Laurea Triennale in informatica-Università di Salerno
Corso di Ingegneria del *Software*- Prof. C. Gravino

Object Design Document
Progetto
SalernArte

Versione	0.1
Data	28/05/2022
Destinatario	Studenti di Ingegneria del Software 2021/22
Presentato da	Martino Lucia [0512105234], Longo Marco [0512105945], Della Pepa Alessia [0512105720]
Approvato da	

Membri del Team

Nome	Matricola	Acronimo	Informazioni di contatto
Della Pepa Alessia	0512105720	DPA	a.dellapepa5@studenti.unisa.it
Longo Marco	0512105945	LM	m.longo36@studenti.unisa.it
Martino Lucia	0512105234	ML	l.martino11@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
28/05/2022	0.1	Prima stesura: struttura del documento e inserimento del package SalernArte	DPA, ML, LM
30/05/2022	0.2	Inserimento package gestione acquisti e class interface package gestione acquisti	LM
06/06/2022	0.3	Inserimento introduzione, object design trade-off, linee guida per la documentazione dell'interfaccia, introduzione alla sezione packages	DPA, ML, LM
07/06/2022	0.4	Inserimento package Registrazione e Autenticazione. Inserimento class interface Package Registrazione e Autenticazione	ML
09/06/2022	0.5	Aggiunta Class Diagram Utente-Registrato/Scolaresca, package control e model	ML
10/06/2022	0.6	Inserimento package gestione eventi, class interfaces per la gestione eventi e class diagram per gestione eventi	DPA
11/06/2022	0.7	Inserimento class diagram amministratore e organizzatore, package control GestioneAcquisti	LM

13/06/2022	0.8	Inserimento del glossario	LM
17/02/2023	1.0	Revisione documento	DPA, LM

Sommario

1. Introduzione

SalernArte si propone come obiettivo principale di realizzare un'agenzia online specializzata nella vendita di biglietti riguardanti mostre d'arte ed eventi teatrali e culturali nel salernitano. L'obiettivo è quello di facilitare a tutti i cittadini la ricerca di iniziative culturali salernitane, raggruppando queste ultime in un unico ambiente semplice ed intuitivo, e di ottimizzare il lavoro di organizzatori di eventi che si interfacciano ai cittadini.

In questa prima sezione del documento, verranno descritti i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

1.1. Object Design Trade-off

Comprensibilità vs Tempo:

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.

Sicurezza vs Costi:
La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su e-mail e password.

Sviluppo rapido vs Features:
Le funzionalità specifiche dell'applicazione verranno realizzate seguendo un sistema basato su delle priorità. Privilegiando uno sviluppo rapido, verrò data la precedenza agli elementi che dispongono di una priorità alta per poi integrare le restanti funzionalità in un secondo momento.

Riusabilità vs Efficienza
Il sistema deve essere costruito per garantirne la riusabilità, attraverso l'utilizzo di design pattern, e componenti COTS nonostante ciò potrebbe portare ad un aumento dei tempi di risposta.

1.2. Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce.

1.2.1 Nomi dei file

- Le classi devono avere nomi al singolare
- I nomi delle classi devono cominciare con la lettera maiuscola
- I nomi del file sorgente Java devono coincidere con il nome della classe
- I nomi dei file sorgenti Java e in generale dei file devono rappresentare il contenuto del file

1.2.2 Strutture dei file sorgente

Il progetto dovrà essere strutturato nel seguente modo:

- **.idea**
- **src** , contiene tutti i file sorgente
    - **main**
        - **java,** contiene le classi java relative alle componenti Control e Model
        - **webapp,** contiene i file relativi alle componenti View
            - **CSS,** contiene i fogli di stile CSS
            - **JS,** contiene gli script JavaScript
            - **immaginiEventi,** contiene le immagini relative agli eventi
            - **jQuery,** contiene il file jQuery
            - **WEB-INF**
                - **lib,** contiene le librerie utili allo sviluppo del progetto
                - **SQL,** contiene l'istanza del database
                - **autenticazione,** contiene i file HTML relativi all'autenticazione
                - **gestioneAcquisti,** contiene i file HTML relativi alla gestione acquisti

    - **gestioneUtente,** contiene i file HTML relativi alla gestione utente
    - **gestioneEventi,** contiene i file HTML relativi alla gestione eventi
  - **test,** contiene tutto il necessario per il testing
    - **java,** contiene le classi java per l'implementazione del testing

Nello specifico dovranno rispettare le seguenti strutture:

- I file relativi all'implementazione del sistema seguono la seguente struttura: *src/main/java/subsystem/package/file.java*
- I file relativi alla view seguono la seguente struttura: *src/main/webapp/WEB-INF/file.html*
- I file di stile CSS seguono la seguente struttura: *src/main/webapp/CSS/file*
- I file JS seguono la seguente struttura: *src/main/webapp/JS/file*
- I file relativi al testing seguono la seguente struttura: *src/test/java/subsystem/package/fileTest.java*

1.2.3 Nomenclatura

- **Package:** lowerCamelCase, solo lettere
- **Classi:** UpperCamelCase, solo lettere
- **Metodi:** lowerCamelCase, solo lettere
- **Costanti:** COSTANT_CASE, solo lettere e underscore
- **Variabili:** lowerCamelCase, solo lettere
- **Parametri di metodi:** lowerCamelCase

1.2.4. Convenzioni metodi DAO

- I metodi del DAO per salvare istanze nel database seguiranno la nomenclatura: doSave**NomeDellaClasse**
- I metodi del DAO per prelevare istanze nel database basandosi sulla chiave seguiranno la nomenclatura: doRetrieve**NomeDellaClasse**
- I metodi del DAO per prelevare istanze nel database basandosi su un parametro diverso dalla chiave seguiranno la nomenclatura: doRetrieve**NomeDellaClasse**By**Parametro**
- I metodi del DAO per cancellare le istanze nel database seguiranno la nomenclatura: **doDelete**
- I metodi del DAO per cancellare le istanze nel database secondo un determinato parametro seguiranno la nomenclatura: **doDeleteByParametro**

1.3. Definizioni, Acronimi e Abbreviazioni

Acronimo/Abbreviazione	Definizione
Package	Raggruppamento di classi, interfacce o file correlati
Design pattern	Template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità

Interfaccia	Insieme di signature delle operazioni offerte dalla classe
View	Nel pattern MVC rappresenta ciò che viene visualizzato a schermo da un utente e che gli permette di interagire con le funzionalità offerte dalla piattaforma
lowerCamelCase	E' la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi
UpperCamelCase	E' la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi
Javadoc	Sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4. Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- Statement of Work
- Requirements Analysis Document
- System Design Document

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.

