e trasparente per il programmatore. Attraverso l’utilizzo di Spring JPA Repository viene quindi rimossa la necessità di avere una classe che implementi tutte le eventuali query continuando a poter usufruire a pieno dei DAO per accedere al database nel resto dell’applicazione.

Qui è riportato un esempio di DAO utilizzato in Biblionet e delle relazioni che ha con altre classi

dell’applicazione.



# Glossario

|  |  |
| --- | --- |
| **Sigla/Termine** | **Definizione** |
| **Package** | Raggruppamento di classi ed interfacce. |
| **DAO** | Data Access Object, implementazione dell’omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti. |
| **Controller** | Classe che si occupa di gestire le richieste effettuate dal client. |
| **Service** | Classe che implementa la logica di business, viene utilizzata dal controller o da un altro sottosistema. |
| **Model** | Parte del design architetturale MVC che fornisce al sistema i metodi per accedere ai dati utili al sistema. |
| **MVC** | Model-View-Controller: design architetturale che permette di separare la logica di presentazione dalla logica di business alla base del sistema. |
| **Facade** | Un oggetto che permette, attraverso un’interfaccia più semplice, l’accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro. |

|  |  |
| --- | --- |
| **Adapter** | È un pattern strutturale che può essere basato sia su classi che su oggetti il cui fine è fornire una soluzione astratta al problema dell’interoperabilità tra interfacce differenti. |
| **Singleton** | È un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga struttura una solo istanza e di fornire un punto di accesso globale a tale istanza. |
| **Observer Pattern** | Design pattern usato come base per la gestione di eventi. |
| **Hibernate** | Piattaforma open source per la gestione della persistenza dei dati sul database. |
| **SRP** | Principio di singola responsabilià: ogni element del programma deve avere una singola responsabilità |