Package SalernArte

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

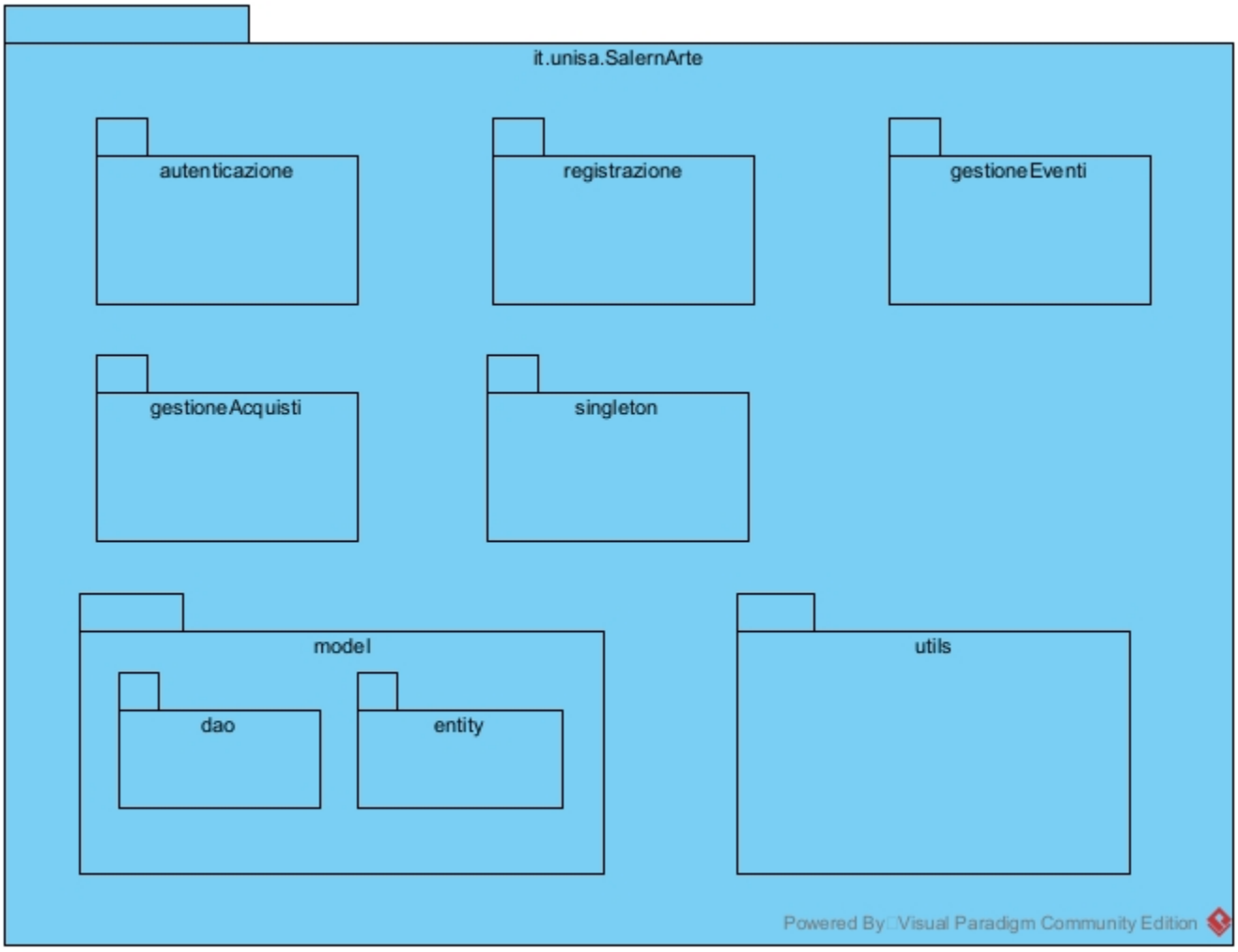
1. Creare un package separato per ogni sottosistema, contenente le classi service e controller del sottosistema, ed eventuali classi di utilità usate unicamente da esso.

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 SalernArte che prevede numerose relazioni

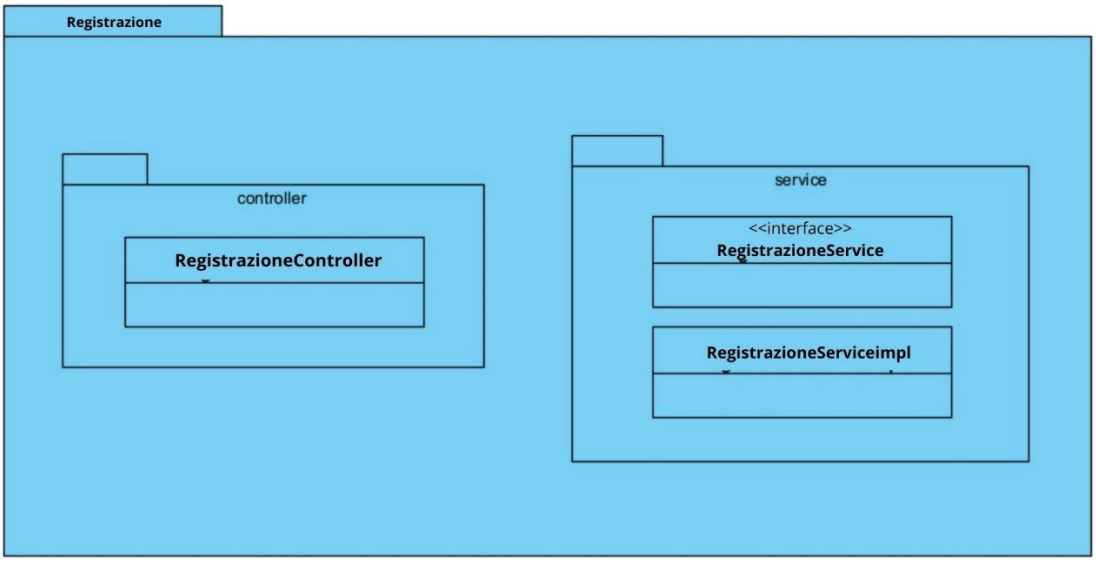
tra le entità. Si è quindi preferito tenere tutto in un package separato e collegato a tutti gli altri package dei sottosistemi.

3. Creare un package chiamato *utils* in cui inserire eventuli classi di utlità per il sistema e usabili da più sotto-sistemi.

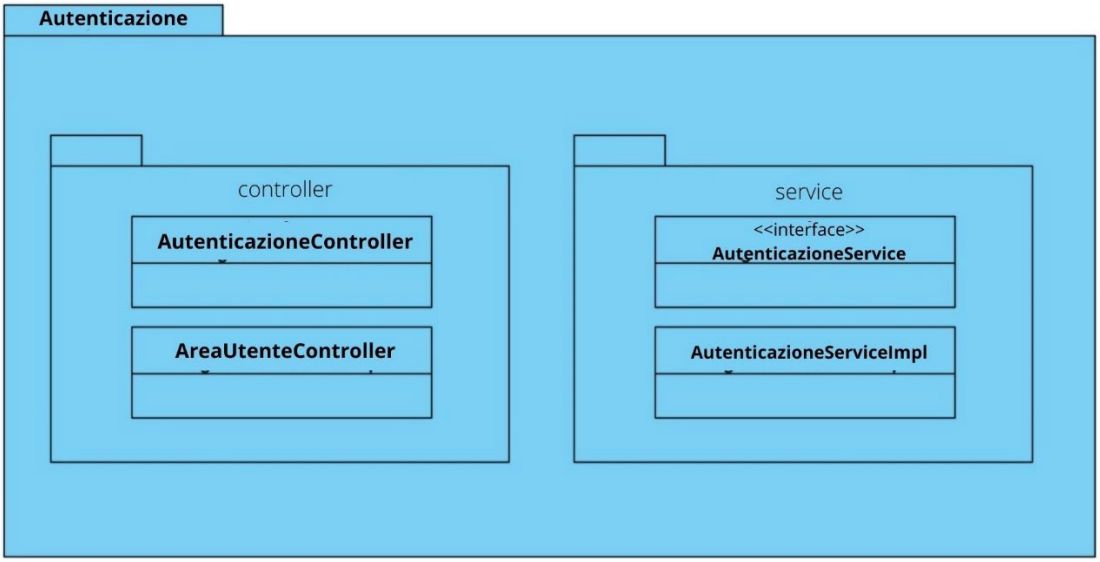
Per ciò che concerne la dipendenza tra i packages, la suddivisione precedentemente illustrata a portato alla creazione di una relazione tra il package model e tutti gli altri package del sistema.



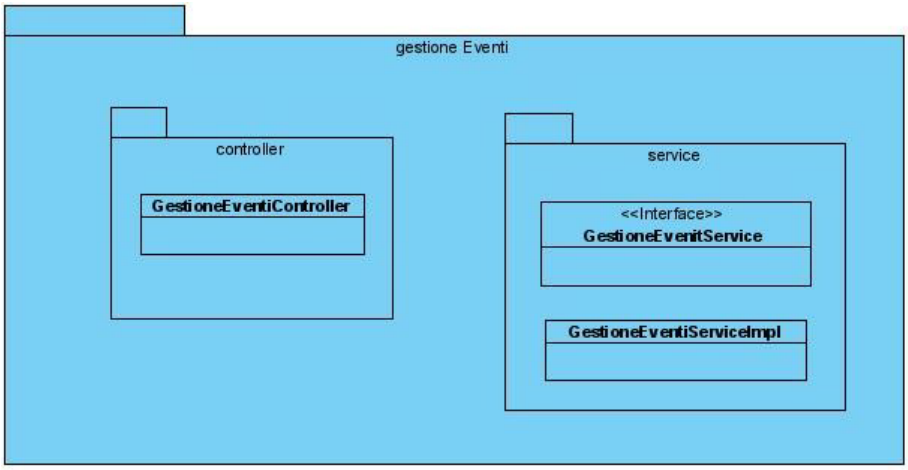
Package Registrazione



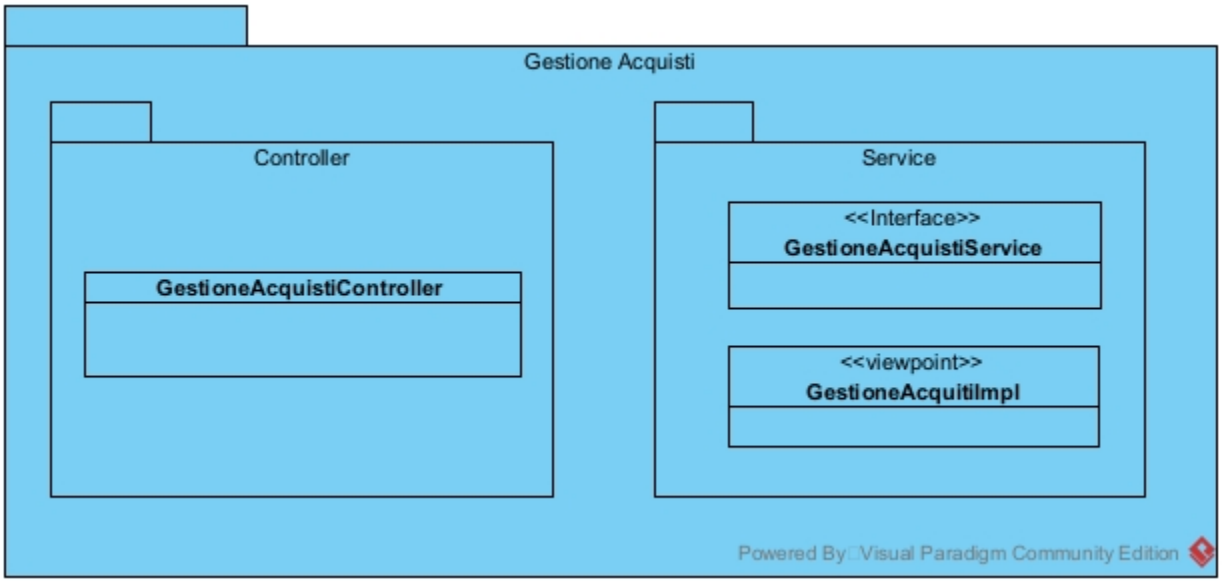
Package Autenticazione



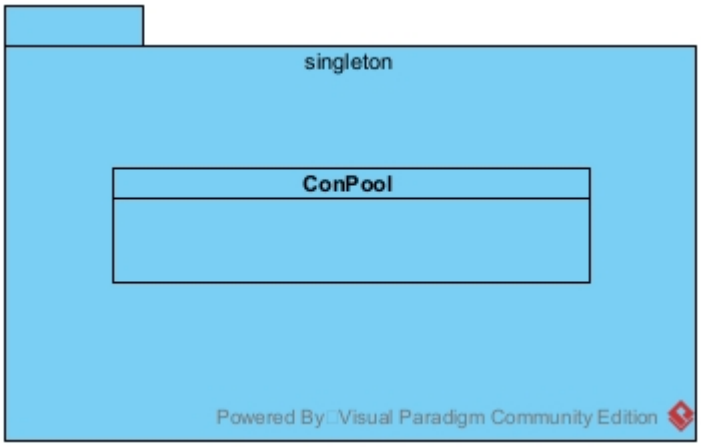
Package gestioneEventi



Package Gestione Acquisti



Package singleton



3. Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package:

Package Registrazione

		Autore	ML
Nome Classe	**RegistrazioneService**		
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione		
Metodi	+registraOspite(Ospite ospite): UtenteRegistrato +registraScolaresca(Scolaresca scolaresca): UtenteRegistrato + registraAdmin(Amministratore amministratore): UtenteRegistrato		
Invariante di classe	/		
Nome Metodo	+registraOspite(Ospite ospite): UtenteRegistrato		
Descrizione	Questo metodo consente la registrazione di un ospite		
Pre-condizione	/		
Post-condizione	context: RegistrazioneService:: registraOspite(Ospite ospite) post: OspiteDAO.save(ospite)==true		
Nome Metodo	+registraScolaresca(Scolaresca scolaresca): UtenteRegistrato		
Descrizione	Questo metodo consente la registrazione di una scolaresca		

Pre-condizione	/
Post-condizione	context: RegistrazioneService:: registraScolaresca(Scolaresca scolaresca) post: ScolarescaDAO.save(ospite)==true
Nome Metodo	+registraAdmin(Amministratore amministratore): UtenteRegistrato
Descrizione	Questo metodo consente la registrazione di un admin
Pre-condizione	/
Post-condizione	**context:** RegistrazioneService:: registraAdmin(Amministratore amministratore) **post:** AmministratoreDAO.save(ospite)==true

Package Autenticazione

	Autore	**ML**
Nome Classe	**AutenticazioneService**	
Descrizione	Questa classe permette di gestire le operazioni relative all'autenticazione	
Metodi	+login(String email, String password): UtenteRegistrato +aggiornaUtenteRegistrato(UtenteRegistrato utenteRegistrato):UtenteRegistrato +aggiornaScolaresca(Scolaresca scolaresca):Scolaresca +aggiornaAdmin(Amministratore amministratore):Admin	
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)
Nome Metodo	+aggiornaUtenteRegistrato(UtenteRegistrato utenteRegistrato)
Descrizione	Questo metodo consente di aggiornare i dati di un utente registrato
Pre-condizione	/
Post-condizione	/

Nome Metodo	+aggiornaScolaresca(Scolaresca scolaresca)
Descrizione	Questo metodo consente di aggiornare i dati di una scolaresca
Pre-condizione	/
Post-condizione	/
Nome Metodo	+aggiornaAdmin(Amministratore amministratore)
Descrizione	Questo metodo consente di aggiornare i dati di un amministratore
Pre-condizione	/
Post-condizione	/

Package Gestione Eventi

	Autore	**DPA**
Nome Classe	**GestioneEventiService**	
Descrizione	Questa classe permette di gestire le operazioni relative alla gestione degli eventi	
Metodi	+richiediInserimentoEvento(int idOrganizzatore, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void +richiediModificaEvento(int idEvDaModificare, UtenteRegistratoBean utenteLoggato, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void +attivaEvento(int idEvento,String tipoUtente): void +rimuoviEvento(int idEvento,UtenteRegistratoBean utente):void +retrieveEventoById(int idEvento); EventoBean +retriveAllRichiesteEventi(String tipoUtente): List<EventoBean> +checkQuantitaCarrello(EventoBean evento, CarrelloBean carrelloSessione): void +checkScaduta(EventoBean evento): boolean +getPrezzoEvento (int idEvento): double +retrieveRichiesteInserimento(String tipoUtente): List<EventoBean> +retrieveRichiesteModifica(String tipoUtente); List<EventoBean> +accettaModifica(int idEvento,String tipoUtente): void +rifiutaModifica(int idEvento,String tipoUtente): void	

	+retrieveEventiOrganizzatore(UtenteRegistratoBean utente): List<EventoBean> +ricercaEventiByNomeOrDescrizione(String query): List<EventoBean>
Invariante di classe	/

Nome Metodo	**+richiediInserimentoEvento(int idOrganizzatore, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void**
Descrizione	Questo metodo consente di inviare una richiesta di inserimento dell'evento da parte dell'organizzatore all'amministratore
Pre-condizione	**Context:** GestioneEventi :: richiediInserimentoEvento(int idOrganizzatore, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void **pre:** dataFine.after(dataInizio) && dataInizio.after(dataAttuale) && nome != null && nome.length() != 0 && descrizione != null && descrizione.length() != null && numBiglietti >0 && prezzoBiglietto > 0 && indirizzo != null && indirizzo.length() != 0 && sede != null && sede.length() != 0 &&
Post-condizione	/
Nome Metodo	**+richiediModificaEvento(int idEvDaModificare, UtenteRegistratoBean utenteLoggato, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void**
Descrizione	Questo metodo consente di inviare una richiesta di modifica dell'evento da parte dell'organizzatore all'amministratore
Pre-condizione	**Context:** GestioneEventi :: richiediModificaEvento(int idEvDaModificare, UtenteRegistratoBean utenteLoggato, String nome, String tipoEvento, String descrizione, String pathContext, Part filePhoto, int numBiglietti, double prezzoBiglietto, Date dataInizio, Date dataFine, String indirizzo, String sede): void **pre:** dataFine.after(dataInizio) && nome != null && nome.length() != 0 && descrizione != null && descrizione.length() != null && numBiglietti >0 && prezzoBiglietto > 0 && indirizzo != null && indirizzo.length() != 0 && sede != null && sede.length() != 0
Post-condizione	/
Nome Metodo	**+attivaEvento(int idEvento,String tipoUtente): void**

Descrizione	Questo metodo consente di attivare un evento e inserirlo nella lista degli eventi attivi
Pre-condizione	**Context:** GestioneEventi :: attivaEvento(int idEvento,String tipoUtente): void **pre:** tipoUtente.compareTo("amministratore) == 0
Post-condizione	/
Nome Metodo	**+rimuoviEvento(int idEvento,UtenteRegistratoBean utente):void**
Descrizione	Questo metodo consente di rimuovere un evento
Pre-condizione	**Context:** GestioneEventi :: rimuoviEvento(int idEvento,UtenteRegistrato-Bean utente):void **pre:** tipoUtente.compareTo("amministratore) == 0
Post-condizione	/
Nome Metodo	**+retrieveEventoById(int idEvento); EventoBean**
Descrizione	Questo metodo cerca l'evento con id = idEvento nel database
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+retriveAllRichiesteEventi(String tipoUtente): List<EventoBean>**
Descrizione	Questo metodo restituisce tutti gli eventi richiesti
Pre-condizione	**Context:** GestioneEventi :: retriveAllRichiesteEventi(String tipoUtente): List<EventoBean> **pre:** tipoUtente.compareTo("amministratore") == 0 && tipoUtente != null
Post-condizione	/
Nome Metodo	**+checkQuantitaCarrello(EventoBean evento, CarrelloBean carrello-Sessione): void**
Descrizione	Questo metodo consente di rimuovere un evento
Pre-condizione	**Context:** GestioneEventi :: checkQuantitaCarrello(EventoBean evento, CarrelloBean carrelloSessione): void **pre:** carrelloSessione != null
Post-condizione	/
Nome Metodo	**+checkScaduta(EventoBean evento): boolean**

Descrizione	Questo metodo consente di modificare lo stato di un evento da attivo a scaduto
Pre-condizione	**Context:** GestioneEventi :: checkScaduta**(EventoBean evento): boolean** **pre:** evento.getDataFine().before(dataAttuale)
Post-condizione	/
Nome Metodo	**+getPrezzoEvento (int idEvento): double**
Descrizione	Questo metodo restituisce il prezzo dell'evento
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+retrieveRichiesteInserimento(String tipoUtente): List<EventoBean>**
Descrizione	Questo metodo restituisce tutte le richieste di inserimento degli eventi inviate dall'organizzatore all'amministratore
Pre-condizione	**Context:** GestioneEventi :: retrieveRichiesteInserimento(String tipoUtente): List<EventoBean> **pre:** tipoUtente.compareTo("amministratore") == 0 && tipoUtente != null
Post-condizione	/
Nome Metodo	**+retrieveRichiesteModifica(String tipoUtente); List<EventoBean>**
Descrizione	Questo metodo restituisce tutte le richieste di modifica degli eventi inviate dall'organizzatore all'amministratore
Pre-condizione	**Context:** GestioneEventi :: retrieveRichiesteModifica(String tipoUtente); List<EventoBean> **pre:** tipoUtente.compareTo("amministratore") == 0 && tipoUtente != null
Post-condizione	/
Nome Metodo	**+accettaModifica(int idEvento,String tipoUtente): void**
Descrizione	Questo metodo accetta la modifica dell'evento
Pre-condizione	**Context:** GestioneEventi :: accettaModifica(int idEvento,String tipoUtente): void **pre:** tipoUtente.compareTo("amministratore") == 0
Post-condizione	/
Nome Metodo	**+rifiutaModifica(int idEvento,String tipoUtente): void**
Descrizione	Questo metodo rifiuta la modifica di un evento
Pre-condizione	**Context:** GestioneEventi :: rifiutaModifica(int idEvento,String tipoUtente): void

	pre: tipoUtente.compareTo("amminstratore") == 0
Post-condizione	/

Nome Metodo	**+retrieveEventiOrganizzatore(UtenteRegistratoBean utente): List<EventoBean>**
Descrizione	Questo metodo restituisce tutti gli eventi dell'organizzatore
Pre-condizione	**Context:** GestioneEventi :: retrieveEventiOrganizzatore(UtenteRegistrato-Bean utente): List<EventoBean> **pre:** utente.getTipoUtente("organizzatore") == 0 && utente != null
Post-condizione	/

Nome Metodo	**+ricercaEventiByNomeOrDescrizione(String query): List<Even-toBean>**
Descrizione	Questo metodo cerca gli eventi inserendo nome o la descrizione
Pre-condizione	/
Post-condizione	/

Package Gestione Acquisti

		Autore	**LM**
Nome Classe	**GestioneAcquistiService**		
Descrizione	Questa classe permette di gestire le operazioni riguardanti gli acquisti degli eventi		
Metodi	+acquistaProdotti(CarrelloBean carrelloSessione, UtenteRegistratoBean utente): void +updateQuantitàCarrello(int idE, int quantità, CarrelloBean carrellosessione, UtenteRegistratoBean utente): void +aggiungiAlCarrello(int idE, int quantità, CarrelloBean carrelloSessione, UtenteRegistratoBean utente): CarrelloBean +removeEventoFromCarrello(int idE, CarrelloBean carrello, UtenteRegistra-toBean utente): void +svuotaCarrello (CarrelloBean carrello, UtenteRegistratoBean utente): void +controlloElementiCarrello(CarrelloBean carrelloSessione, UtenteRegistrato-Bean utente): boolean +retrieveCarrelloUtente (UtenteRegistratobean utente): CarrelloBean		

Invariante di classe	/

Nome Metodo	+ acquistaProdotti(CarrelloBean carrelloSessione, UtenteRegistratoBean utente): void
Descrizione	Questo metodo permette di acquistare gli eventi nel carrello.
Pre-condizione	**context:** Gestione Acquisti :: acquistaProdotti(CarrelloBean carrelloSessione, UtenteRegistratoBean utente): **Pre:** Utente != null || carrelloSessione != null || !(carrelloSessione.getProdotti().isEmpty())
Post-condizione	**context:** Gestione Acquisti :: acquistaProdotti(CarrelloBean carrelloSessione, UtenteRegistratoBean utente): **post:** evento.getNumBiglietti() == @pre evento.getNumBiglietti()-1

Nome Metodo	+ updateQuantitàCarrello(int idE, int quantità, CarrelloBean carrellosessione, UtenteRegistratoBean utente): void
Descrizione	Questo metodo permette di modificare la quantità di ogni singolo evento del carrello
Pre-condizione	**Context**: Gestione Acquisti :: updateQuantitàCarrello(int idE, int quantità, CarrelloBean carrellosessione, UtenteRegistratoBean utente): **Pre**: Quantità <= evento.getNumBiglietti || quantità > 0 || utente != null
Post-condizione	/

Nome Metodo	+ aggiungiAlCarrello(int idE, int quantità, CarrelloBean carrelloSessione, UtenteRegistratoBean utente): CarrelloBean
Descrizione	Questo metodo permette di aggiungere un evento al carrello
Pre-condizione	**Context**: Gestione Acquisti :: aggiungiAlCarrello(int idE, int quantità, CarrelloBean carrelloSessione, UtenteRegistratoBean utente): **Pre**: idE > 0 && quantità > 0 && utente != null && carrelloSessione != null
Post-condizione	/

Nome Metodo	+ removeEventoFromCarrello(int idE, CarrelloBean carrello, UtenteRegistratoBean utente): void
Descrizione	Questo metodo permette di rimuovere un evento dal carrello
Pre-condizione	**Context**: Gestione Acquisti :: removeEventoFromCarrello(int idE, int quantità, CarrelloBean carrelloSessione, UtenteRegistratoBean utente):

	Pre: idE > 0 && eventoDao.doRetrieveById(idE) != null && carrello.get(idE) != null && utente != null
Post-condizione	/

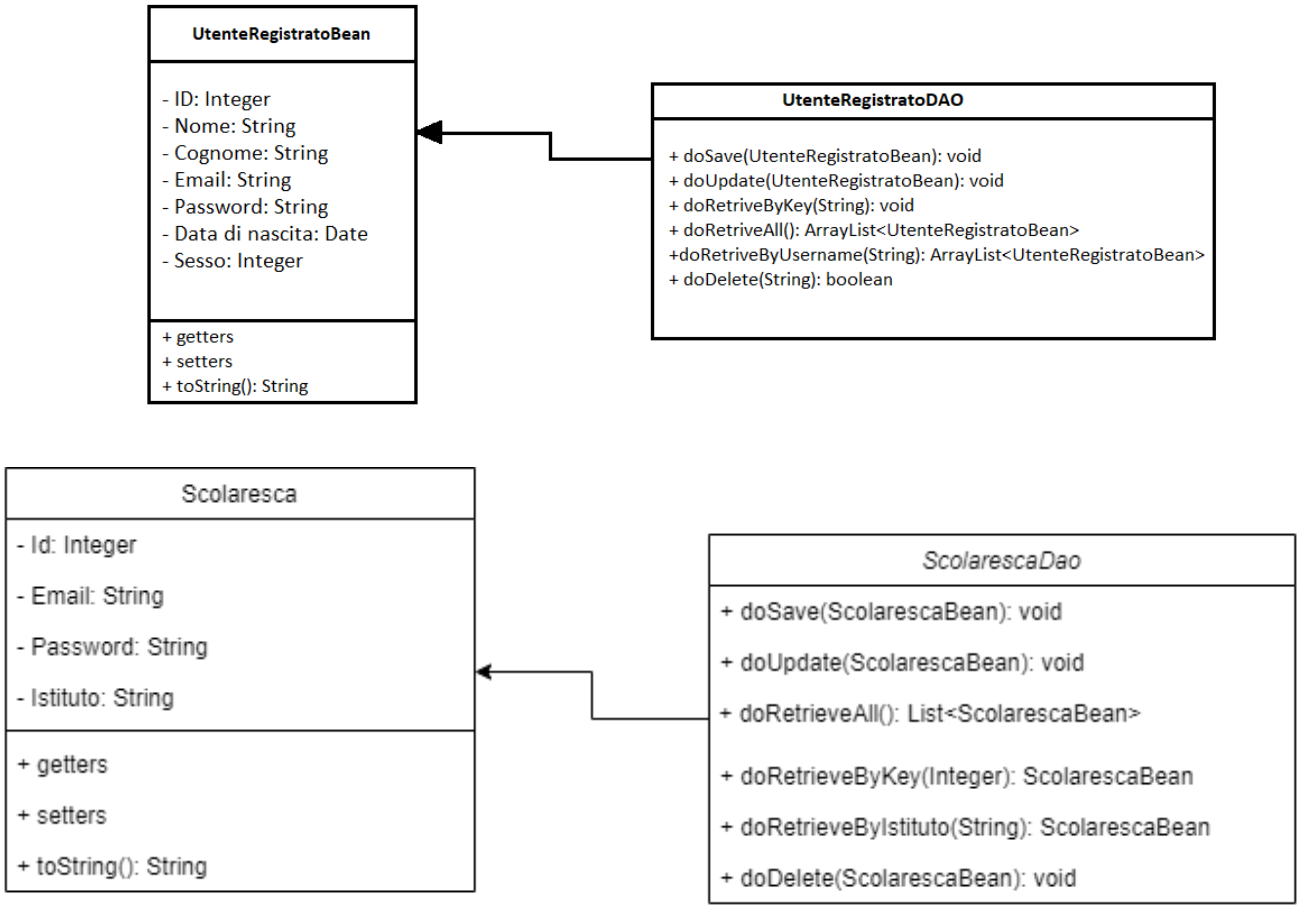
Nome Metodo	+ svuotaCarrello (CarrelloBean carrello, UtenteRegistratoBean utente): void
Descrizione	Questo metodo permette di svuotare il carrello
Pre-condizione	**Context:** Gestione Acquisti :: svuotaCarrello (CarrelloBean carrello, Utente-RegistratoBean utente): **Pre:** utente.getId()!=null
Post-condizione	/

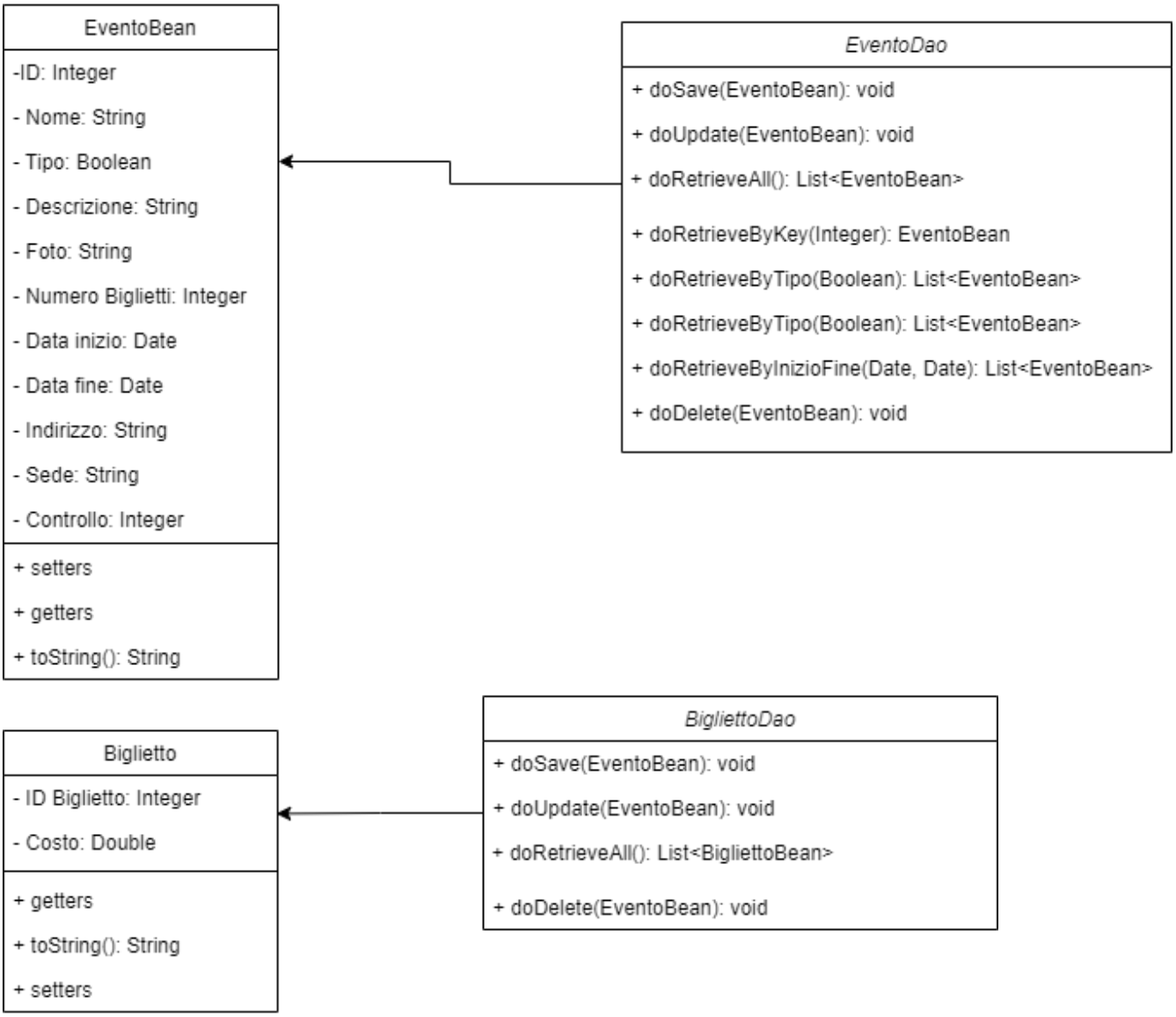
Nome Metodo	+ controlloElementiCarrello(CarrelloBean carrelloSessione, UtenteRegistra-toBean utente): boolean
Descrizione	Questo metodo permette di rimuovere un evento dal carrello
Pre-condizione	**Context:** Gestione Acquisti :: controlloElementiCarrello(CarrelloBean car-relloSessione, UtenteRegistratoBean utente): **Pre**: carrelloSessione.getProdotti().isEmpty() != null
Post-condizione	/

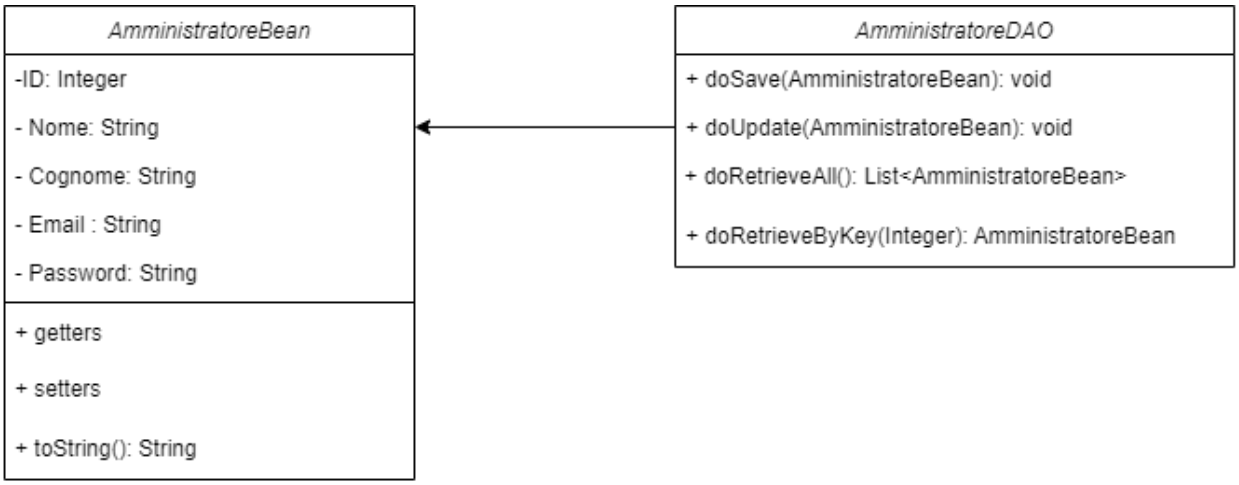
Nome Metodo	+ retrieveCarrelloUtente (UtenteRegistratobean utente): CarrelloBean
Descrizione	Questo metodo permette di recuperare il carrello dell'utente
Pre-condizione	**Context:** Gestione Acquisti :: retrieveCarrelloUtente (UtenteRegistratobean utente): **Pre:** utente.getId()!=null
Post-condizione	/

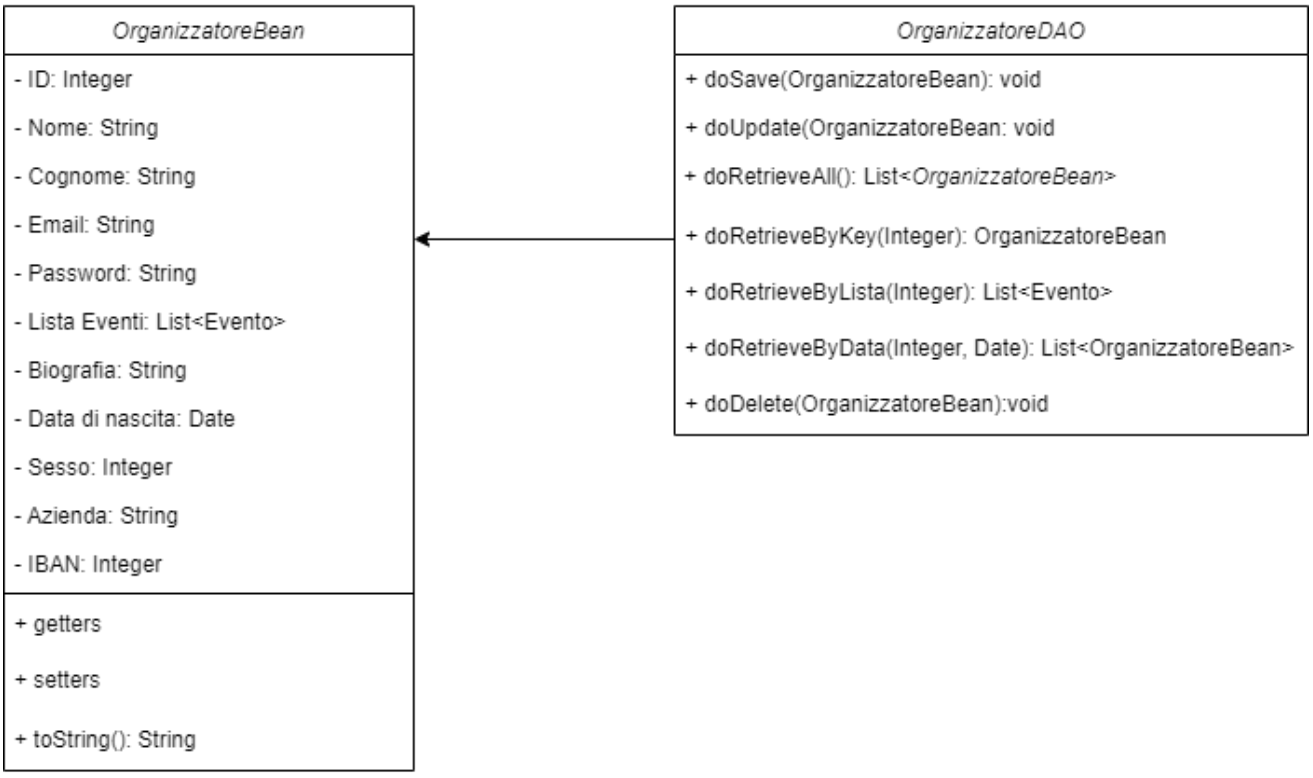
3 Class Diagram

3.1 Package model



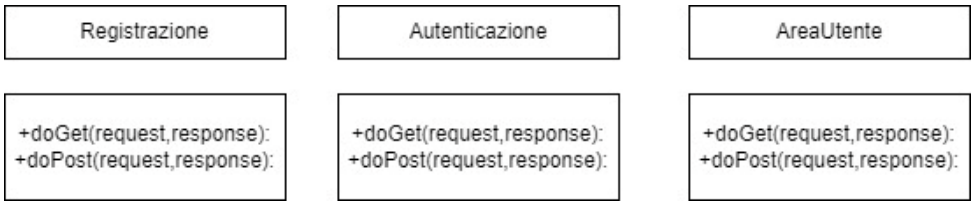

EventoBean
-ID: Integer
- Nome: String
- Tipo: Boolean
- Descrizione: String
- Foto: String
- Numero Biglietti: Integer
- Data inizio: Date
- Data fine: Date
- Indirizzo: String
- Sede: String
- Controllo: Integer
+ setters
+ getters
+ toString(): String
EventoDao
+ doSave(EventoBean): void
+ doUpdate(EventoBean): void
+ doRetrieveAll(): List<EventoBean>
+ doRetrieveByKey(Integer): EventoBean
+ doRetrieveByTipo(Boolean): List<EventoBean>
+ doRetrieveByTipo(Boolean): List<EventoBean>
+ doRetrieveByInizioFine(Date, Date): List<EventoBean>
+ doDelete(EventoBean): void
Biglietto
- ID Biglietto: Integer
- Costo: Double
+ getters
+ toString(): String
+ setters
BigliettoDao
+ doSave(EventoBean): void
+ doUpdate(EventoBean): void
+ doRetrieveAll(): List<BigliettoBean>
+ doDelete(EventoBean): void


AmministratoreBean
-ID: Integer
- Nome: String
- Cognome: String
- Email : String
- Password: String
+ getters
+ setters
+ toString(): String
AmministratoreDAO
+ doSave(AmministratoreBean): void
+ doUpdate(AmministratoreBean): void
+ doRetrieveAll(): List<AmministratoreBean>
+ doRetrieveByKey(Integer): AmministratoreBean


OrganizzatoreBean
- ID: Integer
- Nome: String
- Cognome: String
- Email: String
- Password: String
- Lista Eventi: List<Evento>
- Biografia: String
- Data di nascita: Date
- Sesso: Integer
- Azienda: String
- IBAN: Integer
+ getters
+ setters
+ toString(): String
OrganizzatoreDAO
+ doSave(OrganizzatoreBean): void
+ doUpdate(OrganizzatoreBean: void
+ doRetrieveAll(): List<OrganizzatoreBean>
+ doRetrieveByKey(Integer): OrganizzatoreBean
+ doRetrieveByLista(Integer): List<Evento>
+ doRetrieveByData(Integer, Date): List<OrganizzatoreBean>
+ doDelete(OrganizzatoreBean):void

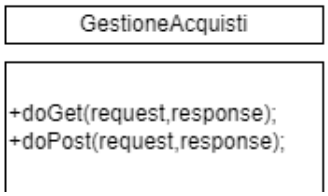
3.2 Package control

[Autore:ML]



[Autore:DPA]

[Autore:LM]

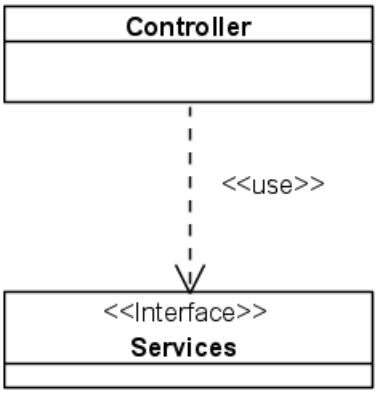


4 Design Patterns

Il primo design pattern scelto è il **Façade**. Il pattern è basato sull'utilizzo di una classe che consente, tramite un'interfaccia più semplice, l'accesso ad un sottosistema che contiene classi con interfacce complesse e diverse tra loro. Questo design pattern offre due vantaggi:

- La riduzione del numero di associazioni
- Semplicità nell'attuazione di cambiamenti

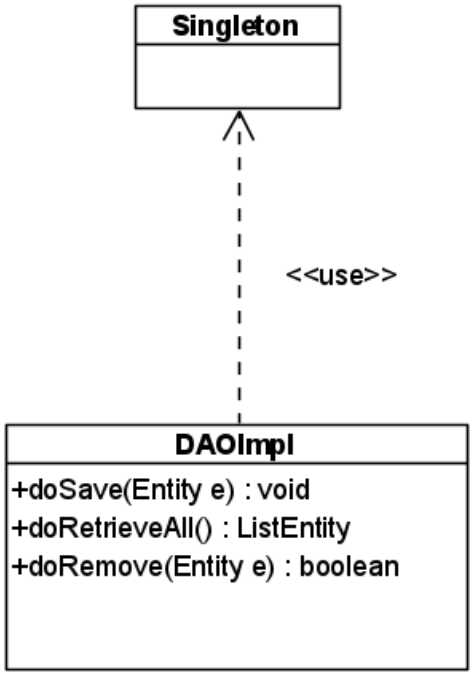
Ogni package è caratterizzato da un sub-package chiamato "services", contenente le classi che implementano i metodi corrispondenti ai servizi offerti dal sottosistema. Questi metodi usufruiranno delle classi situate nei package "model" e "dao". Attraverso questa strutturazione dei package, si ottiene una separazione tra logica di business e logica di controllo.
Di seguito è mostrato un esempio d'uso del design pattern *Façade*.
Il diagramma mostra il collegamento tra Controller e Services grazie all'utilizzo del suddetto design pattern.



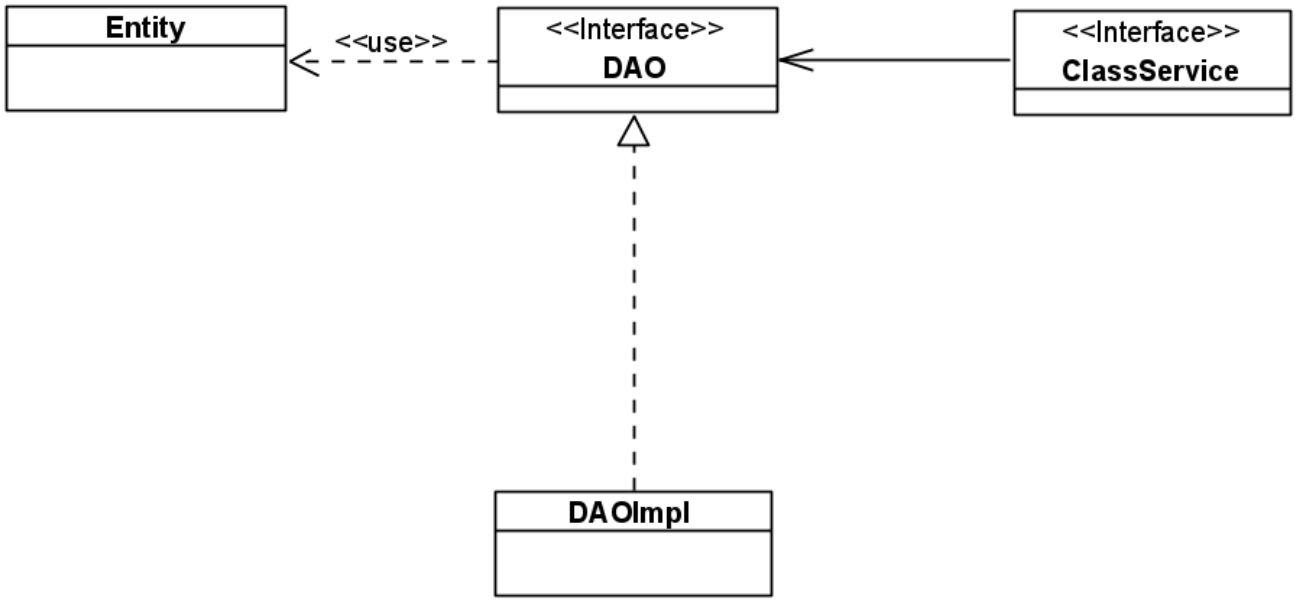
Il secondo design pattern individuato è il **Singleton**, che appartiene alla categoria dei design pattern "creazionali" e ha lo scopo di garantire che una classe possa essere istanziata una sola volta, e di fornire un punto di accesso globale a tale istanza. Inoltre, integra due nuove funzionalità:

- Ha un costruttore privato
- Definisce un operatore public static che viene impiegato per accedere al singleton

Questo design pattern verrà utilizzato nel nostro sistema, per permettere la connessione al database da parte delle classi situate nei package "dao".



Il terzo e ultimo design pattern individuato è il **DAO,** utilizzato per gestire la logica di accesso ai dati persistenti. Di seguito è mostrato un esempio d'uso del design pattern DAO.
Il diagramma illustra come il suddetto design pattern nasconda alla ClasseService le implementazioni delle operazioni di accesso ai dati persistenti, implementando il concetto di "loosely coupled".



5 Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di classi e interfacce
DAO	Data Access Object, design pattern architetturale che si occupa di fornire in modo astratto l'accesso ai dati persistenti
Controller	Classe che gestisce le richieste effettuate dal client
Service	Classe che implementa la logica di business, utilizzata dal controller o altro sottosistema
Model	Parte dell'MVC che fornisce i metodi per accedere ai dati utili al sistema
MVC	Model View Controller, design architetturale che separa la logica di presentazione dalla logica di business
Facade	Classe che utilizza un'interfaccia semplice per accedere ai metodi di un'interfaccia complessa
Singleton	Design pattern creazione con lo scopo di strutturare una sola istanza di una classe fornendo un punto di accesso globale